Information Theory, Inference, and Learning Algorithms

David J.C. MacKay

©1995, 1996, 1997, 1998, 1999, 2000

Draft 2.0.7 February 14, 2000

Part III Physics

Information Theory, Pattern Recognition and Neural Networks

This is the course textbook. It contains the handouts for the lectures (roughly one chapter per lecture), and some additional chapters on related topics that you may find interesting. Sections that are marked '*' are not part of the course.

Note-taking: During lectures, I encourage you to take notes, rather than follow along in the textbook. Learning is an active process, not a passive one. I aim to lecture at the right pace so that you can take notes.

Exercises: The book contains numerous exercises, complete with worked solutions. A subset of these exercises will be designated as the exercises you should work on before supervisions. [Unless otherwise indicated, these exercises are the ones marked 'A'.] *I encourage you not to look at the worked solutions for these exercises before you have attempted them yourself.*

Supervisions: Officially, you are meant to get three supervisions in small groups. However, feedback from students indicates a preference for *six* supervisions in larger groups. Assuming that you would like this, the supervisions will be on Thursdays starting in week 3. Please attend one supervision per week, and sign the attendance register each time.

2pm. Thursday 10/2, 17/2, 24/2, 2/3, 9/3, and **3pm** on 16/3.
4.30pm. Thursday 10/2, 17/2, 24/2, 2/3, 9/3, 16/3.
Location: Ryle seminar room, Rutherford building (Room 930).

Web page: `http://wol.ra.phy.cam.ac.uk/mackay/itprnn/`.

If you have any questions about the course or the textbook, please try the automatic question–answering system.

Your feedback on all aspects of the course is welcomed.

Please let me know of any errors you find in the book, and give me suggestions for how to make it clearer.


```
David J.C. MacKay   email: mackay@mrao.cam.ac.uk
                      www: http://wol.ra.phy.cam.ac.uk/mackay/
tel: 339852          room: 961 Rutherford Building
```


© David J.C. MacKay: Draft 2.0.7. February 14, 2000

Contents

	Part III Physics	1
1	Introduction to Information Theory	8
	Solutions to chapter 1's exercises	22
2	Probability, Entropy and Inference	27
	Solutions to Chapter 2's exercises	43
3	More about Inference	51
	Solutions to Chapter 3's exercises	61
Part I	**Data Compression**	**64**
4	The Source Coding Theorem	66
	Solutions to Chapter 4's exercises	84
5	Symbol Codes	90
	Solutions to Chapter 5's exercises	105
6	Stream Codes	113
	Solutions to Chapter 6's exercises	129
7	Further exercises on data compression	138
	Solutions to Chapter 7's exercises	142
	Codes for integers*	146
Part II	**Noisy Channel Coding**	**153**
8	Correlated random variables	154
	Solution to Chapter 8's exercises	158
9	Communication over a noisy channel	163

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 9's exercises 176

10 The Noisy Channel Coding Theorem 183

Solutions to Chapter 10's exercises 193

11 Error Correcting Codes & Real Channels 195

Solutions to Chapter 11's exercises 208

12 Linear Error Correcting Codes and Perfect Codes* 211

Part III Further Topics in Information Theory 215

13 Units of information content* 216

14 Language models and crosswords* 220

15 Hash Codes: Codes for Efficient Information Retrieval 224

Solutions to Chapter 15's exercises 233

16 Cryptography and cryptanalysis: codes for information concealment* 237

17 Communication over Constrained Noiseless Channels* 238

Solutions to Chapter 17's exercises 248

18 Further exercises on information theory 251

Solutions to Chapter 18's exercises 258

Part IV Probabilities and Inference 259

19 Introduction to Part IV 260

20 Exact Inference Methods 262

More on trellises 269

Solutions to Chapter 20's exercises 272

21 Intermediate Bayesian Stuff 273

Solutions to Chapter 21's exercises 280

22 Laplace's method* 284

23 Monte Carlo methods 286

Solutions to Chapter 23's exercises 318

24 Ising Models 321

Solutions to Chapter 24's exercises 336

25 Exact Monte Carlo Sampling* 337

26 Variational Methods 341

Solutions to Chapter 26's exercises 352

27 Bayesian Inference and Sampling Theory 354

28 Further exercises on inference 355

Part V Neural networks 356

29 Introduction to neural networks 357

30 The single neuron as a classifier 360

Solutions to Chapter 30's exercises 371

31 Capacity of a single neuron 373

Solutions to Chapter 31's exercises 382

32 Learning as Inference 383

Solutions to Chapter 32's exercises 397

33 The Hopfield network 399

Solutions to Chapter 33's exercises 413

34 From Hopfield networks to Boltzmann machines 414

35 Supervised learning in multilayer networks 419

36 Complexity control and model comparison 426

37 Approximate Methods for Handling Hyperparameters 427

38 Gaussian processes 454

39 Deconvolution 455

Part VI Unsupervised learning and latent variable models 459

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

40 Hidden Markov Models 461

41 Clustering 465

42 Latent Variable Modelling and the E-M algorithm 467

43 Continuous latent variable models 474

44 Helmholtz machines 481

45 A single neuron as an unsupervised learning element 482

Part VII Possible chapters, uncertain where to put 483

46 Valiant, PAC 484

47 NP completeness 485

48 More about Graphical models and belief propagation 486

Part VIII Tutti: unifying chapters 491

49 'Bits back' 492

Part IX Sparse Graph Codes 493

50 Introduction to sparse graph codes 494

51 Low density parity check codes 497

52 Convolutional codes 498

53 Turbo codes 508

Part X Appendices 513

A Useful formulae, etc. 514

Bibliography 526

About Chapter 1

I hope you will find the mathematics in the first chapter easy. You will need to be familiar with the binomial distribution. And to solve the exercises in the text — which I urge you to do — you will need to remember Stirling's approximation for the factorial function, $x! \simeq x^x e^{-x}$, and be able to apply it to $\binom{N}{r} = \frac{N!}{(N-r)!r!}$. These topics are reviewed below.

The binomial distribution

Example 0.1: A bent coin has probability f of coming up heads. The coin is tossed N times. What is the probability distribution of the number of heads, r? What are the mean and variance of r?

Solution: The number of heads has a binomial distribution.

$$P(r|f,N) = \binom{N}{r} f^r (1-f)^{N-r} \tag{0.1}$$

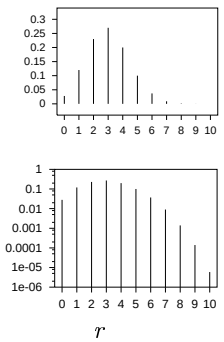


Figure 0.1. The binomial distribution $P(r|f{=}0.3, N{=}10)$, on a linear scale (top) and a logarithmic scale (bottom).

The mean, $\mathcal{E}[r]$, and variance, $\mathrm{var}[r]$, of this distribution are defined by

$$\mathcal{E}[r] \equiv \sum_{r=0}^{N} P(r|f,N)\, r \tag{0.2}$$

$$\mathrm{var}[r] \equiv \mathcal{E}\left[(r - \mathcal{E}[r])^2\right] \tag{0.3}$$

$$= \mathcal{E}[r^2] - (\mathcal{E}[r])^2 = \sum_{r=0}^{N} P(r|f,N) r^2 - (\mathcal{E}[r])^2 . \tag{0.4}$$

Rather than evaluating the sums over r directly, it is easiest to obtain the mean and variance by noting that r is the sum of N *independent* random variables, namely, the number of heads (zero or one) in the first toss, the number of heads in the second toss, and so forth. In general,

$$\begin{array}{rcll} \mathcal{E}[x+y] & = & \mathcal{E}[x] + \mathcal{E}[y] & \text{for any random variables } x \text{ and } y. \\ \mathrm{var}[x+y] & = & \mathrm{var}[x] + \mathrm{var}[y] & \text{if } x \text{ and } y \text{ are independent.} \end{array} \tag{0.5}$$

So the mean of r is the sum of the means of those random variables, and the variance of r is the sum of their variances. The mean number of heads in a single toss is $f \times 1 + (1-f) \times 0 = f$, and the variance of the number of heads in a single toss is

$$\left[f \times 1^2 + (1-f) \times 0^2\right] - f^2 = f - f^2 = f(1-f), \tag{0.6}$$

so the mean and variance of r are:

$$\mathcal{E}[r] = Nf \qquad \text{and} \qquad \mathrm{var}[r] = Nf(1-f). \tag{0.7}$$

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Approximating $x!$ and $\binom{N}{r}$

Let's derive Stirling's approximation by an unconventional route. We start from the Poisson distribution,

$$P(r|\lambda) = e^{-\lambda}\frac{\lambda^r}{r!} \quad r \in \{0, 1, 2, \ldots\}. \tag{0.8}$$

For large λ, this distribution is well approximated — at least in the vicinity of $r \simeq \lambda$ — by a Gaussian distribution with mean λ and variance λ:

$$e^{-\lambda}\frac{\lambda^r}{r!} \simeq \frac{1}{\sqrt{2\pi\lambda}} e^{-\frac{(r-\lambda)^2}{2\lambda}}. \tag{0.9}$$

Let's plug $r = \lambda$ into this formula.

$$e^{-\lambda}\frac{\lambda^\lambda}{\lambda!} \simeq \frac{1}{\sqrt{2\pi\lambda}} \tag{0.10}$$

$$\Rightarrow \lambda! \simeq \lambda^\lambda e^{-\lambda}\sqrt{2\pi\lambda}. \tag{0.11}$$

This is Stirling's approximation, including several of the correction terms that are usually forgotten.

$$x! \simeq x^x e^{-x}\sqrt{2\pi x} \quad \Leftrightarrow \quad \ln x! \simeq x \ln x - x - \tfrac{1}{2}\ln 2\pi x. \tag{0.12}$$

We can use this approximation to approximate $\binom{N}{r} \equiv \frac{N!}{(N-r)!r!}$.

$$\ln \binom{N}{r} \simeq (N-r)\ln\frac{N}{N-r} + r\ln\frac{N}{r}. \tag{0.13}$$

Since all the terms in this equation are logarithms, this result can be rewritten in any base. We will denote natural logarithms ($\log_e$) by 'ln', and logarithms to base 2 ($\log_2$) by 'log'.

If we introduce the *binary entropy function*,

$$H_2(x) \equiv x \log \frac{1}{x} + (1-x)\log\frac{1}{(1-x)} \tag{0.14}$$

then we can rewrite the approximation (0.13) as

$$\log \binom{N}{r} \simeq N H_2(r/N), \tag{0.15}$$

or, equivalently,

$$\binom{N}{r} \simeq 2^{N H_2(r/N)}. \tag{0.16}$$

If we need a more accurate approximation, we can include terms of the next order:

$$\log \binom{N}{r} \simeq N H_2(r/N) + \tfrac{1}{2}\log\left[2\pi N \frac{N-r}{N}\frac{r}{N}\right]. \tag{0.17}$$

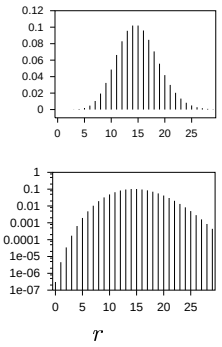


Figure 0.2. The Poisson distribution $P(r|\lambda{=}15)$, on a linear scale (top) and a logarithmic scale (bottom).

Recall that $\log_2 x = \frac{\log_e x}{\log_e 2}$.

Note that $\frac{\partial \log_2 x}{\partial x} = \frac{1}{\log_e 2}\frac{1}{x}$.

1

Introduction to Information Theory

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.
>
> *(Claude Shannon, 1948)*

In the first half of this book we study how to measure information content; we learn how compress data; and we learn how to communicate perfectly over imperfect communication channels.

We start by getting a feeling for this last problem.

1.1 How can we achieve perfect communication over an imperfect, noisy commmunication channel?

Some examples of noisy communication channels are:

- an analogue telephone line, over which two modems communicate digital information;
- the radio communication link from the Jupiter–orbiting spacecraft, Galileo, to earth;
- reproducing cells, in which the daughter cells's DNA contains information from the parent cells;
- a disc drive.

The last example shows that communication doesn't have to involve information going from one *place* to another. When we write a file on a disc drive, we'll read it off in the same location — but at a later *time*.

These channels are noisy. A telephone line suffers from cross-talk with other lines; the hardware in the line distorts and adds noise to the transmitted signal. The deep space network that listens to Galileo's puny transmitter receives background radiation from terrestrial and cosmic sources. DNA is subject to mutations and damage. A disc drive, which writes a binary digit (a one or zero, also known as a *bit*) by aligning a patch of magnetic material in one of two orientations, may later fail to read out the stored binary digit: the patch of material might spontaneously flip magnetization, or a glitch of background noise might cause the reading circuit to report the wrong value

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

for the binary digit, or the writing head might not induce the magnetization in the first place because of interference from neighbouring bits.

In all these cases, if we transmit data, *e.g.*, a string of bits, over the channel, there is some probability that the received message will not be identical to the transmitted message. We would prefer to have a communication channel for which this probability was zero — or so close to zero that for practical purposes it is indistinguishable from zero.

Let's consider a noisy disc drive that transmits each bit correctly with probability $(1-f)$ and incorrectly with probability f. This model communication channel is known as the *binary symmetric channel* (figure 1.1).

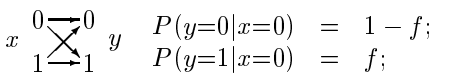


Figure 1.1. The binary symmetric channel. The transmitted symbol is x and the received symbol y. The noise level, the probability of a bit's being flipped, is f.

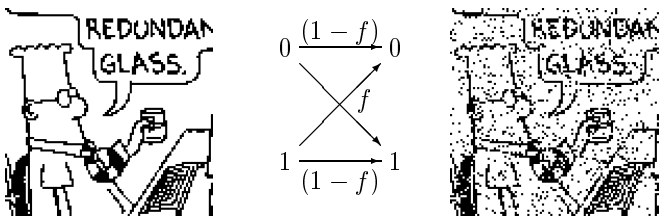


Figure 1.2. A binary data sequence of length 10000 transmitted over a binary symmetric channel with noise level $f = 0.1$. [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

As an example, let's imagine that $f = 0.1$, that is, ten percent of the bits are flipped (figure 1.2). A useful disc drive would flip no bits at all in its entire lifetime. If we expect to read and write a gigabyte per day for ten years, we require a bit error probability of the order of 10^{-15}, or smaller. There are two approaches to this goal.

The physical solution

The physical solution is to improve the physical characteristics of the communication channel to reduce its error probability. We could improve our disc drive by

1. using more reliable components in its circuitry;
2. evacuating the air from the disc enclosure so as to eliminate the turbulent forces that perturb the reading head from the desired track;
3. using a larger magnetic patch to represent each bit; or
4. using higher–power signals or cooling the circuitry in order to reduce thermal noise.

These physical modifications typically increase the cost of the communication channel.

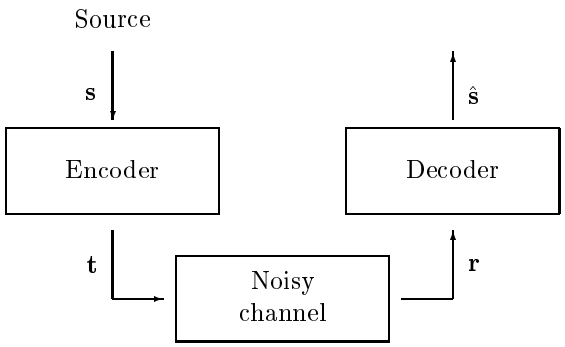


Figure 1.3. The 'system' solution for achieving reliable communication over a noisy channel. The encoding system introduces systematic redundancy into the transmitted vector **t**. The decoding system uses this known redundancy to deduce from the received vector **r** *both* the original source vector *and* the noise introduced by the channel.

The 'system' solution

Information theory and coding theory offer an alternative (and much more exciting) approach: we accept the given noisy channel and add communication *systems* to it so that we can detect and correct the errors introduced by the channel. As shown in figure 1.3, we add an *encoder* before the channel and a *decoder* after it. The encoder encodes the source message **s** into a *transmitted* message **t**; adding *redundancy* to the original message in some way. The channel adds noise to the transmitted message, yielding a received message **r**. The decoder uses the known redundancy introduced by the encoding system to infer both the original signal **s**, and the added noise.

Whereas physical solutions give incremental channel improvements only at an ever-increasing cost, system solutions can turn noisy channels into reliable communication systems with the only cost being a *computational* requirement at the encoder and decoder.

Information theory is concerned with the theoretical limitations and potentials of such systems. 'What is the best error-correcting performance we could achieve?'

Coding theory is concerned with the creation of practical encoding and decoding systems.

1.2 Error correcting codes for the binary symmetric channel

We now consider examples of encoding and decoding systems. What is the simplest way to add useful redundancy to a transmission? We want to be able to detect and correct errors.

Repetition codes

A straightforward idea is to repeat every bit of the message a prearranged number of times — for example, three times, as shown in figure 1.4. We call this *repetition code* 'R_3'.

Imagine that we transmit the source message

$$\mathbf{s} = 0\ 0\ 1\ 0\ 1\ 1\ 0$$

over a binary symmetric channel with noise level $f = 0.1$ using this repetition code. We can describe the channel as adding a sparse noise vector **n** to the

Source sequence **s**	Transmitted sequence **t**
0	000
1	111

Figure 1.4. The repetition code 'R_3'.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

transmitted vector (in modulo 2 arithmetic, *i.e.*, the binary algebra in which 1+1=0). A possible received vector $\mathbf{r} = \mathbf{t} + \mathbf{n}$ is shown in figure 1.5.

s	0	0	1	0	1	1	0
t	0 0 0	0 0 0	1 1 1	0 0 0	1 1 1	1 1 1	0 0 0
n	0 0 0	0 0 1	0 0 0	0 0 0	1 0 1	0 0 0	0 0 0
r	0 0 0	0 0 1	1 1 1	0 0 0	0 1 0	1 1 1	0 0 0

Figure 1.5. An example transmission using R_3.

How should we decode this received vector? The optimal algorithm looks at the received bits three at a time and takes a majority vote.

At the risk of explaining the obvious, let's prove this result. The optimal decoding decision (optimal in the sense of having the smallest probability of being wrong) is to find which value of $\mathbf{s}$ is most probable, given $\mathbf{r}$. Consider the decoding of a single bit s, which, was encoded as $\mathbf{t}(s)$ and gave rise to three received bits $\mathbf{r} = r_1 r_2 r_3$. By Bayes's theorem, the *posterior probability* of s is

$$P(s|r_1r_2r_3) = \frac{P(r_1r_2r_3|s)P(s)}{P(r_1r_2r_3)}. \tag{1.1}$$

We can spell out the posterior probability of the two alternatives thus:

$$P(s=1|r_1r_2r_3) = \frac{P(r_1r_2r_3|s=1)P(s=1)}{P(r_1r_2r_3)}. \tag{1.2}$$

$$P(s=0|r_1r_2r_3) = \frac{P(r_1r_2r_3|s=0)P(s=0)}{P(r_1r_2r_3)}. \tag{1.3}$$

This posterior probability is determined by two factors: the *prior probability* $P(s)$, and the data-dependent term $P(r_1r_2r_3|s)$, which is called the *likelihood* of s. The normalizing constant $P(r_1r_2r_3)$ is irrelevant to the optimal decoding decision, which is to guess $\hat{s}=0$ if $P(s=0|\mathbf{r}) > P(s=1|\mathbf{r})$, and $\hat{s}=1$ otherwise.

To find the optimal decoding decision, we must therefore make an assumption about the prior probabilities of the two hypotheses $s=0$ and $s=1$, and we must make an assumption about the probability of $\mathbf{r}$ given $\mathbf{t}(s)$. We will assume that the prior probabilities are equal: $P(s=0) = P(s=1) = 0.5$; then maximizing the posterior probability $P(s|\mathbf{r})$ is equivalent to maximizing the likelihood $P(\mathbf{r}|s)$. And we will assume that the channel is a binary symmetric channel with noise level $f < 0.5$, so that the likelihood is

$$P(\mathbf{r}|s) = P(\mathbf{r}|\mathbf{t}(s)) = \prod_{n=1}^{N} P(r_n|t_n(s)), \tag{1.4}$$

where $N = 3$ is the number of transmitted bits in the block we are considering, and $P(r_n|t_n)$ is $(1-f)$ if $r_n = t_n$ and f if $r_n \neq t_n$. Thus the likelihood ratio for the two hypotheses is

$$\frac{P(\mathbf{r}|s=1)}{P(\mathbf{r}|s=0)} = \prod_{n=1}^{N} \frac{P(r_n|t_n(1))}{P(r_n|t_n(0))}; \tag{1.5}$$

each factor $\frac{P(r_n|t_n(1))}{P(r_n|t_n(0))}$ equals $\frac{(1-f)}{f}$ if $r_n = 1$ and $\frac{f}{(1-f)}$ if $r_n = 0$. The ratio $\gamma \equiv \frac{(1-f)}{f}$ is greater than 1, since $f < 0.5$, so the winning hypothesis is the one with the most 'votes', each vote counting for a factor of γ in the likelihood ratio.

Thus the majority-vote decoder shown in figure 1.6 is the optimal decoder if we assume that the channel is a binary symmetric channel and that the two possible source messages 0 and 1 have equal prior probability.

If all three received bits are 0, we decode the triplet as a 0. If, as in the second triplet of figure 1.5, there are two 0s and one 1, then we decode the triplet

Received sequence $\mathbf{r}$	Likelihood ratio $\frac{P(\mathbf{r}\|s=1)}{P(\mathbf{r}\|s=0)}$	Decoded sequence $\hat{\mathbf{s}}$
000	γ^{-3}	0
001	γ^{-1}	0
010	γ^{-1}	0
100	γ^{-1}	0
101	γ^{1}	1
110	γ^{1}	1
011	γ^{1}	1
111	γ^{3}	1

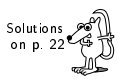
Figure 1.6. Majority–vote decoding algorithm for R_3. Also shown are the likelihood ratios, assuming the channel is a binary symmetric channel; $\gamma \equiv (1-f)/f$.

as a 0 — which in this case corrects the error. Not all errors are corrected, however. If we are unlucky and two errors fall in a single block, as in the fifth triplet of figure 1.5, then the decoding rule gets the wrong answer, as shown in figure 1.7.

s	0	0	1	0	1	1	0
t	000	000	111	000	111	111	000
n	000	001	000	000	101	000	000
r	000	001	111	000	010	111	000
$\hat{\mathbf{s}}$	0	0	1	0	0	1	0
corrected errors		$\star$					
undetected errors					$\star$		

Figure 1.7. Decoding the received vector from figure 1.5.

Thus the error probability is reduced by the use of this code. It is easy to compute the error probability.



Exercise 1.1:[A2] Compute the error probability of R_3 for a binary symmetric channel with noise level f.

The error probability is dominated by the probability that two bits in a block of three are flipped, which scales as f^2. In the case of the binary symmetric channel with $f = 0.1$, the R_3 code has a probability of error, after decoding, of $p_b \simeq 0.03$ per bit. Figure 1.8 shows the result of transmitting a binary image over a binary symmetric channel using the repetition code.

The repetition code R_3 has therefore reduced the probability of error, as desired. Yet we have lost something: our *rate* of information transfer has fallen by a factor of three. So if we use a repetition code to communicate data over a telephone line, it will reduce the error frequency, but it will also reduce our communication rate. We will have to pay three times as much for each phone call. Similarly, we would need three of the original noisy gigabyte disc drives in order to create a one-gigabyte disc drive with $p_b = 0.03$.

Can we push the error probability lower, to the values required for a sellable disc drive — 10^{-15}? We could achieve lower error probabilities by using repetition codes with more repetitions.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

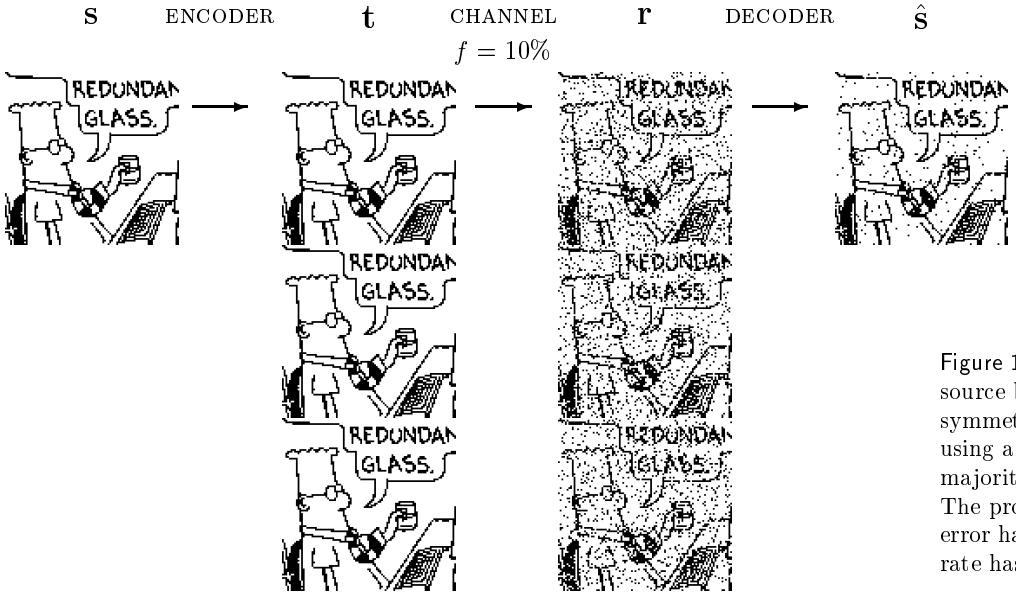


Figure 1.8. Transmitting 10000 source bits over a binary symmetric channel with $f = 10\%$ using a repetition code and the majority vote decoding algorithm. The probability of decoded bit error has fallen to about 3%; the rate has fallen to 1/3.

Solutions on p. 22

Exercise 1.2:[A3] (a) Show that the probability of error of the repetition code with N repetitions is, for odd N,

$$p_{\rm b} = \sum_{n=(N+1)/2}^{N} \binom{N}{n} f^n (1-f)^{N-n}. \tag{1.6}$$

(b) Assuming $f = 0.1$, which of the terms in this sum is the biggest? How much bigger is it than the next biggest term in the sum?

(c) Use Stirling's approximation to get rid of the $\binom{N}{n}$ in the largest term, and find, approximately, the probability of error of the repetition code with N repetitions.

(d) Show that it takes a repetition code with rate about 1/60 to get the probability of error down to 10^{-15}.

So to build a *single* gigabyte disc drive with the required reliability from noisy gigabyte drives with $f = 0.1$, we would need *sixty* of the noisy disc drives. The tradeoff between error probability and rate for repetition codes is shown in figure 1.9.

Block codes — the (7,4) Hamming code

We would like to communicate with tiny probability of error *and* at a substantial rate. Can we improve on repetition codes? What if we add redundancy to *blocks* of data instead of encoding one bit at a time? We now study a simple block code.

A *block code* is a rule for converting a sequence of source bits $\mathbf{s}$, of length K, say, into a transmitted sequence $\mathbf{t}$ of length N bits. To add redundancy, we make N greater than K. In a *linear* block code, the extra $N - K$ bits are linear functions of the original K bits; these extra bits are called *parity check*

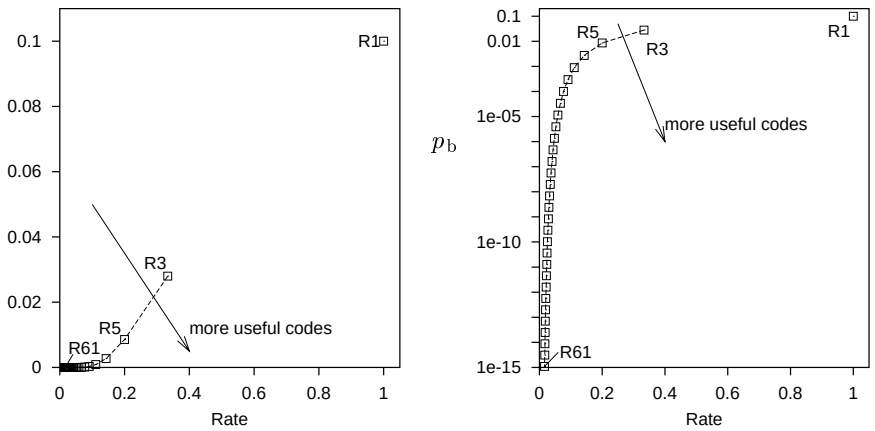


Figure 1.9. Error probability p_b versus rate for repetition codes over a binary symmetric channel with $f = 0.1$. The right hand figure shows p_b on a logarithmic scale. We would like the rate to be large and p_b to be small.

bits. An example of a linear block code is the *(7,4) Hamming* code, which transmits $N = 7$ bits for every $K = 4$ source bits.

The encoding operation for the code is shown pictorially in figure 1.10. We arrange the seven transmitted bits in three intersecting circles. The first four transmitted bits, $t_1t_2t_3t_4$, are set equal to the four source bits, $s_1s_2s_3s_4$. The parity check bits $t_5t_6t_7$ are set so that the *parity* within each circle is even: the first parity check bit is the parity of the first three source bits (that is, it is 0 if the sum of those bits is even, and 1 if the sum is odd); the second is the parity of the last three; and the third parity bit is the parity of source bits one, three and four.

As an example, figure 1.12b shows the transmitted codeword for the case $\mathbf{s} = 1000$.

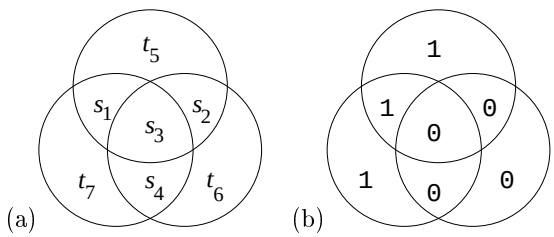


Figure 1.10. Pictorial representation of encoding for the Hamming (7,4) code.

s	t	s	t	s	t	s	t
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Figure 1.11. The sixteen codewords $\{\mathbf{t}\}$ of the (7,4) Hamming code. Any pair of codewords differ from each other in at least three bits.

The table shows the codewords generated by each of the $2^4 =$ sixteen settings of the four source bits.

Because the Hamming code is a linear code, it can be written compactly in terms of matrices as follows. The transmitted codeword $\mathbf{t}$ can be obtained from the source sequence $\mathbf{s}$ by a linear operation,

$$\mathbf{t} = \mathbf{G}^{\mathrm{T}}\mathbf{s}, \tag{1.7}$$

where $\mathbf{G}$ is the *generator matrix* of the code,

$$\mathbf{G}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \tag{1.8}$$

and the encoding equation (1.7) uses modulo-2 arithmetic [$1+1=0$, $0+1=1$, etc.].

In the encoding operation (1.7) I have assumed that $\mathbf{s}$ and $\mathbf{t}$ are column vectors. If instead they are row vectors, then this equation is replaced by

$$\mathbf{t} = \mathbf{s}\mathbf{G}, \tag{1.9}$$

where

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.10}$$

If you are like me, you may find it easier to relate to the right-multiplication (1.7) than the left-multiplication (1.9). Many coding theory texts use the left-multiplying conventions (1.9–1.10), however.

The rows of the generator matrix (1.10) can be viewed as defining four basis vectors lying in a seven–dimensional binary space. The sixteen codewords are obtained by making all possible binary sums of these vectors.

Decoding the (7,4) Hamming code

When we invent a more complex encoder, the task of decoding the received vector $\mathbf{r}$ becomes less straightforward. Remember that *any* of the bits may have been flipped, including the parity bits.

If we assume that the channel is a binary symmetric channel and that all source vectors are equiprobable, then the optimal decoder is one that identifies the source vector $\mathbf{s}$ whose encoding $\mathbf{t}(\mathbf{s})$ differs from the received vector $\mathbf{r}$ in the fewest bits. [Refer to the likelihood function (1.5) to see why this is so.] We could solve the decoding problem by measuring how far $\mathbf{r}$ is from each of the sixteen codewords in figure 1.11. Is there a more efficient way of finding the most probable source vector?

Syndrome decoding for the Hamming code

For the (7,4) Hamming code there is a pictorial solution to the decoding problem.

The received vector can be written into the three circles that we introduced when we were encoding $\mathbf{s}$ (figure 1.10). We then look at each of the three circles

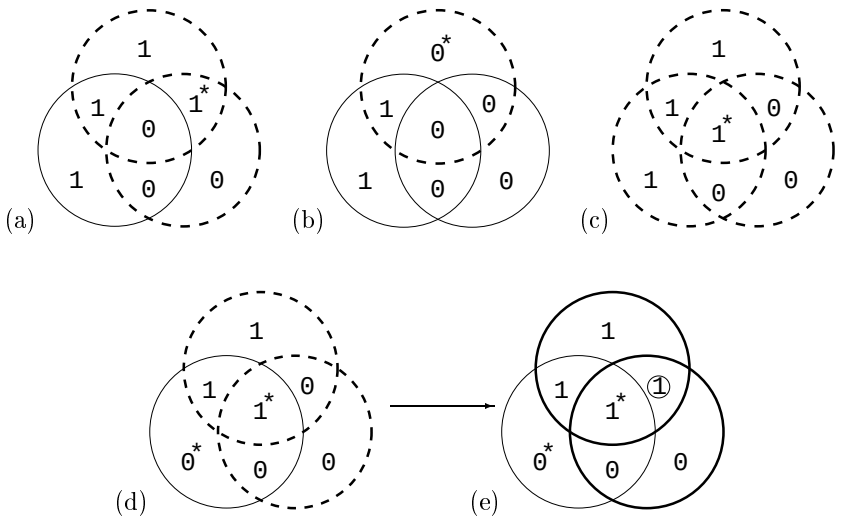


Figure 1.12. Pictorial representation of decoding of the Hamming (7,4) code. The bits that are flipped relative to figure 1.10(b) are labelled by $\star$. The violated parity checks are highlighted by dashed circles. One of the seven bits is the most probable suspect to account for each 'syndrome', *i.e.*, each pattern of violated and satisfied parity checks.
In examples (a), (b) and (c), the most probable suspect is the one bit that was flipped.
In example (d), two bits have been flipped, s_3 and t_7. The most probable suspect is r_2, marked by a circle in (e), which shows the output of the decoding algorithm.

Syndrome $\mathbf{z}$	000	001	010	011	100	101	110	111
Unflip this bit	*none*	7	6	4	5	1	2	3

Figure 1.13. Actions taken by the optimal decoder for the Hamming (7,4) code, assuming a binary symmetric channel with small noise level f. The syndrome vector $\mathbf{z}$ lists whether each parity check is violated (1) or satisfied (0), going through the checks in the order of the bits r_5, r_6, and r_7.

to see whether its parity is even. The fact that all codewords differ from each other in at least three bits means that if the noise has flipped any one or two bits, the received vector will no longer be a valid codeword, and some of the parity checks will be broken. The decoding task is to find the smallest set of flipped bits that can account for the observed violations of the parity checks.

Let us first consider noise patterns of weight one, *i.e.*, restrict attention to the case where just one bit is flipped. For example, if $\mathbf{n} = (0, 1, 0, 0, 0, 0, 0)$, then the received pattern will be as shown in figure 1.12(a). This figure shows highlighted the two circles in which the parity is not even. The bit r_2 that was flipped lies at the intersection of these two circles and outside the third circle, and so it is readily identified as the only single bit capable of explaining the syndrome.

Similarly figure 1.12(b) shows what happens if one of the parity bits, t_5, is flipped by the noise. Just one of the checks is violated, readily identifying this bit as the only single bit capable of explaining the syndrome.

If the central bit r_3 is received flipped, figure 1.12(c) shows that all three checks are violated, so the centre bit is readily identified as the suspect bit.

If you try flipping any one of the seven bits, you'll find that a different syndrome is obtained in each case — seven non–zero syndromes, one for each bit. There is only one other syndrome, the all–zero syndrome. So if the channel is a binary symmetric channel with a small noise level f, the optimal decoder unflips at most one bit, depending on the syndrome, as shown in figure 1.13. Each syndrome could have been caused by other noise patterns too, but any other noise pattern that has the same syndrome must be less probable because it involves a larger number of noise events.

What happens if the noise flips more than one bit? Figure 1.12(d) shows the situation when two bits, r_3 and r_7, are received flipped. The syndrome, 011, makes us suspect the single bit r_2; so our optimal decoding algorithm flips this bit, giving a decoded pattern with three errors as shown in figure 1.12(e). If we use the optimal decoding algorithm, any two–bit error pattern will lead to a decoded seven–bit vector that contains three errors.

General view of decoding for linear codes: syndrome decoding

We can also describe the decoding problem for a linear code in terms of matrices. The first four received bits, $r_1r_2r_3r_4$, purport to be the four source bits; and the received bits $r_5r_6r_7$ purport to be the parities of the source bits, as defined by the generator matrix $\mathbf{G}$. We evaluate the three parity check bits for the received bits, $r_1r_2r_3r_4$, and see whether they match the three received bits, $r_5r_6r_7$. The differences (modulo 2) between these two triplets are called the *syndrome* of the received vector. If the syndrome is zero — if all three parity checks agree with the corresponding received bits — then the received vector is a codeword, and the most probable decoding is given by reading out its first four bits. If the syndrome is non-zero, then the noise sequence for this block was non-zero, and the syndrome is our pointer to the most probable error pattern.

The computation of the syndrome vector is a linear operation. If we define the 3×4 matrix $\mathbf{P}$ such that the matrix of equation (1.8) is

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_4 \\ \mathbf{P} \end{bmatrix}, \tag{1.11}$$

where $\mathbf{I}_4$ is the 4×4 identity matrix, then the syndrome vector is $\mathbf{z} = \mathbf{H}\mathbf{r}$, where the *parity check matrix* $\mathbf{H}$ is given by $\mathbf{H} = \begin{bmatrix} -\mathbf{P} & \mathbf{I}_3 \end{bmatrix}$; in modulo 2 arithmetic, $-1 \equiv 1$, so

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.12}$$

All the codewords $\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}$ of the code satisfy

$$\mathbf{H}\mathbf{t} = 0. \tag{1.13}$$

Since the received vector $\mathbf{r}$ is given by $\mathbf{r} = \mathbf{G}^{\mathsf{T}}\mathbf{s} + \mathbf{n}$, the syndrome–decoding problem is to find the most probable noise vector $\mathbf{n}$ satisfying the equation

$$\mathbf{H}\mathbf{n} = \mathbf{z}. \tag{1.14}$$

A decoding algorithm that solves this problem is called a *maximum-likelihood decoder*. We will discuss decoding problems like this one in later chapters.

Summary of the (7,4) Hamming code's properties

Since there are three parity constraints, each of which might or might not be violated, there are $2 \times 2 \times 2 =$ eight distinct syndromes. They can be divided into seven non-zero syndromes — one for each of the one–bit error patterns — and the all–zero syndrome, corresponding to the zero–noise case.

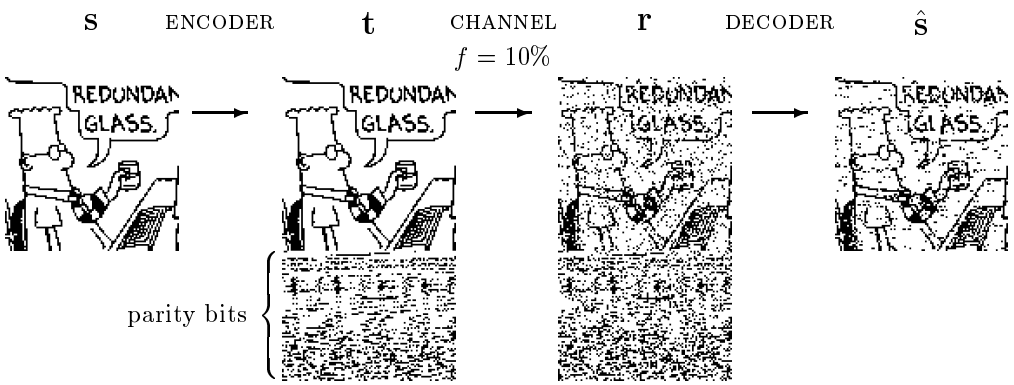


Figure 1.14. Transmitting 10000 source bits over a binary symmetric channel with $f = 10\%$ using a (7,4) Hamming code. The probability of decoded bit error is about 7%.

The optimal decoder takes no action if the syndrome is zero, otherwise it uses this mapping of non–zero syndromes onto one–bit error patterns to unflip the suspect bit.

There is a *decoding error* if the four decoded bits $\hat{s}_1, \ldots, \hat{s}_4$ do not all match the source bits $s_1, \ldots, s_4$. The *probability of block error* $p_{\rm B}$ is the probability that one or more of the decoded bits in one block fail to match the corresponding source bits,

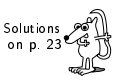
$$p_{\rm B} = P(\hat{\mathbf{s}} \neq \mathbf{s}). \tag{1.15}$$

The *probability of bit error* $p_{\rm b}$ is the average probability that a decoded bit fails to match the corresponding source bit,

$$p_{\rm b} = \frac{1}{K} \sum_{k=1}^{K} P(\hat{s}_k \neq s_k). \tag{1.16}$$

In the case of the Hamming code, a decoding error will occur whenever the noise has flipped more than one bit in a block of seven. The probability of block error is thus the probability that two or more bits are flipped in a block. This probability scales as $O(f^2)$, as did the probability of error for the repetition code R_3. But notice that the Hamming code communicates at a greater rate, $R = 4/7$.

Figure 1.14 shows a binary image transmitted over a binary symmetric channel using the (7,4) Hamming code. About 7% of the decoded bits are in error. Notice that the errors are correlated: often two or three successive decoded bits are flipped.


Solutions on p. 23

Exercise 1.3:[A1] This exercise and the next three refer to the (7,4) Hamming code. Decode the received strings:

(a) $\mathbf{r} = 1101011$

(b) $\mathbf{r} = 0110110$

(c) $\mathbf{r} = 0100111$

(d) $\mathbf{r} = 1111111$.

Exercise 1.4:[A2] (a) Calculate the probability of block error $p_{\rm B}$ of the (7,4) Hamming code as a function of the noise level f and show that to leading order it goes as $21f^2$.

(b) Show that to leading order the probability of bit error $p_{\rm b}$ goes as $9f^2$.

Exercise 1.5:^{A2} Find some noise vectors that give the all-zero syndrome (that is, noise vectors that leave all the parity checks unviolated). How many such noise vectors are there?

Exercise 1.6:^{B2} I asserted above that a block decoding error will result whenever two or more bits are flipped in a single block. Show that this is indeed so. [In principle, there might be error patterns that, after decoding, led only to the corruption of the parity bits, without the source bits's being incorrectly decoded.]

Exercise 1.7:^{B2} Consider the repetition code R_9. Another way of viewing this code is as a *concatenation* of R_3 with R_3. We first encode the source stream with R_3, then encode its output with R_3. We could call this code 'R_3^2'. This idea motivates an alternative decoding algorithm, in which we decode the bits three at a time using the decoder for R_3; then decode the decoded bits from that first decoder using the decoder for R_3.

Evaluate the probability of error for this decoder and compare it with the probability of error for the optimal decoder for R_9.

Do the concatenated encoder and decoder for R_3^2 have advantages over those for R_9?

Summary of codes' performances

Figure 1.15 shows the performance of repetition codes and the Hamming code. It also shows the performance of a family of linear block codes that are generalizations of Hamming codes, BCH codes.

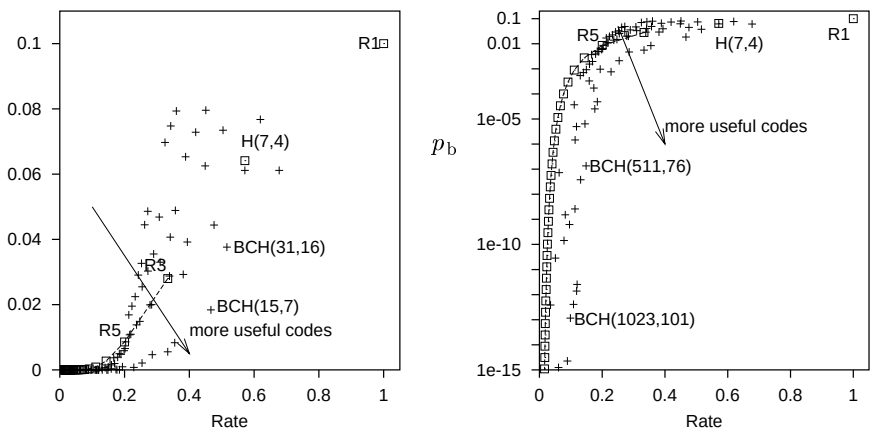


Figure 1.15. Error probability $p_{\rm b}$ versus rate R for repetition codes, the (7,4) Hamming code and BCH codes with block lengths up to 1023 over a binary symmetric channel with $f = 0.1$. The righthand figure shows $p_{\rm b}$ on a logarithmic scale.

This figure shows that we can, using linear block codes, achieve better performance than repetition codes; but the asymptotic situation still looks grim.

Exercise 1.8:^{B5} Design an error–correcting code and a decoding algorithm for it, compute its probability of error, and add it to figure 1.15. [Don't worry if you find it difficult to make a code better than the Hamming code, or if

you find it difficult to find a good decoder for your code; that's the point of this exercise.]

1.3 What performance can the best codes achieve?

There seems to be a trade-off between the decoded bit-error probability $p_{\rm b}$ (which we would like to reduce) and the rate R (which we would like to keep large). How can this trade-off be characterized? What points in the $(R, p_{\rm b})$ plane are achievable? This question was addressed by Shannon in his pioneering paper of 1948, in which he both created the field of information theory and solved most of its fundamental problems.

At that time there was a widespread belief that the boundary between achievable and nonachievable points in the $(R, p_{\rm b})$ plane was a curve passing through the origin $(R, p_{\rm b}) = (0, 0)$; if this were so, then, in order to achieve a vanishingly small error probability $p_{\rm b}$, one would have to reduce the rate correspondingly close to zero. 'No pain, no gain.'

However, Shannon proved the remarkable result that the boundary between achievable and nonachievable points meets the R axis at a *non-zero* value $R = C$, as shown in figure 1.16. For any channel, there exist codes that

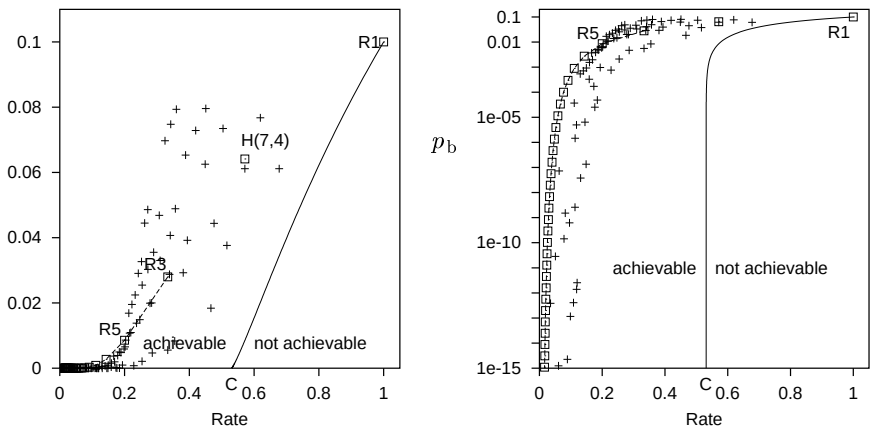


Figure 1.16. Shannon's noisy-channel coding theorem. The solid curve shows the Shannon limit on achievable values of $(R, p_{\rm b})$ for the binary symmetric channel with $f = 0.1$. Rates up to $R = C$ are achievable with arbitrarily small $p_{\rm b}$. The points show the performance of some textbook codes, as in figure 1.15.
The equation defining the Shannon limit (the solid curve) is $R = C/(1 - H_2(p_{\rm b}))$, where C and H_2 are defined in equation (1.17).

make it possible to communicate with *arbitrarily small* probability of error $p_{\rm b}$ at non-zero rates. The first half of this book will be devoted to understanding this remarkable result, which is called the *noisy-channel coding theorem*.

Example: $f = 0.1$

The maximum rate at which communication is possible with arbitrarily small $p_{\rm b}$ is called the *capacity* of the channel. The formula for the capacity of a binary symmetric channel with noise level f is

$$C(f) = 1 - H_2(f) = 1 - \left[f \log_2 \frac{1}{f} + (1 - f) \log_2 \frac{1}{1 - f} \right], \qquad (1.17)$$

so the channel we were discussing earlier with noise level $f = 0.1$ has capacity $C \simeq 0.53$. Let us consider what this means in terms of noisy disc drives.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

A repetition code could communicate over this channel with $p_{\rm b} = 0.03$ at a rate $R = 1/3$. Thus we know how to build a single gigabyte disc drive with $p_{\rm b} = 0.03$ from three noisy gigabyte disc drives. We also know how to make a single gigabyte disc drive with $p_{\rm b} \simeq 10^{-15}$ from sixty noisy one-gigabyte drives (exercise 1.2, p. 13). And now Shannon passes by, notices us juggling with disc drives and codes and says:

> "What performance are you trying to achieve? 10^{-15}? You don't need *sixty* disc drives — you can get that performance with just *two* disc drives (two being greater than $1/0.53$). And if you want $p_{\rm b} = 10^{-18}$ or 10^{-24} or anything, you can get there with two disc drives too!"

[Strictly, the above statements might not be quite right, as Shannon's noisy-channel coding theorem proves the achievability of ever smaller error probabilities at a given rate $R < C$ by studying sequences of block codes with ever increasing blocklengths, and the required blocklength might be bigger than a gigabyte (the size of our disc drive), in which case, Shannon might say "well, I can't do it with those *tiny* disc drives, but if you had two noisy *terabyte* drives, you could make a single high quality terabyte drive from them."]

1.4 Summary

The (7,4) Hamming Code

By including three parity check bits in a block of 7 bits it is possible to detect and correct any single bit error in each block.

Shannon's Noisy-Channel Coding Theorem

Information can be communicated over a noisy channel at a non-zero rate with arbitrarily small error probability.

Information theory addresses both the *limitations* and the *possibilities* of communication. The noisy-channel coding theorem, which we will prove in chapter 10, asserts both that reliable communication at any rate beyond the capacity is impossible, and that reliable communication at all rates up to capacity is possible.

The next few chapters lay the foundations for this result by discussing *how to measure information content* and the intimately related topic of *data compression*.

Solutions to chapter 1's exercises

Solution to exercise 1.1 (p. 12): The error probability of R_3 is a sum of two terms: the probability of all three bits' in a block being flipped, f^3; and the probability of exactly two bits' being flipped, $3f^2(1-f)$.

$$p_b = p_B = 3f^2(1-f) + f^3 = 3f^2 - 2f^3. \tag{1.18}$$

This probability is dominated for small f by the term $3f^2$.

Solution to exercise 1.2 (p. 13): The probability of error for the repetition code RN is dominated by the probability of $\lceil N/2 \rceil$ bits' being flipped, which goes (for odd N) as

$$\binom{N}{\lceil N/2 \rceil} f^{(N+1)/2}(1-f)^{(N-1)/2}. \tag{1.19}$$

The term $\binom{N}{K}$ can be approximated using the binary entropy function:

$$\frac{1}{N+1} 2^{NH_2(K/N)} \leq \binom{N}{K} \leq 2^{NH_2(K/N)} \;\Rightarrow\; \binom{N}{K} \simeq 2^{NH_2(K/N)}, \tag{1.20}$$

where this approximation introduces an error of order $\sqrt{N}$. So

$$p_b = p_B \simeq 2^N (f(1-f))^{N/2} = (4f(1-f))^{N/2}. \tag{1.21}$$

Setting this equal to the required value of 10^{-15} we find $N \simeq 2 \frac{\log 10^{-15}}{\log 4f(1-f)} = 68$. This answer is a little out because the approximation we used overestimated $\binom{N}{K}$ and we did not distinguish between $\lceil N/2 \rceil$ and $N/2$.

A slightly more careful answer (short of explicit computation) goes as follows. Taking the approximation for $\binom{N}{K}$ to the next order, we find:

$$\binom{N}{N/2} \simeq 2^N \frac{1}{\sqrt{2\pi N/4}}. \tag{1.22}$$

This approximation can be proved from an accurate version of Stirling's approximation (0.12), or by considering the binomial distribution with $p = 1/2$ and noting

$$1 = \sum_K \binom{N}{K} 2^{-N} \simeq 2^{-N} \binom{N}{N/2} \sum_{r=-N/2}^{N/2} e^{-r^2/2\sigma^2} \simeq 2^{-N} \binom{N}{N/2} \sqrt{2\pi}\sigma, \tag{1.23}$$

where $\sigma = \sqrt{N/4}$, from which equation (1.22) follows. The distinction between $\lceil N/2 \rceil$ and $N/2$ is not important in this term since $\binom{N}{K}$ has a maximum at $K = N/2$.

Then the probability of error (for odd N) is to leading order

$$p_b \simeq \binom{N}{(N+1)/2} f^{(N+1)/2}(1-f)^{(N-1)/2} \tag{1.24}$$

$$\simeq 2^N \frac{1}{\sqrt{\pi N/2}} f[f(1-f)]^{(N-1)/2} \tag{1.25}$$

$$\simeq \frac{1}{\sqrt{\pi N/8}} f[4f(1-f)]^{(N-1)/2} \tag{1.26}$$

The equation $p_b = 10^{-15}$ can be written

$$(N-1)/2 \simeq \frac{\log 10^{-15} + \log \frac{\sqrt{\pi N/8}}{f}}{\log 4f(1-f)} \tag{1.27}$$

which may be solved for N iteratively, the first iteration starting from $\hat{N}_1 = 68$:

$$(\hat{N}_2 - 1)/2 \simeq \frac{-15 + 1.7}{-0.44} = 29.9 \Rightarrow \hat{N}_2 \simeq 60.9 \tag{1.28}$$

This answer is found to be stable, so $N \simeq 61$ is the block length at which $p_b \simeq 10^{-15}$.

Solution to exercise 1.3 (p. 18): (a) `1100` (b) `0100` (c) `0100` (d) `1111`.

Solution to exercise 1.4 (p. 18):

(a) The probability of block error of the Hamming code is a sum of six terms — the probabilities of 2, 3, 4, 5, 6 and 7 errors's occurring in one block.

$$p_B = \sum_{r=2}^{7} \binom{7}{r} f^r (1-f)^{7-r}. \tag{1.29}$$

To leading order, this goes as

$$p_B \simeq \binom{7}{2} f^2 = 21 f^2. \tag{1.30}$$

(b) The probability of bit error of the Hamming code is smaller than the probability of block error because a block error rarely corrupt all bits in the decoded block. The leading order behaviour is found by considering the outcome in the most probable case where the noise vector has weight two. The decoder will erroneously flip a *third* bit, so that the modified received vector (of length 7) differs in three bits from the transmitted vector. Are parity bits or source bits more likely to be among these three flipped bits, or are all seven bits equally likely to be corrupted when the noise vector has weight two? The Hamming code is in fact completely symmetric in the protection it affords to the seven bits (assuming a binary symmetric channel). [This symmetry can be proved by showing that the role of a parity bits can be exchanged with a source bit and the resulting code is still a (7,4) Hamming code.] The probability that any one bit ends up corrupted is the same for all seven bits. So the probability of bit error is simply three sevenths of the probability of block error.

$$p_b \simeq \frac{3}{7} p_B \simeq 9 f^2. \tag{1.31}$$

Solution to exercise 1.5 (p. 19): There are fifteen non–zero noise vectors which give the all–zero syndrome; these are precisely the fifteen non–zero codewords of the Hamming code. Notice that because the Hamming code is *linear*, the sum of any two codewords is a codeword.

Solution to exercise 1.6 (p. 19): To be a valid hypothesis, a decoded pattern must be a codeword of the code. If there were a decoded pattern in which the parity bits differed from the transmitted parity bits, but the source bits didn't differ, that would mean that there are two codewords with the same source bits but different parity bits. But since the parity bits are a deterministic function of the source bits, this is a contradiction.

So if any linear code is decoded with its optimal decoder, and a decoding error occurs anywhere in the block, some of the source bits must be in error.

Solution to exercise 1.7 (p. 19): The probability of error of R_3^2 is, to leading order,

$$p_b(\mathrm{R}_3^2) \simeq 3\left[p_b(\mathrm{R}_3)\right]^2 = 3(3f^2)^2 + \ldots = 27f^4 + \ldots, \tag{1.32}$$

whereas the probability of error of R_9 is dominated by the probability of five flips,

$$p_b(\mathrm{R}_9) \simeq \binom{9}{5} f^5(1-f)^4 \simeq 126f^5 + \ldots. \tag{1.33}$$

The R_3^2 decoding procedure is therefore suboptimal, since there are noise vectors of weight four which cause a decoding error.

It has the advantage, however, of requiring smaller computational resources: only memorization of three bits, and counting up to three, rather than counting up to nine.

This is an important concept. Concatenated codes are widely used in practice because concatenation allows large codes to be implemented using simple encoding and decoding hardware.

Graphs corresponding to codes

Solution to exercise 1.8 (p. 19): When answering this question, you will probably find that it is easier to invent new codes than to find optimal decoders for them. Here is an example of a linear block code that is similar to, but bigger than, the (7,4) Hamming code.

Many codes can be conveniently expressed in terms of graphs. In figure 1.11, we introduced a pictorial representation of the (7,4) Hamming code. If we replace that figure's big circles, each of which shows that the parity of four particular bits is even, by a 'parity check node' that is connected to the four bits, then we obtain the representation of the (7,4) Hamming code by a *bipartite graph* as shown in figure 1.17. The 7 circles are the 7 transmitted bits. The 3 squares are the parity check nodes (not to be confused with the 3 parity check *bits*, which are the three most peripheral circles). The graph is a 'bipartite' graph because its nodes fall into two classes — bits and checks — and there are edges only between nodes in different classes. The graph and the code's parity check matrix (1.12) are simply related to each other: each parity check node corresponds to one row of $\mathbf{H}$ and each bit node corresponds to a column of $\mathbf{H}$; for every 1 in $\mathbf{H}$, there is an edge between the corresponding pair of nodes.

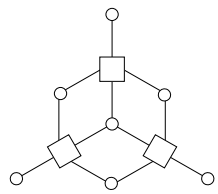


Figure 1.17. The graph of the (7,4) Hamming code. The 7 circles are the bit nodes and the 3 squares are the parity check nodes.

Having noticed this connection between linear codes and graphs, one simple way to invent linear codes is to think of a bipartite graph. A pretty bipartite graph can be obtained from a dodecahedron by calling the vertices of the dodecahedron the parity check nodes, and putting a transmitted bit on each edge in the dodecahedron. This construction defines a parity check matrix in which every column has weight 2 and every row has weight 3. [The weight of a binary vector is the number of 1s it contains.]

This code has $N = 30$ bits, and it appears to have $M_{\text{apparent}} = 20$ parity check constraints. Actually, there are only $M = 19$ *independent* constraints; the 20th constraint is redundant, so the number of source bits is $K = N - M = 11$. The code is a (30,11) code.

It is hard to find a decoding algorithm for this code, but we can estimate its probability of error by finding its lowest weight codewords. The code has 12 codewords of weight 5, one for each face of the original dodecahedron: if we flip all the bits surrounding one face, then all the parity checks will be satisfied. The error probability, assuming a binary symmetric channel, will thus be dominated, at least for low noise levels f, by a term of order

$$12f^5(1-f)^2 5. \tag{1.34}$$

Of course, there is no obligation to make codes whose graphs can be represented on a plane, as this one can; indeed the best linear codes, which have simple graphical descriptions, have graphs that are more tangled.

Furthermore, there is no reason for sticking to linear codes; indeed some nonlinear codes — codes whose codewords cannot be defined by a linear equation like $\mathbf{Ht} = 0$ — have very good properties. But encoding and decoding a nonlinear code is even trickier.

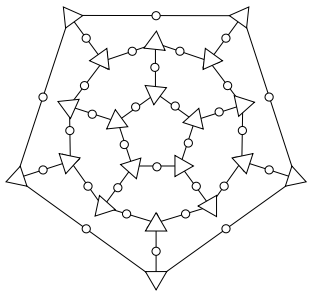


Figure 1.18. The graph defining the (30,11) dodecahedron code. The circles are the 30 transmitted bits and the triangles are the 20 parity checks, one of which is redundant.

About Chapter 2

The next chapter reviews the notation that we will use for probability distributions. I will assume that you are familiar with basic probability theory.

We will also introduce some important definitions to do with *information content* and the *entropy* function which we encountered in equation (1.17).

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

2

Probability, Entropy and Inference

This chapter, and its sibling, chapter 8, devote some time to notation. Just as the White Knight distinguished between the song, the name of the song, and what the name of the song was called (Carroll, 1998), we will sometimes need to be careful to distinguish between a random variable, the value of the random variable, and the proposition that asserts that the random variable has particular value. In any particular chapter, however, I will use the most simple and friendly notation possible, at the risk of upsetting pure-minded readers.

2.1 Probabilities and ensembles

An ensemble X is a triple $(x, \mathcal{A}_X, \mathcal{P}_X)$, where the *outcome* x is the value of a random variable, which takes on one of a set of possible values, $\mathcal{A}_X = \{a_1, a_2, \ldots, a_i, \ldots, a_I\}$, having probabilities $\mathcal{P}_X = \{p_1, p_2, \ldots, p_I\}$, with $P(x=a_i) = p_i$, $p_i \geq 0$ and $\sum_{a_i \in \mathcal{A}_X} P(x=a_i) = 1$.

The name $\mathcal{A}$ is mnemonic for 'alphabet'. One example of an ensemble is a letter that is randomly selected from an English document. This ensemble is shown in figure 2.1. There are twenty-seven possible letters: a–z, and a space character '-'.

Abbreviations. Briefer notation will sometimes be used. For example, $P(x=a_i)$ may be written as $P(a_i)$ or $P(x)$.

Probability of a subset. If T is a subset of $\mathcal{A}_X$ then:

$$P(T) = P(x \in T) = \sum_{a_i \in T} P(x=a_i). \tag{2.1}$$

For example, if we define V to be vowels from figure 2.1, $V = \{\mathtt{a}, \mathtt{e}, \mathtt{i}, \mathtt{o}, \mathtt{u}\}$, then

$$P(V) = 0.06 + 0.09 + 0.06 + 0.07 + 0.03 = 0.31. \tag{2.2}$$

A joint ensemble XY is an ensemble in which each outcome is an ordered pair x, y with $x \in \mathcal{A}_X = \{a_1, \ldots, a_I\}$ and $y \in \mathcal{A}_Y = \{b_1, \ldots, b_J\}$.

We call $P(x, y)$ the joint probability of x and y.

Commas are optional when writing ordered pairs, so $xy \Leftrightarrow x, y$.

N.B. In a joint ensemble XY the two variables are not necessarily independent.

i	a_i	p_i
1	a	0.0575
2	b	0.0128
3	c	0.0263
4	d	0.0285
5	e	0.0913
6	f	0.0173
7	g	0.0133
8	h	0.0313
9	i	0.0599
10	j	0.0006
11	k	0.0084
12	l	0.0335
13	m	0.0235
14	n	0.0596
15	o	0.0689
16	p	0.0192
17	q	0.0008
18	r	0.0508
19	s	0.0567
20	t	0.0706
21	u	0.0334
22	v	0.0069
23	w	0.0119
24	x	0.0073
25	y	0.0164
26	z	0.0007
27	-	0.1928

Figure 2.1. Probability distribution over the 27 outcomes for a randomly selected letter in an English language document (estimated from *The Frequently Asked Questions Manual for Linux*). The picture shows the probabilities by the areas of white squares.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

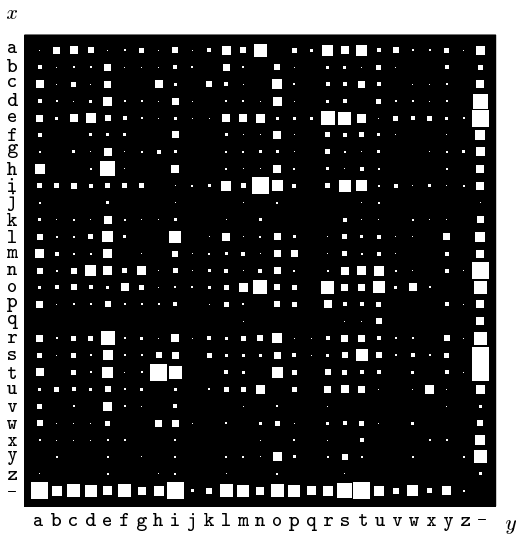


Figure 2.2. The probability distribution over the 27×27 possible bigrams xy in an English language document, *The Frequently Asked Questions Manual for Linux*.

Marginal probability. We can obtain the marginal probability from the joint probability $P(x, y)$ by summation:

$$P(x{=}a_i) \equiv \sum_{y \in \mathcal{A}_Y} P(x{=}a_i, y). \tag{2.3}$$

Similarly, using briefer notation,

$$P(y) \equiv \sum_{x \in \mathcal{A}_X} P(x, y). \tag{2.4}$$

Conditional probability

$$P(x{=}a_i | y{=}b_j) \equiv \frac{P(x{=}a_i, y{=}b_j)}{P(y{=}b_j)} \text{ if } P(y{=}b_j) \neq 0 \tag{2.5}$$

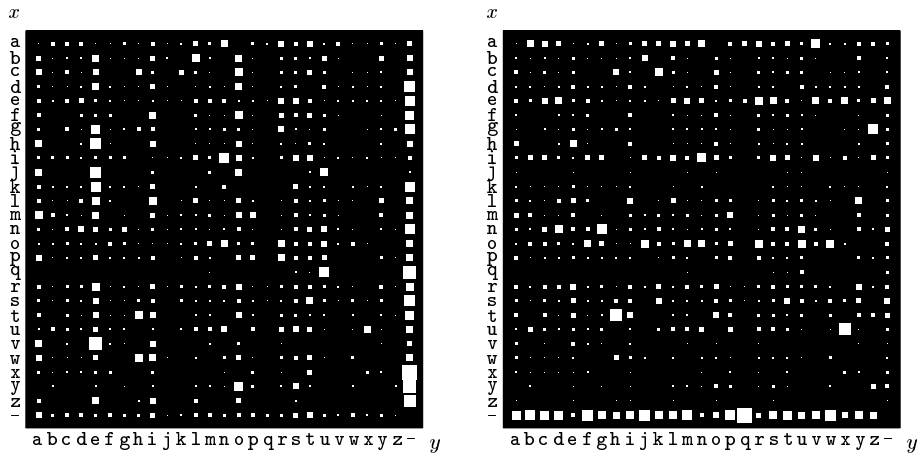
[If $P(y{=}b_j) = 0$ then $P(x{=}a_i|y{=}b_j)$ is undefined.]

We pronounce $P(x{=}a_i|y{=}b_j)$ 'the probability that x equals a_i, given y equals b_j'.

Example 2.1: An example of a joint ensemble is the ordered pair XY consisting of two successive letters in an English document. The possible outcomes are ordered pairs such as `aa`, `ab`, `ac`, and `zz`; of these, we might expect `ab` and `ac` to be more probable than `aa` and `zz`. An estimate of the joint probability distribution for two neighbouring characters is shown graphically in figure 2.2.

This joint ensemble has the special property that its two marginal distributions, $P(x)$ and $P(y)$, are identical. They are both equal to the monogram distribution shown in figure 2.1.

From this joint ensemble we can obtain conditional distributions, $P(y|x)$ and $P(x|y)$. The probability $P(y|x = \mathtt{q})$ is the probability distribution of



(a) $P(y|x)$ (b) $P(x|y)$

Figure 2.3. Conditional probability distributions. (a) $P(y|x)$: Each *row* shows the conditional distribution of the second letter, y, given the first letter, x, in a bigram xy. (b) $P(x|y)$: Each *column* shows the conditional distribution of the first letter, x, given the second letter, y.

the second letter given that the first letter is a `q`; this probability can be found by normalizing the row of the joint probability distribution $P(x, y)$ labelled $x =$ `q`. The two most probable values for y given $x =$ `q` are `u` and `-`. (The space is common after `q` because the document makes heavy use of the word `FAQ`.)

The probability $P(x|y =$ `u`$)$ is the probability distribution of the first letter given that the second letter is a `u`; this probability is a normalized version of the column of the joint probability distribution labelled $y =$ `u`. The two most probable values for x given $y =$ `u` are `n` and `o`.

Rather than writing down the joint probability directly, we often define an ensemble in terms of a collection of conditional probabilities. The following rules of probability theory will be useful. ($\mathcal{H}$ denotes assumptions on which the probabilities are based.)

Product rule — obtained from the definition of conditional probability:

$$P(x, y|\mathcal{H}) = P(x|y, \mathcal{H})P(y|\mathcal{H}) = P(y|x, \mathcal{H})P(x|\mathcal{H}). \tag{2.6}$$

This rule is also known as the chain rule.

Sum rule — a rewriting of the marginal probability definition:

$$P(x|\mathcal{H}) = \sum_y P(x, y|\mathcal{H}) \tag{2.7}$$

$$= \sum_y P(x|y, \mathcal{H})P(y|\mathcal{H}). \tag{2.8}$$

Bayes' theorem — obtained from the product rule:

$$P(y|x, \mathcal{H}) = \frac{P(x|y, \mathcal{H})P(y|\mathcal{H})}{P(x|\mathcal{H})} \tag{2.9}$$

$$= \frac{P(x|y, \mathcal{H})P(y|\mathcal{H})}{\sum_{y'} P(x|y', \mathcal{H})P(y'|\mathcal{H})}. \tag{2.10}$$

Notation: Let 'the degree of belief in proposition x' be denoted by $B(x)$. The negation of x (NOT-x) is written $\overline{x}$. The degree of belief in a conditional proposition, 'x, assuming proposition y to be true', is represented by $B(x|y)$.

Axiom 1: Degrees of belief can be ordered; if $B(x)$ is 'greater' than $B(y)$, and $B(y)$ is 'greater' than $B(z)$, then $B(x)$ is 'greater' than $B(z)$.

[Consequence: beliefs can be mapped onto real numbers.]

Axiom 2: The degree of belief in a proposition x and its negation $\overline{x}$ are related. There is a function f such that

$$B(x) = f[B(\overline{x})].$$

Axiom 3: The degree of belief in a conjunction of propositions x, y (x AND y) is related to the degree of belief in the conditional proposition $x|y$ and the degree of belief in the proposition y. There is a function g such that

$$B(x, y) = g\left[B(x|y), B(y)\right].$$

Figure 2.4. The Cox axioms. If a set of beliefs satisfy these axioms then they can be mapped onto probabilities satisfying $P(\text{FALSE}) = 0$, $P(\text{TRUE}) = 1$, $0 \leq P(x) \leq 1$ and the rules of probability,

$$P(x) = 1 - P(\overline{x}),$$

and

$$P(x, y) = P(x|y)P(y).$$

Independence. Two random variables X and Y are *independent* (sometimes written $X \perp Y$) if and only if

$$P(x, y) = P(x)P(y). \tag{2.11}$$

Exercise 2.2:^{A2} Are the random variables X and Y in the joint ensemble of figure 2.2 independent?

2.2 The meaning of probability

Probabilities can be used in two ways.

Probabilities can be used to describe *frequencies of outcomes in random experiments*, but giving non–circular definitions of the terms 'frequency' and 'random' is a challenge — what does it mean to say that the frequency of a tossed coin's coming up heads is 1/2? If we say that this frequency is the average fraction of heads in long sequences, we have to define 'average'; and it is hard to define 'average' without using a word synonymous to probability! I will not attempt to cut this philosophical knot.

Probabilities can also be used, more generally, to describe *degrees of belief* in propositions that do not involve random variables — for example 'the probability that Mr. S. was the murderer of Mrs. S., given the evidence' [he either was or wasn't, and it's the jury's job to assess how probable it is that he was]; 'the probability that New Guinea has been turned through 180 degrees by continental drift'; or 'the probability that 2050 will be the warmest year on record, assuming people do not change their lifestyle'.

Not all beliefs are rational. Degrees of belief can be mapped onto probabilities if they satisfy simple consistency rules known as the Cox axioms (Cox, 1946). Thus probabilities can be used to describe assumptions, and to describe inferences given those assumptions. The rules of probability ensure that if two people make the same assumptions and receive the same data then they will

draw identical conclusions. This more general use of probability is known as the *Bayesian* viewpoint. It is also known as the *subjective* interpretation of probability, since the probabilities depend on assumptions. Advocates of a Bayesian approach to data modelling and pattern recognition do not view this subjectivity as a defect, since in their view, you cannot do inference without making assumptions. In this book it will from time to time be taken for granted that a Bayesian approach makes sense, but the reader is warned that this is not yet a globally held view — the field of statistics was dominated for most of the 20th century by non-Bayesian methods in which probabilities are allowed to describe only random variables.

2.3 Forward probabilities and inverse probabilities

Probability calculations often fall into one of two categories: *forward probability* and *inverse probability*. Here is an example of a forward probability problem:

Exercise 2.3:[A2] An urn contains K balls, of which B are black and $W = K - B$ are white. Fred draws a ball at random from the urn and replaces it, N times.

(a) What is the probability distribution of the number of times a black ball is drawn, n_B?

(b) What is the expectation of n_B? What is the variance of n_B? What is the standard deviation of n_B? Give numerical answers for the cases $N = 5$ and $N = 400$, when $B = 2$ and $K = 10$.

Forward probability problems involve a *generative model* that describes a process that is assumed to give rise to some data; the task is to compute the probability distribution of some quantity, or the expectation of some function of the data. Here is another example of a forward probability problem:

Exercise 2.4:[B2] An urn contains K balls, of which B are black and $W = K - B$ are white. We define the fraction $f_B \equiv B/K$. Fred draws N times from the urn, exactly as in exercise 2.3, obtaining n_B blacks, and computes the quantity

$$z = \frac{(n_B - f_B N)^2}{N f_B (1 - f_B)}. \tag{2.12}$$

What is the expectation of z? In the case $N = 5$ and $f_B = 1/5$, what is the probability distribution of z? What is the probability that $z < 1$? [Hint: compare z with the quantities computed in the previous exercise.]

Like forward probability problems, *inverse probability problems* involve a generative model of a process, but instead of computing the probability of some function produced by the process, we compute the conditional probability of one or more of the *unobserved variables* in the process, given the observed variables. This invariably requires the use of Bayes' theorem.

Example 2.5: There are eleven urns labelled by $u \in \{0, 1, 2, \ldots, 10\}$, each containing ten balls. Urn u contains u black balls and $10 - u$ white balls. Fred selects an urn u at random and draws N times with replacement from that urn, obtaining n_B blacks and $N - n_B$ whites. Fred's friend, Bill, looks on.

A *graphical model* indicates the causal relationships between the random variables. The fact that no arrows point to node u indicates that u is a 'parent' variable that is set first in the data generation process. The square node N indicates that N is a variable that is fixed by the experimenter rather than a random variable. There is no arrow between N and u because we are assuming that there is no relationship between these variables — Fred is assumed to have chosen a fixed N before he chose u, say. The arrows from u and N to n_B shows that n_B is a 'child' of u and N, that is, the probability distribution of n_B depends on u and N. The graphical model indicates that the joint probability distribution of the variables $P(u, n_B|N)$ has a simple decomposition of the form (2.13).

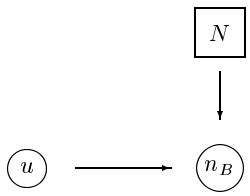


Figure 2.5. Graphical model for Bill and Fred's urn problem.

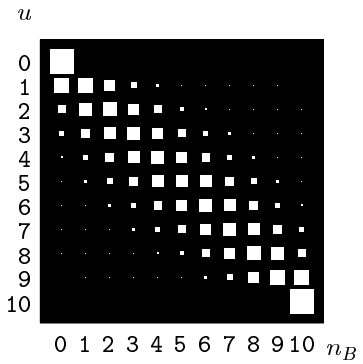


Figure 2.6. Joint probability of u and n_B for Bill and Fred's urn problem.

If after $N = 10$ draws, there are $n_B = 3$ blacks, what is the probability that the urn Fred is using is urn u, from Bill's point of view? (Bill doesn't know the value of u.)

Solution: The joint probability distribution of the random variables u and n_B can be written

$$P(u, n_B|N) = P(u)P(n_B|u, N). \tag{2.13}$$

From the joint probability of u and n_B, we can obtain the conditional distribution of u given n_B:

$$P(u|n_B, N) = \frac{P(u, n_B|N)}{P(n_B|N)} \tag{2.14}$$

$$= \frac{P(u)P(n_B|u, N)}{P(n_B|N)}. \tag{2.15}$$

The marginal probability of u is $P(u) = \frac{1}{11}$ for all u. You wrote down the probability of n_B given u and N, $P(n_B|u, N)$, when you solved exercise 2.3 (p. 31). [You *are* doing the exercises that are marked 'A', aren't you?] If we define $f_u \equiv u/10$ then

$$P(n_B|u, N) = \binom{N}{n_B} f_u^{n_B}(1 - f_u)^{N-n_B}. \tag{2.16}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

What about the denominator, $P(n_B|N)$? This is the marginal probability of n_B, which we can obtain using the sum rule:

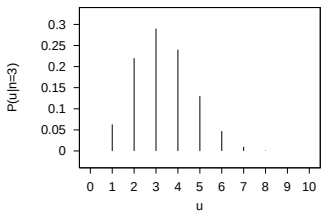
$$P(n_B|N) = \sum_u P(u, n_B|N) = \sum_u P(u)P(n_B|u, N). \tag{2.17}$$

So the conditional probability of u given n_B is

$$P(u|n_B, N) = \frac{P(u)P(n_B|u, N)}{P(n_B|N)} \tag{2.18}$$

$$= \frac{1}{P(n_B|N)} \frac{1}{11} \binom{N}{n_B} f_u^{n_B} (1 - f_u)^{N - n_B}. \tag{2.19}$$

This conditional distribution can be read out from column 3 of figure 2.6(a) and is shown in figure 2.7. The posterior probability that $u = 0$ given $n_B = 3$ is equal to zero, because if Fred were drawing from urn 0 it would be impossible for any black balls to be drawn. The posterior probability that $u = 10$ is also zero, because there are no white balls in that urn. The other hypotheses $u = 1$, $u = 2$, ... $u = 9$ all have non–zero posterior probability.



u	$P(u\|n_B = 3, N)$
0	0
1	0.063
2	0.22
3	0.29
4	0.24
5	0.13
6	0.047
7	0.0099
8	0.00086
9	0.0000096
10	0

Figure 2.7. Conditional probability of u given $n_B = 3$.

In inverse probability problems it is convenient to give names to the probabilities appearing in Bayes' theorem. In equation (2.18), we call the marginal probability $P(u)$ the *prior* probability of u, and $P(n_B|u, N)$ is called the *likelihood* of u. It is important to note that the terms likelihood and probability are not synonyms. The quantity $P(n_B|u, N)$ is a function of n_B and u. For fixed u, $P(n_B|u, N)$ defines a *probability* over n_B. When we observe n_B, the function $P(n_B|u, N)$ becomes of interest for its dependence on u, and we give this function of u a special name — the *likelihood* of u.

The conditional probability $P(u|n_B, N)$ is called the *posterior probability* of u given n_B. The normalizing constant $P(n_B|N)$ has no u–dependence so is not important if we simply wish to evaluate the relative probablities of the alternative hypotheses u. However, in hierarchical models, this quantity becomes important, and it is given various names: $P(n_B|N)$ is known as the *evidence* or the *marginal likelihood*.

If θ denotes the unknown parameters, D denotes the data, and $\mathcal{H}$ denotes the overall hypothesis space, the general equation:

$$P(\theta|D, \mathcal{H}) = \frac{P(D|\theta, \mathcal{H})P(\theta|\mathcal{H})}{P(D|\mathcal{H})} \tag{2.20}$$

is written:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}. \tag{2.21}$$

Exercise 2.6:[A3] Continuing from example 2.5 (p. 31), and assuming still that Bill has observed $n_B = 3$ blacks in $N = 10$ draws, let Fred draw another ball from the same urn. What is the probability that the next drawn ball is a black? [You should make use of the posterior probabilities in figure 2.7.]

Inference as inverse probability

Now consider the following exercise, which has the character of a simple scientific investigation.

Example 2.7: Bill tosses a bent coin N times, obtaining a sequence of heads and tails. We assume that the coin has a probability f_H of coming up heads; we do not know f_H. If n_H heads have occurred in N tosses, what is the probability distribution of f_H? (For example, N might be 10, and n_H might be 3; or, after a lot more tossing, we might have $N = 300$ and $n_H = 29$.) What is the probability that the $N+1$th outcome will be a head?

Unlike example 2.5 (p. 31), this problem has a subjective element. Given a restricted definition of probability that says 'probabilities are the frequencies of random variables', these two examples are different from the eleven–urns example. Whereas the urn u was a random variable, the bias f_H of the coin would not normally be called a random variable. It is just a fixed but unknown parameter that we are interested in. Yet don't the two examples 2.5 and 2.7 seem to have an essential similarity?

To solve example 2.7, we have to make an assumption about what the bias of the coin f_H might be. This prior probability distribution over f_H, $P(f_H)$, corresponds to the prior over u in the eleven–urns problem. In that example, the helpful problem definition specified $P(u)$. In real life, we have to make assumptions in order to assign priors; these assumptions will be subjective, and our answers will depend on them. Exactly the same can be said for the other probabilities in our generative model too. We are assuming, for example, that the balls are drawn from an urn independently; but could there not be correlations in the sequence because Fred's ball–drawing action is not perfectly random? Indeed there could be, so the likelihood function that we use depends on assumptions too. In real data modelling problems, priors are subjective and so are likelihoods.

Exercise 2.8:[B2] Assuming a uniform prior on f_H, $P(f_H) = 1$, solve the problem posed in example 2.7 (p. 34). Sketch the posterior distribution of f_H and compute the probability that the $N+1$th outcome will be a head, for

(a) $N = 3$ and $n_H = 0$;
(b) $N = 3$ and $n_H = 2$;
(c) $N = 10$ and $n_H = 3$;
(d) $N = 300$ and $n_H = 29$.

You will find the beta integral useful:

$$\int_0^1 dp_a \, p_a^{F_a} (1-p_a)^{F_b} = \frac{\Gamma(F_a+1)\Gamma(F_b+1)}{\Gamma(F_a+F_b+2)} = \frac{F_a! F_b!}{(F_a+F_b+1)!}. \quad (2.22)$$

People sometimes confuse assigning a prior distribution to an unknown parameter such as f_H with making an initial guess of the value of the parameter. But the prior over f_H, $P(f_H)$, is not a simple statement like 'initially, I would guess $f_H = 1/2$'. The prior is a probability density over f_H which specifies the prior degree of belief that f_H lies in any interval $(f, f + \delta f)$. It may well be the case that our prior for f_H is symmetric, so that the *mean* of f_H under the prior is $1/2$. In this case, the predictive distribution *for the first toss* x_1 would indeed be

$$P(x_1 = \text{head}) = \int df_H \, P(f_H) P(x_1 = \text{head} | f_H) = \int df_H \, P(f_H) f_H = 1/2. \quad (2.23)$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

But the prediction for subsequent tosses will depend on more properties of the prior than just its mean.

Data compression and inverse probability

Consider the following task.

Example 2.9: Write a program capable of compressing binary files like this one:

```
0000000000000000000010010001000000100000010000000000000000000000000000000000001010000000000000110000
1000000000100001000000000100000000000000000000001000000000000000001000000000110000010000000011000100
0000000001001000000000010001000000000000000011000000000000000000000000000001000000000000000100000000
```

The string shown contains $n_1 = 29$ 1s and $n_0 = 271$ 0s.

Intuitively, compression works by taking advantage of the predictability of a file. In this case, the source appears more likely to emit 0s than 1s. A data compression program that compresses this file must, implicitly or explicitly, be addressing the question 'What is the probability that the next character in this file is a 1?'

Do you think this problem is similar in character to example 2.7 (p. 34)? I do. One of the themes of this book is that data compression and data modelling are one and the same, and that they should both be addressed, like the urn of example 2.5, using inverse probability. example 2.9 is solved in chapter 6.

The likelihood principle

Please solve the following two exercises.

Example 2.10: Urn A contains three balls: one black, and two white; urn B contains three balls: two black, and one white. One of the urns is selected at random and one ball is drawn. The ball is black. What is the probability that the selected urn is urn A?

Example 2.11: Urn A contains five balls: one black, two white, one green and one pink; urn B contains five hundred balls: two hundred black, one hundred white, 50 yellow, 40 cyan, 30 sienna, 25 green, 25 silver, 20 gold, and 10 purple. One of the urns is selected at random and one ball is drawn. The ball is black. What is the probability that the urn is urn A?

What do you notice about your solutions?

The details of the other possible outcomes and their probabilities are irrelevant. All that matters is the probability of the outcome that actually happened (a black ball) given the different hypotheses. We only need to know the *likelihood*, *i.e.*, the probability of the data that happened, given each possible hypothesis. This simple rule about inference is known as the *likelihood principle*.

2.4 Definition of entropy and related functions

The Shannon information content of an outcome x is defined to be

$$h(x) = \log_2 \frac{1}{P(x)}. \tag{2.24}$$

It is measured in bits.

In the next few chapters, we will establish that the Shannon information content $h(a_i)$ is indeed a natural measure of the information content of the event $x{=}a_i$. At that point, we will shorten the name of this quantity to 'the information content'.

The fourth column in figure 2.8 shows the Shannon information content of the 27 possible outcomes when a random character is picked from an English document. The outcome $x =$z has a Shannon information content of 10.4 bits, and $x =$e has an information content of 3.5 bits.

The entropy of an ensemble X is defined to be the average Shannon information content of an outcome:

$$H(X) \equiv \sum_{x \in \mathcal{A}_X} P(x) \log \frac{1}{P(x)}, \tag{2.25}$$

with the convention for $P(x){=}0$ that $0 \times \log 1/0 \equiv 0$, since $\lim_{\theta \to 0^+} \theta \log 1/\theta{=}0$.

When it is convenient, we may also write $H(X)$ as $H(\mathbf{p})$, where $\mathbf{p}$ is the vector $(p_1, p_2, \ldots, p_I)$. Another name for the entropy of X is the uncertainty of X.

i	a_i	p_i	$\log_2 \frac{1}{p_i}$
1	a	0.0575	4.1
2	b	0.0128	6.3
3	c	0.0263	5.2
4	d	0.0285	5.1
5	e	0.0913	3.5
6	f	0.0173	5.9
7	g	0.0133	6.2
8	h	0.0313	5.0
9	i	0.0599	4.1
10	j	0.0006	10.7
11	k	0.0084	6.9
12	l	0.0335	4.9
13	m	0.0235	5.4
14	n	0.0596	4.1
15	o	0.0689	3.9
16	p	0.0192	5.7
17	q	0.0008	10.3
18	r	0.0508	4.3
19	s	0.0567	4.1
20	t	0.0706	3.8
21	u	0.0334	4.9
22	v	0.0069	7.2
23	w	0.0119	6.4
24	x	0.0073	7.1
25	y	0.0164	5.9
26	z	0.0007	10.4
27	-	0.1928	2.4
		$\sum_i p_i \log_2 \frac{1}{p_i}$	4.1

Figure 2.8. Shannon information contents of the outcomes a–z.

Example 2.12: The entropy of a randomly selected letter in an English document is about 4.11 bits, assuming its probability is as given in figure 2.1, p. 27. We obtain this number by averaging $\log 1/p_i$ (shown in the fourth column) under the probability distribution p_i (shown in the third column).

We now note some properties of the entropy function.

- $H(X) \geq 0$ with equality iff $p_i{=}1$ for one i. ['iff' is short for 'if and only if'.]
- Entropy is maximized if $\mathbf{p}$ is uniform:

$$H(X) \leq \log(|X|) \quad \text{with equality iff } p_i{=}1/|X| \text{ for all } i. \tag{2.26}$$

Notation: the vertical bars '$|\cdot|$' have two meanings. If X is an ensemble, then $|X|$ denotes the number of elements in the set $\mathcal{A}_X$; If x is a number, then $|x|$ is the absolute value of x.

The joint entropy of X, Y is:

$$H(X,Y) = \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x,y)}. \tag{2.27}$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \quad \text{iff} \quad P(x,y) = P(x)P(y). \tag{2.28}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Our definitions so far only apply to discrete probability distributions over finite sets $\mathcal{A}_X$. The definitions can be extended to infinite sets, though the entropy may then be infinite. The case of a probability *density* over a continuous set is addressed in section 11.3. Further important definitions and exercises to do with entropy, will come along in section 8.1.

2.5 Other useful definitions

The relative entropy *or* **Kullback-Leibler divergence** between two probability distributions $P(x)$ and $Q(x)$ that are defined over the same alphabet $\mathcal{A}_X$ is

$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}. \tag{2.29}$$

The relative entropy satisfies *Gibbs' inequality*

$$D_{\mathrm{KL}}(P||Q) \geq 0 \tag{2.30}$$

with equality only if P=Q. Note that in general $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$, so D_{KL} is not strictly a 'distance', though it is sometimes called the 'K-L distance'. This quantity is important in pattern recognition and neural networks, as well as in information theory.

Gibbs' inequality is probably the most important inequality in this book. It, and many other inequalities, can be proved using the concept of convexity.

The words 'convex$\cup$' and 'concave$\cap$' may be pronounced 'convex–smile' and 'concave–frown'. This terminology has the advantage of redundancy: while one may forget which way up 'convex' and 'concave' are, it is harder to confuse a smile and a frown.

Convex$\cup$ functions. A function $f(x)$ is *convex*$\cup$ over (a, b) if, as shown in figure 2.9, any chord of the function lies above the function, that is, for all $x_1, x_2 \in (a, b)$ and $0 \leq \lambda \leq 1$,

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2). \tag{2.31}$$

A function f is *strictly convex*$\cup$ if, for all $x_1, x_2 \in (a, b)$, the equality holds only for λ=0 and λ=1.

Similar definitions apply to concave$\cap$ and strictly concave$\cap$ functions.

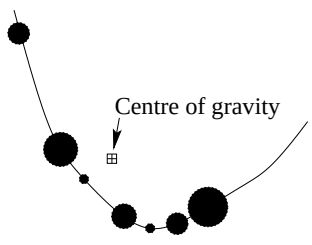
Some strictly convex$\cup$ functions are

- x^2, e^x and e^{-x} for all x;
- $\log(1/x)$ and $x \log x$ for $x > 0$.

Jensen's inequality. If f is a convex$\cup$ function and x is a random variable then:

$$\mathcal{E}\left[f(x)\right] \geq f\left(\mathcal{E}[x]\right), \tag{2.32}$$

where $\mathcal{E}$ denotes expectation. If f is strictly convex$\cup$ and $\mathcal{E}\left[f(x)\right] = f\left(\mathcal{E}[x]\right)$, then the random variable x is a constant (with probability 1).



A physical version of Jensen's inequality runs as follows.

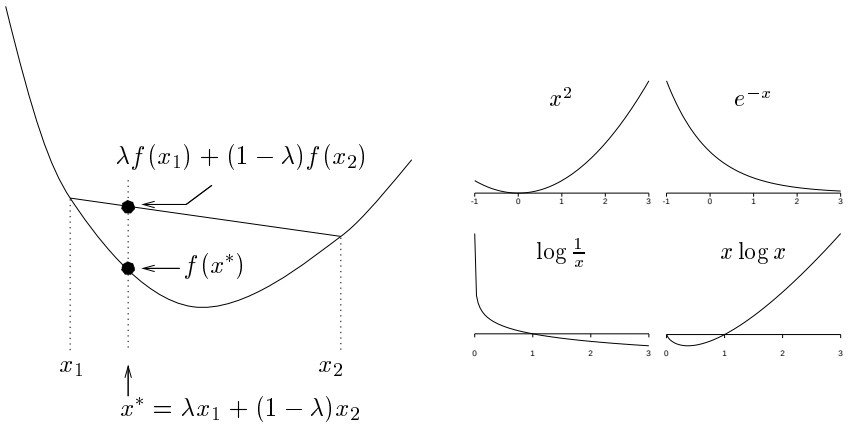


Figure 2.9. Convex $\cup$ functions.

If a collection of masses p_i are placed on a convex $\cup$ curve $f(x)$ at locations $(x_i, f(x_i))$, then the centre of gravity of those masses, which is at $(\mathcal{E}[x], \mathcal{E}[f(x)])$, lies above the curve.

If this fails to convince you, then feel free to do the following exercise.

Exercise 2.13:^{C2} Prove Jensen's inequality.

Jensen's inequality can also be rewritten for a concave $\cap$ function, with the direction of the inequality reversed.

Example 2.14: Three squares have average area $\bar{A} = 100\,\mathrm{m}^2$. The average of the lengths of their sides is $\bar{l} = 10\,\mathrm{m}$. What can be said about the size of the largest of the three squares?

Let x be the length of the side of a square, and let the probability of x be $1/3, 1/3, 1/3$ over the three lengths l_1, l_2, l_3. Then the information that we have is that $\mathcal{E}[x] = 10$ and $\mathcal{E}[f(x)] = 100$, where $f(x) = x^2$ is the function mapping lengths to areas. This is a strictly convex $\cup$ function. We notice that the equality $\mathcal{E}[f(x)] = f(\mathcal{E}[x])$ holds, therefore x is a constant, and the three lengths must all be equal. The area of the largest square is $100\,\mathrm{m}^2$.

Convexity and concavity also relate to maximization: if $f(\mathbf{x})$ is concave $\cap$ and there exists a point such that

$$\frac{\partial f}{\partial x_k} = 0 \text{ for all } k, \tag{2.33}$$

then $f(\mathbf{x})$ has its maximum value at that point.

The converse does not hold: if a concave $\cap$ $f(\mathbf{x})$ is maximized at some $\mathbf{x}$ it is not necessarily true that $\nabla f(\mathbf{x})$ is equal to zero there. For example, $f(x) = -|x|$ is maximized at $x = 0$ where its derivative is undefined; and $f(p) = \log(p)$, for $p \in (0, 1)$, is maximized on the boundary of the range, at $p = 1$, where $f'(p) = 1$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

2.6 Exercises

Most of the exercises can be solved without calculator, but a small number are intended to be solved by writing computer programs.

Exercises have an importance(A–C)/difficulty(1–5) rating next to them, for example A3 = important, and quite difficult; C1 = not so important, and very easy. I strongly recommend that you work on at least all the 'A' (important) problems. Don't skip over the exercises — to truly understand any topic, you must construct it for yourself!

Expectations and entropies

You are probably familiar with the idea of computing the expectation of a function of x,

$$\mathcal{E}\left[f(x)\right] = \langle f(x) \rangle = \sum_x P(x) f(x). \tag{2.34}$$

Maybe you are not so comfortable with computing this expectation in cases where the function $f(x)$ depends on the probability $P(x)$. The next few examples address this concern.

Solutions on p. 44

Exercise 2.15:^{A1} Let $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. Let $f(a) = 10$, $f(b) = 5$, and $f(c) = 10/7$. What is $\mathcal{E}\left[f(x)\right]$? What is $\mathcal{E}\left[1/P(x)\right]$?

Exercise 2.16:^{A2} For an arbitrary ensemble, what is $\mathcal{E}\left[1/P(x)\right]$?

Exercise 2.17:^{B2} Let $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. Let $g(a) = 0$, $g(b) = 1$, and $g(c) = 0$. What is $\mathcal{E}\left[g(x)\right]$?

Exercise 2.18:^{B1} Let $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. What is the probability that $P(x) \in [0.15, 0.5]$? What is

$$P\left(\left|\log \frac{P(x)}{0.2}\right| > 0.05\right)?$$

Exercise 2.19:^{A3} Prove the assertion that $H(X) \leq \log(|X|)$ with equality iff $p_i{=}1/|X|$ for all i. ($|X|$ denotes the number of elements in the set $\mathcal{A}_X$.)

Exercise 2.20:^{B3} The probability p_n of the nth most frequent word in English is roughly approximated by

$$p_n \simeq \begin{cases} \frac{0.1}{n} & \text{for } n \in 1 \ldots 8727. \\ 0 & n > 8727. \end{cases} \tag{2.35}$$

[This remarkable $1/n$ law is known as Zipf's law, and applies to the word frequencies of many languages (Zipf, 1949).] If we assume that English is generated by picking words at random according to this distribution, what is the entropy of English?

Solutions on p. 46

Exercise 2.21:^{B3} Prove that the relative entropy (equation (2.29)) satisfies $D_{\rm KL}(P||Q) \geq 0$ (Gibbs' inequality) with equality only if $P{=}Q$.

Forward probability

Exercise 2.22:^{C2} An unbiased coin is flipped until one head is thrown. What is the expected number of tails and the expected number of heads?

Fred, who doesn't know that the coin is unbiased, estimates the bias using $\hat{f} = h/(h+t)$, where h and t are the numbers of heads and tails tossed. Compute and sketch the probability distribution of $\hat{f}$.

Exercise 2.23:^{A2} Fred rolls an unbiased six-sided die once per second, noting the occasions when the outcome is a six.

(a) What is the mean number of rolls from one six to the next six?

(b) Inbetween two rolls, the clock strikes one. What is the mean number of rolls until the next six?

(c) Now think back before the clock struck. What is the mean number of rolls, going back in time, until the most recent six?

(d) What is the mean number of rolls from the six before the clock struck to the next six?

(e) Is your answer to (d) different from your answer to (a)? Explain.

Another version of this exercise refers to Fred waiting for a bus at a busstop in Poissonville where buses arrive independently at random (a Poisson process), with, on average, one bus every six minutes. What is the average wait for a bus, after Fred arrives at the stop? [6 minutes] So what is the time between the two buses, the one that Fred just missed, and the one that he catches? [12 minutes] Explain the apparent paradox. Note the contrast with the situation in Clockville, where the buses are spaced exactly 6 minutes apart. The mean wait at a bus stop is 3 minutes, and the time between the missed bus and the next one is 6 minutes.

Sums of random variables

Exercise 2.24:^{A3} (a) Two ordinary dice are thrown. What is the probability distribution of the sum of the values? What is the probability distribution of the absolute difference between the values?

(b) One hundred normal dice are thrown. What, roughly, is the probability distribution of the sum of the values?

(c) How can two cubical dice be labelled using the numbers $\{0, 1, 2, 3, 4, 5, 6\}$ so that when the two dice are thrown the sum has a uniform probability distribution over the integers 1–12?

Is there any way that one hundred dice could be labelled with integers such that the probability distribution of the sum is uniform?

Inference problems

Solutions on p. 46

Exercise 2.25:^{A2} If $q = 1 - p$ and $a = \log p/q$, show that

$$p = \frac{1}{1 + \exp(-a)}. \tag{2.36}$$

Sketch this function and find its relationship to $\tanh(a)$.

Exercise 2.26:^{B2} Let x and y be correlated random variables with x a binary variable taking values in $\mathcal{A}_X = \{0, 1\}$. Use Bayes' theorem to show that the log posterior probability ratio for x given y is

$$\log \frac{P(x=1|y)}{P(x=0|y)} = \log \frac{P(y|x=1)}{P(y|x=0)} + \log \frac{P(x=1)}{P(x=0)}. \tag{2.37}$$

Exercise 2.27:^{B2} Let x, d_1 and d_2 be random variables such that d_1 and d_2 are conditionally independent given a binary variable x. Use Bayes' theorem to show that the posterior probability ratio for x given $\{d_i\}$ is

$$\frac{P(x=1|\{d_i\})}{P(x=0|\{d_i\})} = \frac{P(d_1|x=1)}{P(d_1|x=0)} \frac{P(d_2|x=1)}{P(d_2|x=0)} \frac{P(x=1)}{P(x=0)}. \tag{2.38}$$

Life in high-dimensional spaces

Probability distributions and volumes have some unexpected properties in high–dimensional spaces.

Exercise 2.28:^{A2} Consider a sphere of radius r in an N-dimensional real space. Show that the fraction of the volume of the sphere that lies at values of the radius between $r - \epsilon$ and r, where $0 < \epsilon < r$, is:

$$f = 1 - \left(1 - \frac{\epsilon}{r}\right)^N. \tag{2.39}$$

Evaluate f for the cases $N = 2$, $N = 10$ and $N = 1000$, with (a) $\epsilon/r = 0.01$; (b) $\epsilon/r = 0.5$.

Implication: points that are uniformly distributed in a sphere in N dimensions, where N is large, are very likely to be in a thin shell near the surface.

Further inference problems

At this point you have a choice: if you'd like to read more about inference, you can look at these exercises and read chapter 3; if you're eager to get on to information theory, data compression, and noisy channels, you should skip to chapter 4.

Solutions on p. 49

Exercise 2.29:^{B2} A die is selected at random from two twenty–faced dice on which the symbols 1–10 are written with non-uniform frequency as follows.

Symbol	1	2	3	4	5	6	7	8	9	10
Number of faces of die A	6	4	3	2	1	1	1	1	1	0
Number of faces of die B	3	3	2	2	2	2	2	2	1	1

The randomly chosen die is rolled 7 times, with the following outcomes:

$$5, 3, 9, 3, 8, 4, 7.$$

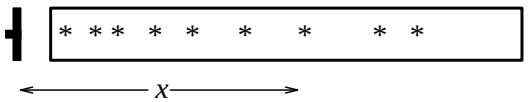
What is the probability that the die is die A?

Exercise 2.30:^{B2} Assume that there is a third twenty–faced die, die C, on which the symbols 1–20 are written once each. As above, one of the three dice is selected at random and rolled 7 times, giving the outcomes: 3, 5, 4, 8, 3, 9, 7.

What is the probability that the die is (a) die A, (b) die B, (c) die C?

Exercise 2.31:^{A3} **Inferring a decay constant**

Unstable particles are emitted from a source and decay at a distance x, a real number that has an exponential probability distribution with characteristic length λ. Decay events can only be observed if they occur in a window extending from $x = 1\,\mathrm{cm}$ to $x = 20\,\mathrm{cm}$. N decays are observed at locations $\{x_1, \ldots, x_N\}$. What is λ?



Exercise 2.32:^{B3} **Forensic evidence**

Two people have left traces of their own blood at the scene of a crime. A suspect, Fred, is tested and found to have type 'O' blood. The blood groups of the two traces are found to be of type 'O' (a common type in the local population, having frequency 60%) and of type 'AB' (a rare type, with frequency 1%). Do these data (type 'O' and 'AB' blood were found at scene) give evidence in favour of the proposition that Fred was one of the two people present at the crime?

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 2's exercises

Solution to exercise 2.3 (p. 31): We define the fraction $f_B \equiv B/K$.

(a) The number of black balls has a binomial distribution.

$$P(n_B|f_B,N) = \binom{N}{n_B} f_B^{n_B}(1-f_B)^{N-n_B} \tag{2.40}$$

(b) The mean and variance of this distribution are:

$$\mathcal{E}[n_B] = Nf_B \tag{2.41}$$

$$\mathrm{var}[n_B] = Nf_B(1-f_B). \tag{2.42}$$

These results were derived in example (p. 6). The standard deviation of n_B is $\sqrt{\mathrm{var}[n_B]} = \sqrt{Nf_B(1-f_B)}$.

When $B/K = 1/5$ and $N = 5$, the expectation and variance of n_B are 1 and 4/5. The standard deviation is 0.89.

When $B/K = 1/5$ and $N = 400$, the expectation and variance of n_B are 80 and 64. The standard deviation is 8.

Solution to exercise 2.4 (p. 31): The numerator of the quantity

$$z = \frac{(n_B - f_B N)^2}{Nf_B(1-f_B)}.$$

can be recognised as $(n_B - \mathcal{E}[n_B])^2$; the denominator is equal to the variance of n_B (2.42), which is by definition the expectation of the numerator. So the expectation of z is 1.

In the case $N = 5$ and $f_B = 1/5$, Nf_B is 1, and $\mathrm{var}[n_B]$ is 4/5. The numerator has five possible values, only one of which is smaller than 1: $(n_B - f_B N)^2 = 0$ has probability $P(n_B = 1) = 0.4096$; So the probability that $z < 1$ is 0.4096.

Solution to exercise 2.6 (p. 33): By the sum rule,

$$P(\text{ball } N+1 \text{ is black}|n_B,N) = \sum_u P(\text{ball } N+1 \text{ is black}|u,n_B,N)P(u|n_B,N). \tag{2.43}$$

Since the balls are drawn with replacement from the chosen urn, the probability $P(\text{ball } N+1 \text{ is black}|u,n_B,N)$ is just $f_u = u/10$, whatever n_B and N are. So

$$P(\text{ball } N+1 \text{ is black}|n_B,N) = \sum_u f_u P(u|n_B,N). \tag{2.44}$$

Using the values of $P(u|n_B,N)$ given in figure 2.7 we obtain

$$P(\text{ball } N+1 \text{ is black}|n_B{=}3,N{=}10) = 0.333. \tag{2.45}$$

Solution to exercise 2.13 (p. 38): We wish to prove, given the property

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2), \tag{2.46}$$

that, if $\sum p_i = 1$ and $p_i \geq 0$,

$$\sum_{i=1}^{I} p_i f(x_i) \geq f\left(\sum_{i=1}^{I} p_i x_i\right). \tag{2.47}$$

We proceed by recursion, working from the right hand side. (This proof does not handle cases where some $p_i = 0$; such details are left to the pedantic reader.) At the first line we use the definition of convexity (2.46) with $\lambda = \frac{p_1}{\sum_{i=1}^{I} p_i} = p_1$; at the second line, $\lambda = \frac{p_2}{\sum_{i=2}^{I} p_i}$, and so forth.

$$\begin{aligned} f\left(\sum_{i=1}^{I} p_i x_i\right) &= f\left(p_1 x_1 + \sum_{i=2}^{I} p_i x_i\right) \\ &\leq p_1 f(x_1) + \left[\sum_{i=2}^{I} p_i\right] \left[f\left(\sum_{i=2}^{I} p_i x_i \Big/ \sum_{i=2}^{I} p_i\right)\right] \qquad (2.48) \\ &\leq p_1 f(x_1) + \left[\sum_{i=2}^{I} p_i\right] \left[\frac{p_2}{\sum_{i=2}^{I} p_i} f(x_2) + \frac{\sum_{i=3}^{I} p_i}{\sum_{i=2}^{I} p_i} f\left(\sum_{i=3}^{I} p_i x_i \Big/ \sum_{i=3}^{I} p_i\right)\right], \end{aligned}$$

and so forth. □

Solution to exercise 2.15 (p. 39): $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. $f(a) = 10$, $f(b) = 5$, and $f(c) = 10/7$.

$$\mathcal{E}\left[f(x)\right] = 0.1 \times 10 + 0.2 \times 5 + 0.7 \times 10/7 = 3. \tag{2.49}$$

For each x, $f(x) = 1/P(x)$, so

$$\mathcal{E}\left[1/P(x)\right] = \mathcal{E}\left[f(x)\right] = 3. \tag{2.50}$$

Solution to exercise 2.16 (p. 39): For general X,

$$\mathcal{E}\left[1/P(x)\right] = \sum_{x \in \mathcal{A}_X} P(x) 1/P(x) = \sum_{x \in \mathcal{A}_X} 1 = |\mathcal{A}_X|. \tag{2.51}$$

Solution to exercise 2.17 (p. 39): $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. $g(a) = 0$, $g(b) = 1$, and $g(c) = 0$.

$$\mathcal{E}\left[g(x)\right] = p_b = 0.2. \tag{2.52}$$

Solution to exercise 2.18 (p. 39):

$$P\left(P(x) \in [0.15, 0.5]\right) = p_b = 0.2. \tag{2.53}$$

$$P\left(\left|\log \frac{P(x)}{0.2}\right| > 0.05\right) = p_a + p_c = 0.8. \tag{2.54}$$

Solution to exercise 2.19 (p. 39): This type of question can be approached in two ways: either by differentiating the function to be maximized, finding the maximum, and proving it is a global maximum; this strategy is somewhat risky since it is possible for the maximum of a function to be at the boundary of the space in question at a place where the derivative is not zero. Alternatively, a carefully chosen inequality can establish the answer. The second method is much neater.

Proof by differentiation:

$$H(X) = \sum_i p_i \log \frac{1}{p_i} \tag{2.55}$$

$$\frac{\partial H(X)}{\partial p_i} = \log \frac{1}{p_i} - 1 \tag{2.56}$$

we maximize subject to the constraint $\sum_i p_i = 1$ which can be enforced with a Lagrange multiplier:

$$G(\mathbf{p}) \equiv H(X) + \lambda \left(\sum_i p_i - 1 \right) \tag{2.57}$$

$$\frac{\partial G(\mathbf{p})}{\partial p_i} = \log \frac{1}{p_i} - 1 + \lambda. \tag{2.58}$$

At a maximum,

$$\log \frac{1}{p_i} - 1 + \lambda = 0 \tag{2.59}$$

$$\Rightarrow \log \frac{1}{p_i} = 1 - \lambda, \tag{2.60}$$

so all the p_i are equal. That this extremum is indeed a maximum is established by finding the curvature:

$$\frac{\partial^2 G(\mathbf{p})}{\partial p_i \partial p_j} = -\frac{1}{p_i} \delta_{ij}, \tag{2.61}$$

which is negative definite. □

Proof using Jensen's inequality: First a reminder of the inequality.

If f is a convex function and x is a random variable then:

$$\mathcal{E}\left[f(x)\right] \geq f\left(\mathcal{E}[x]\right), \tag{2.62}$$

If f is strictly convex and $\mathcal{E}\left[f(x)\right] = f\left(\mathcal{E}[x]\right)$, then the random variable x is a constant (with probability 1).

The secret of a proof using Jensen's inequality is to choose the right function and the right random variable. We could define

$$f(u) = \log \frac{1}{u} = -\log u \tag{2.63}$$

(which is a convex function) and think of $H(X) = \sum p_i \log \frac{1}{p_i}$ as the mean of $f(u)$ where $u = P(x)$, but this would not get us there — it would give us an inequality in the wrong direction. If instead we define

$$u = 1/P(x) \tag{2.64}$$

then we find:

$$H(X) = -\mathcal{E}\left[f(1/P(x))\right] \leq -f\left(\mathcal{E}[1/P(x)]\right); \tag{2.65}$$

now we know from exercise 2.17 (p. 39) that $\mathcal{E}[1/P(x)] = |\mathcal{A}_X|$, so

$$H(X) \leq -f\left(|\mathcal{A}_X|\right) = \log |\mathcal{A}_X|. \tag{2.66}$$

Equality only holds if the random variable $u = 1/P(x)$ is a constant, which means $P(x)$ is a constant for all x. □

Solution to exercise 2.20 (p. 39): The entropy is 11.8 bits per word.

Solution to exercise 2.21 (p. 39):

$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}. \tag{2.67}$$

We prove Gibbs' inequality using Jensen's inequality. Let $f(u) = \log 1/u$ and $u = \frac{Q(x)}{P(x)}$. Then

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \mathcal{E}[f(Q(x)/P(x))] & (2.68)\\ &\geq f\left(\sum_x P(x)\frac{Q(x)}{P(x)}\right) = \log\left(\frac{1}{\sum_x Q(x)}\right) = 0, & (2.69)\end{aligned}$$

with equality only if $u = \frac{Q(x)}{P(x)}$ is a constant, that is, if $Q(x) = P(x)$.

Second solution: In the above proof the expectations were with respect to the probability distribution $P(x)$. A second solution method uses Jensen's inequality with $Q(x)$ instead. We define $f(u) = u \log u$ and let $u = \frac{P(x)}{Q(x)}$. Then

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \sum_x Q(x)\frac{P(x)}{Q(x)} \log \frac{P(x)}{Q(x)} = \sum_x Q(x) f\left(\frac{P(x)}{Q(x)}\right) & (2.70)\\ &\geq f\left(\sum_x Q(x)\frac{P(x)}{Q(x)}\right) = f(1) = 0, & (2.71)\end{aligned}$$

with equality only if $u = \frac{P(x)}{Q(x)}$ is a constant, that is, if $Q(x) = P(x)$. □

Solution to exercise 2.24 (p. 40):

(a) For the outcomes $\{2,3,4,5,6,7,8,9,10,11,12\}$, the probabilities are $\mathcal{P} = \{\frac{1}{36}, \frac{2}{36}, \frac{3}{36}, \frac{4}{36}, \frac{5}{36}, \frac{6}{36}, \frac{5}{36}, \frac{4}{36}, \frac{3}{36}, \frac{2}{36}, \frac{1}{36}\}$.

(b) The value of one die has mean 3.5 and variance 35/12. So the sum of one hundred has mean 350 and variance $3500/12 \simeq 292$, and by the central limit theorem the probability distribution is roughly Gaussian (but confined to the integers!), with this mean and variance.

(c) In order to obtain a sum that has a uniform distribution we have to start from random variables some of which have a spiky distribution with the probability mass concentrated at the extremes. The unique solution is to have one normal die and one with faces 6,6,6,0,0,0.

Inference problems

Solution to exercise 2.25 (p. 40):

$$\begin{aligned} a &= \log \frac{p}{q} & (2.72)\\ \Rightarrow \quad \frac{p}{q} &= e^a & (2.73)\end{aligned}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

And $q = 1 - p$ gives

$$\frac{p}{1-p} = e^a \tag{2.74}$$

$$\Rightarrow p = \frac{e^a}{e^a + 1} = \frac{1}{1 + \exp(-a)}. \tag{2.75}$$

The hyperbolic tangent is

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{2.76}$$

so

$$f(a) \equiv \frac{1}{1 + \exp(-a)} = \frac{1}{2}\left(\frac{1 - e^{-a}}{1 + e^{-a}} + 1\right)$$

$$= \frac{1}{2}\left(\frac{e^{a/2} - e^{-a/2}}{e^{a/2} + e^{-a/2}} + 1\right) = \frac{1}{2}(\tanh(a/2) + 1). \tag{2.77}$$

Solution to exercise 2.26 (p. 41):

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.78}$$

$$\Rightarrow \frac{P(x=1|y)}{P(x=0|y)} = \frac{P(y|x=1)}{P(y|x=0)}\frac{P(x=1)}{P(x=0)} \tag{2.79}$$

$$\Rightarrow \log \frac{P(x=1|y)}{P(x=0|y)} = \log \frac{P(y|x=1)}{P(y|x=0)} + \log \frac{P(x=1)}{P(x=0)}. \tag{2.80}$$

Solution to exercise 2.27 (p. 41): The conditional independence of d_1 and d_2 given x means

$$P(x, d_1, d_2) = P(x)P(d_1|x)P(d_2|x). \tag{2.81}$$

This gives a separation of the posterior probability ratio into a series of factors, one for each data point, times the prior probability ratio.

$$\frac{P(x=1|\{d_i\})}{P(x=0|\{d_i\})} = \frac{P(\{d_i\}|x=1)}{P(\{d_i\}|x=0)}\frac{P(x=1)}{P(x=0)} \tag{2.82}$$

$$= \frac{P(d_1|x=1)}{P(d_1|x=0)}\frac{P(d_2|x=1)}{P(d_2|x=0)}\frac{P(x=1)}{P(x=0)}. \tag{2.83}$$

Life in high-dimensional spaces

Solution to exercise 2.28 (p. 41): The volume of a hypersphere of radius r in N dimensions is

$$V(r, N) = \frac{\pi^{N/2}}{(N/2)!} r^N. \tag{2.84}$$

For this question all that we need is the r–dependence, $V(r, N) \propto r^N$. So the fractional volume in $(r - \epsilon, r)$ is

$$\frac{r^N - (r-\epsilon)^N}{r^N} = 1 - \left(1 - \frac{\epsilon}{r}\right)^N. \tag{2.85}$$

The fractional volumes in the shells for the required cases are:

N	2	10	1000
$\epsilon/r = 0.01$	0.02	0.096	0.99996
$\epsilon/r = 0.5$	0.75	0.999	$1 - 2^{-1000}$

Notice that no matter how small ϵ is, for large enough N essentially all the probability mass is in the surface shell of thickness ϵ.

Solution to exercise 2.22 (p. 40): The probability of the number of tails t is

$$P(t) = \left(\frac{1}{2}\right)^t \frac{1}{2} \quad \text{for } t \geq 0. \tag{2.86}$$

The expected number of tails is

$$\mathcal{E}[t] = \sum_{t=0}^{\infty} t \left(\frac{1}{2}\right)^t \frac{1}{2}, \tag{2.87}$$

which may be shown to be 1 in a variety of ways. For example, since the situation after one tail is thrown is equivalent to the opening situation, we can write down the recurrence relation

$$\mathcal{E}[t] = \frac{1}{2}(1 + \mathcal{E}[t]) + \frac{1}{2}0 \;\Rightarrow\; \mathcal{E}[t] = 1. \tag{2.88}$$

The expected number of heads is 1, by definition of the problem.

The probability distribution of the estimator $\hat{f} = 1/(1+t)$ is plotted in figure 2.10.

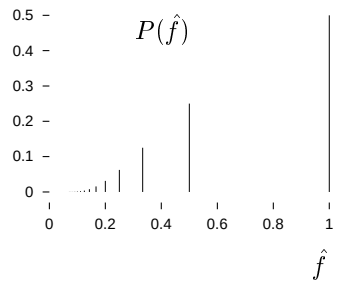


Figure 2.10. The probability distribution of the estimator $\hat{f} = 1/(1+t)$, given that $P(t) = 1/2^{t+1}$.

Solution to exercise 2.23 (p. 40):

(a) The mean number of rolls from one six to the next six is six (assuming we don't count the first of the two sixes). The probability distribution of the number of rolls is an exponential distribution.

(b) The mean number of rolls from the clock until the next six is six.

(c) The mean number of rolls, going back in time, until the most recent six is six.

(d) The mean number of rolls from the six before the clock struck to the six after the clock struck is the sum of the answers to (b) and (c), less one (assuming we don't count the first of the two sixes), that is, eleven.

(e) Rather than explaining the difference between (a) and (d), let me give another hint. Imagine that the buses in Poissonville arrive independently at random (a Poisson process), with, on average, one bus every six minutes. Imagine that passengers turn up at busstops at random also, and are scooped up by the bus without delay, so the space between two buses remains constant. The passengers' representative complains that two-thirds of all passengers found themselves on overcrowded buses. The bus operator claims, 'no, no — only one third of our buses are overcrowded'. Can both these claims be true?

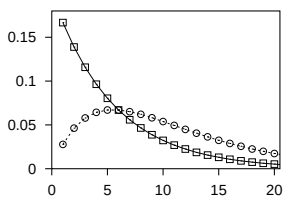


Figure 2.11. The probability distribution of the number of rolls r_1 from one 6 to the next (solid line),

$$P(r_1 = r) = \left(\frac{5}{6}\right)^{r-1} \frac{1}{6},$$

and the probability distribution (dashed line) of the number of rolls from the 6 before 1pm to the next 6, $r_{\rm tot}$,

$$P(r_{\rm tot} = r) = r \left(\frac{5}{6}\right)^{r-1} \left(\frac{1}{6}\right)^2, .$$

The probability $P(r_1 > 6)$ is about 1/3; the probability $P(r_{\rm tot} > 6)$ is about 2/3. The mean of r_1 is 6, and the mean if $r_{\rm tot}$ is 11.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to further inference problems

Solution to exercise 2.29 (p. 41): Let the data be D. Assuming equal prior probabilities,

$$\frac{P(A|D)}{P(B|D)} = \frac{1}{2}\frac{3}{2}\frac{1}{1}\frac{3}{2}\frac{1}{2}\frac{2}{2}\frac{1}{2} = 9/32. \tag{2.89}$$

and $P(A|D) = 9/41$.

Solution to exercise 2.30 (p. 42): The probability of the data given each hypothesis is:

$$P(D|A) = \frac{3}{20}\frac{1}{20}\frac{2}{20}\frac{1}{20}\frac{3}{20}\frac{1}{20}\frac{1}{20} = \frac{18}{20^7}; \tag{2.90}$$

$$P(D|B) = \frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{1}{20}\frac{2}{20} = \frac{64}{20^7}; \tag{2.91}$$

$$P(D|C) = \frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20} = \frac{1}{20^7}. \tag{2.92}$$

So

$$P(A|D) = \frac{18}{18+64+1} = \frac{18}{83}; \qquad P(B|D) = \frac{64}{83}; \qquad P(C|D) = \frac{1}{83}. \tag{2.93}$$

The other exercises are discussed in the next chapter.

About Chapter 3

If you'd like to get on with data compression, information content and entropy, you can skip the next chapter. Data compression and data modelling are intimately connected, however, so you'll probably want to come back to this chapter by the time you get to chapter 6.

Before reading chapter 3, you should have worked on exercises 2.25–2.31 (pp. 40–42).

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

3

More about Inference

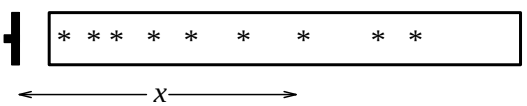
It is not a controversial statement that Bayes' theorem provides the correct language for describing the inference of a message communicated over a noisy channel, as we used it in chapter 1 (p. 11). But strangely, when it comes to other inference problems, the use of Bayes' theorem is not so widespread.

Coherent inference can always be mapped onto probabilities (Cox, 1946). Many textbooks on statistics do not mention this fact, so maybe it is worth using an example to emphasize the contrast between Bayesian inference and the orthodox methods of statistical inference. If this topic interests you, excellent further reading is to be found in the works of Jaynes, for example, Jaynes (1983).

3.1 A first example of probability theory

When I was an undergraduate in Cambridge, I was privileged to receive supervisions from Steve Gull. Sitting at his desk in a dishevelled office in St. John's College, I asked him how one ought to answer an old Tripos question (exercise 2.31):

> Unstable particles are emitted from a source and decay at a distance x, a real number that has an exponential probability distribution with characteristic length λ. Decay events can only be observed if they occur in a window extending from $x = 1\,\mathrm{cm}$ to $x = 20\,\mathrm{cm}$. N decays are observed at locations $\{x_1, \ldots, x_N\}$. What is λ?



I had scratched my head over this for some time. Since the mean of an unconstrained exponential distribution is λ, it seemed reasonable to examine the sample mean. It was easy to invent an 'estimator' $\hat{\lambda} = \bar{x} - 1$ that is appropriate for $\lambda \ll 20\,\mathrm{cm}$, but not for cases where the truncation of the distribution at the right hand side is significant; with a little ingenuity and the introduction of ad hoc bins, promising estimators for $\lambda \gg 20$ cm could be constructed. But there was no obvious estimator that would work under all conditions. What is the general solution to this problem and others like it? Is it always necessary, when confronted by a new inference problem, to grope

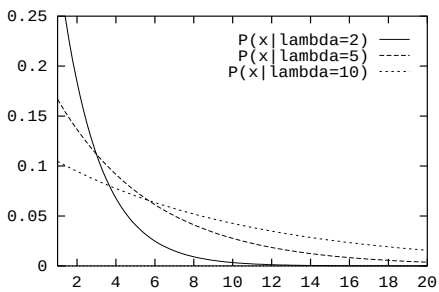


Figure 3.1. The probability density $P(x|\lambda)$ as a function of x.

in the dark for appropriate 'estimators' and worry about finding the 'best' estimator (whatever that means)?

I hope you have already stopped and thought about this question.

Steve wrote:

$$P(x|\lambda) = \begin{cases} \frac{1}{\lambda} e^{-x/\lambda}/Z(\lambda) & 1 < x < 20 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where

$$Z(\lambda) = \int_1^{20} dx \, \frac{1}{\lambda} e^{-x/\lambda} = \left(e^{-1/\lambda} - e^{-20/\lambda} \right). \tag{3.2}$$

This seemed obvious enough. Then he wrote:

$$P(\lambda|\{x_1, \ldots, x_N\}) = \frac{P(\{x\}|\lambda)P(\lambda)}{P(\{x\})} \tag{3.3}$$

$$\propto \frac{1}{(\lambda Z(\lambda))^N} \exp\left(-\textstyle\sum_1^N x_i/\lambda \right) P(\lambda). \tag{3.4}$$

Suddenly, the straightforward distribution $P(\{x_1, \ldots, x_N\}|\lambda)$, defining the probability of the data given the hypothesis λ, was being turned on its head so as to define the probability of a hypothesis given the data. A simple figure showed the probability of a single data point $P(x|\lambda)$ as a familiar function of x, for different values of λ (figure 3.1). Each curve was an innocent exponential, normalized to have area 1. Plotting the same function as a function of λ for a fixed value of x, something remarkable happened: a peak emerges (figure 3.2).

Steve summarised Bayes' theorem (equation 3.3) as embodying the fact that what you know about λ after the data arrive is what you knew before $[P(\lambda)]$, and what the data told you $[P(\{x\}|\lambda)]$. Probabilities are used here to quantify degrees of belief. To nip possible confusion in the bud, it must be emphasized that the hypothesis λ which correctly describes the situation is *not* a stochastic variable, and the fact that the Bayesian uses a probability distribution P does *not* mean that he thinks of the world as stochastically changing its nature between the states described by the different hypotheses. He uses the notation of probabilities to represent his beliefs about the mutually exclusive micro-hypotheses (here, values of λ), of which only one is actually true. That probabilities can denote degrees of belief, given assumptions, seemed intuitive to me.

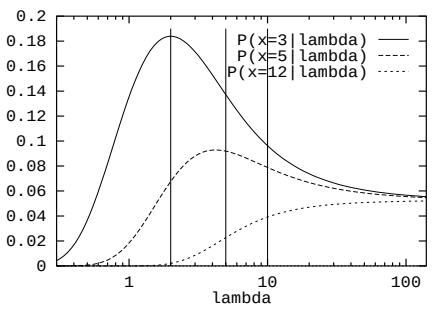


Figure 3.2. The probability density $P(x|\lambda)$ as a function of λ. When plotted this way round, the function is known as the *likelihood* of λ.

The posterior probability distribution of equation (3.4) represents the unique and complete solution to the problem. There is no need to invent estimators; nor do we need to invent criteria for comparing alternative estimators with each other. Whereas orthodox statisticians offer twenty ways of solving a problem, and another twenty different criteria for deciding which of these solutions is the best, Bayesian statistics only offers one answer to a well–posed problem.

Our inference is conditional on our assumptions [for example, the prior $P(\lambda)$]. Critics view such priors as a difficulty because they are 'subjective', but I don't see how it could be otherwise. How can one perform inference without making assumptions? I believe that it is of great value that Bayesian methods force one to make these tacit assumptions explicit.

First, once assumptions are made, the inferences are objective and unique, reproduceable with complete agreement by anyone who has the same information and makes the same assumptions. For example, given the assumptions listed above $\mathcal{H}$ and the data D, everyone will agree about the posterior probability of the decay length λ:

$$P(\lambda|D,\mathcal{H}) = \frac{P(D|\lambda,\mathcal{H})P(\lambda|\mathcal{H})}{P(D|\mathcal{H})}. \tag{3.5}$$

Second, when the assumptions are explicit, they are easier to criticize, and we can quantify the sensitivity of our inferences to the details of the assumptions. We can note from the likelihood curves in figure 3.2 that in the case of a single data point at $x = 5$, the likelihood function is less strongly peaked than in the case $x = 3$; and the details of the prior $P(\lambda)$ become more important as the sample mean $\bar{x}$ gets closer to the middle of the window, 10.5. In the case $x = 12$, the likelihood function doesn't have a peak at all — such data merely rule out small values of λ, and don't give any information about the relative probabilities of large values of λ. So in this case, the details of the prior at the small λ end of things are not important, but at the large λ end, our prior is important.

Third, when we are not sure which of various alternative assumptions is the most appropriate for a problem, we can treat this question as another inference task. Thus, given data D, we can compare alternative assumptions $\mathcal{H}$ using Bayes' theorem:

$$P(\mathcal{H}|D,I) = \frac{P(D|H,I)P(\mathcal{H}|I)}{P(D|I)}, \tag{3.6}$$

where I denotes the highest assumptions, which we are not questioning.

Fourth, we can take into account our uncertainty regarding such assumptions when we make subsequent predictions. Rather than choosing one particular assumption $\mathcal{H}^*$, and working out our predictions about some quantity $\mathbf{t}$, $P(\mathbf{t}|D, \mathcal{H}^*, I)$, we obtain predictions that take into account our uncertainty about $\mathcal{H}$ by using the sum rule:

$$P(\mathbf{t}|D, I) = \sum_{\mathcal{H}} P(\mathbf{t}|D, \mathcal{H}, I) P(\mathcal{H}|D, I). \tag{3.7}$$

This is another contrast with orthodox statistics, in which it is conventional to 'test' a default model, and then, if the test 'accepts' the model, to use exclusively that model to make predictions.

Steve thus persuaded me that

> Probability theory reaches parts that ad hoc methods cannot reach.

Let's look at a few more examples of simple inference problems.

3.2 The bent coin

A bent coin is tossed F times; we observe a sequence $\mathbf{s}$ of heads and tails. We wish to know the bias of the coin, and predict the probability that the next toss will result in a head. We first encountered this task in example 2.7 (p. 34), and we will encounter it again in chapter 6, when we discuss adaptive data compression. It is also the original inference problem studied by Rev. Thomas Bayes in his essay published in 1763.

As in chapter 2, we will assume a uniform prior distribution and obtain a posterior distribution by multiplying by the likelihood. A critic might object, 'where did this prior come from?' I will not claim that the uniform prior is in any way fundamental; indeed we'll give examples of nonuniform priors later. The prior is a subjective assumption. One of the themes of this book is:

> You can't do inference — or data compression — without making assumptions.

Likelihood function

We give the name $\mathcal{H}_1$ to our assumptions. [We'll be introducing an alternative set of assumptions in a moment.] The probability, given p_a, that F tosses result in a sequence $\mathbf{s}$ that contains $\{F_a, F_b\}$ counts of the two outcomes is

$$P(\mathbf{s}|p_a, F, \mathcal{H}_1) = p_a^{F_a}(1 - p_a)^{F_b}. \tag{3.8}$$

Our first model assumes a uniform prior distribution for p_a,

$$P(p_a|\mathcal{H}_1) = 1, \quad p_a \in [0, 1] \tag{3.9}$$

and $p_b \equiv 1 - p_a$.

Inferences and predictions

Given a string of length F of which F_a are as and F_b are bs we are interested in inferring what p_a might be.

Assuming $\mathcal{H}_1$ to be true, the posterior probability of p_a, given a string $\mathbf{s}$ of length F that has counts $\{F_a, F_b\}$, is by Bayes' theorem

$$P(p_a|\mathbf{s}, F, \mathcal{H}_1) = \frac{P(\mathbf{s}|p_a, F, \mathcal{H}_1)P(p_a|\mathcal{H}_1)}{P(\mathbf{s}|F, \mathcal{H}_1)}. \tag{3.10}$$

The factor $P(\mathbf{s}|p_a, F, \mathcal{H}_1)$ was given in equation (3.8) and the prior $P(p_a|\mathcal{H}_1)$ in equation (3.9).

$$P(p_a|\mathbf{s}, F, \mathcal{H}_1) = \frac{p_a^{F_a}(1-p_a)^{F_b}}{P(\mathbf{s}|F, \mathcal{H}_1)}. \tag{3.11}$$

The normalizing constant is given by the beta integral

$$P(\mathbf{s}|F, \mathcal{H}_1) = \int_0^1 dp_a \, p_a^{F_a}(1-p_a)^{F_b} = \frac{\Gamma(F_a+1)\Gamma(F_b+1)}{\Gamma(F_a+F_b+2)} = \frac{F_a!F_b!}{(F_a+F_b+1)!}. \tag{3.12}$$

Our inference of p_a, assuming $\mathcal{H}_1$ to be true, is thus given by equation (3.11).

The bent coin and model comparison

Imagine that a scientist introduces another theory for our data. He asserts that the source is not really a bent coin but is really a perfectly formed die with one face painted heads ('a') and the other five painted tails ('b'). Thus the parameter p_a, which in the original model above, $\mathcal{H}_1$, could take any value between 0 and 1, is according to the new hypothesis, $\mathcal{H}_0$, not a free parameter at all; rather, it is equal to 1/6. [This hypothesis is termed $\mathcal{H}_0$ so that the suffix of each model indicates its number of free parameters.]

How can we compare these two models in the light of data? We wish to infer how probable $\mathcal{H}_1$ is relative to $\mathcal{H}_0$.

Model comparison

In order to perform model comparison, we write down Bayes' theorem again, but this time with a different argument on the left hand side. We wish to know how probable $\mathcal{H}_1$ is given the data.

$$P(\mathcal{H}_1|\mathbf{s}, F) = \frac{P(\mathbf{s}|F, \mathcal{H}_1)P(\mathcal{H}_1)}{P(\mathbf{s}|F)} \tag{3.13}$$

Similarly, the posterior probability of $\mathcal{H}_0$ is

$$P(\mathcal{H}_0|\mathbf{s}, F) = \frac{P(\mathbf{s}|F, \mathcal{H}_0)P(\mathcal{H}_0)}{P(\mathbf{s}|F)}. \tag{3.14}$$

The normalizing constant in both cases is $P(\mathbf{s}|F)$, which is the probability of getting the observed data regardless of which model is true. If $\mathcal{H}_1$ and $\mathcal{H}_0$ are the only models under consideration, this probability is given by the sum rule:

$$P(\mathbf{s}|F) = P(\mathbf{s}|F, \mathcal{H}_1)P(\mathcal{H}_1) + P(\mathbf{s}|F, \mathcal{H}_0)P(\mathcal{H}_0). \tag{3.15}$$

F	Data (F_a, F_b)	$\frac{P(\mathcal{H}_1\|\mathbf{s},F)}{P(\mathcal{H}_0\|\mathbf{s},F)}$	
6	(5,1)	222.2	
6	(3,3)	2.67	
6	(2,4)	0.71	$= 1/1.4$
6	(1,5)	0.356	$= 1/2.8$
6	(0,6)	0.427	$= 1/2.3$
20	(10,10)	96.5	
20	(3,17)	0.2	$= 1/5$
20	(0,20)	1.83	

Table 3.1. Outcome of model comparison between models $\mathcal{H}_1$ and $\mathcal{H}_0$ for the 'bent coin'. Model $\mathcal{H}_0$ states that $p_a = 1/6, p_b = 5/6$.

To evaluate the posterior probabilities of the hypotheses we need to assign values to the prior probabilities $P(\mathcal{H}_1)$ and $P(\mathcal{H}_0)$; in this case, we might set these to 1/2 each. And we need to evaluate the data–dependent terms $P(\mathbf{s}|F,\mathcal{H}_1)$ and $P(\mathbf{s}|F,\mathcal{H}_0)$. We can give names to these quantities. The quantity $P(\mathbf{s}|F,\mathcal{H}_1)$ is a measure of how much the data favour $\mathcal{H}_1$, and we call it the *evidence* for model $\mathcal{H}_1$. We already encountered this quantity in equations (3.10) and (3.11) where it appeared as the normalizing constant of the first inference we made — the inference of p_a given the data.

Model comparison — take home message number 1:
The evidence for a model is usually the normalizing constant of an earlier Bayesian inference.

The evidence for model $\mathcal{H}_0$ is very simple because this model has no parameters to infer. Defining p_0 to be 1/6, we have

$$P(\mathbf{s}|F,\mathcal{H}_0) = p_0^{F_a}(1-p_0)^{F_b}. \tag{3.16}$$

Thus the posterior probability ratio of model $\mathcal{H}_1$ to model $\mathcal{H}_0$ is

$$\frac{P(\mathcal{H}_1|\mathbf{s},F)}{P(\mathcal{H}_0|\mathbf{s},F)} = \frac{P(\mathbf{s}|F,\mathcal{H}_1)P(\mathcal{H}_1)}{P(\mathbf{s}|F,\mathcal{H}_0)P(\mathcal{H}_0)} \tag{3.17}$$

$$= \frac{\frac{F_a!F_b!}{(F_a+F_b+1)!}}{p_0^{F_a}(1-p_0)^{F_b}}. \tag{3.18}$$

Some values of this posterior probability ratio are illustrated in table 3.1. The first five lines illustrate that there are outcomes that favour one model, and there are outcomes that favour the other. No outcome is completely incompatible with either model. With small amounts of data (six tosses) it is typically not the case that one of the two models is overwhelmingly more probable than the other. But with more data, the evidence against $\mathcal{H}_0$ given by any data set with the ratio $F_a : F_b$ differing from 1:5 mounts up.

The simpler model, $\mathcal{H}_1$, since it has no adjustable parameters, is able to lose out by the biggest margin. The odds may be hundreds to one against it. The more complex model can never lose out by a large margin; there's no data set that is actually *unlikely* given model $\mathcal{H}_1$.

We will discuss model comparison more in a later chapter.

3.3 An example of legal evidence

The following example (exercise 2.32) illustrates that there is more to Bayesian inference than the priors.

> Two people have left traces of their own blood at the scene of a crime. A suspect, Fred, is tested and found to have type 'O' blood. The blood groups of the two traces are found to be of type 'O' (a common type in the local population, having frequency 60%) and of type 'AB' (a rare type, with frequency 1%). Do these data (type 'O' and 'AB' blood were found at scene) give evidence in favour of the proposition that Fred was one of the two people present at the crime?

A careless lawyer might claim that the fact that the suspect's blood type was found at the scene is positive evidence for the theory that he was present. But this is not so.

Denote the proposition 'the suspect and one unknown person were present' by S. The alternative, $\bar{S}$, states 'two unknown people from the population were present'. The prior in this problem is the prior probability ratio between the propositions S and $\bar{S}$. This quantity is important to the final verdict and would be based on all other available information in the case. Our task here is just to evaluate the contribution made by the data D, that is, the likelihood ratio, $P(D|S,\mathcal{H})/P(D|\bar{S},\mathcal{H})$. In my view, a jury's task should generally be to multiply together carefully evaluated likelihood ratios from each independent piece of admissible evidence. [This is the view of many statisticians but learned judges have recently disagreed and actually overturned the verdict of a trial, on appeal, because the jurors *had* been taught to use Bayes' theorem to handle complicated evidence.]

The probability of the data given S is the probability that one unknown person drawn from the population has blood type AB.

$$P(D|S,\mathcal{H}) = p_{AB} \tag{3.19}$$

The probability of the data given $\bar{S}$ is the probability that two unknown people drawn from the population have types O and AB.

$$P(D|\bar{S},\mathcal{H}) = 2\,p_O\,p_{AB} \tag{3.20}$$

In these equations $\mathcal{H}$ denotes the assumptions that two people were present and left blood there, and that the probability distribution of the blood groups of unknown people in an explanation is the same as the population frequencies.

Dividing, we obtain the likelihood ratio:

$$\frac{P(D|S,\mathcal{H})}{P(D|\bar{S},\mathcal{H})} = \frac{1}{2p_0} = \frac{1}{2\times 0.6} = 0.83 \tag{3.21}$$

Thus the data in fact provide weak evidence *against* the supposition that this suspect was present.

This result may be found surprising, so let us examine it from various points of view. First consider the case of another suspect who has type AB. Intuitively, the data do provide evidence in favour of the theory S' that this

suspect was present, relative to the null hypothesis $\bar{S}$. And indeed the likelihood ratio in this case is:

$$\frac{P(D|S',\mathcal{H})}{P(D|\bar{S},\mathcal{H})} = \frac{1}{2\,p_{AB}} = 50. \tag{3.22}$$

Now let us change the situation slightly; imagine that 99% of people are of blood type O, and the rest are of type AB. Only these two blood types exist in the population. The data at the scene are the same as before. Consider again how these data influence our beliefs about a particular suspect of type O and another of type AB. Intuitively, we still believe that the presence of the rare AB blood provides positive evidence that the suspect of type AB was there. But does the fact that type O blood was detected at the scene favour the hypothesis that the type O suspect was present? If this were the case, that would mean that regardless of who the suspect is, the data make it more probable they were present; everyone in the population would be under greater suspicion, which would be absurd. The data may be *compatible* with any suspect of either blood type being present, but if they provide positive evidence for some theories, they must also provide evidence against other theories.

Here is another way of thinking about this: imagine that instead of two people's blood stains there are ten, and that in the entire local population of one hundred, there are ninety type O suspects and ten type AB suspects. Consider a particular type O suspect: without any other information, and before the blood test results come in, there is a one in 10 chance that he was at the scene. We now get the results of blood tests, and find that nine of the ten stains are of type AB, and one of the stains is of type O. Does this make it more likely that the type O suspect was there? No, although he could have been, there is now only a one in ninety chance that he was, since we know that only one person present was of type O.

Maybe the intuition is aided finally by writing down the formulae for the general case where n_O blood stains of individuals of type O are found, and n_{AB} of type AB, a total of N individuals in all, and unknown people come from a large population with fractions p_O, p_{AB}. (There may be other blood types too.) The task is to evaluate the likelihood ratio for the two hypotheses: S, 'the type O suspect and $N-1$ unknown others left N stains'; and $\bar{S}$, 'N unknowns left N stains'. The probability of the data under hypothesis $\bar{S}$ is just the probability of getting n_O, n_{AB} individuals of the two types when N individuals are drawn at random from the population:

$$P(n_O, n_{AB}|\bar{S}) = \frac{N!}{n_O!\,n_{AB}!} p_O^{n_O} p_{AB}^{n_{AB}}. \tag{3.23}$$

In the case of hypothesis S, we need the distribution of the $N-1$ other individuals:

$$P(n_O, n_{AB}|S) = \frac{(N-1)!}{(n_O-1)!\,n_{AB}!} p_O^{n_O-1} p_{AB}^{n_{AB}}. \tag{3.24}$$

The likelihood ratio is:

$$\frac{P(n_O, n_{AB}|S)}{P(n_O, n_{AB}|\bar{S})} = \frac{n_O/N}{p_O}. \tag{3.25}$$

This is an instructive result. The likelihood ratio, *i.e.* the contribution of this data to the question of whether the type O suspect was present, depends

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

simply on a comparison of the frequency of type O blood in the observed data with the background frequency of type O blood in the population. There is no dependence on the counts of the other types found at the scene, or their frequencies in the population. If there are more type O stains than the average expected under hypothesis $\bar{S}$, then the data gives evidence in favour of the presence of this type O suspect. Conversely, if there are fewer type O stains than the expected number under $\bar{S}$, then the data reduce the probability of the hypothesis that he was there. In the special case $n_O/N = p_O$, the data contribute no evidence either way, regardless of the fact that the data are compatible with the hypothesis S.

3.4 Exercises

Exercise 3.1:^{B2} Mrs. S is found stabbed in her family garden. Mr. S behaves strangely after her death and is considered as a suspect. On investigation of police and social records it is found that Mr. S had beaten up his wife on at least nine previous occasions. The prosecution advances this data as evidence in favour of the hypothesis that Mr. S is guilty of the murder. "Ah no," says Mr. S's highly paid lawyer, "*statistically*, only one in a thousand wife-beaters actually goes on to murder his wife.[1] So the wife-beating, which is not denied by Mr. S, is not strong evidence at all. In fact, given the wife–beating evidence alone, it's extremely unlikely that he would be the murderer of his wife — only a 1/1000 chance."

Is the lawyer right? Or is he a lying slimey trickster? If the latter, what is wrong with his argument?

Exercise 3.2:^{A2} The three doors, normal rules.

On a game show, a contestant is told the rules as follows:

> There are three doors, labelled 1, 2, 3. A single prize has been hidden behind one of them. You get to select one door. Initially your chosen door will *not* be opened. Instead, the gameshow host will open one of the other two doors, and *he will do so in such a way as not to reveal the prize.* For example, if you first choose door 1, he will then open one of doors 2 and 3, and it is guaranteed that he will choose which one to open so that the prize will not be revealed.
>
> At this point, you will be given a fresh choice of door: you can either stick with your first choice, or you can switch to the other closed door. All the doors will then be opened and you will receive whatever is behind your final choice of door.

Imagine that the contestant chooses door 1 first; then the gameshow host opens door 3, revealing nothing behind the door, as promised. Should the contestant (a) stick with door 1, or (b) switch to door 2, or (c) does it make no difference?

[1] In the U.S.A., it is estimated that 2 million women are abused each year by their partners. In 1994 4,739 women were victims of homicide; of those, 28 percent, 1,326 women were slain by husbands and boyfriends.
(Sources: `http://www.umn.edu/mincava/papers/factoid.htm`, `http://www.gunfree.inter.net/vpc/womenfs.htm`)

Exercise 3.3:[A2] The three doors, earthquake scenario.

Imagine that the game happens again and just as the gameshow host is about to open one of the doors a violent earthquake!during gameshow rattles the building and one of the three doors flies open. It happens to be door 3, and it happens not to have the prize behind it. The contestant had initially chosen door 1.

Repositioning his toupée, the host suggests, 'OK, since you chose door 1 initially, door 3 is a valid door for me to open, according to the rules of the game; I'll let door 3 stay open. Let's carry on as if nothing happened.'

Should the contestant stick with door 1, or switch to door 2, or does it make no difference? Assume that the prize was placed randomly, that the gameshow host does not know where it is, and that the door flew open because its latch was broken by the earthquake.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 3's exercises

Solution to exercise 3.1 (p. 59): The statistic quoted indicates the probability that a randomly selected wife-beater will murder his wife. The probability that he was the murderer, *given* that the wife has been murdered, is a completely different quantity.

To deduce the latter, we need to make additional assumptions about the probability of the wife's being murdered by someone else. If she lives in a neighbourhood with frequent random murders, then this probability is large and the posterior probability that the husband did it (in the abscence of other evidence) may not be very large. But in more peaceful regions, it may well be that the most likely person to have murdered you, if you are found murdered, is one of your closest relatives.

Let's work out some illustrative numbers with the help of the statistics on page 59. Let M denote the proposition that a woman has been murdered; H, the proposition that the husband did it; and B, the proposition that he beat her in the year preceding the murder. The negation of H ('someone else did it') is denoted by $\overline{H}$. We need to define $P(H|M)$, $P(B|H,M)$, and $P(B|\overline{H},M)$ in order to compute the posterior probability $P(H|B,M)$. From the statistics, we can read out $P(H|M) = 0.28$. And if two million women out of 100 million are beaten, then $P(B|\overline{H},M) = 0.02$. Finally, we need a value for $P(B|H,M)$: if a man murders his wife, how likely is it that this is the first time he laid a finger on her? I expect it's pretty unlikely; so maybe $P(B|H,M)$ is 0.9 or larger.

By Bayes' theorem, then,

$$P(H|B,M) = \frac{.9 \times .28}{.9 \times .28 + .02 \times .72} \simeq 95\%. \tag{3.26}$$

One way to make obvious the dishonesty of the slimey lawyer is to construct arguments with the same logical structure that are clearly wrong. For example, the lawyer could say 'Not only was Mrs. S. murdered, she was murdered between 4.02pm and 4.03pm. *statistically*, only one in a *million* wife-beaters actually goes on to murder his wife between 4.02pm and 4.03pm. So the wife-beating, which is not denied by Mr. S, is not strong evidence at all. In fact, given the wife–beating evidence alone, it's extremely unlikely that he would murder his wife in this way — only a 1/1000000 chance."

Solution to exercise 3.2 (p. 59): Let $\mathcal{H}_i$ denote the hypothesis that the prize is behind door i. We make the following assumptions: the three hypotheses $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ are equiprobable *a priori*, *i.e.*,

$$P(\mathcal{H}_1) = P(\mathcal{H}_1) = P(\mathcal{H}_1) = \frac{1}{3}. \tag{3.27}$$

The datum we receive, after choosing door 1, is one of $D = 3$ and $D = 2$ (meaning door 3 or 2 is opened, respectively. We assume that these two

possible outcomes have the following probabilities. If the prize is behind door 1 then the host has a free choice; in this case we assume that the host selects at random between $D = 2$ and $D = 3$. Otherwise the choice of the host is forced and the probabilities are 0 and 1.

$$\left| \begin{array}{c} P(D{=}2|\mathcal{H}_1) = \frac{1}{2} \\ P(D{=}3|\mathcal{H}_1) = \frac{1}{2} \end{array} \right| \left| \begin{array}{c} P(D{=}2|\mathcal{H}_2) = 0 \\ P(D{=}3|\mathcal{H}_2) = 1 \end{array} \right| \left| \begin{array}{c} P(D{=}2|\mathcal{H}_3) = 1 \\ P(D{=}3|\mathcal{H}_3) = 0 \end{array} \right| \tag{3.28}$$

Now, using Bayes' theorem, we evaluate the posterior probabilities of the hypotheses:

$$P(\mathcal{H}_i|D{=}3) = \frac{P(D{=}3|\mathcal{H}_i)P(\mathcal{H}_i)}{P(D{=}3)} \tag{3.29}$$

$$\left| \; P(\mathcal{H}_1|D{=}3) = \frac{(1/2)(1/3)}{P(D=3)} \; \right| \; P(\mathcal{H}_2|D{=}3) = \frac{(1)(1/3)}{P(D=3)} \; \left| \; P(\mathcal{H}_3|D{=}3) = \frac{(0)(1/3)}{P(D=3)} \; \right| \tag{3.30}$$

The denominator $P(D{=}3)$ is $(1/2)$ because it is the normalizing constant for this posterior distribution. So

$$\left| \; P(\mathcal{H}_1|D{=}3) \;=\; \frac{1}{3} \; \right| \; P(\mathcal{H}_2|D{=}3) \;=\; \frac{2}{3} \; \left| \; P(\mathcal{H}_3|D{=}3) \;=\; 0. \; \right| \tag{3.31}$$

So the contestant should switch to door 2 in order to have the biggest chance of getting the prize. Many people find this outcome surprising. There are two ways to make it more intuitive. One is to play the game twenty times with a friend and keep track of the frequency with which switching gets the prize. Alternatively, one can perform a thought experiment in which the game is played with a million doors. The rules are now that the contestant chooses one door, then the game show host opens 999,998 doors in such a way as not to reveal the prize, leaving the selected door and one other door closed. The contestant may now stick or switch. Imagine the contestant confronted by a million doors, of which doors 1 and 234,598 have not been opened, door 1 having been the contestant's initial guess. Where do you think the prize is?

Solution to exercise 3.3 (p. 60): If door 3 is opened by an earthquake, the inference comes out differently — even though visually the scene looks the same. The nature of the data, and the probability of the data, are both now different. The possible data outcomes are, firstly, that any number of the doors might have opened. We could label the eight possible outcomes $\mathbf{d} = (0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,1,1), \ldots, (1,1,1)$. Secondly, it might be that the prize is visible after the earthquake has opened one or more doors. So the data D consists of the value of $\mathbf{d}$, and a statement of whether the prize was revealed. It is hard to say what the probabilities of these outcomes are, since they depend on our beliefs about the reliability of the door latches and the properties of earthquakes, but it is possible to extract the desired posterior probability without naming the values of $P(\mathbf{d}|\mathcal{H}_i)$ for each $\mathbf{d}$. All that matters are the relative values of the quantities $P(D|\mathcal{H}_1)$, $P(D|\mathcal{H}_2)$, $P(D|\mathcal{H}_3)$, for the value of D that actually occured. [This is the *likelihood principle* which we met in the last chapter.] The value of D that actually occured is $\mathbf{d} = (0,0,1)$, and no prize visible. First, it is clear that $P(D|\mathcal{H}_3) = 0$, since the datum that no prize is visible is incompatible with $\mathcal{H}_3$. Now, assuming that the contestant selected door 1, how does the probability $P(D|\mathcal{H}_1)$ compare with $P(D|\mathcal{H}_2)$? Assuming that earthquakes are not sensitive to decisions of game show contestants, it would seem that by symmetry, these two quantities have

to be equal. We don't know how likely it is that door 3 falls off its hinges, but however likely it is, it's just as likely to do so whether the prize is behind door 1 or door 2. So, if $P(D|\mathcal{H}_1)$ and $P(D|\mathcal{H}_2)$ are equal, we obtain:

$$\left| \begin{array}{c|c|c} P(\mathcal{H}_1|D)=\frac{P(D|\mathcal{H}_1)(1/3)}{P(D)} & P(\mathcal{H}_2|D)=\frac{P(D|\mathcal{H}_2)(1/3)}{P(D)} & P(\mathcal{H}_3|D)=\frac{P(D|\mathcal{H}_3)(1/3)}{P(D)} \\ =\frac{1}{2} & =\frac{1}{2} & =0. \end{array} \right| \tag{3.32}$$

The two possible hypotheses are now equally likely.

If we assume that the host knows where the prize is and might be acting deceptively, then the answer might be further modified, because we have to view the host's words as part of the data.

Part I

Data Compression

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

About Chapter 4

In this chapter we discuss how to measure the information content of the outcome of a random experiment.

This chapter has some tough bits. If you find the mathematical details hard to follow, skim through them and keep going — you'll be able to enjoy chapters 5 and 6 without this chapter's tools.

Before reading chapter 4, you should have read chapter 2 and worked on exercises 2.15–2.19 and 2.24 (pp. 39–40)

The following exercise is intended to help you think about how to measure information content.

Exercise 3.4: — *Please work on this problem before reading chapter 4.*

You are given 12 balls, all equal in weight except for one that is either heavier or lighter. You are also given a two-pan balance to use. In each use of the balance you may put any number of the 12 balls on the left pan, and the same number on the right pan, and push a button to initiate the weighing; there are three possible outcomes: either the weights are equal, or the balls on the left are heavier, or the balls on the left are lighter. Your task is to design a strategy to determine which is the odd ball *and* whether it is heavier or lighter than the others *in as few uses of the balance as possible.*

While thinking about this problem, you may find it helpful to consider the following questions:

(a) How can one measure *information*?

(b) When you have identified the odd ball and whether it is heavy or light, how much information have you gained?

(c) Once you have designed a strategy, draw a tree showing, for each of the possible outcomes of a weighing, what weighing you perform next. At each node in the tree, how much information have the outcomes so far given you, and how much information remains to be gained?

(d) How much information is gained when you learn (i) the state of a flipped coin; (ii) the states of two flipped coins; (iii) the outcome when a four–sided die is rolled?

(e) How much information is gained on the first step of the weighing problem if 6 balls are weighed against the other 6? How much is gained if 4 are weighed against 4 on the first step, leaving out 4?

4

The Source Coding Theorem

4.1 How can we measure the information content of a random variable?

An ensemble X is a triple $(x, \mathcal{A}_X, \mathcal{P}_X)$, where the *outcome* x is the value of a random variable, which takes on one of a set of possible values, $\mathcal{A}_X = \{a_1, a_2, \ldots, a_i, \ldots, a_I\}$, having probabilities $\mathcal{P}_X = \{p_1, p_2, \ldots, p_I\}$, with $P(x=a_i) = p_i$, $p_i \geq 0$ and $\sum_{a_i \in \mathcal{A}_X} P(x=a_i) = 1$.

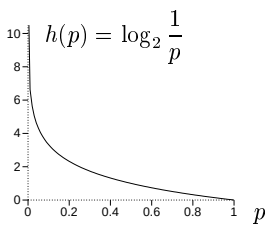
How can we measure the information content of an outcome $x = a_i$ from such an ensemble? In this chapter we examine the assertions that the Shannon information content,

$$h(x=a_i) \equiv \log_2 \frac{1}{p_i}, \tag{4.1}$$

is a sensible measure of the information content of the outcome $x = a_i$, and that the entropy of the ensemble,

$$H(X) = \sum_i p_i \log_2 \frac{1}{p_i}, \tag{4.2}$$

is a sensible measure of the ensemble's average information content. Figure 4.1 shows the Shannon information content of an outcome with probability p, as a function of p. Notice that the less probable an outcome is, the greater its Shannon information content. Figure 4.1 also shows the binary entropy function,



p	$h(p)$	$H_2(p)$
0.001	10.0	0.011
0.01	6.6	0.081
0.1	3.3	0.47
0.2	2.3	0.72
0.5	1.0	1.0

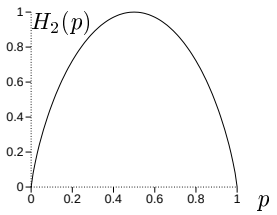


Figure 4.1. The Shannon information content $h(p) = \log_2 \frac{1}{p}$ and the binary entropy function $H_2(p) = H(p, 1-p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}$ as a function of p.

$$H_2(p) = H(p, 1-p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}, \tag{4.3}$$

which is the entropy of the ensemble X whose alphabet and probability distribution are $\mathcal{A}_X = \{a, b\}, \mathcal{P}_X = \{p, (1-p)\}$.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Let us explore these ideas with some examples; then, in section 4.3 and in chapters 5 and 6, we prove that the Shannon information content and the entropy are related to the number of bits needed to describe the outcome of an experiment.

The weighing problem: designing informative experiments

Have you solved the weighing problem (exercise 3.4, p. 65) yet? Are you sure? Notice that in three uses of the balance — which reads either 'left heavier', 'right heavier', or 'balanced' — the number of conceivable outcomes is $3^3 = 27$, whereas the number of possible states of the world is 24: the odd ball could be any of twelve balls, and it could be heavy or light. So in principle, the problem might be solvable in three weighings, but not two, since $3^2 < 24$. If you know how you can determine the odd weight *and* whether it is heavy or light in *three* weighings, then you may read on.

Why is your strategy optimal? What is it about your series of weighings that allows useful information to be gained as quickly as possible? The answer is that at each step of an optimal procedure, the three outcomes ('left heavy', 'right heavy', and 'balance') are *as close as possible to equiprobable*. An optimal solution is shown in figure 4.2.

Suboptimal strategies, such as weighing balls 1–6 against 7–12 on the first step, do not achieve all outcomes with equal probability: these two sets of balls can never balance, so the only possible outcomes are 'left heavy' and 'right heavy'. Some suboptimal strategies produce binary trees rather than ternary trees like the one in figure 4.2, and binary trees are necessarily deeper than balanced ternary trees with the same number of leaves.

The insight that the outcomes should be as near as possible to equiprobable makes it easier to search for an optimal strategy. The first weighing must divide the 24 possible hypotheses into three groups of eight. Then the second weighing must be chosen so that there is a 3:3:2 split of the hypotheses.

Thus we might conclude:

> *An outcome of a random experiment is guaranteed to be most informative if the probability distribution over outcomes is uniform.*

This conclusion agrees with the property of the entropy that you proved when you solved exercise 2.19 (p. 39): the entropy of an ensemble X is biggest if all the outcomes have equal probability $p_i = 1/|\mathcal{A}_X|$.

Guessing games

Example 4.1: The game 'sixty–three'. What's the smallest number of yes/no questions needed to identify an integer x between 0 and 63?

Intuitively, the best questions successively divide the 64 possibilities into equal sized sets. One reasonable strategy asks the following questions:

1: is $x \geq 32$?
2: is $x \bmod 32 \geq 16$?
3: is $x \bmod 16 \geq 8$?
4: is $x \bmod 8 \geq 4$?
5: is $x \bmod 4 \geq 2$?
6: is $x \bmod 2 = 1$?

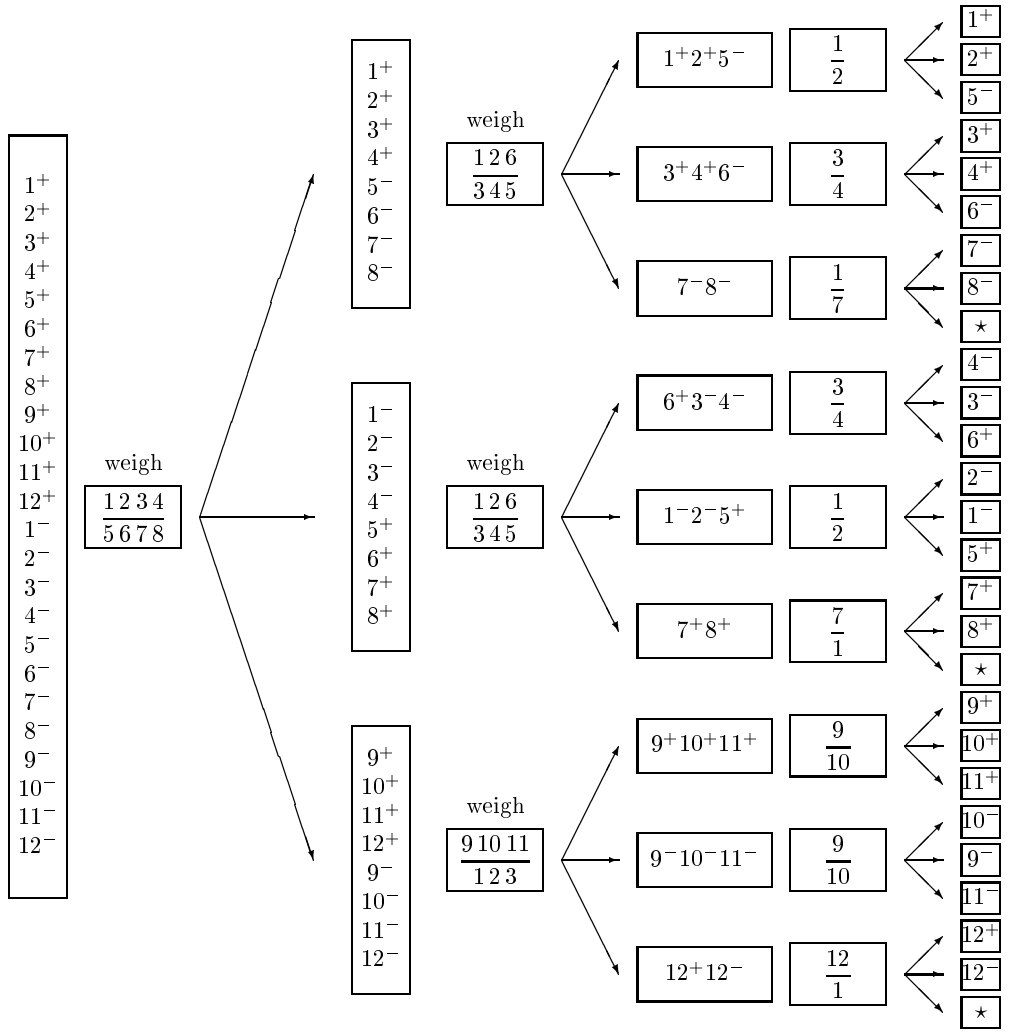
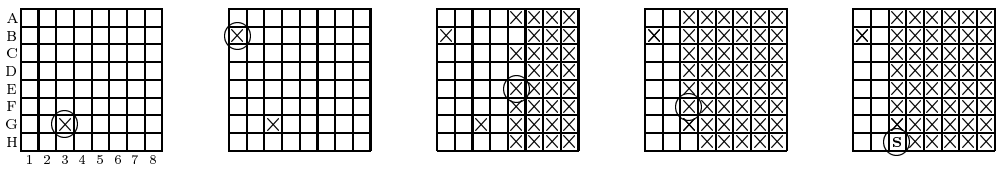


Figure 4.2. An optimal solution to the weighing problem. At each step a pair of boxes shows (a) which hypotheses are still possible, and (b) the next weighing. The 24 hypotheses are written $1^+, \ldots, 12^-$, with, *e.g.*, 1^+ denoting that 1 is the odd ball and it is heavy. Weighings are written by listing the names of the balls on the two pans, separated by a line; for example, in the first weighing, balls 1, 2, 3, and 4 are put on the left hand side and 5, 6, 7 and 8 on the right. In each triplet of arrows the upper arrow leads to the situation when the left side is heavier, the middle arrow when the right side is heavier, and the lower arrow when the outcome is balanced. The three points labelled $\star$ correspond to impossible outcomes.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000



move #	1	2	32	48	49
question	G3	B1	E5	F3	H3
outcome	$x = \mathtt{n}$	$x = \mathtt{n}$	$x = \mathtt{n}$	$x = \mathtt{n}$	$x = \mathtt{y}$
$P(x)$	$\frac{63}{64}$	$\frac{62}{63}$	$\frac{32}{33}$	$\frac{16}{17}$	$\frac{1}{16}$
$h(x)$	0.0227	0.0230	0.0443	0.0874	4.0
Total info.	0.0227	0.0458	1.0	2.0	6.0

Figure 4.3. A game of submarine. The submarine is hit on the 49th attempt.

[The notation $x \bmod 32$, pronounced 'x modulo 32', denotes the remainder when x is divided by 32; for example, $35 \bmod 32 = 3$ and $32 \bmod 32 = 0$.]

The answers to these questions, if translated from $\{\text{yes}, \text{no}\}$ to $\{1, 0\}$, give the binary expansion of x, for example $35 \Rightarrow 100011$. Six questions suffice.

What are the Shannon information contents of the outcomes in this example? If we assume that all values of x are equally likely, then the answers to the questions are independent and each has Shannon information content $\log_2(1/0.5) = 1\,\text{bit}$; the total Shannon information gained is always six bits. Furthermore, the number x that we learn from these questions is a six–bit binary number. Our questioning strategy defines a way of encoding the random variable x as a binary file. So far, the Shannon information content makes sense: it measures the length of a binary file that encodes x.

However, we have not yet studied ensembles where the outcomes have unequal probabilities. Does the Shannon information content make sense there?

The game of submarine

In the game of battleships, each player hides a fleet of ships in a sea represented by a square grid. On each turn, one player attempts to hit the other's ships by firing at one square in the opponent's sea. The response to a selected square such as 'G3' is either 'miss', 'hit', or 'hit and destroyed'.

In a boring version of battleships called `submarine`, each player hides just one submarine in one square of an eight–by–eight grid. Figure 4.3 shows a few pictures of this game in progress; the submarine is hit on the 49th attempt.

Each shot made by a player defines an ensemble. The two possible outcomes are $\{\mathtt{y}, \mathtt{n}\}$, corresponding to a hit and a miss, and their probabilities depend on the state of the board. At the beginning, $P(\mathtt{y}) = 1/64$ and $P(\mathtt{n}) = 63/64$. At the second shot, if the first shot missed, $P(\mathtt{y}) = 1/63$ and $P(\mathtt{n}) = 62/63$. At the third shot, if the first two shots missed, $P(\mathtt{y}) = 1/62$ and $P(\mathtt{n}) = 61/62$.

The Shannon information gained from an outcome x is $h(x) = \log(1/P(x))$. If we are lucky, and hit the submarine on the first shot, then

$$h(x) = h_{(1)}(\mathtt{y}) = \log_2 64 = 6\,\text{bits}. \tag{4.4}$$

This makes sense. We have learnt the hiding place, which could have been any of 64 squares; so we have, by one lucky binary question, learnt six bits.

What if the first shot misses? The Shannon information that we gain from this outcome is

$$h(x) = h_{(1)}(\mathtt{n}) = \log_2 \frac{64}{63} = 0.0227 \text{ bits.} \tag{4.5}$$

Does this make sense? It is not so obvious. If our second shot also misses, the Shannon information content of the second outcome is

$$h_{(2)}(\mathtt{n}) = \log_2 \frac{63}{62} = 0.0230 \text{ bits.} \tag{4.6}$$

If we miss thirty-two times (firing at a new square each time), the total Shannon information gained is

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{33}{32} \\ &= 0.0227 + 0.0230 + \cdots + 0.0430 \;=\; 1.0 \text{ bits.} \end{aligned} \tag{4.7}$$

Why this round number? Well, what have we learnt? We now know that the submarine is not in any of the 32 squares we fired at; learning that fact is just like playing a game of `sixty-three` (page 67), asking as our first question 'is x one of the thirty–two numbers corresponding to these squares I fired at?', and receiving the answer 'no'. This answer rules out half of the hypotheses, so it gives us one bit.

After 48 unsuccessful shots, the information gained is 2 bits: the unknown location has been narrowed down to one quarter of the original hypothesis space.

What if we hit the submarine on the 49th shot, when there were 16 squares left? The Shannon information content of this outcome is

$$h_{(49)}(\mathtt{y}) = \log_2 16 = 4.0 \text{ bits.} \tag{4.8}$$

The total Shannon information content of all the outcomes is

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{17}{16} + \log_2 \frac{16}{1} \\ &= \; 0.0227 + 0.0230 + \cdots + 0.0874 + 4.0 \;=\; 6.0 \text{ bits.} \end{aligned} \tag{4.9}$$

So once we know where the submarine is, the total Shannon information content gained is 6 bits. This result holds regardless of when we hit the submarine. If we hit it when there are n squares left to choose from — n was 16 in equation (4.9) — then the total information gained is:

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{n+1}{n} + \log_2 \frac{n}{1} \\ &= \; \log_2 \left[\frac{64}{63} \times \frac{63}{62} \times \cdots \times \frac{n+1}{n} \times \frac{n}{1} \right] \;=\; \log_2 \frac{64}{1} \;=\; 6 \text{ bits.} \end{aligned} \tag{4.10}$$

4.2 Data compression

The preceding examples justify the idea that the Shannon information content of an outcome is a natural measure of its information content. Improbable

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

outcomes do convey more information than probable outcomes. We now discuss the information content of a source by considering how many bits are needed to describe the outcome of an experiment, that is, by studying data compression.

If we can show that we can compress data from a particular source into a file of b bits per source symbol and recover the data reliably, then we will say that the information content of that source is at most b bits per symbol. We will show that, for any source, the information content of the source is intimately related to its entropy.

Example: Compression of text files

A file is composed of a sequence of bytes. A byte is composed of 8 bits and can have a decimal value between 0 and 255. A typical text file is composed of the ASCII character set (decimal values 0 to 127). This character set uses only seven of the eight bits in a byte.

Here we use the word 'bit' with its meaning, 'a symbol with two values', not to be confused with the unit of information content.

Solutions on p. 84

Exercise 4.2:[B1] By how much could the size of a file be reduced given that it is an ASCII file? How would you achieve this reduction?

Intuitively, it seems reasonable to assert that an ASCII file contains 7/8 as much information as an arbitrary file of the same size, since we already know one out of every eight bits before we even look at the file. This is a simple example of redundancy. Most sources of data have further redundancy: English text files use the ASCII characters with non-equal frequency; certain pairs of letters are more probable than others; and entire words can be predicted given the context and a semantic understanding of the text.

Some simple data compression methods that define measures of information content

One way of measuring the information content of a random variable is simply to count the *possible* outcomes, $|\mathcal{A}_X|$. (The number of elements in a set $\mathcal{A}$ is denoted by $|\mathcal{A}|$.) If we imagine giving a binary name to each outcome, the length of each name would be $\log_2 |\mathcal{A}_X|$ bits, if $|\mathcal{A}_X|$ happened to be a power of 2. We thus make the following definition.

The raw bit content of X is

$$H_0(X) = \log_2 |\mathcal{A}_X|. \tag{4.11}$$

$H_0(X)$ is a lower bound for the number of binary questions that are always guaranteed to identify an outcome from the ensemble X. It is an additive quantity: the raw bit content of an ordered pair x, y, having $|\mathcal{A}_X||\mathcal{A}_Y|$ possible outcomes, satisfies

$$H_0(X, Y) = H_0(X) + H_0(Y). \tag{4.12}$$

This measure of information content does not include any probabilistic element, and the encoding rule it corresponds to does not 'compress' the source data, it simply maps each outcome to a constant-length binary string.

Exercise 4.3:[A2] Could there be a compressor that maps an outcome x to a binary code $c(x)$, and a decompressor that maps c back to x, such that *every possible outcome* is compressed into a binary code of length *shorter* than $H_0(X)$ bits?

Even though a simple counting argument shows that it is impossible to make a reversible compression program that reduces the size of *all files*, amateur compression enthusiasts frequently announce that they have invented a program that can do this — indeed that they can further compress compressed files by putting them through their compressor several times. Stranger yet, patents have been granted to these modern–day alchemists. See the `comp.compression` frequently asked questions for further reading.[1]

There are only two ways in which a 'compressor' can actually compress files:

1. A *lossy* compressor compresses some files, but maps some files to the *same* encoding. We'll assume that the user requires perfect recovery of the source file, so the occurrence of one of these confusable files leads to a failure (though in applications such as image compression, lossy compression is viewed as satisfactory). A lossy compressor has a probability δ of failure, where δ is the probability of the source string's being one of the confusable files. If δ can be made very small then a lossy compressor may be practically useful.

2. A *lossless* compressor maps all files to different encodings; if a lossless compressor shortens some files, it necessarily *makes others longer*. We try to design the compressor so that the probability that a file is lengthened is very small.

In this chapter we discuss a simple lossy compressor. In subsequent chapters we discuss lossless compression methods.

Whichever type of compressor we construct, we need somehow to take into account the *probabilities* of the different outcomes. Imagine comparing the information contents of two text files — one in which all 128 ASCII characters are used with equal probability, and one in which the characters are used with their frequencies in English text. Can we define a measure of information content that distinguishes between these two files? Intuitively, the latter file contains less information per character because it is more predictable. One simple way to use our knowledge that some symbols have a smaller probability is to imagine recoding the observations into a smaller alphabet — thus losing the ability to encode some of the more improbable symbols — and then measuring the raw bit content of the new alphabet. For example, we might take a risk when compressing English text, guessing that the most infrequent characters won't occur, and make a reduced ASCII code that omits { `!`, `@`, `#`, `%`, `^`, `*`, `~`, `<`, `>`, `/`, `\`, `_`, `{`, `}`, `[`, `]`, `|` }, thereby reducing the total number of characters by seventeen. Thus we can give new names to a *subset* of the possible outcomes and count how many names we use. The larger the risk we are willing to take, the smaller the number of names we need.

We introduce a parameter δ that describes the risk we are taking when using this compression method: δ is the probability that there will be no name for an outcome x.

Example 4.4: Let

$$\begin{aligned} \mathcal{A}_X &= \{ \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h} \}, \\ \text{and} \quad \mathcal{P}_X &= \{ \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{3}{16}, \tfrac{1}{64}, \tfrac{1}{64}, \tfrac{1}{64}, \tfrac{1}{64} \}. \end{aligned} \tag{4.13}$$

[1] `http://www.lib.ox.ac.uk/internet/news/faq/`

The raw bit content of this ensemble is 3 bits, corresponding to 8 binary names. But notice that $P(x \in \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}) = 15/16$. So if we are willing to run a risk of $\delta = 1/16$ of not having a name for x, then we can get by with four names — half as many names as are needed if every $x \in \mathcal{A}_X$ has a name.

Figure 4.4 shows binary names that could be given to the different outcomes in the cases $\delta = 0$ and $\delta = 1/16$. When $\delta = 0$ we need 3 bits to encode the outcome: when $\delta = 1/16$ we only need 2 bits.

$\delta = 0$		$\delta = 1/16$	
x	$c(x)$	x	$c(x)$
a	000	a	00
b	001	b	01
c	010	c	10
d	011	d	11
e	100	e	—
f	101	f	—
g	110	g	—
h	111	h	—

Figure 4.4. Binary names for the outcomes, for two failure probabilities δ.

Let us now formalize this idea. To make a compression strategy with risk δ, we make the smallest possible subset S_δ such that the probability that x is not in S_δ is less than or equal to δ, *i.e.*, $P(x \notin S_\delta) \leq \delta$. For each value of δ we can then define a new measure of information content — the log of the size of this smallest subset S_δ. [In ensembles in which several elements have the same probability, there may be several smallest subsets that contain different elements, but all that matters is their sizes (which are equal), so we will not dwell on this ambiguity.]

The smallest δ–sufficient subset S_δ is the smallest subset of $\mathcal{A}_X$ satisfying

$$P(x \in S_\delta) \geq 1 - \delta. \tag{4.14}$$

The subset S_δ can be constructed by ranking the elements of $\mathcal{A}_X$ in order of decreasing probability and adding successive elements starting from the most probable elements until the total probability exceeds $(1-\delta)$.

We can make a data compression code by assigning a binary name to each element of the smallest sufficient subset. This compression scheme motivates the following measure of information content:

The essential bit content of X is:

$$H_\delta(X) = \log_2 |S_\delta| = \log_2 \min \{|S| : S \subseteq \mathcal{A}_X, P(x \in S) \geq 1 - \delta\}. \tag{4.15}$$

Note that $H_0(X)$ is the special case of $H_\delta(X)$ with $\delta = 0$ (if $P(x) > 0$ for all $x \in \mathcal{A}_X$). [Caution: Do not confuse $H_0(X)$ and $H_\delta(X)$ with the function $H_2(p)$ displayed in figure 4.1.]

Figure 4.5 shows $H_\delta(X)$ for the ensemble defined in example 4.4 as a function of δ.

Extended ensembles

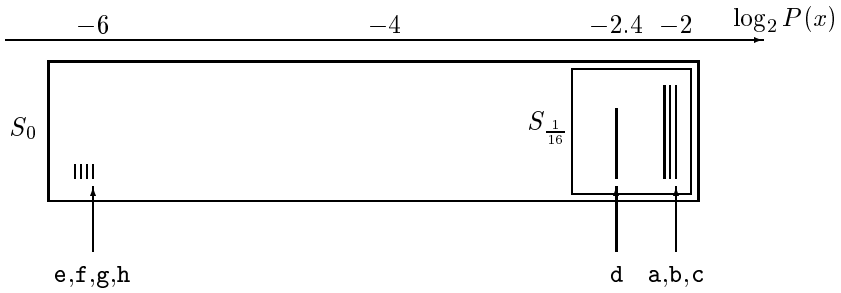
Is this compression method any more useful if we compress *blocks* of symbols from a source?

We now turn to examples where the outcome $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is a string of independent identically distributed random variables from a single ensemble X. We will denote by X^N the ensemble $(X_1, X_2, \ldots, X_N)$. Remember that entropy is additive for independent variables,[2] so $H(X^N) = NH(X)$.

Example 4.5: Consider $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ where $x_n \in \{\mathtt{0}, \mathtt{1}\}$, with probabilities p_0=0.9, p_1=0.1. The most probable strings $\mathbf{x}$ are those with most 0s. If $r(\mathbf{x})$ is the number of 1s in $\mathbf{x}$ then

$$P(\mathbf{x}) = p_0^{N-r(\mathbf{x})} p_1^{r(\mathbf{x})}. \tag{4.16}$$

[2] There should have been an exercise on this by now.



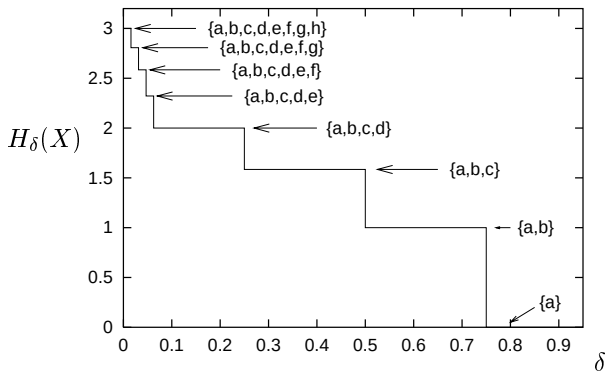


Figure 4.5. The outcomes of X (from example 4.4 (p. 72)), ranked by their probability, and the essential bit content $H_\delta(X)$. The labels on the graph show the smallest sufficient set as a function of δ. Note $H_0(X) = 3$ bits and $H_{1/16}(X) = 2$ bits.

To evaluate $H_\delta(X^N)$ we must find the smallest sufficient subset S_δ. This subset will contain all $\mathbf{x}$ with $r(\mathbf{x}) = 0, 1, 2, \ldots$, up to some $r_{\max}(\delta) - 1$, and some of the $\mathbf{x}$ with $r(\mathbf{x}) = r_{\max}(\delta)$. Figures 4.6 and 4.7 show graphs of $H_\delta(X^N)$ against δ for the cases $N = 4$ and $N = 10$. The steps are the values of δ at which $|S_\delta|$ changes by 1, and the cusps where the slope of the staircase changes are the points where $r_{\max}$ changes by 1.

Exercise 4.6:^{C2} What are the mathematical shapes of the curves between the cusps?

For the examples shown in figures 4.5–4.7, $H_\delta(X^N)$ depends strongly on the value of δ, so it might not seem a fundamental or useful definition of information content. But we will consider what happens as N, the number of independent variables in X^N, increases. We will find the remarkable result that $H_\delta(X^N)$ becomes almost independent of δ — and for all δ it is very close to $NH(X)$, where $H(X)$ is the entropy of one of the random variables.

Figure 4.8 illustrates this asymptotic tendency for the binary ensemble of example 4.5. As N increases, $\frac{1}{N}H_\delta(X^N)$ becomes an increasingly flat function, except for tails close to $\delta = 0$ and 1. As long as we are allowed a tiny probability of error δ, compression down to NH bits is possible. Even if we are allowed a large probability of error, we still can compress only down to NH bits. This is the source coding theorem.

Theorem 4.1 *Let X be an ensemble with entropy $H(X) = H$ bits. Given $\epsilon > 0$ and $0 < \delta < 1$, there exists a positive integer N_0 such that for $N > N_0$,*

$$\left| \frac{1}{N} H_\delta(X^N) - H \right| < \epsilon. \tag{4.17}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

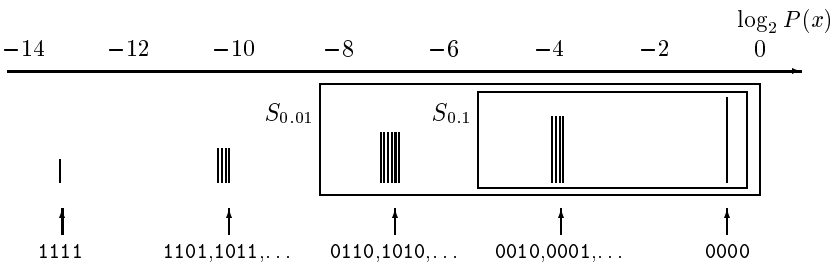
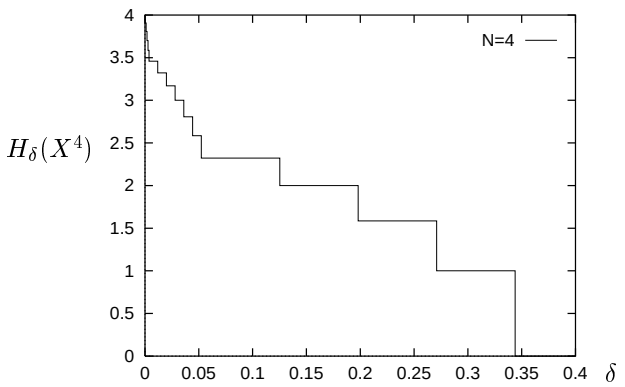


Figure 4.6. The sixteen outcomes of the ensemble X^4 with $p_1 = 0.1$, ranked by probability, and the essential bit content $H_\delta(X^4)$. The upper schematic diagram indicates the strings's probabilities by the vertical lines's lengths (not to scale).



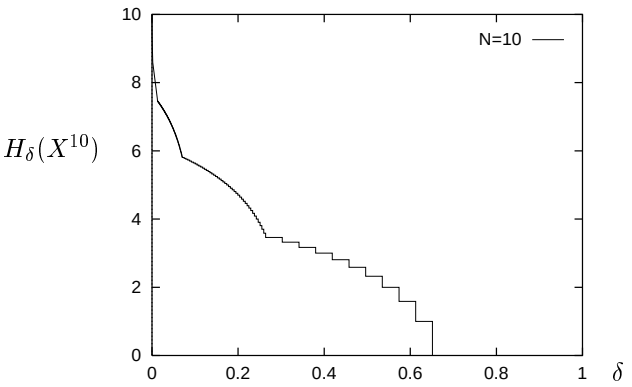


Figure 4.7. $H_\delta(X^N)$ for $N = 10$ binary variables with $p_1 = 0.1$.

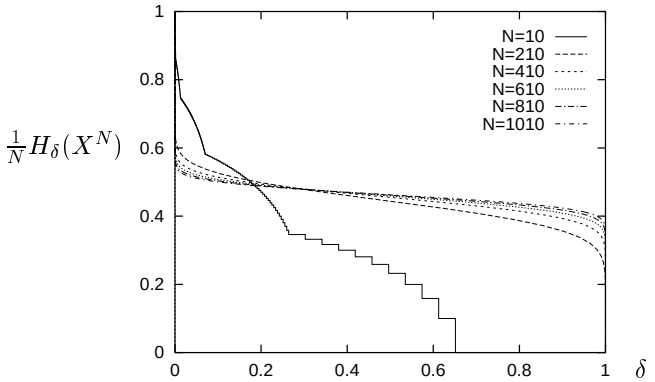


Figure 4.8. $\frac{1}{N}H_\delta(X^N)$ for $N = 10, 210, \ldots, 1010$ binary variables with $p_1 = 0.1$.

$\mathbf{x}$	$\log_2(P(\mathbf{x}))$
...1.....1....1....1.1.....1.....1.....1.....1.....1.....11...	-50.1
.....1....1....1.....1....1.....1.....1.....1....	-37.3
.....1...1..1...1....11..1.1.....11.....1...1.1..1...1.....1.	-65.9
1.1...1.....1.....11.1..1.....1.....1.....1.....1..1.11.....	-56.4
...11.....1...1....1.1.....1.....1....1...1....1.....1.....	-53.2
.....1.....1.....1.1.....1.....1.....1...1..1.....1.....	-43.7
.....1.....1.....1...1.....1.....1.....1.....1..11.....	-46.8
.....1..1..1.....111.....1.....1.....1.....1.1...1...1.....1	-56.4
.....1.....1.....1.....1.....1....1.....1...	-37.3
.....1.....1.....1.....1....1.1.1.1..1.....1.	-43.7
1.....1.....1...1.....1...1....1...1.....1..11..1.1...1.....	-56.4
.....11.1.....1.....1.....1.....1.....	-37.3
.1.....1...1.1.....1.....11.....1.1...1.....1.....11.....	-56.4
.....1...1..1.....1..11.1.1.1...1.....1.....1.....1..1.....	-59.5
.....11.1.....1....1..1.....1.....1.....1.....1.....	-46.8
.....	-15.2
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111	-332.1

Figure 4.9. The top 15 strings are samples from X^{100}, where $p_1 = 0.1$ and $p_0 = 0.9$. The bottom two are the most and least probable strings in this ensemble. The final column shows the probabilities of the random strings, which may be compared with the entropy $H(X^{100}) = 46.9$ bits.

4.3 Typicality

Why does increasing N help? Let's examine long strings from X^N. Table 4.9 shows fifteen samples from X^N for $N = 100$ and $p_1 = 0.1$. The probability of a string $\mathbf{x}$ that contains r 1s and $N-r$ 0s is

$$p(\mathbf{x}) = p_1^r (1-p_1)^{N-r}. \tag{4.18}$$

The number of strings that contain r 1s is

$$n(r) = \binom{N}{r}. \tag{4.19}$$

So the number of 1s, r, has a binomial distribution:

$$p(r) = \binom{N}{r} p_1^r (1-p_1)^{N-r}. \tag{4.20}$$

These functions are shown in figure 4.10. The mean of r is Np_1, and its standard deviation is $\sqrt{Np_1(1-p_1)}$. If N is 100 then

$$r \sim Np_1 \pm \sqrt{Np_1(1-p_1)} \simeq 10 \pm 3. \tag{4.21}$$

If $N = 1000$ then

$$r \sim 100 \pm 10. \tag{4.22}$$

Notice that as N gets bigger, the probability distribution of r becomes more concentrated in the sense that the total range of possible values of r grows as N, but the standard deviation of r only grows as $\sqrt{N}$. That r is most likely to fall in a small range of values implies that the outcome $\mathbf{x}$ is also most likely to fall in a small corresponding subset of outcomes that we will call the *typical set*.

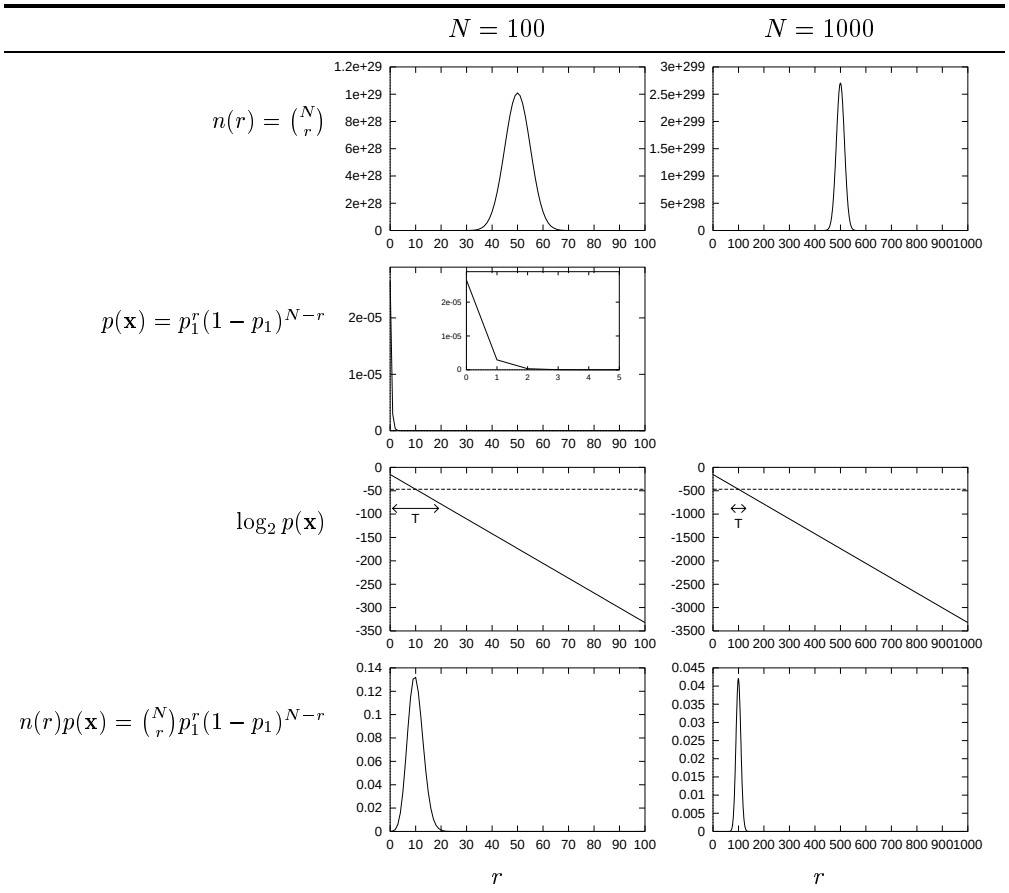


Figure 4.10. Anatomy of the typical set T. For $p_1 = 0.1$ and $N = 100$ and $N = 1000$, these graphs show $n(r)$, the number of strings containing r 1s; the probability $p(\mathbf{x})$ of a single string that contains r 1s; the same probability on a log scale; and the total probability $n(r)p(\mathbf{x})$ of all strings that contain r 1s. The number r is on the horizontal axis. The plot of $\log_2 p(\mathbf{x})$ also shows by a dotted line the mean value of $\log_2 p(\mathbf{x}) = -NH_2(p_1)$ which equals -46.9 when $N = 100$ and -469 when $N = 1000$. The typical set includes only the strings that have $\log_2 p(\mathbf{x})$ close to this value. The range marked T shows the set $T_{N\beta}$ for $N = 100$ and $\beta = 0.29$ (left) and $N = 1000$, $\beta = 0.09$ (right).

Definition of the typical set

Let us generalize our discussion to an arbitrary ensemble X with alphabet $\mathcal{A}_X$ and define typicality. Our definition of a typical string will involve the string's probability. A long string of N symbols will contain with high probability about p_1N occurrences of the first symbol, p_2N occurrences of the second, etc. Hence the probability of this string is roughly

$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3)\ldots p(x_N) \simeq p_1^{(p_1N)}p_2^{(p_2N)}\ldots p_I^{(p_IN)} \tag{4.23}$$

so that

$$\log_2 p(\mathbf{x}) \simeq N\sum_i p_i \log_2 p_i \simeq -NH. \tag{4.24}$$

So the random variable $\frac{1}{N}\log_2\frac{1}{p(\mathbf{x})}$ is very likely to have value very close to H. We define the typical elements of $\mathcal{A}_X^N$ to be those elements that have probability close to 2^{-NH} (note that the typical set, unlike the smallest sufficient subset, does *not* include the most probable elements of $\mathcal{A}_X^N$, but we will show that these most probable elements contribute negligible probability).

We introduce a parameter β that defines how close the probability has to be to 2^{-NH} for an element to be 'typical'. We call the set of typical elements the typical set, $T_{N\beta}$

$$T_{N\beta} \equiv \left\{ \mathbf{x} \in \mathcal{A}_X^N : \left| \frac{1}{N}\log_2\frac{1}{P(\mathbf{x})} - H \right| < \beta \right\}. \tag{4.25}$$

We will show that whatever value of β we choose, the typical set contains almost all the probability as N increases.

This important result is sometimes called the 'Asymptotic Equipartition' Principle.

'Asymptotic Equipartition' Principle. For an ensemble of N independent identically distributed (i.i.d.) random variables $X^N \equiv (X_1, X_2, \ldots, X_N)$, with N sufficiently large, the outcome $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is almost certain to belong to a subset of $\mathcal{A}_X^N$ having only $2^{NH(X)}$ members, each having probability 'close to' $2^{-NH(X)}$.

Notice that if $H(X) < H_0(X)$ then $2^{NH(X)}$ is a *tiny* fraction of the number of possible outcomes $|\mathcal{A}_X^N| = |\mathcal{A}_X|^N = 2^{NH_0(X)}$.

> The term equipartition is chosen to describe the idea that the members of the typical set have *roughly equal* probability. [This should not be taken too literally, hence my use of quotes around 'asymptotic equipartition'; see page 81.] A second meaning for equipartition, in thermal physics, is the idea that each degree of freedom of a classical system has equal average energy, $\frac{1}{2}kT$. This second meaning is not intended here.

The 'asymptotic equipartition' principle is equivalent to:

Shannon's source coding theorem. N i.i.d. random variables each with entropy $H(X)$ can be compressed into more than $NH(X)$ bits with negligible risk of information loss, as $N \to \infty$; conversely if they are compressed into fewer than $NH(X)$ bits it is virtually certain that information will be lost.

These two theorems are equivalent because we can define a compression algorithm that gives a distinct name of length $NH(X)$ bits to each $\mathbf{x}$ in the typical set.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

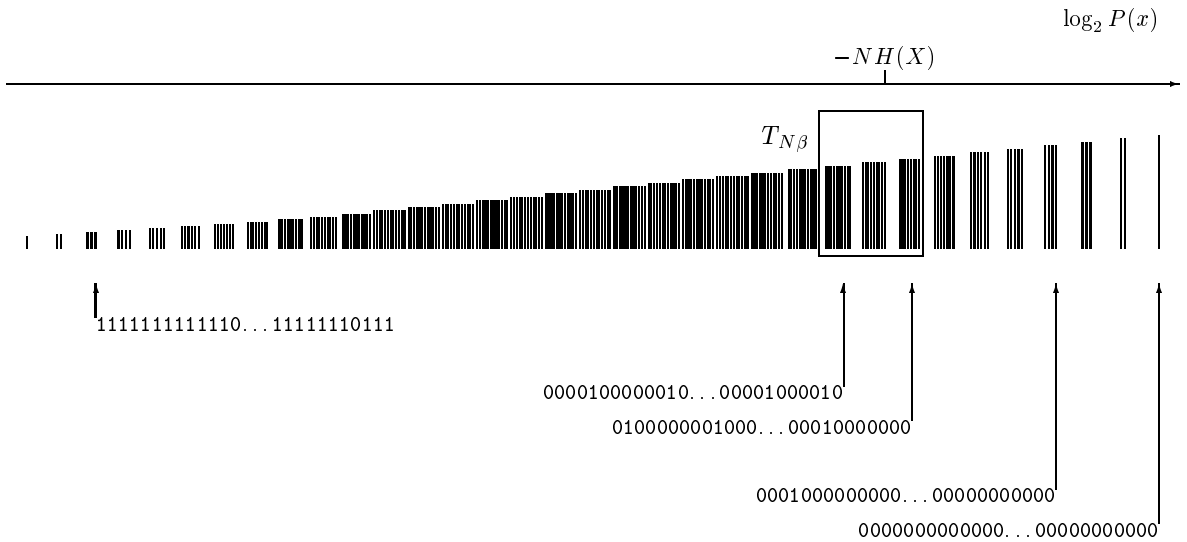


Figure 4.11. Schematic diagram showing all strings in the ensemble X^N ranked by their probability.

4.4 Theorems and proofs

We will prove the 'asymptotic equipartition' principle first; then we will show that for any given δ there is a sufficiently big N such that $H_\delta(X^N) \simeq NH$.

The law of large numbers

Our proof of the source coding theorem uses the law of large numbers.

Mean and variance of a real random variable[3] are $\mathcal{E}[u] = \bar{u} = \sum_u P(u)u$ and $\mathrm{var}(u) = \sigma_u^2 = \mathcal{E}[(u-\bar{u})^2] = \sum_u P(u)\,(u-\bar{u})^2$.

Chebyshev's inequality 1. Let t be a non-negative real random variable, and let α be a positive real number. Then

$$P(t \geq \alpha) \;\leq\; \bar{t}/\alpha. \tag{4.26}$$

Proof: $P(t \geq \alpha) = \sum_{t \geq \alpha} P(t)$. We multiply each term by $t/\alpha \geq 1$ and obtain: $P(t \geq \alpha) \leq \sum_{t \geq \alpha} P(t)t/\alpha$. We add the (non-negative) missing terms and obtain: $P(t \geq \alpha) \leq \sum_t P(t)t/\alpha = \bar{t}/\alpha$.

Chebyshev's inequality 2. Let x be a random variable, and let α be a positive real number. Then

$$P\left((x-\bar{x})^2 \geq \alpha\right) \;\leq\; \sigma_x^2/\alpha. \tag{4.27}$$

Proof: Take $t = (x-\bar{x})^2$ and apply the previous proposition.

[3] Technical note: strictly I am assuming here that u is a function $u(x)$ of a sample x from a finite discrete ensemble X. Then the summations $\sum_u P(u)f(u)$ should be written $\sum_x P(x)f(u(x))$. This means that $P(u)$ is a finite sum of delta functions. This restriction guarantees that the mean and variance of u do exist, which is not necessarily the case for general $P(u)$.

Weak law of large numbers. Take x to be the average of N independent random variables $h_1, \ldots, h_N$, having common mean $\bar{h}$ and common variance σ_h^2: $x = \frac{1}{N}\sum_{n=1}^{N} h_n$. Then

$$P((x-\bar{h})^2 \geq \alpha) \leq \sigma_h^2/\alpha N. \tag{4.28}$$

Proof: obtained by showing that $\bar{x} = \bar{h}$ and that $\sigma_x^2 = \sigma_h^2/N$.

We are interested in x being very close to the mean (α very small). No matter how large σ_h^2 is, and no matter how small the required α is, and no matter how small the desired probability of $(x-\bar{h})^2 \geq \alpha$, we can always achieve it by taking N large enough.

Proof of theorem 4.1

We apply the law of large numbers to the random variable $\frac{1}{N}\log_2 \frac{1}{P(\mathbf{x})}$ defined for $\mathbf{x}$ drawn from the ensemble X^N. This random variable can be written as the average of N terms $h_n = \log_2(1/P(x_n))$, each of which is a random variable with mean $H = H(X)$ and variance $\sigma^2 \equiv \mathrm{var}[\log_2(1/P(x_n))]$. We again define a typical set with parameters N and β thus:

$$T_{N\beta} = \left\{ \mathbf{x} \in \mathcal{A}_X^N : \left[\frac{1}{N} \log_2 \frac{1}{P(\mathbf{x})} - H \right]^2 < \beta^2 \right\}. \tag{4.29}$$

For all $\mathbf{x} \in T_{N\beta}$, the probability of $\mathbf{x}$ satisfies

$$2^{-N(H+\beta)} < P(\mathbf{x}) < 2^{-N(H-\beta)}. \tag{4.30}$$

And by the law of large numbers,

$$P(\mathbf{x} \in T_{N\beta}) \geq 1 - \frac{\sigma^2}{\beta^2 N}. \tag{4.31}$$

We have thus proved the 'asymptotic equipartition' principle. As N increases, the probability that $\mathbf{x}$ falls in $T_{N\beta}$ approaches 1, for any β. How does this result relate to source coding?

Part 1: $\frac{1}{N} H_\delta(X^N) < H + \epsilon$.

We show how *small* $H_\delta(X^N)$ must be by calculating the largest cardinality that $T_{N\beta}$ could have. We are free to set β to any convenient value. The smallest possible probability that a member of $T_{N\beta}$ can have is $2^{-N(H+\beta)}$, and the total probability that $T_{N\beta}$ contains can't be any bigger than 1. So

$$|T_{N\beta}| 2^{-N(H+\beta)} < 1, \tag{4.32}$$

that is, the size of the typical set is bounded by

$$|T_{N\beta}| < 2^{N(H+\beta)}. \tag{4.33}$$

If we set $\beta = \epsilon$ and N_0 such that $\frac{\sigma^2}{\epsilon^2 N} \leq \delta$, then $P(T_{N\beta}) \geq 1 - \delta$, and the set $T_{N\beta}$ becomes a witness to the fact that $H_\delta(X^N) \leq \log_2 |T_{N\beta}| < N(H+\epsilon)$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part 2: $\frac{1}{N}H_\delta(X^N) > H - \epsilon$.

Imagine that someone claims this second part is not so — that, for any N, the smallest δ–sufficient subset S_δ is smaller than the above inequality would allow. We make use of our typical set to show that they must be mistaken. Remember that we are free to set β to any value we choose. We will set $\beta = \epsilon/2$, so that our task is to prove that a subset S' having $|S'| \leq 2^{N(H-2\beta)}$ and achieving $P(\mathbf{x} \in S') \geq 1 - \delta$ cannot exist (for N greater than an N_0 that we will specify).

So, let us consider the probability of falling in this rival smaller subset S'. The probability of the subset S' is

$$P(\mathbf{x} \in S' \cap T_{N\beta}) + P(\mathbf{x} \in S' \cap \overline{T_{N\beta}}), \tag{4.34}$$

where $\overline{T_{N\beta}}$ denotes the complement $\{\mathbf{x} \notin T_{N\beta}\}$. The maximum value of the first term is found if $S' \cap T_{N\beta}$ contains $2^{N(H-2\beta)}$ outcomes all with the maximum probability, $2^{-N(H-\beta)}$. The maximum value the second term can have is $P(\mathbf{x} \notin T_{N\beta})$. So:

$$P(\mathbf{x} \in S') \leq 2^{N(H-2\beta)}\, 2^{-N(H-\beta)} + \frac{\sigma^2}{\beta^2 N} = 2^{-N\beta} + \frac{\sigma^2}{\beta^2 N}. \tag{4.35}$$

We can now set $\beta = \epsilon/2$ and N_0 such that $P(\mathbf{x} \in S') < 1 - \delta$, which shows that S' cannot satisfy the definition of a sufficient subset S_δ. Thus *any* subset S' such that $|S'| \leq 2^{N(H-\epsilon)}$ has probability less than $1-\delta$, so by the definition of H_δ, $H_\delta(X^N) > N(H - \epsilon)$.

Thus for large enough N, the function $\frac{1}{N}H_\delta(X^N)$ is essentially a constant function of δ, as was illustrated in figure 4.8. □

4.5 Comments

The source coding theorem has two parts, $\frac{1}{N}H_\delta(X^N) < H + \epsilon$, and $\frac{1}{N}H_\delta(X^N) > H - \epsilon$. Both results are interesting.

The first part tells us that even if δ is extremely small, the number of bits per symbol needed to specify a long N-symbol string $\mathbf{x}$ with vanishingly small error probability, $\frac{1}{N}H_\delta(X^N)$, does not have to exceed $H + \epsilon$ bits. We need to have only a tiny tolerance to error, and the number of bits required drops significantly from $H_0(X)$ to $(H + \epsilon)$.

What happens if we are yet more tolerant to compression errors? Part 2 tells us that even if δ is very close to 1, so that errors are made most of the time, the average number of bits per symbol needed to specify $\mathbf{x}$ must still be at least $H - \epsilon$ bits. These two extremes tell us that regardless of our specific allowance for error, the number of bits per symbol needed to specify $\mathbf{x}$ is H bits; no more and no less.

Caveat regarding 'asymptotic equipartition'

I put the words 'asymptotic equipartition' in quotes because it is important not to think that the elements of the typical set $T_{N\beta}$ really do have roughly the same probability as each other. They are similar in probability only in the sense that their values of $\log_2 \frac{1}{P(\mathbf{x})}$ are within $2N\beta$ of each other. Now, as β is decreased, how does N have to increase, if we are to keep our bound on

the mass of the typical set, $P(\mathbf{x} \in T_{N\beta}) \geq 1 - \frac{\sigma^2}{\beta^2 N}$, constant? N must grow as $1/\beta^2$, so, if we write β in terms of N as $\alpha/\sqrt{N}$, for some constant α, then the most probable string in the typical set will be of order $2^{\alpha\sqrt{N}}$ times greater than the least probable string in the typical set. As β decreases, N increases, and this ratio $2^{\alpha\sqrt{N}}$ grows exponentially. Thus we have 'equipartition' only in a weak sense!

4.6 Exercises

Weighing problems

Solutions on p. 84

Exercise 4.7:^{B2} You are given 16 balls, all of which are equal in weight except for one that is either heavier or lighter. You are also given a bizarre two-pan balance that can report only two outcomes: 'the two sides balance' or 'the two sides do not balance'. Design a strategy to determine which is the odd ball in as few uses of the balance as possible.

Exercise 4.8:^{B2} You have a two-pan balance; your job is to weigh out bags of flour with integer weights 1 to 40 pounds inclusive. How many weights do you need? [You are allowed to put weights on either pan. You're only allowed to put one flour bag on the balance at a time.]

Solutions on p. 84

Exercise 4.9:^{C3} (a) Is it possible to solve exercise 2.15 (p. 39) (the weighing problem with 12 balls and the three–outcome balance) using a sequence of three *fixed* weighings, such that the balls chosen for the second weighing do not depend on the outcome of the first, and the third weighing does not depend on the first or second?

(b) Find a general solution to the weighing problem in which exactly one of N balls is odd. Show that in W weighings, an odd ball can be identified from among $N = (3^W - 3)/2$ balls.

Solutions on p. 85

Exercise 4.10:^{C3} You are given 12 balls and the three–outcome balance of exercise 2.15 (p. 39); this time, *two* of the balls are odd; each odd ball may be heavy or light, and we don't know which. We want to identify the odd balls and in which direction they are odd.

(a) *Estimate* how many weighings are required by the optimal strategy. And what if there are three odd balls?

(b) How do your answers change if it is known that all the regular balls weigh 100 g, that light balls weigh 99 g, and heavy ones weigh 110 g?

Source coding

Exercise 4.11:^{A2} Let $\mathcal{P}_X = \{0.4, 0.6\}$. Sketch $\frac{1}{N}H_\delta(X^N)$ as a function of δ for $N = 1, 2$ and 100.

Exercise 4.12:^{B2} Let $\mathcal{P}_Y = \{0.5, 0.5\}$. Sketch $\frac{1}{N}H_\delta(Y^N)$ as a function of δ for $N = 1, 2, 3$ and 100.

Solutions on p. 87

Exercise 4.13:^{B2} Discuss the relationship between the proof of the 'asymptotic equipartition' principle and the equivalence (for large systems) of the Boltzmann entropy and the Gibbs entropy.

Distributions that don't obey the law of large numbers

The law of large numbers, which we used in this chapter, shows that the mean of a set of N i.i.d. random variables has a probability distribution that becomes narrower, with width $\propto 1/\sqrt{N}$, as N increases. However, we have proved this property only for discrete random variables, that is, for real numbers taking on a *finite* set of possible values. While many random variables with continuous probability distributions also satisfy the law of large numbers, there are important distributions that do not. Some continuous distributions do not have a mean or variance.

Exercise 4.14:^{B2} Sketch the Cauchy distribution

$$P(x) = \frac{1}{Z}\frac{1}{x^2+1}, \quad x \in (-\infty, \infty). \tag{4.36}$$

What is its normalizing constant Z? Can you evaluate its mean or variance?

Consider the sum $z = x_1 + x_2$, where x_1 and x_2 are independent random variables from a Cauchy distributions. What is $P(z)$? What is the probability distribution of the mean of x_1 and x_2, $\bar{x} = (x_1 + x_2)/2$? What is the probability distribution of the mean of N samples from this Cauchy distribution?

Solutions to Chapter 4's exercises

Solution to exercise 4.2 (p. 71): An ASCII file can be reduced in size by 7/8. This reduction could be achieved by a block code that maps 8 byte blocks into 7 byte blocks by copying the 56 information–carrying bits into 7 bytes.

Solution to exercise 4.3 (p. 71): The pigeon–hole principle states: you can't put 16 pigeons into 15 holes without using one of the holes twice.

Similarly, you can't give $\mathcal{A}_X$ outcomes binary names of some length l shorter than $\log_2 |\mathcal{A}_X|$ bits, because there are only 2^l such binary names, and $l < \log_2 |\mathcal{A}_X|$ implies $2^l < |\mathcal{A}_X|$, so at least two different inputs to the compressor would compress to the same output file.

Solution to exercise 4.6 (p. 74): Between the cusps, all the changes in probability are equal, and the number of elements in T changes by one at each step. So H_δ varies logarithmically with $(-\delta)$.

Solution to exercise 4.7 (p. 82): Going by the rule of thumb that the most efficient strategy is the most informative strategy, in the sense of having all possible outcomes as near as possible to equiprobable, we want the first weighing to have outcomes 'the two sides balance' in eight cases and 'the two sides do not balance' in eight cases. This is achieved by initially weighing 1,2,3,4 against 5,6,7,8, leaving the other eight balls aside. Iterating this binary division of the possibilities, we arrive at a strategy requiring 4 weighings.

The above strategy for designing a sequence of binary experiments by constructing a binary tree from the top down is actually not always optimal; the optimal method of constructing a binary tree will be explained in the next chapter.

Solution to exercise 4.9 (p. 82): The following solution was found by Dyson and Lyness in 1946 and presented in the following elegant form by John Conway in 1999.[4] Be warned: the symbols A, B, and C are used to name the balls, to name the pans of the balance, to name the outcomes, and to name the possible states of the odd ball!

(a) Label the 12 balls by the sequences

```
AAB  ABA  ABB  ABC  BBC  BCA  BCB  BCC  CAA  CAB  CAC  CCA
```

and in the

```
1st                AAB ABA ABB ABC            BBC BCA BCB BCC
2nd weighings put AAB CAA CAB CAC in pan A, ABA ABB ABC BBC in pan B.
3rd                ABA BCA CAA CCA            AAB ABB BCB CAB
```

[4] Posting to `geometry-puzzles@forum.swarthmore.edu` Thu, 28 Jan 1999.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Now in a given weighing, a pan will either end up in the

- Canonical position (C) that it assumes when the pans are balanced, or
- Above that position (A), or
- Below it (B),

so the weighings determine a sequence of three of these letters.

If this sequence is CCC, then there's no odd ball. Otherwise, for *just one* of the two pans, the sequence is among the 12 above, and names the odd ball, whose weight is Above or Below the proper one according as the pan is A or B.

(b) In W weighings the odd ball can be identified from among

$$N = (3^W - 3)/2 \tag{4.37}$$

balls in the same way, by labelling them with all the non-constant sequences of W letters from A, B, C whose first change is A–to–B or B–to–C or C–to–A, and at the wth weighing putting those whose wth letter is A in pan A and those whose wth letter is B in pan B.

Solution to exercise 4.10 (p. 82):

(a) A sloppy answer to this question counts the number of possible states, $\binom{12}{2}2^2 = 264$, and takes its base 3 logarithm, which is 5.07, which exceeds 5. We might estimate that six weighings suffice to find the state of the two odd balls among 12. If there are three odd balls then there are $\binom{12}{3}2^3 = 1760$ states, whose logarithm is 6.80, so seven weighings might be estimated to suffice.

However, these answers neglect the possibility that we will learn something more from our experiments than which are the odd balls; there is a good chance that we will also learn something about the relative oddnesses of the two odd balls. If balls A and B are the odd balls, there is a good chance that the optimal weighing strategy will at some point put ball A on one side of the balance and ball B on the other, along with a load of regular balls; if, A and B are both heavy balls, say, the outcome of this weighing will reveal, at the end of the day, whether A was heavier than B, or lighter, or the same, which is not something we were asked to find out. From the point of view of the task, finding the relative oddnesses of the two balls is a waste of experimental capacity.

A more careful estimate takes this annoying possibility into account. Let us define the oddness of an odd ball to be the absolute value of the difference between its weight and the regular weight.

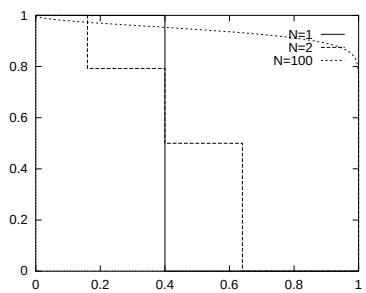
In the case of two odd balls, a complete description of the balls, including a ranking of their oddnesses, has three times as many states as we counted above (the two odd balls could be odd by the same amount, or by amounts that differ, *i.e.*, $264 \times 3 = 792$ outcomes, whose logarithm is 6.07. Thus to identify the *full* state of the system in 6 weighings is impossible — at least seven are needed — but I can believe that we might be able to solve the

required problem in 6 weighings with a strategy which sometimes avoids finding the ranking of the oddnesses.

In the case of three odd balls, there are 3! = 6 possible rankings of the oddnesses if the oddnesses are different (*e.g.*, $0 < A < B < C$), six if two of them are equal (*e.g.*, $0 < A < B = C$ and $0 < A = B < C$), and just one if they are equal ($0 < A = B = C$). So we have to multiply the sloppy answer by 13. We thus find that the number of *full* system states is 13×1760, whose logarithm is 9.13. So at least ten weighings are needed to guarantee identification of the full state. I can believe that nine weighings might suffice to solve the required problem, but it is not clear.

(b) If the weights of heavy, regular and light balls are known in advance, the original sloppy method becomes correct. At least six weighings are needed to guarantee identification of the two–odd–out–of–twelve, and at least seven to identify three out of twelve.

Solution to exercise 4.11 (p. 82): The curves $\frac{1}{N}H_\delta(X^N)$ as a function of δ for $N = 1, 2$ and 100 are shown in figure 4.12. Note that $H_2(0.4) = 0.971$ bits.

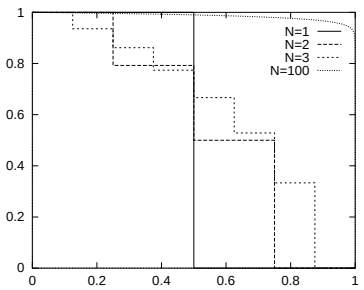


$N = 1$		
δ	$\frac{1}{N}H_\delta(\mathbf{X})$	$2^{H_\delta(\mathbf{X})}$
0–0.4	1	2
0.4–1	0	1

$N = 2$		
δ	$\frac{1}{N}H_\delta(\mathbf{X})$	$2^{H_\delta(\mathbf{X})}$
0–0.16	1	4
0.16–0.4	0.79248	3
0.4–0.64	0.5	2
0.64–1	0	1

Figure 4.12. $\frac{1}{N}H_\delta(\mathbf{X})$ (vertical axis) against δ (horizontal), for $N = 1, 2, 100$ binary variables with $p_1 = 0.4$.

Solution to exercise 4.12 (p. 82): The curves $\frac{1}{N}H_\delta(Y^N)$ as a function of δ for $N = 1, 2, 3$ and 100 are shown in figure 4.13 and table 4.13. Note that $H_2(0.5) = 1$ bit.



$N = 2$		
δ	$\frac{1}{N}H_\delta(\mathbf{Y})$	$2^{H_\delta(\mathbf{Y})}$
0–0.25	1	4
0.25–0.5	0.79248	3
0.5–0.75	0.5	2
0.75–1	0	1

$N = 3$		
δ	$\frac{1}{N}H_\delta(\mathbf{Y})$	$2^{H_\delta(\mathbf{Y})}$
0–0.125	1	8
0.125–0.25	0.93578	7
0.25–0.375	0.86165	6
0.375–0.5	0.77398	5
0.5–0.625	0.66667	4
0.625–0.75	0.52832	3
0.75–0.875	0.33333	2
0.875–1	0	1

Figure 4.13. $\frac{1}{N}H_\delta(\mathbf{Y})$ (vertical axis) against δ (horizontal), for $N = 1, 2, 3, 100$ binary variables with $p_1 = 0.5$.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solution to exercise 4.13 (p. 82): The Gibbs entropy is $k_{\rm B}\sum_i p_i \log \frac{1}{p_i}$, where i runs over all states of the system. This entropy is equivalent (apart from the factor of $k_{\rm B}$) to the Shannon entropy of the ensemble.

Whereas the Gibbs entropy can be defined for any ensemble, the Boltzmann entropy is only defined for *microcanonical* ensembles, which have a probability distribution that is uniform over a set of accessible states. The Boltzmann entropy is defined to be $S_{\rm B} = k_{\rm B} \log \Omega$ where Ω is the number of accessible states of the microcanonical ensemble. This is equivalent (apart from the factor of $k_{\rm B}$) to the perfect information content H_0 of that ensemble. The Gibbs entropy of a microcanonical ensemble is trivially equal to the Boltzmann entropy.

What if we consider a thermal distribution (the *canonical* ensemble), where the probability of a state $\mathbf{x}$ is

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{E(\mathbf{x})}{k_{\rm B}T}\right)? \tag{4.38}$$

What is the corresponding microcanonical ensemble? It is an ensemble with fixed total energy (to within some precision ϵ). Now, fixing the total energy to a precision ϵ is equivalent to fixing the value of $\log P(\mathbf{x})$ to within $\epsilon k_{\rm B}T$. Our definition of the typical set $T_{N\beta}$ was precisely that it consisted of all elements that have a value of $\log P(\mathbf{x})$ very close to the mean value of $\log P(\mathbf{x})$ under the canonical ensemble, $-NH(X)$. Thus the microcanonical ensemble is equivalent to a uniform distribution over the typical set of the canonical ensemble.

Our proof of the 'asymptotic equipartition' principle thus proves — for the case of a system whose energy is separable into a sum of independent terms — that the Boltzmann entropy of the microcanonical ensemble is very close (for large N) to the Gibbs entropy of the canonical ensemble, if the energy of the microcanonical ensemble is constrained to equal the mean energy of the canonical ensemble.

Solution to exercise 4.14 (p. 83): The normalizing constant of the Cauchy distribution

$$P(x) = \frac{1}{Z}\frac{1}{x^2+1}$$

is

$$Z = \int_{-\infty}^{\infty} dx\, \frac{1}{x^2+1} = \left[\tan^{-1}x\right]_{-\infty}^{\infty} = \frac{\pi}{2} - \frac{-\pi}{2} = \pi. \tag{4.39}$$

The mean and variance of this distribution are both undefined. (The distribution may be symmetrical about zero, but this does not imply that its mean is zero. The mean is the value of a divergent integral.) The sum $z = x_1 + x_2$, where x_1 and x_2 both have Cauchy distributions, has probability density given by the convolution

$$P(z) = \frac{1}{\pi^2}\int_{-\infty}^{\infty} dx_1\, \frac{1}{x_1^2+1}\frac{1}{(z-x_2)^2+1}, \tag{4.40}$$

which after a considerable labour using standard methods gives

$$P(z) = \frac{1}{\pi^2} 2\frac{\pi}{z^2+4} = \frac{2}{\pi}\frac{1}{z^2+2^2}, \tag{4.41}$$

which we recognize as a Cauchy distribution with width parameter 2 (where the original distribution has width parameter 1). This implies that the mean of the two points, $\bar{x} = (x_1 + x_2)/2 = z/2$, has a Cauchy distribution with width parameter 1. Generalizing, the mean of N samples from a Cauchy distribution is Cauchy distributed with the *same parameters* as the individual samples. The probability distribution of the mean does *not* become narrower as $1/\sqrt{N}$.

The central limit theorem does not apply to the Cauchy distribution, because it does not have a finite variance.

An alternative neat method for getting to equation (4.41) makes use of the Fourier transform of the Cauchy distribution, which is a biexponential $e^{-|\omega|}$. Convolution in real space corresponds to multiplication in Fourier space, so the Fourier transform of z is simply $e^{-|2\omega|}$. Reversing the transform, we obtain equation (4.41).

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

About Chapter 5

In the last chapter, we saw a proof of the fundamental status of the entropy as a measure of average information content. We defined a data compression scheme using *fixed length block codes*, and proved that as N increases, it is possible to encode N i.i.d. variables $\mathbf{x} = (x_1, \ldots, x_N)$ into a block of $N(H(X)+\epsilon)$ bits with vanishing probability of error, whereas if we attempt to encode X^N into $N(H(X)-\epsilon)$ bits, the probability of error is virtually 1.

We thus verified the *possibility* of data compression, but the block coding defined in the proof did not give a truly practical algorithm. In this chapter and the next, we study practical data compression algorithms. Whereas the last chapter's compression scheme used large blocks of fixed size and was lossy, in the next chapter we discuss variable-length compression schemes that are practical for small block sizes and that are not lossy.

Before reading chapter 5, you should have worked on exercise 2.21 (p. 39).

We will use the following notation for intervals: the statement $x \in [1, 2)$ means that $x \geq 1$ and $x < 2$; the statement $x \in (1, 2]$ means that $x > 1$ and $x \leq 2$.

5

Symbol Codes

5.1 Symbol codes

In this chapter, we discuss *variable–length symbol codes*, which encode one source symbol at a time, instead of encoding huge strings of N source symbols. These codes are *lossless:* unlike the last chapter's block codes, they are guaranteed to compress and decompress without any errors; but there is a chance that the codes may sometimes produce encoded strings longer than the original source string.

We again verify the fundamental status of the Shannon information content and the entropy, proving:

Source coding theorem (symbol codes). There exists a variable–length encoding C of an ensemble X such that the average length of an encoded symbol, $L(C, X)$, satisfies $L(C, X) \in [H(X), H(X) + 1)$.

The average length is equal to the entropy $H(X)$ only if the codelength for each outcome is equal to its Shannon information content.

We will also define a constructive procedure, the Huffman coding algorithm, that produces optimal symbol codes.

Notation for alphabets. $\mathcal{A}^N$ denotes the set of ordered N-tuples of elements from the set $\mathcal{A}$, *i.e.*, all strings of length N. The symbol $\mathcal{A}^+$ will denote the set of all strings of finite length composed of elements from the set $\mathcal{A}$.

Example 5.1: $\{0,1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

Example 5.2: $\{0,1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$.

A (binary) symbol code C for an ensemble X is a mapping from the range of x, $\mathcal{A}_X$=$\{a_1, \ldots, a_I\}$, to $\{0,1\}^+$. $c(x)$ will denote the 'codeword' corresponding to x, and $l(x)$ will denote its length, with $l_i = l(a_i)$.

The *extended code* C^+ is a mapping from $\mathcal{A}_X^+$ to $\{0,1\}^+$ obtained by concatenation, without punctutation, of the corresponding codewords:

$$c^+(x_1 x_2 \ldots x_N) = c(x_1)c(x_2)\ldots c(x_N). \tag{5.1}$$

[The term 'mapping' here is a synonym for 'function'.]

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Example 5.3: A symbol code for the ensemble X : $\mathcal{A}_X = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, $\mathcal{P}_X = \{1/2, 1/4, 1/8, 1/8\}$ is C_0:

	a_i	$c(a_i)$	l_i
C_0:	a	1000	4
	b	0100	4
	c	0010	4
	d	0001	4

Using the extended code, we may encode acdbac as

$$c^+(\mathtt{acdbac}) = \mathtt{100000100001010010000010} \tag{5.2}$$

There are basic requirements for a useful symbol code. First, an encoded string must have a unique decoding.

A code $C(X)$ is uniquely decodeable if, under the extended code C^+, every element of $\mathcal{A}_X^+$ maps into a different string, *i.e.*,

$$\forall \mathbf{x}, \mathbf{y} \in \mathcal{A}_X^+, \mathbf{x} \neq \mathbf{y} \Rightarrow c^+(\mathbf{x}) \neq c^+(\mathbf{y}). \tag{5.3}$$

The code C_0 defined above is an example of a uniquely decodeable code.

Second, the symbol code must be easy to decode. A symbol code is easiest to decode if it is possible to identify the end of a codeword as soon as it arrives, which means that no codeword can be a *prefix* of another codeword. We will show later that we don't lose any performance if we constrain our symbol code to be a prefix code.

[A word c is a *prefix* of another word d if there exists a tail string t such that the concatenation ct is identical to d. For example, 1 is a prefix of 101, and so is 10.]

A code is called a prefix code if no codeword is a prefix of any other codeword.

A prefix code is also known as an *instantaneous* code, because an encoded string can be decoded from left to right without looking ahead to subsequent codewords. The end of a codeword is immediately recognizable. A prefix code is self–punctuating, and is uniquely decodeable.

Prefix codes are also known as 'prefix–free codes' or 'prefix condition codes'.

Example 5.4: The code $C_1 = \{\mathtt{0}, \mathtt{101}\}$ is a prefix code because 0 is not a prefix of 101, nor is 101 a prefix of 0.

Example 5.5: Let $C_2 = \{\mathtt{1}, \mathtt{101}\}$. This code is not a prefix code because 1 is a prefix of 101.

Solutions on p. 105

Exercise 5.6:[A1] Is C_2 uniquely decodeable?

Example 5.7: Consider exercise 3.4 (p. 65). Any weighing strategy that identifies the odd ball and whether it is heavy or light can be viewed as assigning a ternary code to each of the 24 possible states. This code is a prefix code.

Third, the code should achieve as much compression as possible.

The expected length $L(C, X)$ of a symbol code C for ensemble X is

$$L(C, X) = \sum_{x \in \mathcal{A}_X} P(x) l(x). \tag{5.4}$$

We may also write this quantity as

$$L(C, X) = \sum_{i=1}^{I} p_i l_i \tag{5.5}$$

where $I = |\mathcal{A}_X|$.

Example 5.8: Let

$$\begin{array}{r} \mathcal{A}_X = \{ \texttt{a}, \quad \texttt{b}, \quad \texttt{c}, \quad \texttt{d} \}, \\ \text{and } \mathcal{P}_X = \{1/2, 1/4, 1/8, 1/8\}, \end{array} \tag{5.6}$$

and consider the code C_3:

	a_i	$c(a_i)$	p_i	$\log_2 \frac{1}{p_i}$	l_i
C_3:	a	0	0.5	1.0	1
	b	10	0.25	2.0	2
	c	110	0.125	3.0	3
	d	111	0.125	3.0	3

The entropy of X is 1.75 bits, and the expected length $L(C_3, X)$ of this code is also 1.75 bits. The sequence of symbols $\mathbf{x}$=(acdbac) is encoded as $c^+(\mathbf{x})$ = 0110111100110. You can confirm that no other sequence of symbols $\mathbf{x}$ has the same encoding. In fact, C_3 is a prefix code and is therefore uniquely decodeable. Notice that the codeword lengths satisfy $l_i = \log_2(1/p_i)$, or equivalently, $p_i = 2^{-l_i}$.

Example 5.9: Consider the fixed length code for the same ensemble X, C_4:

	a_i	$c(a_i)$	p_i	$\log_2 \frac{1}{p_i}$	l_i
C_4:	a	00	0.5	1.0	2
	b	01	0.25	2.0	2
	c	10	0.125	3.0	2
	d	11	0.125	3.0	2

The expected length $L(C_4, X)$ is 2 bits.

Example 5.10: Consider C_5:

	a_i	$c(a_i)$	p_i	$\log_2 \frac{1}{p_i}$	l_i
C_5:	a	0	0.5	1.0	1
	b	1	0.25	2.0	1
	c	00	0.125	3.0	2
	d	11	0.125	3.0	2

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

The expected length $L(C_5, X)$ is 1.25 bits, which is less than $H(X)$. But the code is not uniquely decodeable. The sequence $\mathbf{x}$=(acdbac) encodes as 000111000, which can also be decoded as (cabdca).

Example 5.11: Consider the code C_6:

	a_i	$c(a_i)$	p_i	$\log_2 \frac{1}{p_i}$	l_i
C_6:	a	0	0.5	1.0	1
	b	01	0.25	2.0	2
	c	011	0.125	3.0	3
	d	111	0.125	3.0	3

The expected length $L(C_6, X)$ of this code is 1.75 bits. The sequence of symbols $\mathbf{x}$=(acdbac) is encoded as $c^+(\mathbf{x}) =$ 0011111010011.

Is C_6 a prefix code? It is not, because c(a) = 0 is a prefix of c(b) and c(c).

Is C_6 uniquely decodeable? This is not so obvious. If you think that it might *not* be uniquely decodeable, try to prove it so by finding a pair of strings $\mathbf{x}$ and $\mathbf{y}$ that have the same encoding. [The definition of unique decodeability is given in equation (5.3).]

C_6 certainly isn't *easy* to decode. When we receive '00', it is possible that $\mathbf{x}$ could start 'aa', 'ab' or 'ac'. Once we have received '001111', the second symbol is still ambiguous, as $\mathbf{x}$ could be 'abd...' or 'acd...'. But eventually a unique decoding crystallizes, once the next 0 appears in the encoded stream.

C_6 *is* in fact uniquely decodeable. Comparing with the prefix code C_3, we see that the codewords of C_6 are the reverse of C_3's. That C_3 is uniquely decodeable proves that C_6 is too, since any string from C_6 is identical to a string from C_3 read backwards.

5.2 What limit is imposed by unique decodeability?

We now ask, given a list of positive integers $\{l_i\}$, does there exist a uniquely decodeable code with those integers as its codeword lengths? At this stage, we ignore the probabilities of the different symbols; once we understand unique decodeability better, we'll reintroduce the probabilities and discuss how to make an *optimal* uniquely decodeable symbol code.

In the examples above, we have observed that if we take a code such as $\{00, 01, 10, 11\}$, and shorten one of its codewords, for example 00 $\rightarrow$ 0, then we can retain unique decodeability only if we lengthen other codewords. Thus there seems to be a constrained budget that we can spend on codewords, with shorter codewords being more expensive.

Let us explore the nature of this budget. If we build a code purely from codewords of length l equal to three, how many codewords can we have and retain unique decodeability? The answer is $2^l = 8$. Once we have chosen all eight of these codewords, is there any way we could add to the code another codeword of some *other* length and retain unique decodeability? It would seem not.

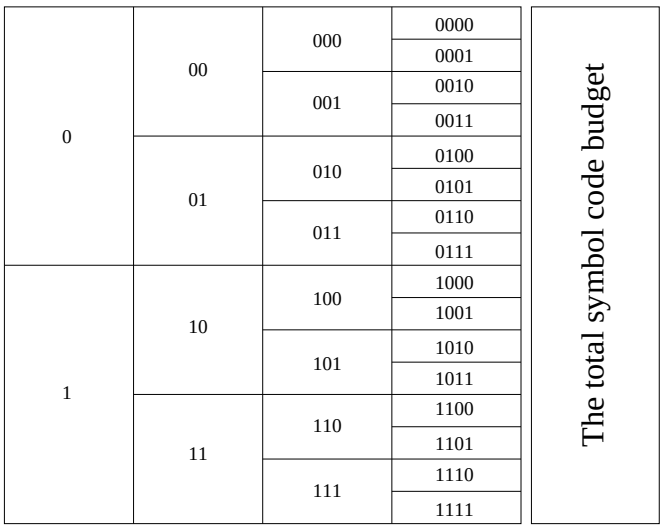


Figure 5.1. The symbol coding budget. The 'cost' 2^{-l} of each codeword (with length l) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.

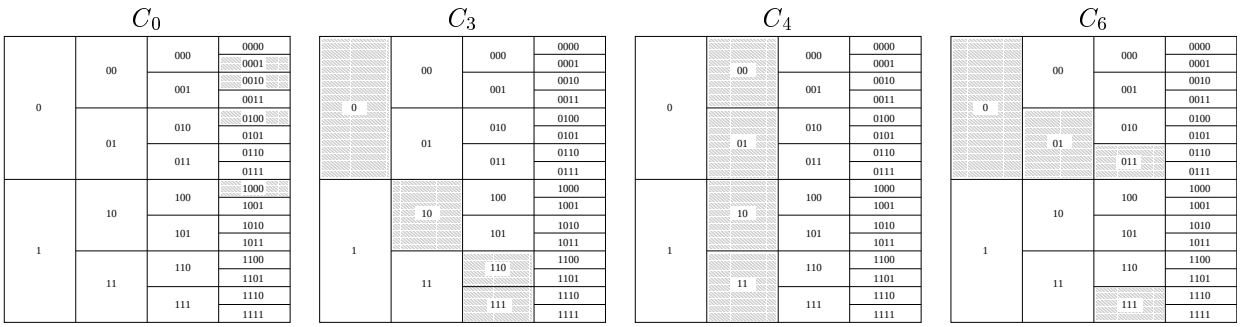


Figure 5.2. Selections of codewords made by codes C_0, C_3, C_4 and C_6 from section 5.1.

What if we make a code that includes a length–one codeword, '0', with the other codewords being of length three? How many length–three codewords can we have? If we restrict attention to prefix codes, then we can have only four codewords of length three, namely $\{100, 101, 110, 111\}$. What about other codes? Is there any other way of choosing codewords of length 3 that can give more codewords? Intuitively, we think this unlikely. A codeword of length 3 appears to have a cost 2^2 times smaller than a codeword of length 1.

Let's define a total budget of size 1, which we can spend on codewords. If we set the cost of a codeword whose length is l to 2^{-l}, then we have a pricing system that fits the examples discussed above. Codewords of length 3 cost 1/8 each; codewords of length 1 cost 1/2 each. We can spend our budget on any codewords. If we go over our budget then the code will certainly not be uniquely decodeable. If, on the other hand,

$$\sum_i 2^{-l_i} \leq 1, \tag{5.7}$$

then the code may be uniquely decodeable. This inequality is the Kraft inequality.

Kraft inequality. For any uniquely decodeable code C over the binary al-

phabet $\{0, 1\}$, the codeword lengths must satisfy:

$$\sum_{i=1}^{I} 2^{-l_i} \leq 1, \tag{5.8}$$

where $I = |\mathcal{A}_X|$.

Completeness. If a uniquely decodeable code satisfies the Kraft inequality with equality then it is called a *complete* code.

We want codes that are uniquely decodeable; and prefix codes are uniquely decodeable. Codes that are not prefix codes are less straightforward to decode than prefix codes. So life would be simpler for us if we could restrict attention to prefix codes. Fortunately, for any source there *is* an optimal symbol code that is also a prefix code.

Kraft inequality and prefix codes. Given a set of codeword lengths that satisfy the Kraft inequality, there exists a uniquely decodeable prefix code with these codeword lengths.

The Kraft inequality might be more accurately referred to as the Kraft-McMillan inequality: Kraft (1949) proved that if the inequality is satisfied, then a prefix code exists with the given lengths. McMillan (1956) proved the converse, that unique decodeability implies that the inequality holds.

Proof of the Kraft inequality: Define $S = \sum_i 2^{-l_i}$. Consider the quantity

$$S^N = \left[\sum_i 2^{-l_i}\right]^N = \sum_{i_1=1}^{I} \sum_{i_2=1}^{I} \cdots \sum_{i_N=1}^{I} 2^{-(l_{i_1} + l_{i_2} + \ldots l_{i_N})} \tag{5.9}$$

The quantity in the exponent, $(l_{i_1} + l_{i_2} + \ldots l_{i_N})$, is the length of the encoding of the string $\mathbf{x} = a_{i_1} a_{i_2} \ldots a_{i_N}$. For every string $\mathbf{x}$ of length N, there is one term in the above sum. Introduce an array A_l that counts how many strings $\mathbf{x}$ have encoded length l. Then, defining $l_{\min} = \min_i l_i$ and $l_{\max} = \max_i l_i$:

$$S^N = \sum_{l=Nl_{\min}}^{Nl_{\max}} 2^{-l} A_l. \tag{5.10}$$

Now assume C is uniquely decodeable, so that for all $\mathbf{x} \neq \mathbf{y}$, $c^+(\mathbf{x}) \neq c^+(\mathbf{y})$. Concentrate on the $\mathbf{x}$ that have encoded length l. There are a total of 2^l distinct bit strings of length l, so it must be the case that $A_l \leq 2^l$. So

$$S^N = \sum_{l=Nl_{\min}}^{Nl_{\max}} 2^{-l} A_l \leq \sum_{l=Nl_{\min}}^{Nl_{\max}} 1 \;\leq\; Nl_{\max}. \tag{5.11}$$

Thus $S^N \leq l_{\max} N$ for all N. Now if S were greater than 1, then as N increases, S^N would be an exponentially growing function, and for large enough N, an exponential always exceeds a polynomial such as $l_{\max} N$. But our result ($S^N \leq l_{\max} N$) is true for *any* N. Therefore $S \leq 1$. □

Solutions on p. 105

Exercise 5.12:^{B3} Prove the result stated above, that for any set of codeword lengths $\{l_i\}$ satisfying the Kraft inequality, there is a prefix code having those lengths.

A pictorial view of the Kraft inequality may help you solve this exercise. Imagine that we are choosing the codewords to make a symbol code. We can draw the set of all candidate codewords in a figure that shows the 'cost' of the codeword by the area of a box (figure 5.1). The total budget available — the '1' on the right hand side of the Kraft inequality — is shown at one side. Some of the codes discussed in section 5.1 are illustrated in figure 5.2. Notice that the codes that are prefix codes, C_0, C_3, and C_4, have the property that to the right of any selected codeword, there are no other selected codewords.

We are now ready to put back the symbols's probabilities $\{p_i\}$. Given a set of symbol probabilities (the English language probabilities of figure 2.1, for example), how do we make the best symbol code — one with the smallest possible expected length $L(C, X)$? And what is that smallest possible expected length? It's not obvious how to assign the codeword lengths. If we give short codewords to the more probable symbols then the expected length might be reduced; on the other hand, shortening some codewords necessarily causes others to lengthen, by the Kraft inquality.

5.3 What's the most compression that we can hope for?

As you might have guessed, the entropy appears as the lower bound on the expected length of a code.

Lower bound on expected length. The expected length $L(C, X)$ of a uniquely decodeable code is bounded below by $H(X)$.

Proof: We define the *implicit probabilities* $q_i \equiv 2^{-l_i}/z$, where $z = \sum_{i'} 2^{-l_{i'}}$, so that $l_i = \log 1/q_i - \log z$. We then use Gibbs' inequality, $\sum_i p_i \log 1/q_i \geq \sum_i p_i \log 1/p_i$, with equality if $q_i = p_i$, and the Kraft inequality $z \leq 1$:

$$\begin{aligned} L(C, X) &= \sum_i p_i l_i = \sum_i p_i \log 1/q_i - \log z \\ &\geq \sum_i p_i \log 1/p_i - \log z \\ &\geq H(X). \end{aligned}$$

The equality $L(C, X) = H(X)$ is achieved only if the Kraft equality $z = 1$ is satisfied, and if the code lengths satisfy $l_i = \log(1/p_i)$. □

This is an important result so let's say it again:

Optimal source code lengths. The expected length is minimized and is equal to $H(X)$ only if the code lengths are equal to the *Shannon information contents*:

$$l_i = \log_2(1/p_i). \tag{5.12}$$

Implicit probabilities defined by code lengths. Conversely, any choice of codelengths $\{l_i\}$ *implicitly* defines a probability distribution $\{q_i\}$,

$$q_i \equiv 2^{-l_i}/z, \tag{5.13}$$

for which those codelengths would be the optimal codelengths. If the code is complete then $z = 1$ and the implicit probabilities are given by $q_i = 2^{-l_i}$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

5.4 How much can we compress?

So, we can't compress below the entropy. How close can we expect to get to the entropy?

Theorem 5.1 *For an ensemble X there exists a prefix code C with expected length satisfying*

$$H(X) \leq L(C,X) < H(X) + 1. \tag{5.14}$$

Proof: We set the code lengths to integers slightly larger than the optimum lengths:

$$l_i = \lceil \log_2(1/p_i) \rceil \tag{5.15}$$

where $\lceil l^* \rceil$ denotes the smallest integer greater than or equal to l^*. [We are not asserting that the *optimal* code necessarily uses these lengths, we are simply choosing these lengths because we can use them to prove the theorem.]

We check that there *is* a prefix code with these lengths by confirming that the Kraft inequality is satisfied.

$$\sum_i 2^{-l_i} = \sum_i 2^{\lceil \log_2(1/p_i) \rceil} \leq \sum_i 2^{\log_2(1/p_i)} = \sum_i p_i = 1. \tag{5.16}$$

Then we confirm

$$L(C,X) = \sum_i p_i \lceil \log(1/p_i) \rceil < \sum_i p_i (\log(1/p_i) + 1) = H(X) + 1. \tag{5.17}$$

□

The cost of using the wrong codelengths

If we use a code whose lengths are not equal to the optimal codelengths, the average message length will be larger than the entropy.

If the true probabilities are $\{p_i\}$ and we use a complete code with lengths l_i, we can view those lengths as defining an implicit probabilistic model $q_i = 2^{-l_i}$. The average length is

$$L(C,X) = H(X) + \sum_i p_i \log p_i / q_i, \tag{5.18}$$

i.e., it exceeds the entropy by the Kullback–Leibler divergence $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q})$ (as defined in chapter 1, p. 37).

5.5 Optimal source coding with symbol codes: Huffman coding

Given a set of probabilities $\mathcal{P}$, how can we design an optimal prefix code? For example, what is the best symbol code for the English language ensemble shown in the margin?

x	$P(x)$
a	0.0575
b	0.0128
c	0.0263
d	0.0285
e	0.0913
f	0.0173
g	0.0133
h	0.0313
i	0.0599
j	0.0006
k	0.0084
l	0.0335
m	0.0235
n	0.0596
o	0.0689
p	0.0192
q	0.0008
r	0.0508
s	0.0567
t	0.0706
u	0.0334
v	0.0069
w	0.0119
x	0.0073
y	0.0164
z	0.0007
-	0.1928

Figure 5.3. An ensemble in need of a symbol code.

How not to do it

One might try to roughly split the set $\mathcal{A}_X$ in two, and continue bisecting the subsets so as to define a binary tree from the top. This construction has the right spirit, but it is not necessarily optimal; it achieves $L(C,X) \leq H(X) + 2$.

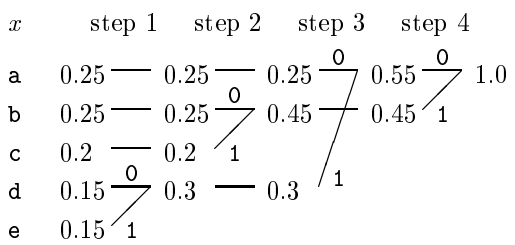
The Huffman coding algorithm

The trick is to construct the code backwards starting from the tails of the codewords.

1. Take the two least probable symbols in the alphabet. These two symbols will be given the longest codewords, which will have equal length, and differ only in the last digit.
2. Combine these two symbols into a single symbol, and repeat.

Since each step reduces the size of the alphabet by one, this algorithm will have assigned strings to all the symbols after $|\mathcal{A}_X| - 1$ steps.

Example 5.13: Let $\mathcal{A}_X = \{$ a, b, c, d, e $\}$
and $\mathcal{P}_X = \{ 0.25, 0.25, 0.2, 0.15, 0.15 \}$.

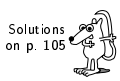


The codewords are then obtained by concatenating the symbols in reverse order: $C = \{00, 10, 11, 010, 011\}$.

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.25	2.0	2	00
b	0.25	2.0	2	10
c	0.2	2.3	2	11
d	0.15	2.7	3	010
e	0.15	2.7	3	011

The codelengths selected by the Huffman algorithm (column 4) are in some cases longer and in some cases shorter than the ideal codelengths, the Shannon information contents $\log_2 \frac{1}{p_i}$ (column 3). The expected length of the code is $L = 2.30$ bits, whereas the entropy is $H = 2.2855$ bits.

If at any point there is more than one way of selecting the two least probable symbols then the choice may be made in any manner — the expected length of the code will be the same, no matter what choice is made.



Exercise 5.14:[B3] Prove that there is no better symbol code for a source than the Huffman code.

Example 5.15: We can make a Huffman code for the probability distribution over the alphabet introduced in figure 2.1. The result is shown in figure 5.4. This code has an expected length of 4.15 bits; the entropy of the ensemble is 4.11 bits. Observe the disparities between the assigned codelengths and the ideal codelengths $\log_2 \frac{1}{p_i}$.

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.0575	4.1	4	0000
b	0.0128	6.3	6	001000
c	0.0263	5.2	5	00101
d	0.0285	5.1	5	10000
e	0.0913	3.5	4	1100
f	0.0173	5.9	6	111000
g	0.0133	6.2	6	001001
h	0.0313	5.0	5	10001
i	0.0599	4.1	4	1001
j	0.0006	10.7	10	1101000000
k	0.0084	6.9	7	1010000
l	0.0335	4.9	5	11101
m	0.0235	5.4	6	110101
n	0.0596	4.1	4	0001
o	0.0689	3.9	4	1011
p	0.0192	5.7	6	111001
q	0.0008	10.3	9	110100001
r	0.0508	4.3	5	11011
s	0.0567	4.1	4	0011
t	0.0706	3.8	4	1111
u	0.0334	4.9	5	10101
v	0.0069	7.2	8	11010001
w	0.0119	6.4	7	1101001
x	0.0073	7.1	7	1010001
y	0.0164	5.9	6	101001
z	0.0007	10.4	10	1101000001
-	0.1928	2.4	2	01

Figure 5.4. Huffman code for the English language ensemble.

Constructing a binary tree top-down is suboptimal

In previous chapters we studied weighing problems in which we built ternary or binary trees. We noticed that balanced trees — ones in which at every step, the two possible outcomes were as close as possible to equiprobable — appeared to describe the most efficient experiments. This gave an intuitive motivation for entropy as a measure of information content.

It is not the case, however, that optimal codes can *always* be constructed by a greedy top–down method in which the alphabet is successively divided into subsets that are as near as possible to equiprobable.

Example 5.16: Find the optimal binary symbol code for the ensemble:

$$\begin{array}{l} \mathcal{A}_X = \{ \ \mathtt{a} \ , \ \mathtt{b} \ , \ \mathtt{c} \ , \ \mathtt{d} \ , \ \mathtt{e} \ , \ \mathtt{f} \ , \ \mathtt{g} \ \} \\ \mathcal{P}_X = \{ \, 0.01, 0.24, 0.05, 0.20, 0.47, 0.01, \, 0.02 \, \} \end{array} . \tag{5.19}$$

Notice that a greedy top–down method can split this set into two subsets $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}$ and $\{\mathtt{e},\mathtt{f},\mathtt{g}\}$ which both have probability $1/2$, and that $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}$ can be divided into subsets $\{\mathtt{a},\mathtt{b}\}$ and $\{\mathtt{c},\mathtt{d}\}$, which have probability $1/4$; so a greedy top–down method gives the code:

a_i	p_i	$c(a_i)$
a	0.01	000
b	0.24	001
c	0.05	010
d	0.20	011
e	0.47	10
f	0.01	110
g	0.02	111

which has expected length 2.53. The Huffman coding algorithm yields this code:

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.01	6.6	6	000000
b	0.24	2.1	2	01
c	0.05	4.3	4	0001
d	0.20	2.3	3	001
e	0.47	1.1	1	1
f	0.01	6.6	6	000001
g	0.02	5.6	5	00001

which has expected length 1.97.

5.6 Disadvantages of the Huffman code

The Huffman algorithm produces an optimal symbol code for an ensemble, but this is not the end of the story. Both the word 'ensemble' and the phrase 'symbol code' need careful attention.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Changing ensemble

If we wish to communicate a sequence of outcomes from one unchanging ensemble, then a Huffman code may be convenient. But often the appropriate ensemble changes. If, for example, we are compressing text, then the symbol frequencies will vary with context: the letter `u` is much more probable after a `q` than after an `e`. And furthermore, our knowledge of these context-dependent symbol frequencies will also change as we learn the statistical properties of the text source.

Huffman codes do not handle changing ensemble probabilities with any elegance. One brute-force approach would be to recompute the Huffman code every time the probability over symbols changes. Another attitude is to deny the option of adaptation, and instead to run through the entire file in advance and compute a good probability distribution, which will then remain fixed throughout transmission. The code itself must also be communicated in this scenario. Such a technique is not only cumbersome and restrictive, it is also suboptimal, since the initial message specifying the code and the document itself are partially redundant. This technique therefore wastes bits.

The extra bit

An equally serious problem with Huffman codes is the innocuous-looking 'extra bit' relative to the ideal average length of $H(X)$ — a Huffman code achieves a length that satisfies $H(X) \leq L(C,X) < H(X)+1$, as proved in theorem 5.1. A Huffman code thus incurs an overhead of between 0 and 1 bits per symbol. If $H(X)$ were large, then this overhead would be an unimportant fractional increase. But for many applications, the entropy may be as low as one bit per symbol, or even smaller, so the overhead $L(C,X) - H(X)$ may dominate the encoded file length. Consider English text: in some contexts, long strings of characters may be highly predictable. For example, in the context '`strings_of_ch`', one might predict the next nine symbols to be '`aracters_`' with a probability of 0.99 each. A traditional Huffman code would be obliged to use one bit per character, making a total cost of nine bits where virtually no information is being conveyed. The entropy of English, given a good model, is about one bit per character (Shannon, 1993b), so a Huffman code is likely to be highly inefficient.

A traditional patch-up of Huffman codes uses them to compress *blocks* of symbols, for example the 'extended sources' X^N we discussed in chapter 2. The overhead per block is at most 1 bit so the overhead per symbol is at most $1/N$ bits. For sufficiently large blocks, the problem of the extra bit may be removed — but only at the expenses of (a) losing the elegant instantaneous decodeability of simple Huffman coding; and (b) having to compute the probabilities of all relevant strings and build the associated Huffman tree. One will end up explicitly computing the probabilities and codes for a huge number of strings, most of which will never actually occur.

A further problem is that it may not be appropriate to model successive symbols as coming independently from a single ensemble X. As we already asserted, any decent model for text will assign a probability over symbols that depends on the context. A changing probability distribution over symbols is not incompatible with the construction of Huffman codes for blocks of symbols. One could consider each possible sequence, computing the relevant probability

distributions along the way to evaluate the probability of the entire sequence, then build a Huffman tree for the sequences. One could account for dependencies between blocks as well, if one were willing to use a different Huffman code each time. But this modified encoder would be computationally expensive, since for large block sizes an exponentially large number of possible sequences would have to be considered along with their adaptive probabilities.

Beyond symbol codes

Huffman codes, therefore, although widely trumpeted as 'optimal', have many defects for practical purposes. They *are* optimal *symbol* codes, but for practical purposes *we don't want a symbol code.*

The defects of Huffman codes are rectified by *arithmetic coding*, which dispenses with the restriction that each symbol must translate into an integer number of bits. Arithmetic coding is the main topic of the next chapter.

5.7 Summary

Kraft inequality. If a code is *uniquely decodeable* its lengths must satisfy

$$\sum_i 2^{-l_i} \leq 1. \tag{5.20}$$

For any lengths satisfying the Kraft inequality, there exists a prefix code with those lengths.

Optimal source code lengths for an ensemble are equal to the Shannon information contents

$$l_i = \log_2 \frac{1}{p_i}, \tag{5.21}$$

and conversely, any choice of codelengths defines *implicit probabilities*

$$q_i = \frac{2^{-l_i}}{z}. \tag{5.22}$$

The relative entropy $D_{KL}(\mathbf{p}||\mathbf{q})$ measures how many bits per symbol are wasted by using a code whose implicit probabilities are $\mathbf{q}$, when the ensemble's true probability distribution is $\mathbf{p}$.

Source coding theorem for symbol codes. For an ensemble X, there exists a prefix code whose expected length satisfies

$$H(X) \leq L(C, X) < H(X) + 1. \tag{5.23}$$

The Huffman coding algorithm generates an optimal symbol code iteratively. At each iteration, the two least probable symbols are combined.

5.8 Exercises

Solutions on p. 106

Exercise 5.17:^{B2} Is the code $\{00, 11, 0101, 111, 1010, 100100, 0110\}$ uniquely decodeable?

Exercise 5.18:^{B2} Is the ternary code $\{00, 012, 0110, 0112, 100, 201, 212, 22\}$ uniquely decodeable?

Exercise 5.19:^{A3} Make Huffman codes for X^2, X^3 and X^4 where $\mathcal{A}_X = \{0,1\}$ and $\mathcal{P}_X = \{0.9, 0.1\}$. Compute their expected lengths and compare them with the entropies $H(X^2)$, $H(X^3)$ and $H(X^4)$.

Repeat this exercise for X^2 and X^4 where $\mathcal{P}_X = \{0.6, 0.4\}$.

Exercise 5.20:^{A2} Find a probability distribution $\{p_1, p_2, p_3, p_4\}$ such that there are *two* optimal codes that assign different lengths $\{l_i\}$ to the four symbols.

Exercise 5.21:^{C3} (Continuation of exercise 5.20.) Assume that the four probabilities $\{p_1, p_2, p_3, p_4\}$ are ordered such that $p_1 \geq p_2 \geq p_3 \geq p_4 \geq 0$. Call the set of all probability vectors $\mathbf{p}$ such that there are *two* optimal codes with different lengths the set '$\mathcal{Q}$'. Give a complete description of $\mathcal{Q}$. Find three probability vectors $\mathbf{q}^{(1)}$, $\mathbf{q}^{(2)}$, $\mathbf{q}^{(3)}$, which are the convex hull of $\mathcal{Q}$, *i.e.*, such that any $\mathbf{p} \in \mathcal{Q}$ can be written as

$$\mathbf{p} = \mu_1 \mathbf{q}^{(1)} + \mu_2 \mathbf{q}^{(2)} + \mu_3 \mathbf{q}^{(3)}, \tag{5.24}$$

where $\{\mu_i\}$ are positive.

Exercise 5.22:^{B2} Make ensembles for which the difference between the entropy and the expected length of the Huffman code is as big as possible.

Exercise 5.23:^{B2} Consider a sparse binary source with $\mathcal{P}_X = \{0.99, 0.01\}$. Discuss how Huffman codes could be used to compress this source *efficiently*.

Exercise 5.24:^{A2} *Scientific American* carried the following puzzle in 1975.

> **The poisoned glass.** 'Mathematicians are curious birds', the police commissioner said to his wife. 'You see, we had all those partly filled glasses lined up in rows on a table in the hotel kitchen. Only one contained poison, and we wanted to know which one before searching that glass for fingerprints. Our lab could test the liquid in each glass, but the tests take time and money, so we wanted to make as few of them as possible by simultaneously testing mixtures of small samples from groups of glasses. The university sent over a mathematics professor to help us. He counted the glasses, smiled and said:
>
> ' "Pick any glass you want, Commissioner. We'll test it first."
>
> ' "But won't that waste a test?" I asked.
>
> ' "No," he said, "it's part of the best procedure. We can test one glass first. It doesn't matter which one." '
>
> 'How many glasses were there to start with?' the commissioner's wife asked.
>
> 'I don't remember. Somewhere between 100 and 200.'
>
> What was the exact number of glasses?

Solve this puzzle and then explain why the professor was in fact wrong and the commissioner was right. What is in fact the optimal procedure for identifying the one poisoned glass? What is the expected waste relative to this optimum if one followed the professor's strategy? Explain the relationship to symbol coding.

Exercise 5.25:^{A2} Assume that a sequence of symbols from the ensemble X introduced at the beginning of this chapter is compressed used the code

C_3:

a_i	$c(a_i)$	p_i	$\log_2 \frac{1}{p_i}$	l_i
a	0	0.5	1.0	1
b	10	0.25	2.0	2
c	110	0.125	3.0	3
d	111	0.125	3.0	3

Imagine picking one bit at random from the binary encoded sequence $\mathbf{c} = c(x_1)c(x_2)c(x_3)\ldots$. What is the probability that this bit is a 1?

Solutions on p. 109

Exercise 5.26:^{B2} How should the binary Huffman encoding scheme be modified to make optimal symbol codes in an encoding alphabet with q symbols? (Also known as 'radix q'.)

Mixture codes

It is a tempting idea to construct a 'metacode' from several symbol codes that assign different–length codewords to the alternative symbols. For example, we might switch from one code to another, choosing whichever assigns the shortest codeword to the current symbol. Clearly we cannot do this for free. If one wishes to choose between two codes, then it is necessary to lengthen the message in a way that indicates which of the two codes is being used. If we indicate this choice by a single leading bit, it will be found that the resulting code is suboptimal because it is incomplete (that is, it fails the Kraft equality).

Exercise 5.27:^{A3} Prove that this metacode is incomplete, and explain why this combined code is suboptimal.

Solutions to Chapter 5's exercises

Solution to exercise 5.6 (p. 91): Yes, $C_2 = \{1, 101\}$ is uniquely decodeable, even though it is not a prefix code, because no two different strings can map onto the same string; only the codeword $c(2) = 101$ contains the symbol 0.

Solution to exercise 5.12 (p. 95): We wish to prove that for any set of codeword lengths $\{l_i\}$ satisfying the Kraft inequality, there is a prefix code having those lengths. This is readily proved by thinking of the codewords illustrated in

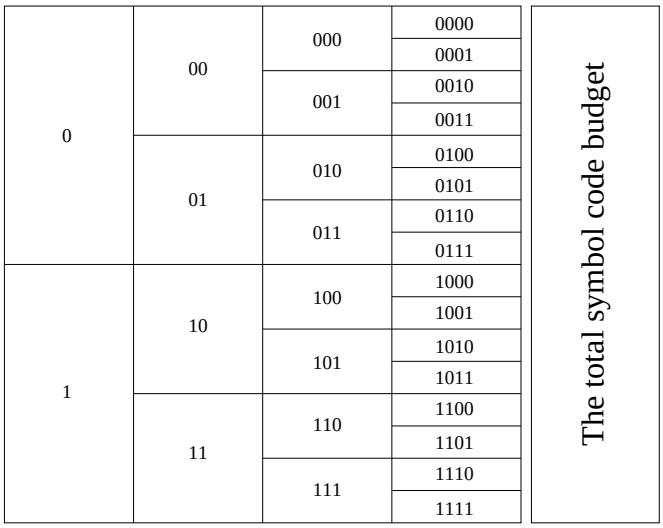


Figure 5.5. The symbol coding budget. The 'cost' 2^{-l} of each codeword (with length l) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.

figure 5.5 as being in a 'codeword supermarket', with size indicating cost. We imagine purchasing codewords one at a time, starting from the shortest codewords (*i.e.*, the biggest purchases), using the budget shown at the right of figure 5.5. We start at one side of the codeword supermarket, say the top, and purchase the first codeword of the required length. We advance down the supermarket a distance 2^{-l}, and purchase the next codeword of the next required length, and so forth. Because the codeword lengths are getting longer, and the corresponding intervals are getting shorter, we can always buy an adjacent codeword to the latest purchase, so there is no wasting of the budget. Thus at the Ith codeword we have advanced a distance $\sum 2^{-l_i}$ down the supermarket; if $\sum 2^{-l_i} \leq 1$, we will have purchased all the codewords without running out of budget.

Solution to exercise 5.14 (p. 98): The proof that Huffman coding is optimal depends on proving that the key step in the algorithm — the decision to give the two symbols with smallest probability equal encoded lengths — cannot

lead to a larger expected length than any other code. We can prove this by contradiction.

Assume that the two symbols with smallest probability, called a and b, to which the Huffman algorithm would assign equal length codewords, do *not* have equal lengths in *any* optimal symbol code. And assume that the optimal symbol code is some other rival code in which these two codewords have unequal lengths l_a and l_b with $l_a < l_b$. Without loss of generality we can assume that this other code is a complete prefix code, because any codelengths of a uniquely decodeable code can be realized by a prefix code.

In this rival code, there must be some other symbol c whose probability p_c is greater than p_a and whose length in the rival code is greater than or equal to l_b, because the code for b must have an adjacent codeword of equal or greater length — a complete prefix code never has a solo codeword of the maximum length. Consider exchanging the codewords for a and c, so that

symbol	probability	Huffman codewords	Rival code's codewords	Modified rival code
a	p_a	$c_{\mathrm{H}}(a)$	$c_{\mathrm{R}}(a)$	$c_{\mathrm{R}}(c)$
b	p_b	$c_{\mathrm{H}}(b)$	$c_{\mathrm{R}}(b)$	$c_{\mathrm{R}}(b)$
c	p_c	$c_{\mathrm{H}}(c)$	$c_{\mathrm{R}}(c)$	$c_{\mathrm{R}}(a)$

Figure 5.6. Proof that Huffman coding makes an optimal symbol code. We assume that the rival code, which is said to be optimal, assigns *unequal* length codewords to the two symbols with smallest probability, a and b. By interchanging codewords a and c of the rival code, where c is a symbol with rival codelength as long as b's, we can make a code better than the rival code. This shows that the rival code was not optimal.

a is encoded with the longer codeword that was c's, and c, which is more probable than a, gets the shorter codeword. Clearly this reduces the expected length of the code. The change in expected length is $(p_a - p_c)(l_c - l_a)$. Thus we have contradicted the assumption that the rival code is optimal. So it is valid to give the two symbols with smallest probability equal encoded lengths. Therefore Huffman coding produces optimal symbol codes.

Solution to exercise 5.17 (p. 102): The code $\{00, 11, 0101, 111, 1010, 100100, 0110\}$ is not uniquely decodeable because 11111 can be realized from $c(2)c(4)$ and $c(4)c(2)$.

Solution to exercise 5.18 (p. 102): The ternary code $\{00, 012, 0110, 0112, 100, 201, 212, 22\}$ *is* uniquely decodeable because it is a prefix code.

Solution to exercise 5.19 (p. 103): A Huffman code for X^2 where $\mathcal{A}_X = \{0, 1\}$ and $\mathcal{P}_X = \{0.9, 0.1\}$ is $\{00, 01, 10, 11\} \rightarrow \{1, 01, 000, 001\}$. This code has $L(C, X^2) = 1.29$, whereas the entropy $H(X^2)$ is 0.938.

A Huffman code for X^3 is

$$\{000, 100, 010, 001, 101, 011, 110, 111\} \rightarrow \{1, 011, 010, 001, 00000, 00001, 00010, 00011\}.$$

This has expected length $L(C, X^3) = 1.598$ whereas the entropy $H(X^3)$ is 1.4069.

A Huffman code for X^4 maps the sixteen source strings to the following codelengths:

$$\{0000, 1000, 0100, 0010, 0001, 1100, 0110, 0011, 0101, 1010, 1001, 1110, 1101, 1011, 0111, 1111\} \rightarrow \{1, 3, 3, 3, 4, 6, 7, 7, 7, 7, 7, 9, 9, 9, 10, 10\}.$$

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
0000	0.1296	2.9	3	000
0001	0.0864	3.5	4	0100
0010	0.0864	3.5	4	0110
0100	0.0864	3.5	4	0111
1000	0.0864	3.5	3	100
1100	0.0576	4.1	4	1010
1010	0.0576	4.1	4	1100
1001	0.0576	4.1	4	1101
0110	0.0576	4.1	4	1110
0101	0.0576	4.1	4	1111
0011	0.0576	4.1	4	0010
1110	0.0384	4.7	5	00110
1101	0.0384	4.7	5	01010
1011	0.0384	4.7	5	01011
0111	0.0384	4.7	4	1011
1111	0.0256	5.3	5	00111

Figure 5.7. Huffman code for X^4 when $p_0 = 0.6$. Column 4 shows the assigned codelengths and column 5 the codewords. Notice the asymmetries between strings whose probabilities are identical.

This has expected length $L(C, X^4) = 1.9702$ whereas the entropy $H(X^4)$ is 1.876.

When $\mathcal{P}_X = \{0.6, 0.4\}$, the Huffman code for X^2 has lengths $\{2, 2, 2, 2\}$; the expected length is 2 bits, and the entropy is 1.94 bits. A Huffman code for X^4 is shown in figure 5.7. The expected length is 3.92 bits, and the entropy is 3.88 bits.

Solution to exercise 5.20 (p. 103): The set of probabilities $\{p_1, p_2, p_3, p_4\} = \{1/6, 1/6, 1/3, 1/3\}$ give rise to two different optimal sets of codelengths, because at the second step of the Huffman coding algorithm we can choose any of the three possible pairings. We may either put them in a constant length code $\{00, 01, 10, 11\}$ or the code $\{000, 001, 01, 1\}$. Both codes have expected length 2.

Another solution is $\{p_1, p_2, p_3, p_4\} = \{1/5, 1/5, 1/5, 2/5\} = \{0.2, 0.2, 0.2, 0.4\}$.

And a third is $\{p_1, p_2, p_3, p_4\} = \{1/3, 1/3, 1/3, 0\}$.

Solution to exercise 5.21 (p. 103): Probability vectors leading to a free choice in the Huffman coding algorithm satisfy $p_1 \geq p_2 \geq p_3 \geq p_4 \geq 0$ and

$$p_1 = p_3 + p_4. \tag{5.25}$$

The convex hull of $\mathcal{Q}$ is most easily obtained by turning two of the three inequalities $p_1 \geq p_2 \geq p_3 \geq p_4$ into equalities, and then solving equation (5.25) for $\mathbf{p}$. Each choice of equalities gives rise to one of the set of three vectors

$$\{1/3, 1/3, 1/6, 1/6\},\ \{2/5, 1/5, 1/5, 1/5\} \text{ and } \{1/3, 1/3, 1/3, 0\}. \tag{5.26}$$

Solution to exercise 5.23 (p. 103): The sparse source $\mathcal{P}_X = \{0.99, 0.01\}$ could be compressed with a Huffman code based on blocks of length N, but N would

need to be quite large for the code to be efficient. The probability of the all–0 sequence has to be reduced to about 0.5 or smaller for the code to be efficient. This sets $N \simeq \log 0.5 / \log 0.99 = 69$. The Huffman code would then have 2^{69} entries in its tree, which probably exceeds the memory capacity of all the computers in this universe and several others.

There are other ways that we could describe the data stream. One is run-length encoding. We could chop the source into the substrings $1, 01, 001, 0001, 00001, \ldots$ with the last elements in the set being, say, two strings of equal maximum length $00\ldots01$ and $00\ldots00$. We can give names to each of these strings and compute their probabilities, which are not hugely dissimilar to each other. This list of probabilities starts $\{0.01, 0.0099, 0.009801, \ldots\}$. For this code to be efficient, the string with largest probability should have probability about 0.5 or smaller; this means that we would make a code out of about 69 such strings. It is perfectly feasible to make such a code. The only difficulty with this code is the issue of termination. If a sparse file ends with a string of 20 0s still left to transmit, what do we do? This problem has arisen because we failed to include the end–of–file character in our source alphabet. The best solution to this problem is to use an arithmetic code as described in the next chapter.

Solution to exercise 5.24 (p. 103): The poisoned glass problem is intended to have the solution '129', this being the only number of the form $2^m + 1$ between 100 and 200. However the optimal strategy, assuming all glasses have equal probability, is to design a Huffman code for the glasses. This produces a binary tree in which each pair of branches have almost equal weight. On the first measurement, either 64 or 65 of the glasses are tested. (Given the assumption that one of the glasses is poisoned, it makes no difference which; however, going for 65 might be viewed as preferable if there were any uncertainty over this assumption.) There is a 2/129 probability that an extra test is needed after seven tests have occurred. So the expected number of tests is $7\frac{2}{129}$, whereas the strategy of the professor takes 8 tests with probability 128/129 and one test with probability 1/129, giving a mean number of tests $7\frac{122}{129}$. The expected waste is 40/43 tests.

Exercise 5.28:[A2] There are two ways to answer this problem correctly, and one popular way to answer it incorrectly. Let's give the incorrect answer first:

Erroneous answer. We can pick a random bit by first picking a random source symbol x_i with probability p_i, then picking a random bit from $c(x_i)$. If we define f_i to be the fraction of the bits of $c(x_i)$ that are 1s, we find

$$P(\text{bit is } 1) = \sum_i p_i f_i \tag{5.27}$$

$$= \frac{1}{2} \times 0 + \frac{1}{4} \times 1/2 + \frac{1}{8} \times 2/3 + \frac{1}{8} \times 1 = 1/3 \tag{5.28}$$

This answer is wrong because it falls for the busstop fallacy, which was introduced in exercise 2.23 (p. 40): if buses arrive at random, and we are interested in the average time from one bus until the next, we must distinguish two possible averages: (a) the average time from a randomly chosen bus until the next; (b) the average time between the bus you missed

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

and the next bus, which is twice as big. The second average is twice as big because, by waiting for a bus at a random time, you bias your selection of a bus in favour of buses that follow a large gap. You're unlikely to catch a bus that comes 10 seconds after a preceding bus! Similarly, the symbols `c` and `d` get encoded into longer-length binary strings than `a`, so when we pick a bit from the compressed string at random, we are more likely to land in a bit belonging to a `c` or a `d` than would be given by the probabilities p_i in the expectation (5.27). All the probabilities need to be scaled up by l_i, and renormalised.

Correct answer in the same style. Every time symbol x_i is encoded, l_i bits are added to the binary string, of which $f_i l_i$ are `1`s. The expected number of `1`s added per symbol is

$$\sum_i p_i f_i l_i; \tag{5.29}$$

and the expected total number of bits added per symbol is

$$\sum_i p_i l_i. \tag{5.30}$$

So the fraction of `1`s in the transmitted string is

$$P(\text{bit is } \mathtt{1}) = \frac{\sum_i p_i f_i l_i}{\sum_i p_i l_i} \tag{5.31}$$

$$= \frac{\frac{1}{2}\times 0 + \frac{1}{4}\times 1 + \frac{1}{8}\times 2 + \frac{1}{8}\times 3}{7/4} = (7/8)/(7/4) = 1/2. \tag{5.32}$$

For a general symbol code and a general ensemble, the expectation (5.31) is the correct answer. But in this case, there is a more powerful argument we can use.

Information-theoretic answer. The encoded string $\mathbf{c}$ is the output of an optimal compressor that compresses samples from X down to an expected length of $H(X)$ bits. We can't expect to compress this data any further. But if the probability $P(\text{bit is } \mathtt{1})$ were not equal to $1/2$ then it would be possible to compress the binary string further (using a block compression code, say). Therefore $P(\text{bit is } \mathtt{1})$ must be equal to $1/2$; indeed the probability of any sequence of l bits in the compressed stream taking on any particular value must be 2^{-l}. The output of a perfect compressor is always perfectly random bits.

To put it another way, if the probability $P(\text{bit is } \mathtt{1})$ were not equal to $1/2$, then the information content per bit of the compressed string would be at most $H_2(P(\mathtt{1}))$, which would be less than 1; but this contradicts the fact that we can recover the original data from $\mathbf{c}$, so the information content per bit of the compressed string must be $H(X)/L(C,X) = 1$.

Solution to exercise 5.26 (p. 104): The general Huffman coding algorithm for an encoding alphabet with q symbols has one difference from the binary case. The process of combining q symbols into 1 symbol reduces the number of symbols by $q-1$. So if we start with A symbols, we'll only end up with a

complete q–ary tree if $A \bmod (q-1)$ is equal to 1. Otherwise, we know that whatever prefix code we make, it will be an incomplete tree with a number of missing leaves equal, modulo $(q-1)$, to $A \bmod (q-1) - 1$. For example, if a ternary tree is built for eight symbols, then there will unavoidably be one missing leaf in the tree.

The optimal q–ary code is made by putting these extra leaves in the longest branch of the tree. This can be achieved by adding the appropriate number of symbols to the original source symbol set, all of these extra symbols having probability zero. The total number of states is then equal to $k(q-1)+1$, for some integer k. The symbols are then repeatedly combined by taking the q symbols with smallest probability and replacing them by a single symbol, as in the binary Huffman coding algorithm.

Solution to exercise 5.27 (p. 104): We wish to show that a greedy metacode, which picks the code which gives the shortest encoding, is actually suboptimal, because it violates the Kraft inequality.

We'll assume that each symbol x is assigned lengths $l_k(x)$ by each of the candidate codes C_k. Let us assume there are K alternative codes and that we can encode which code is being used with a header of length $\log K$ bits. Then the metacode assigns lengths $l'(x)$ that are given by

$$l'(x) = \log_2 K + \min_k l_k(x). \tag{5.33}$$

We compute the Kraft sum:

$$S = \sum_x 2^{-l'(x)} = \frac{1}{K} \sum_x 2^{-\min_k l_k(x)} \tag{5.34}$$

Let's divide the set $\mathcal{A}_X$ into non–overlapping subsets $\{\mathcal{A}_k\}_{k=1}^K$ such that subset $\mathcal{A}_k$ contains all the symbols x that get encoded using code k by the metacode. Then

$$S = \frac{1}{K} \sum_k \sum_{x \in \mathcal{A}_k} 2^{-l_k(x)}. \tag{5.35}$$

Now if one sub–code k satisfies the Kraft equality $\sum_{x \in \mathcal{A}_X} 2^{-l_k(x)}=1$, then it must be the case that

$$\sum_{x \in \mathcal{A}_k} 2^{-l_k(x)} \leq 1, \tag{5.36}$$

with equality only if all the symbols x are in $\mathcal{A}_k$, which would mean that we are only using one of the K codes. So

$$S \leq \frac{1}{K} \sum_{k=1}^{K} 1 = 1, \tag{5.37}$$

with equality only if equation (5.36) is an equality for all codes k. But it's impossible for all the symbols to be in *all* the non–overlapping subsets $\{\mathcal{A}_k\}_{k=1}^K$, so we can't have equality (5.36) holding for *all* k. So

$$S < 1. \tag{5.38}$$

Another way of seeing that a mixture code is suboptimal is to consider the binary tree that it defines. Think of the special case of two codes. The

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

first bit we send identifies which code we are using. Now, in a complete code, any subsequent binary string is a valid string. But once we know that we are using, say, code A, we know that what follows can only be a codeword corresponding to a symbol x whose encoding is shorter under code A than code B. So some strings are invalid continuations, and the mixture code is incomplete and suboptimal.

We will further discuss this issue and its relationship to probabilistic modelling in the chapter on 'bits back coding'.

About Chapter 6

Before reading chapter 6, you should have read the previous chapter and worked on most of the exercises in it.

The description of Lempel–Ziv coding is based on that of Cover and Thomas (1991).

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

6

Stream Codes

In this chapter we discuss two data compression schemes.

Arithmetic coding is a beautiful method that goes hand in hand with the philosophy that compression of data from a source entails probabilistic modelling of that source. As of 1999, the best compression methods for text files use arithmetic coding, and several state-of-the-art image compression systems use it too.

Lempel-Ziv coding is a 'universal' method, designed under the philosophy that we would like a single compression algorithm that will do a reasonable job for *any* source. In fact, for many real life sources, this algorithm's universal properties hold only in the limit of unfeasibly large amounts of data, but, all the same, Lempel-Ziv compression is widely used and often effective.

6.1 The guessing game

As a motivation for these two compression methods, let us consider the redundancy in a typical English text file. Such files have redundancy at several levels: for example, they contain the ASCII characters with non-equal frequency; certain consecutive pairs of letters are more probable than others; and entire words can be predicted given the context and a semantic understanding of the text.

To illustrate the redundancy of English, and a curious way in which it could be compressed, we can imagine a guessing game in which an English speaker repeatedly attempts to predict the next character in a text file.

For simplicity, let us assume that the allowed alphabet is just the 26 upper case letters `A,B,C,..., Z` and a space '-'. The game involves asking the subject to guess the next character repeatedly, the only feedback being whether the guess is correct or not, until the character is correctly guessed. After a correct guess, we note the number of guesses that were made when the character was identified, and ask the subject to guess the next character.

One sentence gave the following result when a human was asked to guess a sentence. The numbers of guesses are listed below each character.

```
T H E R E - I S - N O - R E V E R S E - O N - A - M O T O R C Y C L E -
1 1 1 5 1 1 2 1 1 2 1 1 15 1 17 1 1 1 2 1 3 2 1 2 2 7 1 1 1 1 4 1 1 1 1 1
```

Notice that in many cases, the next letter is guessed immediately, in one guess. In other cases, particularly at the start of syllables, far more guesses are needed.

What do this game and these results offer us? First, they demonstrate the redundancy of English from the point of view of an English speaker. Second, this game might be used in a data compression scheme, as follows.

The string of numbers '1, 1, 1, 5, 1, ...', listed above, was obtained by presenting the text to the subject. The maximum number of guesses that the subject will make for a given letter is twenty-seven, so what the subject is doing for us is performing a time-varying mapping of the twenty-seven letters $\{\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots, \mathtt{Z}, -\}$ onto the twenty-seven numbers $\{1, 2, 3, \ldots, 27\}$, which we can view as symbols in a new alphabet. The total number of symbols has not been reduced, but since he uses some of these symbols much more frequently than others — for example, 1 and 2 — it should be easy to compress this new string of symbols.

How would the *uncompression* of the sequence of numbers '1, 1, 1, 5, 1, ...' work? At uncompression time, we do not have the original string 'THERE...', we just have the encoded sequence. Imagine that our subject has an absolutely identical twin who also plays the guessing game with us, as if we knew the source text. If we stop him whenever he has made a number of guesses equal to the given number, then he will have just guessed the correct letter, and we can then say 'yes, that's right', and move to the next character. Alternatively, if the identical twin is not available, we could design a compression system with the help of just one human as follows. We choose a window length L, that is, a number of characters of context to show the human. For every one of the 27^L possible strings of length L, we ask them, 'What would you predict is the next character?', and 'If that prediction were wrong, what would your next guesses be?'. After tabulating their answers to these 26×27^L questions, we could use two copies of these enormous tables at the encoder and the decoder in place of the two human twins.

These systems are clearly unrealistic for practical compression, but they illustrate several principles that we will make use of later.

6.2 Arithmetic Codes

When we discussed variable-length symbol codes, and the optimal Huffman algorithm for constructing them, we concluded by pointing out two practical and theoretical problems with Huffman codes (section 5.6).

These defects are rectified by *arithmetic codes*, which were invented by Elias, by Rissanen and by Pasco, and subsequently promoted by Witten *et al.* (1987). In an arithmetic code, the probabilistic modelling is clearly separated from the encoding operation. The system is rather similar to the guessing game. The human predictor is replaced by a *probabilistic model* of the source. As each symbol is produced by the source, the probabilistic model supplies a *predictive distribution* over all possible values of the next symbol, that is, a list of positive numbers $\{p_i\}$ that sum to one. If we choose to model the source as producing i.i.d. symbols with some known distribution, then the predictive distribution is the same every time; but arithmetic coding can with equal ease handle complex adaptive models that produce context-dependent predictive distributions. Such a model would in general be implemented in a computer program.

The encoder makes use of the model's predictions to create a binary string. The decoder makes use of an identical twin of the model (just as in the guessing

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

game) to interpret the binary string.

Let the source alphabet be $\mathcal{A}_X = \{a_1, \ldots, a_I\}$, and let the Ith symbol a_I have the special meaning 'end of transmission'. The source spits out a sequence $x_1, x_2, \ldots, x_n, \ldots$. The source does *not* necessarily produce i.i.d. symbols. We will assume that a computer program is provided to the encoder that assigns a predictive probability distribution over a_i given the sequence that has occurred thus far, $P(x_n{=}a_i|x_1, \ldots, x_{n-1})$. The receiver has an identical program that produces the same predictive probability distribution $P(x_n{=}a_i|x_1, \ldots, x_{n-1})$.

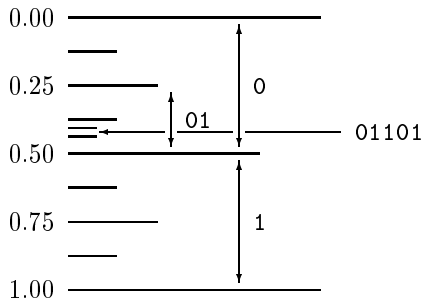


Figure 6.1. Binary strings define real intervals within the real line [0,1). We first encountered a picture like this when we discussed the symbol-code supermarket in chapter 5.

Concepts for understanding arithmetic coding

Notation for intervals. The interval $[0.01, 0.10)$ is all numbers between 0.01 and 0.10, including $0.01\dot{0} \equiv 0.01000\ldots$ but not $0.10\dot{0} \equiv 0.10000\ldots$.

A binary transmission defines an interval within the real line from 0 to 1. For example, the string `01` is interpreted as a binary real number $0.01\ldots$, which corresponds to the interval $[0.01, 0.10)$ (binary), *i.e.*, the interval $[0.25, 0.50)$ (base ten).

The longer string `01101` corresponds to a smaller interval [0.01101, 0.01110). Because `01101` has the first string, `01`, as a prefix, the new interval is a sub-interval of the interval $[0.01, 0.10)$. A one-megabyte binary file (2^{23} bits) is thus viewed as specifying a number between 0 and 1 to a precision of about ten million decimal places.

Similarly, we can divide the real line [0,1) into I intervals of lengths equal to the probabilities $P(x_1{=}a_i)$, as shown in figure 6.2.

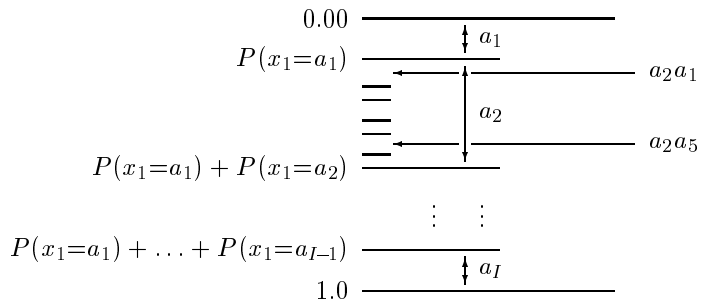


Figure 6.2. A probabilistic model defines real intervals within the real line [0,1).

We may then take each interval a_i and subdivide it into intervals denoted $a_ia_1, a_ia_2, \ldots, a_ia_I$, such that the length of a_ia_j is proportional to

$P(x_2{=}a_j|x_1{=}a_i)$. Indeed the length of the interval $a_i a_j$ will be precisely the joint probability

$$P(x_1{=}a_i, x_2{=}a_j) = P(x_1{=}a_i)P(x_2{=}a_j|x_1{=}a_i). \tag{6.1}$$

Iterating this procedure, the interval $[0, 1)$ can be divided into a sequence of intervals corresponding to all possible finite length strings $x_1x_2\ldots x_N$, such that the length of an interval is equal to the probability of the string given our model.

Formulae describing arithmetic coding

The process depicted in figure 6.2 can be written out explicitly as follows. The intervals are defined in terms of the lower and upper cumulative probabilities

$$Q_n(a_i|x_1,\ldots,x_{n-1}) \equiv \sum_{i'=1}^{i-1} P(x_n{=}a_{i'}|x_1,\ldots,x_{n-1}), \tag{6.2}$$

$$R_n(a_i|x_1,\ldots,x_{n-1}) \equiv \sum_{i'=1}^{i} P(x_n{=}a_{i'}|x_1,\ldots,x_{n-1}). \tag{6.3}$$

As the nth symbol arrives, we subdivide the $n-1$th interval at the points defined by Q_n and R_n. For example, starting with the first symbol, the intervals 'a_1', 'a_2', and 'a_I' are

$$a_1 \leftrightarrow [Q_1(a_1), R_1(a_1)) = [0, P(x_1{=}a_1)), \tag{6.4}$$

$$a_2 \leftrightarrow [Q_1(a_2), R_1(a_2)) = [P(x{=}a_1), P(x{=}a_1) + P(x{=}a_2)), \tag{6.5}$$

and

$$a_I \leftrightarrow [Q_1(a_I), R_1(a_I)) = [P(x_1{=}a_1) + \ldots + P(x_1{=}a_{I-1}), 1.0). \tag{6.6}$$

The following algorithm describes the general procedure.

Iterative procedure to find the interval $[u, v)$ for the string $x_1x_2\ldots x_N$

```
u := 0.0
v := 1.0
p := v − u
for n = 1 to N {
        Compute {R_n(i|x_1,...,x_{n−1})}_{i=0}^{I} using equation (6.3)
        v := u + pR_n(x_n|x_1,...,x_{n−1})
        u := u + pQ_n(x_n|x_1,...,x_{n−1})
        p := v − u
}
```

The method of encoding a string $x_1x_2\ldots x_N$ is straightforward. We simply locate the interval corresponding to $x_1x_2\ldots x_N$, and send a binary string whose interval lies within that interval.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Example: compressing the tosses of a bent coin

Imagine that we watch as a bent coin is tossed some number of times. [c.f. example 2.7 (p. 34) and section 3.2 (p. 54).] The two outcomes when the coin is tossed are denoted `a` and `b`. A third possibility is that the experiment is halted, an event denoted by the 'end of file' symbol, '$\square$'. Because the coin is bent, we expect that the probabilities of the outcomes `a` and `b` are not equal, though beforehand we don't know which is the more probable outcome.

Encoding

Let the source string be '`bbba`$\square$'. We pass along the string one symbol at a time and use our model to compute the probability distribution of the next symbol given the string thus far. Let these probabilities be:

Context (sequence thus far)	Probability of next symbol		
	$P(\mathtt{a})=0.425$	$P(\mathtt{b})=0.425$	$P(\square)=0.15$
`b`	$P(\mathtt{a}\|\mathtt{b})=0.28$	$P(\mathtt{b}\|\mathtt{b})=0.57$	$P(\square\|\mathtt{b})=0.15$
`bb`	$P(\mathtt{a}\|\mathtt{bb})=0.21$	$P(\mathtt{b}\|\mathtt{bb})=0.64$	$P(\square\|\mathtt{bb})=0.15$
`bbb`	$P(\mathtt{a}\|\mathtt{bbb})=0.17$	$P(\mathtt{b}\|\mathtt{bbb})=0.68$	$P(\square\|\mathtt{bbb})=0.15$
`bbba`	$P(\mathtt{a}\|\mathtt{bbba})=0.28$	$P(\mathtt{b}\|\mathtt{bbba})=0.57$	$P(\square\|\mathtt{bbba})=0.15$

The corresponding intervals are shown in figure 6.3. The interval `b` is the middle 0.425 of $[0,1)$. The interval `bb` is the middle 0.567 of `b`, and so forth.

When the first symbol '`b`' is observed, the encoder knows that the encoded string will start '`01`', '`10`', or '`11`', but does not know which. The encoder takes no action and examines the next symbol, which is '`b`'. The interval '`bb`' lies wholly within interval '`1`', so the encoder can write the first bit: '`1`'. The third symbol '`b`' narrows down the interval a little, but not quite enough for it to lie wholly within interval '`10`'. Only when the next '`a`' is read from the source can we transmit some more bits. Interval '`bbba`' lies wholly within the interval '`1001`', so the encoder adds '`001`' to what it has written. Finally when the '$\square$' arrives, we need a procedure for terminating the encoding. Magnifying the interval '`bbba`$\square$' (figure 6.3, right) we note that '`100111101`' is wholly contained by `bbba`$\square$, so the encoding can be completed by appending '`11101`'.

Solutions on p. 129

Exercise 6.1:[A2] Show that the overhead required to terminate a message is never more than 2 bits, relative to the ideal message length, $h(\mathbf{x}|\mathcal{H}) = \log[1/P(\mathbf{x}|\mathcal{H})]$, given the probabilistic model $\mathcal{H}$.

This is an important result. Arithmetic coding is very nearly optimal. The message length is always within two bits of the Shannon information content of the entire source string, so the expected message length is within two bits of the entropy of the entire message.

Decoding

The decoder receives the string '`100111101`' and passes along it one symbol at a time. First, the probabilities $P(\mathtt{a}), P(\mathtt{b}), P(\square)$ are computed using the

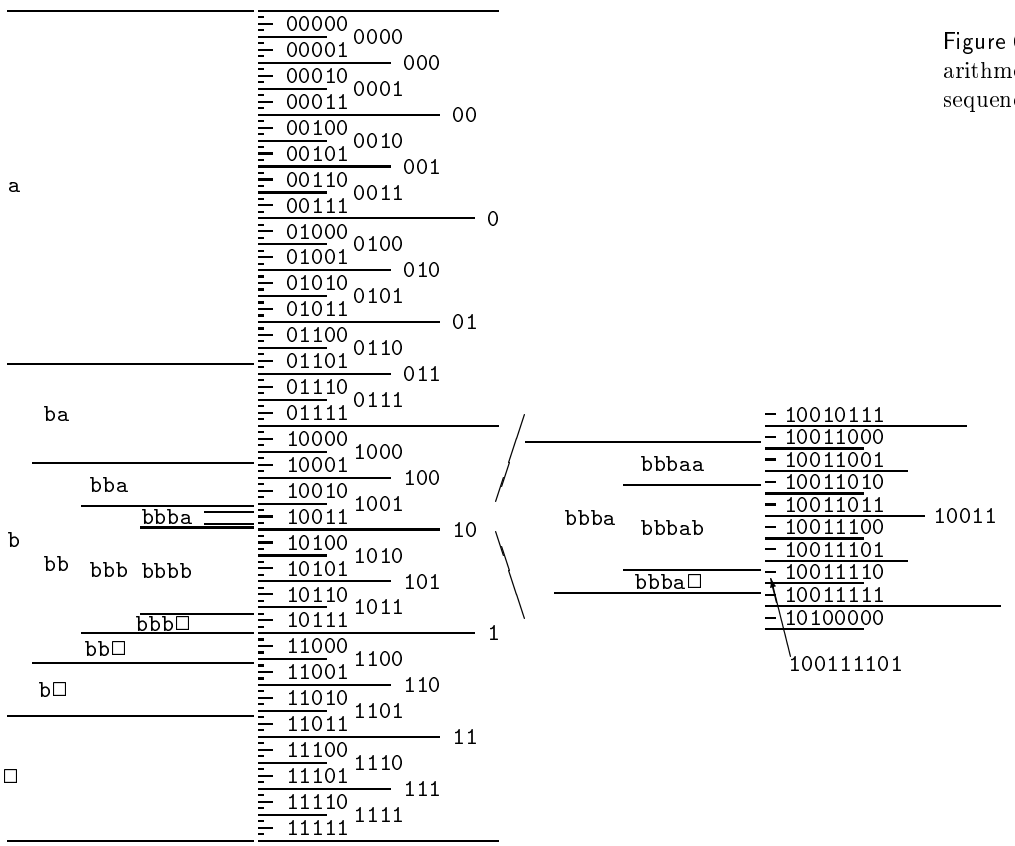


Figure 6.3. Illustration of the arithmetic coding process as the sequence bbba□ is transmitted

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

identical program that the encoder used and the intervals 'a', 'b' and '□' are deduced. Once the first two bits '10' have been examined, it is certain that the original string must have been started with a 'b', since the interval '10' lies wholly within interval 'b'. The decoder can then use the model to compute $P(\mathtt{a}|\mathtt{b}), P(\mathtt{b}|\mathtt{b}), P(\square|\mathtt{b})$ and deduce the boundaries of the intervals 'ba', 'bb' and 'b□'. Continuing, we decode the second b once we reach '1001', the third b once we reach '100111', and so forth, with the unambiguous identification of 'bbba□' once the whole binary string has been read. With the convention that '□' denotes the end of the message, the decoder knows to stop decoding.

Transmission of multiple files

How might one use arithmetic coding to communicate several distinct files over the binary channel? Once the □ character has been transmitted, we imagine that the decoder is reset into its initial state. There is no transfer of the learnt statistics of the first file to the second file. If, however, we did believe that there is a relationship among the files that we are going to compress, we could define our alphabet differently, introducing a second end-of-file character that marks the end of the file but instructs the encoder and decoder to continue using the same probabilistic model. If we went this route, we would only be able to uncompress the second file after first uncompressing the first file.

The big picture

Notice that to communicate a string of N letters both the encoder and the decoder only needed to compute $N|\mathcal{A}|$ conditional probabilities — the probabilities of each possible letter in each context actually encountered — just as in the guessing game. This cost can be contrasted with the alternative of using a Huffman code with a large block size (in order to reduce the possible one-bit-per-symbol overhead discussed in section 5.6), where *all* block sequences that could occur must be considered and their probabilities evaluated.

Notice how flexible arithmetic coding is: it can be used with any encoded alphabet and any source alphabet. The size of the source alphabet can change with time. Arithmetic coding can be used with any probability distribution, which can change utterly from context to context.

How the probabilistic model might make its predictions

The technique of arithmetic coding does not force one to produce the predictive probability in any particular way, but the predictive distributions might naturally be produced by a Bayesian model.

Figure 6.3 was generated using a simple model that always assigns a probability of 0.15 to □, and assigns the remaining 0.85 to a and b, divided in proportion to probabilities given by Laplace's rule, which we encountered in exercise 2.8 (p. 34),

$$P_{\mathrm{L}}(\mathtt{a}|x_1,\ldots,x_{n-1}) = (F_{\mathtt{a}}+1)/(F_{\mathtt{a}}+F_{\mathtt{b}}+2), \tag{6.7}$$

where $F_{\mathtt{a}}(x_1,\ldots,x_{n-1})$ is the number of times that a has occurred so far, and $F_{\mathtt{b}}$ is the count of bs. These predictions corresponds to a simple Bayesian model that is able to learn a non-equal frequency of use of the source symbols

a and b within a file. The end result will be an encoder that can adapt to a non-uniform source.

Figure 6.4 displays the intervals corresponding to a large number of strings. Note that if the string so far has contained a large number of bs then the probability of b relative to a is increased, and conversely if many as occur then as are made more probable. Larger intervals, remember, require fewer bits to encode.

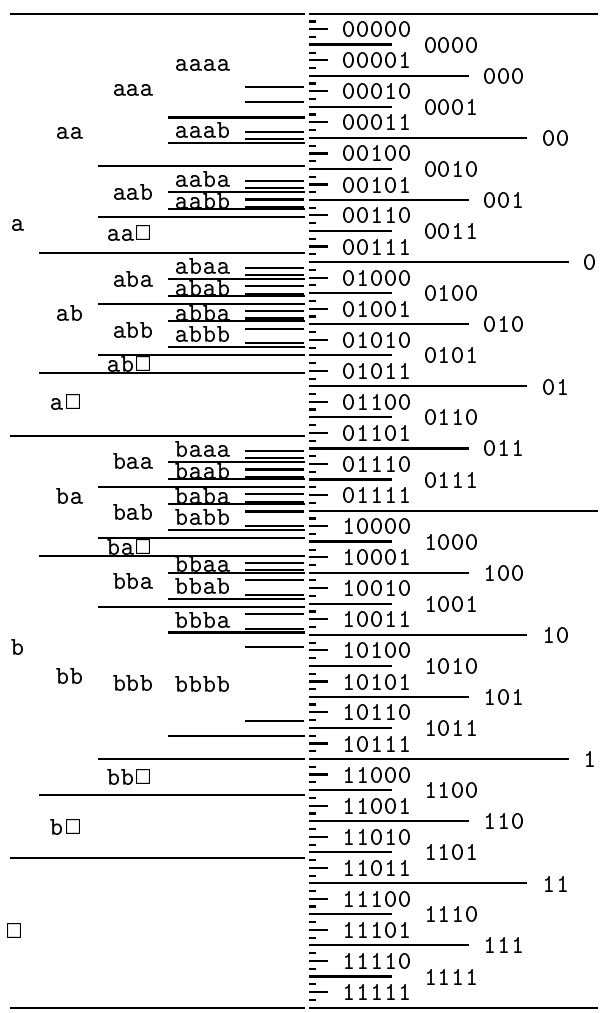


Figure 6.4. Illustration of the intervals defined by a simple Bayesian probabilistic model. The size of an intervals is proportional to the probability of the string. This model anticipates that the source is likely to be biased towards one of a and b, so sequences having lots of as or lots of bs have larger intervals than sequences of the same length that are 50:50 as and bs.

Details of the Bayesian model

Having emphasized that any model could be used — arithmetic coding is not wedded to any particular set of probabilities — let me explain the simple adaptive probabilistic model used in the preceding example.

Assumptions

The model will be described using parameters $p_\Box$, $p_{\mathtt{a}}$ and $p_{\mathtt{b}}$, defined below, which should not be confused with the predictive probabilities *in a particular context*, for example, $P(\mathtt{a}|\mathtt{s}{=}\mathtt{baa})$. An analogy for this model, as I indicated at the start, is the tossing of a bent coin. A bent coin labelled `a` and `b` is tossed some number of times l, which we don't know beforehand. The coin's probability of coming up `a` when tossed is $p_{\mathtt{a}}$, and $p_{\mathtt{b}} = 1 - p_{\mathtt{a}}$; the parameters $p_{\mathtt{a}}, p_{\mathtt{b}}$ are not known beforehand. The source string $\mathtt{s} = \mathtt{baaba}\Box$ indicates that l was 5 and the sequence of outcomes was `baaba`.

1. It is assumed that the length of the string l has an exponential probability distribution
$$P(l) = (1 - p_\Box)^l p_\Box. \tag{6.8}$$
This distribution corresponds to assuming a constant probability $p_\Box$ for the termination symbol '$\Box$' at each step.

2. It is assumed that the non-terminal characters in the string are selected independently at random from an ensemble with probabilities $\mathcal{P} = \{p_{\mathtt{a}}, p_{\mathtt{b}}\}$; the probability $p_{\mathtt{a}}$ is fixed throughout the string to some unknown value that could be anywhere between 0 and 1. The probability of an `a` occurring as the next symbol, given $p_{\mathtt{a}}$ (if only we knew it), is $(1 - p_\Box)p_{\mathtt{a}}$. The probability, given $p_{\mathtt{a}}$, that an unterminated string of length F is a given string $\mathtt{s}$ that contains $\{F_{\mathtt{a}}, F_{\mathtt{b}}\}$ counts of the two outcomes is the Bernoulli distribution
$$P(\mathtt{s}|p_{\mathtt{a}}, F) = p_{\mathtt{a}}^{F_{\mathtt{a}}}(1 - p_{\mathtt{a}})^{F_{\mathtt{b}}}. \tag{6.9}$$

3. We assume a uniform prior distribution for $p_{\mathtt{a}}$,
$$P(p_{\mathtt{a}}) = 1, \quad p_{\mathtt{a}} \in [0, 1] \tag{6.10}$$
and $p_{\mathtt{b}} \equiv 1 - p_{\mathtt{a}}$. It would be easy to assume other priors on $p_{\mathtt{a}}$, with beta distributions being the most convenient to handle.

Inferences and predictions

Given a string of length F of which $F_{\mathtt{a}}$ symbols are `a`s and $F_{\mathtt{b}}$ are `b`s, we are interested in (a) inferring what $p_{\mathtt{a}}$ might be; (b) predicting the probability of an `a` or `b` being the next character (assuming that it is not '$\Box$'; this last assumption is just for convenience so we don't have factors of $(1 - p_\Box)$ sitting around).

The posterior probability of $p_{\mathtt{a}}$ given an unterminated string $\mathtt{s}$ of length F that has counts $\{F_{\mathtt{a}}, F_{\mathtt{b}}\}$ is, by Bayes' theorem,
$$P(p_{\mathtt{a}}|\mathtt{s}, F) = \frac{P(\mathtt{s}|p_{\mathtt{a}}, F)P(p_{\mathtt{a}})}{P(\mathtt{s}|F)}. \tag{6.11}$$
The factor $P(\mathtt{s}|p_{\mathtt{a}}, F)$, which as a function of $p_{\mathtt{a}}$ is known as the likelihood function, was given in equation (6.9); the prior $P(p_{\mathtt{a}})$ was given in equation (6.10). Our inference of $p_{\mathtt{a}}$ is thus given by:
$$P(p_{\mathtt{a}}|\mathtt{s}, F) = \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}}(1 - p_{\mathtt{a}})^{F_{\mathtt{b}}}}{P(\mathtt{s}|F)}. \tag{6.12}$$

The normalizing constant is given by the beta integral

$$P(\mathbf{s}|F) = \int_0^1 dp_{\mathtt{a}}\, p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}} P(p_{\mathtt{a}}) = \frac{\Gamma(F_{\mathtt{a}}+1)\Gamma(F_{\mathtt{b}}+1)}{\Gamma(F_{\mathtt{a}}+F_{\mathtt{b}}+2)} = \frac{F_{\mathtt{a}}!F_{\mathtt{b}}!}{(F_{\mathtt{a}}+F_{\mathtt{b}}+1)!}. \tag{6.13}$$

Exercise 6.2: Sketch the posterior probability $P(p_{\mathtt{a}}|\mathbf{s}{=}\mathtt{aba}, F{=}3)$. What is the most probable value of $p_{\mathtt{a}}$ (*i.e.*, the value that maximizes the posterior probability density)? What is the mean value of $p_{\mathtt{a}}$ under this distribution?

Answer the same questions for the posterior probability $P(p_{\mathtt{a}}|\mathbf{s}{=}\mathtt{bbb}, F{=}3)$.

Our prediction of the probability of the next character's being an a (assuming it is not '□') is obtained by integrating over $p_{\mathtt{a}}$. This has the effect of taking into account our uncertainty about $p_{\mathtt{a}}$ when making predictions. By the sum rule,

$$P(\mathtt{a}|\mathbf{s}, F) = \int dp_{\mathtt{a}}\, P(\mathtt{a}|p_{\mathtt{a}}) P(p_{\mathtt{a}}|\mathbf{s}, F). \tag{6.14}$$

The probability of an a given $p_{\mathtt{a}}$ is simply $p_{\mathtt{a}}$, so

$$P(\mathtt{a}|\mathbf{s}, F) = \int dp_{\mathtt{a}}\, p_{\mathtt{a}} \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}}}{P(\mathbf{s}|F)} \tag{6.15}$$

$$= \int dp_{\mathtt{a}}\, \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}+1}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}}}{P(\mathbf{s}|F)} \tag{6.16}$$

$$= \left[\frac{(F_{\mathtt{a}}+1)!F_{\mathtt{b}}!}{(F_{\mathtt{a}}+F_{\mathtt{b}}+2)!}\right] \Big/ \left[\frac{F_{\mathtt{a}}!F_{\mathtt{b}}!}{(F_{\mathtt{a}}+F_{\mathtt{b}}+1)!}\right] = \frac{F_{\mathtt{a}}+1}{F_{\mathtt{a}}+F_{\mathtt{b}}+2} \tag{6.17}$$

which is Laplace's rule.

Exercise 6.3:[B3] Compare the expected message length when an ASCII file is compressed by the following two methods.

Huffman-with-header. Read the whole file, find the empirical frequency of each symbol, construct a Huffman code for those frequencies, transmit the code by transmitting the lengths of the Huffman codewords, then transmit the file using the Huffman code. (The actual codewords don't need to be transmitted, since we can use a deterministic method for building the tree given the codelengths.)

Arithmetic code using Laplace model.

$$P_{\mathrm{L}}(a|x_1, \ldots, x_{n-1}) = \frac{F_{\mathtt{a}}+1}{\sum_{\mathtt{a}'}(F_{\mathtt{a}'}+1)}. \tag{6.18}$$

Arithmetic code using a Dirichlet model. This model's predictions are:

$$P_{\mathrm{L}}(a|x_1, \ldots, x_{n-1}) = \frac{F_{\mathtt{a}}+\alpha}{\sum_{\mathtt{a}'}(F_{\mathtt{a}'}+\alpha)}, \tag{6.19}$$

where α is fixed to a number such as 0.01. This corresponds to a more responsive version of the Laplace model; the probability over characters is expected to be more non-uniform; $\alpha = 1$ reproduces the Laplace model.

Take care that the header of your Huffman message is self-delimiting. Special cases worth considering are (a) short files with just a few hundred characters; (b) large files in which some characters are never used.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Further applications of arithmetic coding

Arithmetic coding not only offers a way to compress strings believed to come from a given model; it also offers a way to generate random strings from a model. Imagine sticking a pin into the unit interval at random, that line having been divided into subintervals in proportion to probabilities p_i; the probability that your pin will lie in interval i is p_i.

So to generate a sample from a model, all we need to do is feed ordinary random bits into an arithmetic decoder for that model. An infinite random bit sequence corresponds to the selection of a point at random from the line $[0, 1)$, so the decoder will then select a string at random from the assumed distribution. This arithmetic method is guaranteed to use very nearly the smallest number of random bits possible to make the selection — an important point in communities where random numbers are expensive!

A simple example of the use of this technique is in the generation of random bits with a non-uniform distribution $\{p_0, p_1\}$.

Exercise 6.4:^{A2} Compare the following two techniques for generating random symbols from a non-uniform distribution $\{p_0, p_1\} = \{0.99, 0.01\}$:

(a) The standard method: use a standard random number generator to generate an integer between 1 and 2^{32}. Rescale the integer to (0,1). Test whether this uniformly distributed random variable is less than 0.99, and emit a `0` or `1` accordingly.

(b) Arithmetic coding using the correct model, fed with standard random bits.

Roughly how many random bits will each method use to generate a thousand samples from this sparse distribution?

6.3 Lempel-Ziv Coding

The Lempel-Ziv algorithms, which are widely used for data compression (*e.g.*, the `compress` command in `UNIX` and `gzip` in `GNU`), are different in philosophy to arithmetic coding. There is no separation between modelling and coding, and no opportunity for explicit modelling.

Basic Lempel-Ziv algorithm

The method of compression is to replace a substring with a pointer to an earlier occurrence of the same substring. For example if the string is `1011010100010`..., we parse it into a *dictionary* of substrings that have not appeared so far as follows: λ,`1`,`0`,`11`,`01`,`010`,`00`,`10`,.... We include the empty substring λ as the first substring in the list. After every comma, we look along the next part of the input sequence until we have read a substring that has not been marked off before. A moment's reflection will confirm that this substring is longer by one bit than a substring that has occurred earlier in the list. This means that we can encode each substring by giving a *pointer* to the earlier occurrence of that prefix and then sending the extra bit by which the new substring in the stack differs from the earlier substring. If, at the nth bit, we have enumerated $s(n)$ substrings, then we can give the value of the pointer in $\lceil \log_2 s(n) \rceil$ bits. The code for the above sequence is then as follows (with

punctuation included for clarity), the upper lines indicating the source string and the value of $s(n)$:

source substrings	λ	1	0	11	01	010	00	10
$s(n)$	0	1	2	3	4	5	6	7
$s(n)_{\text{binary}}$	000	001	010	011	100	101	110	111
(pointer,bit)		(, 1)	(0, 0)	(01, 1)	(10, 1)	(100, 0)	(010, 0)	(001, 0)

Notice that the first pointer we send is the empty string, because, given that there is only one substring on the stack, no bits are needed to convey the 'choice' of that substring as the prefix. Thus the encoded string is 100011101100001000010. The encoding, in this simple case, is actually a longer string than the source string, because there was no obvious redundancy in the source string.

Exercise 6.5:^{B2} Prove that *any* uniquely decodeable code from $\{0,1\}^+$ to $\{0,1\}^+$ necessarily makes some strings longer if it makes some strings shorter.

This is not necessarily the explanation for the above lengthening, however, because the algorithm described is certainly inefficient: once a substring in the list has been joined there by both of its children, then we can be sure that it will not be needed; so at this point we could drop it from our dictionary of substrings and shuffle them all along one, thereby reducing the length of subsequent pointer messages. Equivalently, we could write the second prefix into the dictionary at the point previously occupied by the parent. A second unnecessary overhead is the transmission of the new bit in these cases — the second time a prefix is used, we can be sure of the identity of the next bit.

Decoding

The decoder again involves an identical twin at the decoding end who constructs the dictionary of substrings as the data is decoded.

Exercise 6.6:^{B2} Encode the string 000000000000100000000000 using the basic Lempel-Ziv algorithm described above.

Exercise 6.7:^{B2} Decode the string 00101011101100100100011010101000011 that was encoded using the basic Lempel-Ziv algorithm.

Practicalities

In this description I have not discussed the method for terminating a string.

There are many variations on the Lempel-Ziv algorithm, all exploiting the same idea but using different procedures for dictionary management, etc.. The resulting programs are fast, but their performance on compression of English text, although useful, is poor by the standards set in the arithmetic coding literature.

Theoretical properties

In contrast to the block code, Huffman code, and arithmetic coding methods we discussed in the last three chapters, the Lempel-Ziv algorithm is defined

without making any mention of a probabilistic model for the source. Yet, given any ergodic[1] source, the Lempel-Ziv algorithm can be proven *asymptotically* to compress down to the entropy of the source. This is why it is called a 'universal' compression algorithm. For a proof of this property, see Cover and Thomas (1991).

It achieves its compression, however, only by *memorizing* substrings that have happened so that it has a short name for them the next time they occur. The asymptotic timescale on which this universal performance is achieved is likely to be the time that it takes for *all* typical sequences of length n^* to occur, where n^* is the longest lengthscale associated with the statistical fluctuations in the source. For many sources the time for all typical sequences to occur is unfeasibly long. The useful performance of the algorithm in practice is a reflection of the fact that many files contain multiple repetitions of particular short sequences of characters, a form of redundancy to which the algorithm is well suited.

Common ground

I have emphasized the difference in philosophy behind arithmetic coding and Lempel-Ziv coding. There is an overlap between them, though: in principle, one can design adaptive probabilistic models, and thence arithmetic codes, that are 'universal', that is, models that will asymptotically compress any source in some class to within some factor (preferably 1) of its entropy. However, for practical purposes, I think such universal models can only be constructed if the class of sources is severely restricted. A general purpose compressor that can discover the probability distribution of *any* source would be a general purpose artificial intelligence! A general purpose artificial intelligence does not yet exist.

6.4 Demonstration

A demonstration arithmetic-coding software package written by Radford Neal is available from
`ftp://ftp.cs.toronto.edu/pub/radford/www/ac.software.html`. This package consists of encoding and decoding modules to which the user adds a module defining the probabilistic model. It should of course be emphasized that there is no single general-purpose arithmetic-coding compressor; a new model has to be written for each type of source.

Radford Neal's package includes a simple adaptive model similar to the Bayesian model demonstrated in section 6.2. The results using this Laplace model should be viewed as a basic benchmark since it is perhaps the simplest possible probabilistic model — it simply assumes the characters in the file come independently from a fixed ensemble. The counts $\{F_i\}$ of the symbols $\{a_i\}$ are rescaled and rounded as the file is read such that all the counts lie between 1 and 256.

Yann Le Cun, Leon Bottou and colleagues at AT&T Labs have written a state-of-the-art compressor for documents containing text and images, `DjVu`, which uses arithmetic coding.[2] The JBIG image compression standard for

[1] Need to clarify this. It means the source is memoryless on sufficiently long timescales.
[2] `DjVu` is described here: `http://djvu.research.att.com/`

binary images uses arithmetic coding with a context-dependent model which adapts using a rule similar to Laplace's rule. PPM *(need reference)* is a leading method for text compression, and it uses arithmetic coding. `bzip` is a block-sorting file compressor; how does it work? *(need details here.)*

There are many Lempel-Ziv-based programs. `gzip` is based on a version of Lempel-Ziv called 'LZ77'. `compress` is based on 'LZW'. In my experience the best is `gzip`, with `compress` being inferior on most files.

Compression of a text file

The following table gives the computer time in seconds taken and the compression achieved when these programs are applied to the LaTeX file containing the text of this chapter, of size 20942 bytes.

Method	Compression time / sec	Compressed size (%age of 20942)	Uncompression time / sec
Laplace model	0.28	12974 (61%)	0.32
`gzip`	0.10	8177 (39%)	0.01
`compress`	0.05	10816 (51%)	0.05
`bzip`		7495 (36%)	
`bzip2`		7640 (36%)	
`ppmz`		6800 (32%)	

Compression of a sparse file

Interestingly, `gzip` does not always do so well. The following table gives the compression achieved when these programs are applied to a text file containing 10^6 characters, each of which is either `0` and `1` with probabilities 0.99 and 0.01. The Laplace model is quite well matched to this source, and the benchmark arithmetic coder gives good performance, followed closely by `compress`; `gzip` is worst. An ideal model for this source would compress the file into about $10^6 H_2(0.01)/8 \simeq 10100$ bytes. The Laplace model compressor falls short of this performance because it is implemented using only eight-bit precision. The `ppmz` compressor compresses the best of all, but takes much more computer time.

Method	Compression time / sec	Compressed size	Uncompression time / sec
Laplace model	0.45	14143 (1.4%)	0.57
`gzip`	0.22	20646 (2.1%)	0.04
`gzip --best`	1.63	15553 (1.6%)	0.05
`compress`	0.13	14785 (1.5%)	0.03
`bzip`	0.30	10903 (1.09%)	0.17
`bzip2`	0.19	11260 (1.12%)	0.05
`ppmz`	533	10447 (1.04%)	535

6.5 Summary

In the last three chapters we have studied three classes of data compression codes.

Fixed-length block codes (Chapter 2). These are mappings from a fixed number of source symbols to a fixed length binary message. Only a tiny fraction of the source strings are given an encoding. These codes were fun for identifying the entropy as the measure of compressibility but they are of little practical use.

Symbol codes (Chapter 3). Symbol codes employ a variable length code for each symbol in the source alphabet, the codelengths being integer lengths determined by the probabilities of the symbols. Huffman's algorithm constructs an optimal symbol code for a given set of symbol probabilities.

Every source string has a uniquely decodeable encoding, and if the source symbols come from the assumed distribution then the symbol code will compress to an expected length L lying in the interval $[H, H+1)$. Statistical fluctuations in the source may make the actual length longer or shorter than this mean length.

If the source is not well matched to the assumed distribution then the mean length is increased by the relative entropy $D_{\rm KL}$ between the source distribution and the code's implicit distribution. For sources with small entropy, the symbol has to emit at least one bit per source symbol; compression below one bit per source symbol can only be achieved by the cumbersome procedure of putting the source data into blocks.

Stream codes. The distinctive property of stream codes, compared with symbol codes, is that they are not constrained to emit at least one bit for every symbol read from the source stream. Large numbers of source symbols can be coded into a smaller number of bits without the source stream being explicitly put into blocks.

- Arithmetic codes combine a probabilistic model with an encoding algorithm that identifies each string with a sub-interval of $[0, 1)$ of size equal to the probability of that string under the model. This code is almost optimal in the sense that the compressed length of a string $\mathbf{x}$ closely matches the Shannon information content of $\mathbf{x}$ given the probabilistic model. Arithmetic codes fit with the philosophy that good compression requires *data modelling*, in the form of an adaptive Bayesian model.

- Lempel-Ziv codes are adaptive in the sense that they memorize strings that have already occurred. They are built on the philosophy that we don't know anything at all about what the probability distribution of the source will be, and we want a compression algorithm that will perform reasonably well whatever that distribution is.

Both arithmetic codes and Lempel-Ziv codes will fail to decode correctly if any of the bits of the compressed file are altered. So if compressed files are to be stored or transmitted over noisy media, error-correcting codes will be essential. Reliable communication over unreliable channels is the topic of the next few chapters.

6.6 Problems

Solutions on p. 131

Exercise 6.8:^{A2} Describe an arithmetic coding algorithm to encode random bit strings of length N and weight K (*i.e.*, K ones and $N-K$ zeroes) where N and K are given.

For the case $N = 5, K = 2$ show in detail the intervals corresponding to all source substrings of lengths 1–5.

Exercise 6.9:^{B2} How many bits are needed to specify a selection of K objects from N objects? (N and K are assumed to be known and the selection of K objects is unordered.) How might such a selection be made at random without being wasteful of random bits?

Exercise 6.10:^{B2} Describe an arithmetic coding algorithm to generate random bit strings of length N with density f (*i.e.*, each bit has probability f of being a one) where N is given.

Exercise 6.11:^{C2} *This question needs to have its meaning made more clear.*

Assume that a Bayesian model and arithmetic encoder are to be used to encode the values of N variables $(x_1, \ldots, x_N)$ in a *correlated* ensemble. Show that, in principle, the variables could be communicated in any fixed order (known to the encoder and the decoder) and the length of the arithmetic interval would be just the same — so the message length will be the same (give or take a couple of bits).

Exercise 6.12:^{A2} Use a modified Lempel-Ziv algorithm in which, as discussed on 124, the dictionary of prefixes is pruned by writing new prefixes into the space occupied by prefixes that will not be needed again. Such prefixes can be identified when both their children have been added to the dictionary of prefixes. (You may neglect the issue of termination of encoding.) Use this algorithm to encode the string 0100001000100010101000001. Highlight the bits that follow a prefix on the second occasion that that prefix is used. (As discussed earlier, these bits could be omitted.)

Exercise 6.13:^{B2} Show that this modified Lempel-Ziv code is still not 'complete', that is, there are binary strings that are not encodings of any string.

Exercise 6.14:^{A3} Give examples of simple sources that have low entropy but would not be compressed well by the Lempel-Ziv algorithm.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 6's exercises

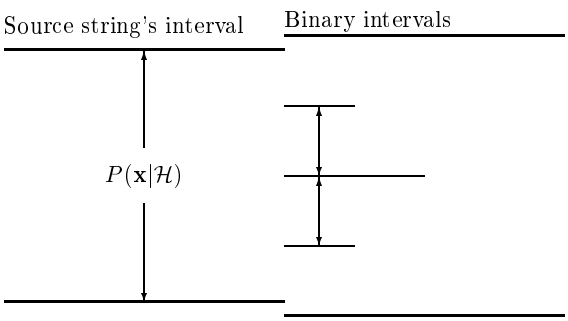
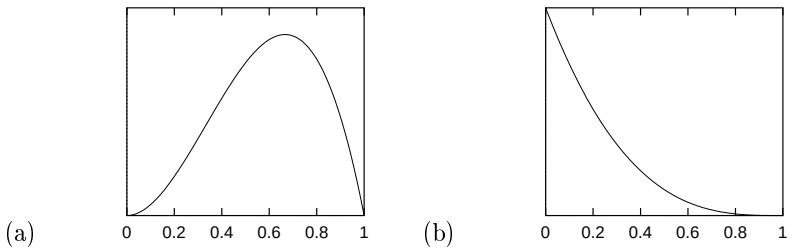


Figure 6.5. Termination of arithmetic coding in the worst case, where there is a two bit overhead. Either of the two binary intervals marked on the right hand side may be chosen. These binary intervals are no smaller than $P(\mathbf{x}|\mathcal{H})/4$.

Solution to exercise 6.1 (p. 117): The worst case situation is when the interval to be represented lies just inside a binary interval. In this case, we may choose either of two binary intervals as shown in figure 6.5. These binary intervals are no smaller than $P(\mathbf{x}|\mathcal{H})/4$, so the binary encoding has a length no greater than $\log_2 1/P(\mathbf{x}|\mathcal{H}) + \log_2 4$, which is two bits more than the ideal message length.



$P(p_a|\mathbf{s} = aba, F = 3) \propto p_a^2(1 - p_a)$ $P(p_a|\mathbf{s} = bbb, F = 3) \propto (1 - p_a)^3$

Figure 6.6. Posterior probability for the bias p_a of a bent coin given two different data sets.

Solution to exercise 2.8 (p. 34):

(a) $P(p_a|\mathbf{s} = aba, F = 3) \propto p_a^2(1 - p_a)$. The most probable value of p_a (*i.e.*, the value that maximizes the posterior probability density) is 2/3. The mean value of p_a is 3/5.

See figure 6.6 (a).

(b) $P(p_a|\mathbf{s}=bbb, F=3) \propto (1-p_a)^3$. The most probable value of p_a (*i.e.*, the value that maximizes the posterior probability density) is 0. The mean value of p_a is 1/5.

See figure 6.6 (b).

Solution to exercise 6.3 (p. 122): The Huffman-with-header is bound to be suboptimal because it is not a complete code, just as the mixture code of exercise 5.27 (p. 104) was not complete. We assume the encoder uses a fixed mapping from files to Huffman codelengths. Once the Huffman codelengths have been received, the receiver knows, in principle, that the file can only be a file whose mapping to codelengths matches those codelengths. This implies that all files that have other Huffman codelengths are invalid. Yet the Huffman code used to communicate the file is capable of encoding *all* source files. So the code is not complete.

How might the header be built? We need to communicate the codelengths. At the outset, we know that the codelengths of the 128 source symbols are integers between 1 and 127. They might all be equal to 7, but for a typical non-uniform file, there will be a variety of codelengths, with some shorter than 7 and some longer. An efficient header would make use of these correlations among the codelengths. An inefficient method would be to encode the 128 lengths by 128 seven-bit integers, giving a total header size of 896 bits (a little over 100 bytes). An alternative self-delimiting variable-length code could convey each codelength using the code for integers C_α from section 7.1. The shortest header would be obtained if all the codelengths were 7 ($c_\alpha(7) = \overline{00}11$), in which case the header length would be $128 \times 5 = 640$ bits $= 80$ bytes. By being smarter about the code for the header, we might be able to reduce this header size somewhat. Smarter methods might include sending an initial pre-header indicating the total file length; then the encoding of the codelengths could depend on the file length, with cheaper and cruder codes being used for shorter files.

In the case of a short file with length N equal to a few hundred characters, the Huffman-with-header method is obliged to send a header listing all the codelengths. The codelengths will all be roughly 7 bits, so the header size will be something like 80 bytes, depending on the encoding method used. The length of the subsequent file will depend on the statistics of the original file, but the shortest it could be is about N bits, if the source file were made up of a single character. If $N = 100$, the total file length would be about $640 + 100$ bits. Notice how the header dominates the encoded cost.

In contrast, the arithmetic code using the Laplace model would give a total file length close to

$$\log 1/P(\mathbf{x}|\mathbf{F}) = \log \frac{(N+126)!}{\prod_a F_a!}. \tag{6.20}$$

For very small N, there is no constant overhead equivalent to the header of the preceding code. Let's take a length of $N = 100$ characters as a concrete example. The encoded file length is largest if all the F_a are equal to 1, in which case

$$\log 1/P(\mathbf{x}|\mathbf{F}) = \log 226! \simeq 1447 \text{ bits}.. \tag{6.21}$$

If one F_a were equal to N, then

$$\log 1/P(\mathbf{x}|\mathbf{F}) = \log 226!/100! \simeq 922 \text{ bits}. \tag{6.22}$$

(Compare this with the length of the original file encoded with plain 7-bit ascii, which would be 700 bits.)

Using the Dirichlet model, ... *(more here).*

Solution to exercise 6.4 (p. 123): The standard method uses 32 random bits per generated symbol and so requires 32,000 bits to generate one thousand samples.

Arithmetic coding uses on average about $H_2(0.01) = 0.081$ bits per generated symbol, and so requires about 83 bits to generate one thousand samples (assuming an overhead of roughly two bits associated with termination).

Solution to exercise 6.5 (p. 124): Task: to prove that if *all* binary strings are encoded in a way that makes them shorter or leaves their length unchanged, the compression code must not be uniquely decodeable.

Proof: Consider the set S_N of all source strings of length up to N. There are $2 + 2^2 + \ldots + 2^N = 2^{N+1} - 2$ such strings. Assuming that one or more of these strings is made shorter, and none of them are made longer, then by a simple counting argument the code must define a mapping from S_N into S_N which is many-to-one and is thus not uniquely decodeable.

(more detail needed here.)

Solution to exercise 6.6 (p. 124): The encoding is 010100110010110001100, which comes from the parsing

$$0, 00, 000, 0000, 001, 00000, 000000 \tag{6.23}$$

which is encoded thus

$$(, 0), (1, 0), (10, 0), (11, 0), (010, 1), (100, 0), (110, 0). \tag{6.24}$$

Solution to exercise 6.7 (p. 124): The decoding is 0100001000100010101000001.

Solution to exercise 6.8 (p. 128): Figure 6.7 shows the arithmetic encoder for the case $N = 5$, $K = 2$.

0	0	0	1	1
		1	0	1
			1	0
	1	0	0	1
			1	0
		1	0	0
1	0	0	0	1
			1	0
		1	0	0
	1	0	0	0

Figure 6.7. Arithmetic encoder for binary strings of length $N = 5$ with fixed weight $K = 2$.

Solution to exercise 6.9 (p. 128): This problem is equivalent to the previous one.

The selection of K objects from N objects requires $\lceil \log_2 \binom{N}{K} \rceil$ bits $\simeq NH_2(K/N)$ bits. This selection could be made using arithmetic coding. The selection corresponds to a binary string of length N in which the 1 bits represent which objects are selected. Initially the probability of a 1 is K/N and the probability of a 0 is $(N-K)/N$. Thereafter, given that the emitted string thus far, of length n, contains k 1s, the probability of a 1 is $(K-k)/(N-n)$ and the probability of a 0 is $1-(K-k)/(N-n)$.

Solution to exercise 6.10 (p. 128): One can generate strings with density f by running dense random bits into the decoder corresponding to the arithmetic encoder for a sparse source with density f. See figure 6.8.

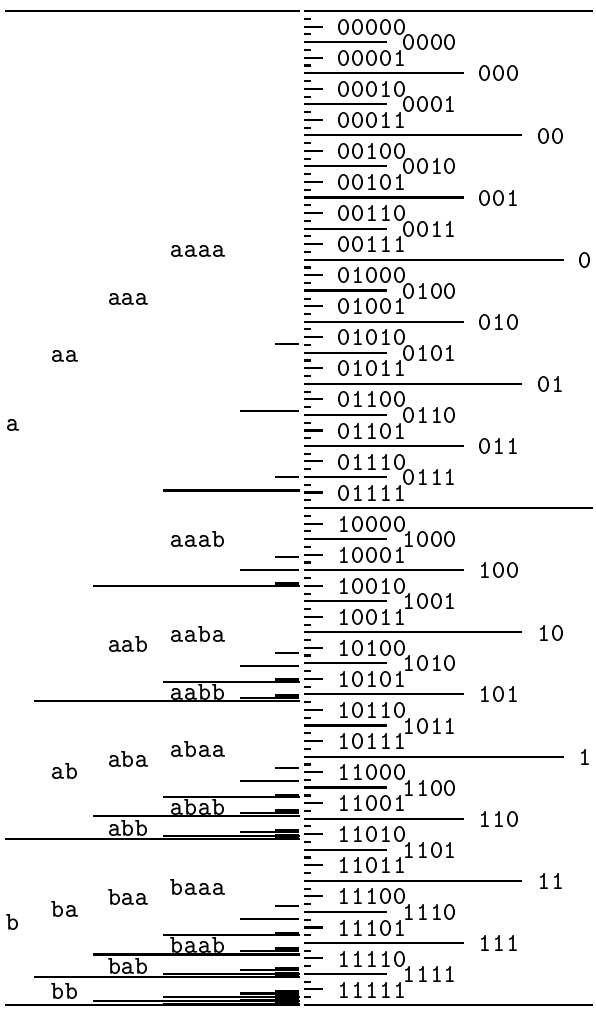


Figure 6.8. Arithmetic encoder for a sparse source with $f = 1/6$.

Solution to exercise 6.11 (p. 128): The joint probability may be written thus:

$$P(x_1, \ldots, x_N) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \ldots P(x_N|x_1, \ldots, x_{N-1}) \quad (6.25)$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

and arithmetic coding associates an interval of this length with the sequence. The right hand side defines the sequence of factors by which the initial unit interval is multiplied during arithmetic coding. If the order in which $x_1, \ldots, x_N$ are communicated is changed in some fixed way then a new decomposition can be written, for example:

$$P(x_1, \ldots, x_N) = P(x_N)P(x_{N-1}|x_N) \ldots P(x_1|x_2, \ldots, x_N). \tag{6.26}$$

The factors on the right hand side are all different from what we had before, but by the product rule, their product (the left hand side) is the same as the product in equation (6.25).

Solution to exercise 6.12 (p. 128): The encoding is $\mathtt{00}\mathbf{1}\mathtt{1010110}\mathbf{0}\mathtt{0000110100101000001}\mathbf{1}$, coming from the following parsed message:

$$(,0), (0,\mathbf{1}), (1,0), (10,1), (10,\mathbf{0}), (00,0), (011,0), (100,1), (010,0), (001,\mathbf{1}) \tag{6.27}$$

The highlighted symbols could be omitted in the further improved coding system.

Solution to exercise 6.13 (p. 128): This modified Lempel-Ziv code is still not 'complete', because, for example, after five prefixes have been collected, the pointer could be any of the strings $000, 001, 010, 011, 100$, but it cannot be $101, 110$ or 111. Thus there are some binary strings that cannot be produced as encodings.

Solution to exercise 6.14 (p. 128): Sources with low entropy that are not well compressed by Lempel-Ziv include:

(a) Sources with some symbols that have long range correlations and intervening random junk. An ideal model should capture what's correlated and compress it. Lempel-Ziv can only compress the correlated features by memorizing all cases of the intervening junk. As a simple example, consider a telephone book in which every line contains an (old number, new number) pair:

```
285-3820:572-5892□
258-8302:593-2010□
```

The number of characters per line is 18, drawn from the 13-character alphabet $\{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}, \mathtt{-}, \mathtt{:}, \square\}$. The characters '-', ':' and '□' occur in a predictable sequence, so the true information content per line, assuming all the phone numbers are seven digits long, and assuming that they are random sequences, is about 14 bans. (A ban is the information content of a random integer between 0 and 9.) A finite state language model could easily capture the regularities in these data. A Lempel-Ziv algorithm will take a long time before it compresses such a file down by a factor of 14/18, however, because in order for it to 'learn' that the string $\mathtt{:}ddd$ is always followed by -, for any three digits ddd, it will have to *see* all those strings. So near-optimal compression will only be achieved after thousands of lines of the file have been read.

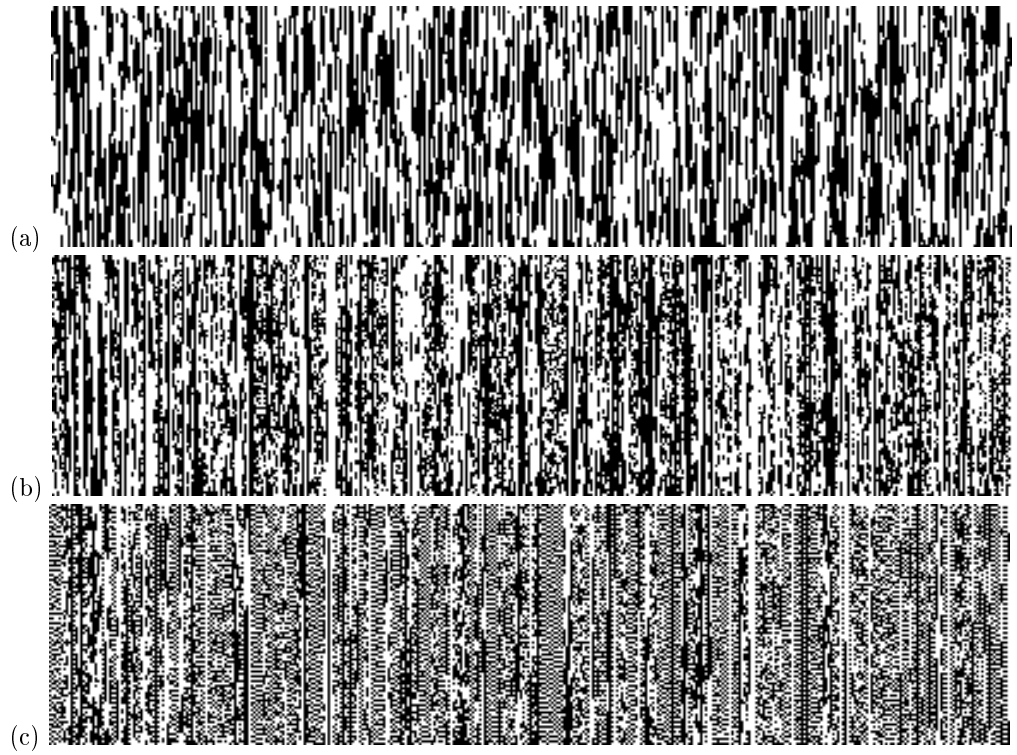


Figure 6.9. Three sources with low entropy that are not well compressed by Lempel-Ziv. The bit sequence is read from left to right. The image width is 400 pixels in each case. (a) Each line differs from the line above in $p = 5\%$ of its bits. (b) Each column c has a transition probability p_c such that successive vertical bits are identical with probability p_c. The probabilities p_c are drawn from a distribution over $[0, 0.5]$. (c) As in b, but the probabilities p_c are drawn from a distribution over $[0, 1]$; in columns with $p_c \simeq 1$, successive vertical bits are likely to be opposite to each other.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

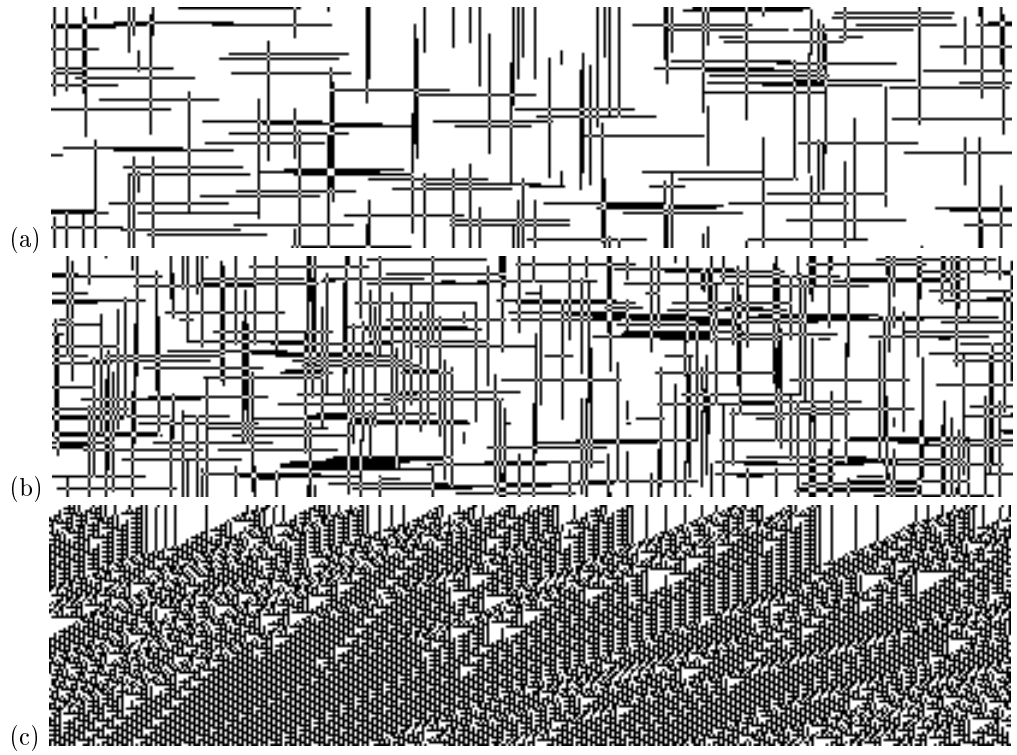


Figure 6.10. Three more sources with low entropy that are not optimally compressed by Lempel-Ziv.

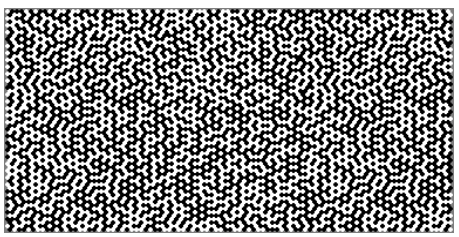


Figure 6.11. Frustrated triangular Ising model in one of its ground states.

(b) Sources with long range correlations, for example two-dimensional images that are represented by a sequence of pixels, row by row, so that vertically adjacent pixels are a distance w apart in the source stream, where w is the image width. Consider, for example, a fax transmission in which each line is very similar to the previous line (figure 6.9). Each line is somewhat similar to the previous line but not identical, so there is no previous occurrence of a long string to point to; some algorithms in the Lempel-Ziv class will achieve a certain degree of compression by memorizing recent short strings, but the compression achieved will not equal the true entropy. Lempel-Ziv algorithms will only commence compressing down to the entropy once *all* strings of length $2^w = 2^{400}$ have occurred and their successors have been memorized. There are only about 2^{300} particles in the universe, so we can confidently say that Lempel-Ziv codes will *never* capture the redundancy of such an image. The true entropy is only $H_2(f)$ per pixel, where f is the probablity that a pixel differs from its parent.

Two more highly redundant textures are shown in figure 6.10a,b. They were made by dropping horizontal and vertical pins randomly on the plane. These images contain both long-range vertical correlations and long-range horizontal correlations. There is no practical way that Lempel-Ziv, fed with a pixel-by-pixel scan of this image, could capture both these correlations.

Biological computational systems can readily identify the redundancy in these images and in images that are much more complex; thus we might anticipate that the best data compression algorithms will result from the development of artificial intelligence methods.

(c) Sources with intricate redundancy, such as files generated by computers. For example, a LaTeX file followed by its encoding into a postscript file. The information content of this pair of files is roughly equal to the information content of the LaTeX file alone.

(d) A picture of the Mandelbrot set. The picture has an information content equal to the number of bits required to specify the range of the complex plane studied, the pixel sizes, and the colouring rule used.

(e) A picture of a ground state of a frustrated antiferromagnetic Ising model (figure 6.11), which we will discuss in chapter 24. Like figure 6.10, this binary image has interesting correlations in two directions.

(f) Cellular automata — figure 6.10(c) shows the state history of 100 steps of a cellular automaton with 400 cells. The update rule, in which each cell's new state depends on the state of five preceding cells, was selected at random. The information content is equal to the information in the boundary (400 bits), and the propagation rule, which here can be described in 32 bits. An optimal compressor will thus give a compressed file length which is essentially constant, independent of the vertical height of the image. Lempel-Ziv would only give this zero-cost compression once the cellular automaton has entered a periodic limit cycle, which could easily take something like 2^{100} iterations.

In contrast, the JBIG compression method, which models the probability of a pixel given its local context and uses arithmetic coding, should do a good job on these images.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

(g) And finally, an example relating to error-correcting codes: the received transmissions arising when encoded transmissions are sent over a noisy channel. Such received strings have an entropy equal to the source entropy plus the channel noise's entropy. If a Lempel-Ziv algorithm could compress these strings, this would be tantamount to solving the decoding problem for the error-correcting code!

We have not got to this topic yet, but we will see later that the decoding of a general error-correcting code is a challenging intractable problem.

7

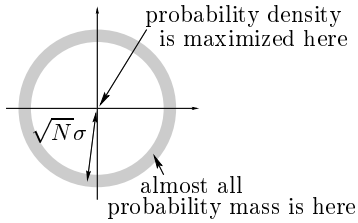
Further exercises on data compression

The following exercises may be skipped by the reader who is eager to learn about noisy channels.

Exercise 7.1:^{A3} Consider a Gaussian distribution in N dimensions,

$$P(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\sum_n x_n^2}{2\sigma^2}\right). \tag{7.1}$$

Define the radius of a point $\mathbf{x}$ to be $r = \left(\sum_n x_n^2\right)^{1/2}$. Estimate the mean and variance of the square of the radius, $r^2 = (\sum_n x_n^2)$.



You may find helpful the integral

$$\int dx \, \frac{1}{(2\pi\sigma^2)^{1/2}} \, x^4 \exp\left(-\frac{x^2}{2\sigma^2}\right) = 3\sigma^4, \tag{7.2}$$

though you should be able to estimate the required quantities without it.

Assuming that N is large, show that nearly all the probability of a Gaussian is contained in a thin shell of radius $\sqrt{N}\sigma$. Find the thickness of the shell.

Evaluate the probability density in $\mathbf{x}$ space (7.1) at a point in that thin shell and at the origin $\mathbf{x} = 0$ and compare them. Use the case $N = 1000$ as an example.

Notice that nearly all the probability mass is located in a different part of the space from the region of highest probability density.

Exercise 7.2:^{B3} In the guessing game, we imagined predicting the next letter in a document starting from the beginning and working towards the end. Consider the task of predicting the *reversed* text, that is, predicting the letter that precedes those already known. Most people find this a harder task. Assuming that we model the language using an N-gram model (which says the probability of the next character depends only on the $N-1$ preceding characters), is there any difference between the entropies of the reversed language and the forward language?

Exercise 7.3:^{A2} Explain what is meant by an *optimal binary symbol code*.

Find an optimal binary symbol code for the ensemble:

$$\mathcal{A} = \{a, b, c, d, e, f, g, h, i, j\},$$

$$\mathcal{P} = \left\{ \frac{1}{100}, \frac{2}{100}, \frac{4}{100}, \frac{5}{100}, \frac{6}{100}, \frac{8}{100}, \frac{9}{100}, \frac{10}{100}, \frac{25}{100}, \frac{30}{100} \right\},$$

and compute the expected length of the code.

Exercise 7.4:^{A2} A string $\mathbf{y} = x_1 x_2$ consists of *two* independent samples from an ensemble

$$X : \mathcal{A}_X = \{a, b, c\}; \mathcal{P}_X = \left\{ \frac{1}{10}, \frac{3}{10}, \frac{6}{10} \right\}$$

What is the entropy of $\mathbf{y}$? Construct an optimal binary symbol code for the string $\mathbf{y}$, and find its expected length.

Exercise 7.5:^{A2} (Cambridge University Part III Maths examination, 1998.)

Strings of N independent samples from an ensemble with $\mathcal{P} = \{0.1, 0.9\}$ are compressed using an arithmetic code that is matched to that ensemble. Estimate the mean and standard deviation of the compressed strings' lengths for the case $N = 1000$.

$[H_2(0.1) \simeq 0.47]$

Exercise 7.6:^{B3} (Cambridge University Part III Maths examination, 1998.)
Source coding with variable-length symbols

> In the lectures on source coding, we assumed that we were encoding into a binary alphabet $\{0, 1\}$ in which both symbols should be used with equal frequency. In this question we explore how the encoding alphabet should be used if the symbols take different times to transmit.

A poverty-stricken student communicates for free with a friend using a telephone by selecting an integer $n \in \{1, 2, 3 \ldots\}$ and making the friend's phone ring n times then hanging up in the middle of the nth ring. This process is repeated so that a string of symbols $n_1 n_2 n_3 \ldots$ is received. If large integers n are selected then the message takes longer to communicate. If only small integers n are used then the information content per symbol is small. The aim of this question is to ascertain the maximum rate of information transfer, per unit time, that can be achieved.

Assume that the time taken to transmit a number of rings n and to redial is l_n seconds. Consider a probability distribution over n, $\{p_n\}$. Defining the average duration *per symbol* to be

$$L(\mathbf{p}) = \sum_n p_n l_n \tag{7.3}$$

and the entropy *per symbol* to be

$$H(\mathbf{p}) = \sum_n p_n \log_2 \frac{1}{p_n}, \tag{7.4}$$

show that for the average information rate *per second* to be maximized, the symbols must be used with probabilties of the form

$$p_n = \frac{1}{Z} 2^{-\beta l_n} \tag{7.5}$$

where $Z = \sum_n 2^{-\beta l_n}$ and β satisfies the implicit equation

$$\beta = \frac{H(\mathbf{p})}{L(\mathbf{p})}, \tag{7.6}$$

that is, β is the rate of communication. Show that these two equations (7.5, 7.6) imply that β must be set such that

$$\log Z = 0. \tag{7.7}$$

Assuming that the channel has the property

$$l_n = n \,\text{seconds}, \tag{7.8}$$

find the optimal distribution $\mathbf{p}$ and show that the maximal information rate is 1 bit per second.

How does this compare with the information rate per second achieved if $\mathbf{p}$ is set to $(1/2, 1/2, 0, 0, 0, 0, \ldots)$ — that is, only the symbols $n = 1$ and $n = 2$ are selected, and they have equal probability?

Discuss the relationship between the results (7.5, 7.7) derived above, and the Kraft inequality from source coding theory.

How might a random binary source be efficiently encoded into a sequence of symbols $n_1 n_2 n_3 \ldots$ for transmission over the channel defined in equation (7.8)?

Exercise 7.7:^{B2} In the card game Bridge, the four players receive 13 cards each from the deck of 52 and start each game by looking at their own hand and bidding. The legal bids are, in ascending order $1\clubsuit, 1\diamondsuit, 1\heartsuit, 1\spadesuit$, $1NT$, $2\clubsuit$, $2\diamondsuit$, ... $7\heartsuit, 7\spadesuit, 7NT$, and successive bids must follow this order; a bid of, say, $2\heartsuit$ may only be followed by higher bids such as $2\spadesuit$ or $3\clubsuit$ or $7NT$. (Let us neglect the 'double' bid.)

The players' aims when bidding are several, but one of the aims is for two partners to communicate to each other as much as possible about what cards are in their hands.

Let us concentrate on this task.

(a) After the cards have been dealt, how many bits are needed for North to convey to South what her hand is?

(b) Assuming that E and W do not bid at all, what is the maximum total information that N and S can convey to each other while bidding? Assume that N starts the bidding, and that once either N or S stops bidding, the bidding stops.

Exercise 7.8:^{B2} My old 'arabic' microwave oven had 11 buttons for entering cooking times, and my new 'roman' microwave has just five. The buttons of the roman microwave are labelled '10 minutes', '1 minute', '10 seconds', '1 second', and 'Start'; I'll abbreviate these five strings to the symbols M, C, X, I, □. To enter one minute and twenty-three seconds (1:23), the arabic sequence is

$$123\square, \tag{7.9}$$

Arabic	Roman
1 2 3 4 5 6 7 8 9 0 □	M X C I □

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

and the roman sequence is

$$\texttt{CXXIII}\square. \tag{7.10}$$

Each of these keypads defines a code mapping the 3599 cooking times from 0:01 to 59:59 into a string of symbols.

(a) Which times can be produced with two or three symbols? (For example, 0:20 can be produced by three symbols in either code: XX□ and 20□.)

(b) Are these two codes complete? Discuss all the ways in which these two codes are not complete.

(c) For each code, name a couple of times that it can produce in four symbols that the other code cannot.

(d) Discuss the implicit probability distributions over times to which each of these codes is best matched.

(e) Concoct a plausible probability distribution over times that a real user might use, and evaluate roughly the expected number of symbols, and maximum number of symbols, that each code requires. Discuss the ways in which each code is inefficient or efficient.

(f) Invent a more efficient time-encoding system for a microwave oven.

Exercise 7.9:^{C3} Is the standard binary representation for positive integers (*e.g.*, $c_b(5) = \texttt{101}$) a uniquely decodeable code?

Design a binary code for the positive integers, *i.e.*, a mapping from $n \in \{1, 2, 3, \ldots\}$ to $c(n) \in \{\texttt{0}, \texttt{1}\}^+$, that is uniquely decodeable. Try to design codes that are prefix codes and that satisfy the Kraft equality $\sum_n 2^{-l_n}=1$.

> (Motivations: Note that any data file terminated by a special end of file character can be mapped onto an integer, so a prefix code for integers can be used as a self-delimiting encoding of files too. Large files correspond to large integers. Also, one of the building blocks of a 'universal' coding scheme — that is, a coding scheme that will work OK for a large variety of sources — is the ability to encode integers.)

Discuss criteria by which one might compare alternative codes for integers (or, equivalently, alternative self-delimiting codes for files).

Solutions to Chapter 7's exercises

Solution to exercise 7.1 (p. 138): For a one–dimensional Gaussian, the variance of x, $\mathcal{E}[x^2]$, is σ^2. So the mean value of r^2 in N dimensions, since the components of $\mathbf{x}$ are independent random variables, is

$$\mathcal{E}[r^2] = N\sigma^2. \tag{7.11}$$

The variance of r^2, similarly, is N times the variance of x^2, where x is a one-dimensional Gaussian variable.

$$\mathrm{var}(x^2) = \int dx \, \frac{1}{(2\pi\sigma^2)^{1/2}} x^4 \exp\left(-\frac{x^2}{2\sigma^2}\right) - \sigma^4. \tag{7.12}$$

The integral is found to be $3\sigma^4$ (equation (7.2)), so $\mathrm{var}(x^2) = 2\sigma^4$. Thus the variance of r^2 is $2N\sigma^4$.

For large N, the central limit theorem indicates that r^2 has a Gaussian distribution with mean $N\sigma^2$ and standard deviation $\sqrt{2N}\sigma^2$, so the probability density of r must similarly be concentrated about $r \simeq \sqrt{N}\sigma$.

The thickness of this shell is given by turning the standard deviation of r^2 into a standard deviation on r: for small $\delta r/r$, $\delta \log r = \delta r/r = (1/2)\delta \log r^2 = (1/2)\delta(r^2)/r^2$, so setting $\delta(r^2) = \sqrt{2N}\sigma^2$, r has standard deviation $\delta r = (1/2)r\delta(r^2)/r^2 = \sigma/\sqrt{2}$.

The probability density of the Gaussian at a point $\mathbf{x}_{\mathrm{shell}}$ where $r = \sqrt{N}\sigma$ is

$$P(\mathbf{x}_{\mathrm{shell}}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N\sigma^2}{2\sigma^2}\right) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2}\right). \tag{7.13}$$

Whereas the probability density at the origin is

$$P(\mathbf{x}{=}0) = \frac{1}{(2\pi\sigma^2)^{N/2}}. \tag{7.14}$$

Thus $P(\mathbf{x}_{\mathrm{shell}})/P(\mathbf{x}{=}0) = \exp\left(-N/2\right)$. The probability density at the typical radius is $e^{-N/2}$ times smaller than the density at the origin. If $N = 1000$, then the probability density at the origin is e^{500} times greater.

Solution to exercise 7.2 (p. 138): (Draft.)

Write down the entropies of the reversed language and the forward language, and show they are equal.

Solution to exercise 7.3 (p. 138): Using the Huffman coding algorithm, we arrive at the answer shown, which is unique (apart for trivial modifications to the codewords).

The expected length is 2.81. The entropy is 2.78.

a_i	p_i	$h(p_i)$	l_i	c_i
a	0.01	6.6	6	000000
b	0.02	5.6	6	000001
c	0.04	4.6	5	00001
d	0.05	4.3	4	0010
e	0.06	4.1	4	0011
f	0.08	3.6	4	0001
g	0.09	3.5	3	100
h	0.1	3.3	3	101
i	0.25	2.0	2	11
j	0.3	1.7	2	01

Solution to exercise 7.5 (p. 139): (Draft.) my answer: 470 ± 30.

Solution to exercise 7.8 (p. 140):

(a) The arabic keypad can produce the times 0:01–0:09 in two symbols and the times 0:10–0:59 in three symbols. The roman keypad can produce the times 0:01, 0:10, 1:00, and 10:00 in two symbols, and 0:02, 0:11, 0:20, 1:01, 1:10, 2:00, 20:00, 11:00, 10:10, and 10:01 in three symbols. The times 0:11, 1:01, 1:10, 11:00, 10:10, and 10:01 can all be produced in two different ways, because the two keys with numbers can be pressed in either sequence.

(b) The arabic code is incomplete because

  i. The keys `0` and □ are both illegal first symbols.

  ii. After a four–digit number has been entered, the only legal symbol is □.

(c) The roman code is incomplete because

  i. The key □ is an illegal first symbol.

  ii. Some times can be produced by multiple symbol–strings.

  iii. After a key has been pressed a number of times (five or nine, depending which key) it may not be pressed any more times.

(d) The arabic code can produce 3:15, 2:30, and 5:00 in four symbols, and the roman code cannot. The roman code can produce 12:00 and 21:00 in four symbols, and the arabic code cannot.

(e) Both codes allow the time 0:01 to be encoded with a very short sequence, length two. This is implicitly one of the most probable times for both models. In the arabic model, the implicit probability of a time is roughly $1/11^{l+1}$, where l is the length of the time when encoded in decimal. In the roman model, times that contain many ones and zeroes are the most probable, with the probability of a time decreasing roughly as the sum of its digits: $P(\mathbf{x}) \sim 1/5^s$, where $s = \sum_i x_i$.

(f) When I use the microwave, my probability distribution is roughly:

$\mathbf{x}$	0:10	0:20	0:30	1:00	1:10	1:20	1:30	2:00	3:00
$P(\mathbf{x})$	0.1	0.05	0.02	0.1	0.1	0.4	0.1	0.03	0.03
$\mathbf{x}$	5:00	7:00	8:00	10:00	12:00	20:00	other		
$P(\mathbf{x})$	0.02	0.02	0.02	0.01	0.01	0.01	ϵ		

(7.15)

The arabic system is poorly matched to my distribution because it forces me to push the zero button at the end of every time, to specify 'zero seconds', which I always want. The roman system similarly wastes an entire button (the `I` button) which I never use. The arabic system is otherwise quite well matched to my probability distribution, except that my favourite time (1:20 for a cafe latte) could do with a shorter sequence. The roman system is well–matched to some of my times, but terribly matched to others, in particular, the time 8:00.

The arabic system has a maximum codelength of five symbols. The roman system has a terrible maximum codelength of 28 symbols, for the time 59:59.

(g) An alternative encoder using five buttons would work as follows.

i. The display starts by offering as a default the median of the last one hundred times selected. If the user has a favourite time, then this will probably be offered. It can be accepted by pressing the □ key.

ii. The other four keys are marked `+`, `++`, `-`, and `--`.

- The `+` symbol increments the displayed time by a little bit, *e.g.*16%.
- The `-` symbol decrements the displayed time by a little bit, *e.g.*16%.
- The `++` symbol increments the displayed time by a lot, *e.g.*, a factor of two.
- The `--` symbol decrements the displayed time by a lot, *e.g.*, a factor of two.

To make this system even more adaptive to its user, these four buttons could have their effect by moving the percentile around the distribution of times recently used by the user. The initial time is the median. The `+` button takes us to the 63rd percentile and `++` takes us to the 87th, say, with the step size decreasing adaptively as the time is selected. If a user has five preferred times, these could adaptively be discovered by the system so that, after time, the five times would be invoked by the sequences `--`□, `-`□, □, `+`□, and `++`□ respectively.

Solution to exercise 7.7 (p. 140): (Draft.)

(a) After the cards have been dealt, the number of bits needed for North to convey her hand to South (remember that he already knows his own hand) is

$$\log_2 \binom{39}{13} \simeq 39 H_2(13/39) \simeq 36 \text{ bits.} \tag{7.16}$$

[This is wrong by 3 bits! Use a better approximation than this leading order approximation.] Now, North does not know South's hand, so how, in practice, could this information be conveyed efficiently? This relates to the Slepian Wolf correlated information comunication problem.

(b) The maximum number of bits is equal to 35, the number of distinct bids in the list $1\clubsuit \ldots 7NT$. Given the assumption that E and W do not bid, the bidding process can be viewed as defining a binary string of length 35, with a 1 against every bid that was made by N or S, and a 0 against every bid that was skipped. The complete bidding history can be reconstructed from this binary string, since N and S alternate (we assumed that the bidding stops if either of them does not bid). So the maximum total information conveyed cannot exceed 35 bits.

A bidding system which achieved this maximum information content would be one in which a binary code is agreed such that 0s and 1s are equally probable; then each bidder chooses the next bid by raising the bid by the appropriate number of notches. There will be a probability of 1/2 that they raise the bid by one notch; a probability of 1/4 that they raise it by two notches; and so forth.

Solution to exercise 7.4 (p. 139): The entropy of $\mathbf{y} = x_1 x_2$ is twice $H(X)$; $H(X) = 1.295$ bits so $H(\mathbf{Y}) = 2.59$ bits.

The optimal binary symbol code is constructed using the Huffman coding algorithm. There are several valid answers; the codelengths should be identical to one of the two lists below. The strings ab and ba, marked $\star$, are interchangeable.

a_i	p_i	$h(p_i)$	$l_i^{(1)}$	$c(a_i)$	$l_i^{(2)}$	$c^{(2)}(a_i)$
aa	0.01	6.6	6	000000	6	000000
ab$\star$	0.03	5.1	6	000001	6	000001
ba$\star$	0.03	5.1	5	00001	5	00001
ac	0.06	4.1	4	0010	4	0010
ca	0.06	4.1	4	0011	4	0011
bb	0.09	3.5	4	0001	4	0001
bc	0.18	2.5	3	010	2	10
cb	0.18	2.5	3	011	2	11
cc	0.36	1.5	1	1	2	01

The expected length is 2.67.

*Codes for integers**

7.1 Solution to exercise 7.9 (p. 141)

To discuss the coding of integers we need some definitions.

The standard binary representation of a positive integer n will be denoted by $c_b(n)$, *e.g.*, $c_b(5) = \texttt{101}$, $c_b(45) = \texttt{101101}$.

The standard binary length of a positive integer n is denoted by $l_b(n)$. For example, $l_b(5) = 3$, $l_b(45) = 6$..

The standard binary representation $c_b(n)$ is *not* a uniquely decodeable code for integers since there is no way of knowing when an integer has ended. For example, $c_b(5)c_b(5)$ is identical to $c_b(45)$. It would be uniquely decodeable if we knew the length of each integer before it was received.

Noticing that all positive integers have a standard binary representation that starts with a 1, we might define another representation:

The headless binary representation of a positive integer n will be denoted by $c_B(n)$, *e.g.*, $c_B(5) = \texttt{01}$, $c_B(45) = \texttt{01101}$ and $c_B(1) = \lambda$ (where λ denotes the null string).

This representation would be uniquely decodeable if we knew the length of the integer.

So, how can we make a uniquely decodeable code for integers? Two strategies can be distinguished.

1. **Codes with 'end of file' characters.** We code the integer into blocks of length b bits, and reserve one of the 2^b symbols to have the special meaning 'end of file'. The coding of integers into blocks is arranged so that this reserved symbol is not needed for any other purpose.

2. **Self-delimiting codes.** An alternative strategy is to make the code self-delimiting by first communicating the length of the integer $l_b(n)$ somehow, then communicating the integer itself using $c_B(n)$.

The simplest uniquely decodeable code for integers is the unary code, which can be viewed as a code with an end of file character.

Unary code. An integer n is encoded by sending a string of $n-1$ 0s followed

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

by a 1.

n	$c_U(n)$
1	1
2	01
3	001
4	0001
5	00001
⋮	
45	000000000000000000000000000000000000000000001

The unary code has length $l_U(n) = n$.

The unary code is the optimal code for integers if the probability distribution over n is $p_U(n) = 2^{-n}$.

Self-delimiting codes

We can use a unary code to encode the *length* of the binary encoding of n and make a self-delimiting code:

Code C_α. The length of the standard binary representation is a positive integer $l_b(n)$. We send the unary code for this length, followed by the headless binary representation of n.

$$c_\alpha(n) = c_U[l_b(n)]c_B(n) \tag{7.17}$$

The following table shows the codes for some integers. The overlining indicates the division of each string into the parts $c_U[l_b(n)]$ and $c_B(n)$.

n	$c_b(n)$	$l_b(n)$	$c_\alpha(n) = c_U[l_b(n)]c_B(n)$
1	1	1	$\overline{1}$
2	10	2	$\overline{01}0$
3	11	2	$\overline{01}1$
4	100	3	$\overline{001}00$
5	101	3	$\overline{001}01$
6	110	3	$\overline{001}10$
⋮			
45	101101	6	$\overline{000001}01101$

We might equivalently view $c_\alpha(n)$ as consisting of a string of $(l_b(n) - 1)$ zeroes followed by the standard binary representation of n, $c_b(n)$.

The codeword $c_\alpha(n)$ has length $l_\alpha(n) = 2l_b(n) - 1$.

The implicit probability distribution over n for the code C_α is separable into the product of a probability distribution over the length l,

$$P(l) = 2^{-l}, \tag{7.18}$$

and a uniform distribution over integers having that length,

$$P(n|l) = \begin{cases} 2^{-l+1} & l_b(n) = l \\ 0 & \text{otherwise} \end{cases}. \tag{7.19}$$

Now, for the above code, the header always occupies the same number of bits as the standard binary representation of the integer (give or take one). If we are expecting to encounter large integers (large files) then this representation seems suboptimal, since it leads to all files occupying a size that is double their original uncoded size. Instead of using the unary code to encode the length $l_b(n)$, we could use C_α.

Code C_β. The length of the standard binary representation is a positive integer $l_b(n)$. We send this length using C_α, followed by the headless binary representation of n.

$$c_\beta(n) = c_\alpha[l_b(n)]c_B(n) \tag{7.20}$$

n	$c_b(n)$	$l_b(n)$	$c_\beta(n) = c_\alpha[l_b(n)]c_B(n)$
1	1	1	$\overline{1}$
2	10	2	$\overline{10}0$
3	11	2	$\overline{10}1$
4	100	3	$\overline{11}00$
5	101	3	$\overline{11}01$
6	110	3	$\overline{11}10$
⋮			
45	101101	6	$\overline{00110}01101$

Iterating this procedure, we can define a sequence of codes.

Code C_γ.

$$c_\gamma(n) = c_\beta[l_b(n)]c_B(n) \tag{7.21}$$

n	$c_b(n)$	$l_b(n)$	$c_\gamma(n) = c_\beta[l_b(n)]c_B(n)$
1	1	1	$\overline{1}$
2	10	2	$\overline{100}0$
3	11	2	$\overline{100}1$
4	100	3	$\overline{101}00$
5	101	3	$\overline{101}01$
6	110	3	$\overline{101}10$
⋮			
45	101101	6	$\overline{01110}01101$

Code C_δ.

$$c_\delta(n) = c_\gamma[l_b(n)]c_B(n) \tag{7.22}$$

Codes with end-of-file symbols

We can also make byte-based representations. (Let's use the term byte flexibly here, to denote any fixed length string of bits, not just a string of length 8 bits.) If we encode the number in some base, for example decimal, then we can represent each digit in a byte. In order to represent a digit from 0 to 9 in a byte we need four bits. Because $2^4 = 16$, this leaves 6 extra four-bit symbols, {1010, 1011, 1100, 1101, 1110, 1111}. which correspond to no decimal digit.

We can use these as end-of-file symbols to indicate the end of our positive integer. Such a code can also code the integer zero, which we have not been providing a code for up till now.

Clearly it is redundant to have more than one end-of-file symbol, so a more efficient code would encode the integer into base 15, and use just the sixteenth symbol, `1111`, as the punctuation character.

This gives a code which is almost complete. (Recall that a code's being 'complete' means that it satisfies the Kraft inequality with equality.) The code's remaining inefficiency is that it provides the ability to encode an empty string, which was not required.

Generalizing this idea, we can make similar byte-based codes for integers in bases 3 and 7, and in any base of the form $2^n - 1$.

Exercise 7.10: Consider the implicit probability distribution over integers corresponding to the code with EOF character.

(a) If the code has eight-bit blocks (*i.e.*, the integer is coded in base 255), what is the mean length in bits of the integer, under the implicit distribution?

(b) If one wishes to encode binary files of expected size about one hundred Kbytes using a code with an EOF character, what is the optimal block size?

Encoding a tiny file

To illustrate the codes we have discussed, let us now encode a small file consisting of just 14 characters,

`Claude Shannon`,

using each code.

- If we map the ASCII characters onto seven-bit symbols (*e.g.*, in decimal, `C`=67, `l`=108, etc.), this 14 character file corresponds to the integer

$$n = 167987786364950891085602469870 \text{ (decimal)}.$$

- The unary code for n consists of this many (less one) zeroes, followed by a one. If all the oceans were turned into ink, and if we wrote a hundred bits with every cubic millimeter, there would be roughly enough ink to write $c_U(n)$.

- The standard binary representation of n is the length 98 sequence of bits:

$c_b(n)$ = `10000111101100110000111101011100100110010101000001010011110`
`10001100001110111011011101101111111101110`.

- The self-delimiting representations of n are:

$c_\alpha(n)$ = $\overline{\texttt{00000000000000000000000000000000000000000000000000000000000000}}$
$\overline{\texttt{0000000000000000000000000000000000000000}}$
$\overline{\texttt{1}}$`0000111101100110000111101011100100110010101000001010011110`
`10001100001110111011011101101111111101110`.

$c_\beta(n)$ = 000000
1$\overline{100010}$
000011110110011000011110101110010011001010100000101001111 0
100011000011101110110111011011111101110.

$c_\gamma(n)$ = $\overline{00}$111
$\overline{100010}$
000011110110011000011110101110010011001010100000101001111 0
100011000011101110110111011011111101110.

$c_\delta(n)$ = 01$\overline{1}$
11
$\overline{100010}$
000011110110011000011110101110010011001010100000101001111 0
100011000011101110110111011011111101110.

- Byte based, base 3.

  010010100110101010011010010110000010101001010101010110001010010
  110011000010000010010100101000010101010010100101001100001100011

- base 7

  011000101000101000100011100110010100001100000110011001101100000
  100000100000010101000011001011100100010100111

- base 15

  100111101010010000100110010000101011100011101011101000001110111
  000111011101010101110011111101110100001111

Comparing the codes

One could answer the question 'which of two codes is superior?' by a single sentence of the form "For $N > K$, code 1 is superior, for $N < K$, code 2 is superior" but I contend that such an answer misses the point: any complete code corresponds to a prior for which it is optimal; you should not say that any other code is superior to it. Other codes are just optimal for other priors. These implicit priors should be thought about so as to achieve the best code for one's application.

Notice that one cannot, for free, switch from one code to another, choosing whichever is shorter. If one wishes to do this, then it is necessary to lengthen the message in some way that indicates which of the two codes is being used. If this is done by a single leading bit, it will be found that the resulting code is suboptimal because it fails the Kraft equality, as was discussed in exercise 5.27 (p. 104).

Another way to compare codes for integers is to consider a sequence of probability distributions, such as monotonic probability distributions over $n \geq$

n	$c_\omega(n)$	n	$c_\omega(n)$
1	0	31	10100111110
2	100	32	101011000000
3	110	45	101011011010
4	101000	63	101011111110
5	101010	64	1011010000000
6	101100	127	1011011111110
7	101110	128	10111100000000
8	1110000	255	10111111111110
9	1110010	256	1110001000000000
10	1110100	365	1110001011011010
11	1110110	511	1110001111111110
12	1111000	512	11100110000000000
13	1111010	719	11100110110011110
14	1111100	1023	11100111111111110
15	1111110	1024	111010100000000000
16	10100100000	1025	111010100000000010

Table 7.1. Elias's 'universal' code for integers. Examples from 1 to 1024.

1, and rank the codes as to how well they encode *any* of these distributions. A code is called a 'universal' code if for any distribution in a given class, it encodes into an average length that is within some factor of the ideal average length.

Let me say this again. We are meeting an alternative world view — rather than figuring out a good prior over integers, as advocated above, many theorists have studied the problem of creating codes that, while not perfectly matched to all priors, are reasonably good codes for *any* priors in a broad class. Here the class of priors conventionally considered is the set of priors that (a) assign a monotonically decreasing probability over integers and (b) have finite entropy.

Several of the codes we have discussed above are universal. Another code which elegantly transcends the sequence of self-delimiting codes is Elias's 'universal code for integers' (Elias, 1975), which effectively chooses from all the codes $C_\alpha, C_\beta, \ldots$. It works by sending a sequence of messages each of which encodes the length of the next message, and indicates whether that message is the final integer (in its standard binary representation) or not by a single bit. Because a length is a positive integer and all positive integers begin with '1', all the leading 1's can be omitted. Table 7.1 shows the resulting codes.

The encoder of C_ω operates on the integer n as follows. Note that the encoding is generated from right to left.

Write '0'
Loop {
  If $\lfloor \log n \rfloor = 0$ halt
  Prepend $c_b(n)$ to the written string
  n:=$\lfloor \log n \rfloor$
}

The Elias code has been used to number the pages of this book.

The Elias code is not actually the best code for very large integers. A code that has shorter codelengths asymptotically (*e.g.*, for $n > 2^{100}$) uses the same idea but first encodes the number of levels of recursion that the encoder will go through, using any convenient prefix code for integers, for example C_ω; then the encoder does not need to transmit all those initial 1's and the final zero. Maybe this code should be called C_{ω^ω}? I would need to check with John Conway whether there is a connection between these codes and transcendental numbers.

Solutions

Solution to exercise 7.10 (p. 149): The EOF symbol in a code that represents the integer in some base q corresponds to a belief that there is a probability of $(1/(q+1))$ that the current character is the last character of the number. Thus the prior to which this code is matched puts an exponential prior distribution over the length of the integer.

...

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part II

Noisy Channel Coding

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

8

Correlated random variables

In the last three chapters on data compression we concentrated on random vectors $\mathbf{x}$ coming from an extremely simple probability distribution, namely the separable distribution in which each component x_n is independent of the others.

In this chapter, we consider joint ensembles that have correlations, and, carrying on from chapter 2, we look at further definitions and exercises to do with the information content of correlated variables. This material has three motivations. First, data from the real world has interesting correlations in it, so to do data compression well, we need to know how to work with models that include correlations. Second, a noisy channel with input x and output y defines a joint ensemble in which x and y are correlated — if they were independent, it would be impossible to communicate over the channel — so communication over noisy channels is described in terms of the entropy of joint ensembles. And third, we will study a particular model of correlated variables, the Markov chain, because this model, and its accompanying computational methods, will be used several times in other parts of this book. [This bit ain't written yet.]

8.1 More about entropy

This section picks up with definitions and exercises to do with entropy, carrying on from section 2.4.

The joint entropy of X, Y is:

$$H(X,Y) = \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x,y)}. \tag{8.1}$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \;\; \text{iff} \;\; P(x,y) = P(x)P(y). \tag{8.2}$$

The conditional entropy of X **given** $y = b_k$ is the entropy of the probability distribution $P(x|y{=}b_k)$.

$$H(X|y{=}b_k) \equiv \sum_{x \in \mathcal{A}_X} P(x|y{=}b_k) \log \frac{1}{P(x|y{=}b_k)}. \tag{8.3}$$

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

The conditional entropy of X given Y is the average over y of the conditional entropy of X given y.

$$H(X|Y) \equiv \sum_{y\in\mathcal{A}_Y} P(y)\left[\sum_{x\in\mathcal{A}_X} P(x|y)\log\frac{1}{P(x|y)}\right] = \sum_{xy\in\mathcal{A}_X\mathcal{A}_Y} P(x,y)\log\frac{1}{P(x|y)}. \quad (8.4)$$

This measures the average uncertainty that remains about x when y is known.

The marginal entropy of X is another name for the entropy of X, $H(X)$, used to contrast it with the conditional entropies listed above.

Chain rule for information content. From the product rule for probabilities, equation (2.6), we obtain:

$$\log\frac{1}{P(x,y)} = \log\frac{1}{P(x)} + \log\frac{1}{P(y|x)} \quad (8.5)$$

so

$$h(x,y) = h(x) + h(y|x). \quad (8.6)$$

In words, this says that the information content of x and y is the information content of x plus the information content of y given x.

Chain rule for entropy. The joint entropy, conditional entropy and marginal entropy are related by:

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y). \quad (8.7)$$

In words, this says that the uncertainty of X and Y is the uncertainty of X plus the uncertainty of Y given X.

The mutual information between X and Y is

$$I(X;Y) \equiv H(X) - H(X|Y), \quad (8.8)$$

and satisfies $I(X;Y) = I(Y;X)$, and $I(X;Y) \geq 0$. It measures the average reduction in uncertainty about x that results from learning the value of y, **or vice versa**, the average amount of information that y conveys about x.

The conditional mutual information between X and Y given $z = c_k$ is the mutual information between the random variables X and Y in the joint ensemble $P(x,y|z=c_k)$,

$$I(X;Y|z=c_k) = H(X|z=c_k) - H(X|Y,z=c_k). \quad (8.9)$$

The conditional mutual information between X and Y given Z is the average over z of the above conditional mutual information.

$$I(X;Y|Z) = H(X|Z) - H(X|Y,Z). \quad (8.10)$$

No other 'three–term entropies' will be defined. For example, expressions such as $I(X;Y;Z)$ and $I(X|Y;Z)$ are illegal. But you may put conjunctions of arbitrary numbers of variables in each of the three spots in the expression $I(X;Y|Z)$ — for example, $I(A,B;C,D|E,F)$ is fine.

Figure 8.1 shows how the total entropy $H(X,Y)$ of a joint ensemble can be broken down.

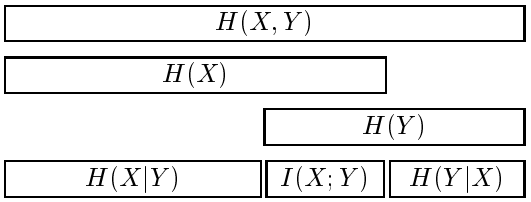


Figure 8.1. The relationship between joint information, marginal entropy, conditional entropy and mutual entropy.

8.2 Exercises

Exercise 8.1:^{A2} Prove the assertion that entropy is additive for independent random variables (equation (8.2)).

Exercise 8.2:^{B2} Consider three independent random variables u, v, w with entropies H_u, H_v, H_w. Let $X \equiv (U,V)$ and $Y \equiv (V,W)$. What is $H(X,Y)$? What is $H(X|Y)$? What is $I(X;Y)$?

Exercise 8.3:^{B3} Referring to the definitions of conditional entropy (8.3–8.4), confirm (with an example) that it is possible for $H(X|y{=}b_k)$ to exceed $H(X)$, but that the average, $H(X|Y)$ is less than $H(X)$. So data is helpful — it doesn't increase uncertainty, on average.

Exercise 8.4:^{B2} Prove the chain rule for entropy, equation (8.7). [$H(X,Y) = H(X) + H(Y|X)$].

Exercise 8.5:^{A2} Prove that the mutual information $I(X;Y) \equiv H(X) - H(X|Y)$ satisfies $I(X;Y) = I(Y;X)$ and $I(X;Y) \geq 0$.

[Hint: see exercise 2.21 (p. 39) and note that

$$I(X;Y) = D_{\mathrm{KL}}(P(x,y)||P(x)P(y)).] \tag{8.11}$$

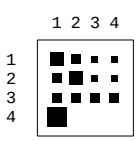
Exercise 8.6:^{C4} The 'entropy distance' between two random variables can be defined to be the difference between their joint entropy and their mutual information:

$$D_H(X,Y) \equiv H(X,Y) - I(X;Y). \tag{8.12}$$

Prove that the entropy distance satisfies the axioms for a distance — $D_H(X,Y) \geq 0$, $D_H(X,X) = 0$, $D_H(X,Y){=}D_H(Y,X)$, and $D_H(X,Z) \leq D_H(X,Y) + D_H(Y,Z)$. [Incidentally, we are unlikely to see $D_H(X,Y)$ again but it is a good function on which to practice inequality-proving.]

Exercise 8.7:^{A2} A joint ensemble XY has the following joint distribution.

$P(x,y)$		x			
		1	2	3	4
y	1	1/8	1/16	1/32	1/32
	2	1/16	1/8	1/32	1/32
	3	1/16	1/16	1/16	1/16
	4	1/4	0	0	0



 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

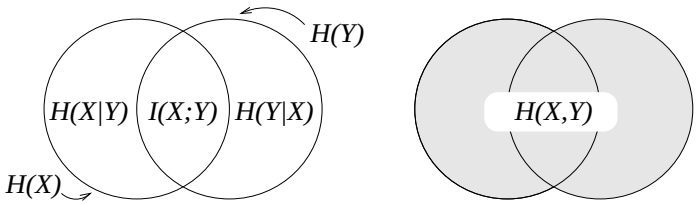


Figure 8.2. A misleading representation of entropies (contrast with figure 8.1).

What is the joint entropy $H(X,Y)$? What are the marginal entropies $H(X)$ and $H(Y)$? For each value of y, what is the conditional entropy $H(X|y)$? What is the conditional entropy $H(X|Y)$? What is the conditional entropy of Y given X? What is the mutual information between X and Y?

Solutions on p. 161

Exercise 8.8:^{A2} Consider the ensemble XYZ in which $\mathcal{A}_X = \mathcal{A}_Y = \mathcal{A}_Z = \{0,1\}$, x and y are independent with $\mathcal{P}_X = \{p, 1-p\}$ and $\mathcal{P}_Y = \{q, 1-q\}$ and

$$z = x + y \bmod 2. \tag{8.13}$$

(a) If $q = 1/2$, what is $\mathcal{P}_Z$? What is $I(Z;X)$?

(b) For general p and q, what is $\mathcal{P}_Z$? What is $I(Z;X)$?

Three term entropies

Exercise 8.9:^{C3} Many texts draw figure 8.1 in the form of a Venn diagram (figure 8.2). Discuss why this diagram is a misleading representation of entropies. Hint: consider the three-variable ensemble XYZ in which $x \in \{0,1\}$ and $y \in \{0,1\}$ are independent binary variables and $z \in \{0,1\}$ is defined to be $z = x + y \bmod 2$.

The data–processing theorem

The data processing theorem states that data processing can only destroy information.

Exercise 8.10:^{A3} Prove this theorem by considering an ensemble WDR in which w is the state of the world, d is data gathered, and r is the processed data, so that these three variables form a *Markov chain*

$$w \to d \to r, \tag{8.14}$$

that is, the probability $P(w,d,r)$ can be written as

$$P(w,d,r) = P(w)P(d|w)P(r|d). \tag{8.15}$$

Show that the average information that R conveys about W, $I(W;R)$, is less than or equal to the average information that D conveys about W, $I(W;D)$.

This theorem is as much a caution about our definition of 'information' as it is a caution about data processing!

Solution to Chapter 8's exercises

Solution to exercise 8.5 (p. 156): Symmetry of mutual information:

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) && (8.16)\\ &= \sum_x P(x) \log \frac{1}{P(x)} - \sum_{xy} P(x,y) \log \frac{1}{P(x|y)} && (8.17)\\ &= \sum_{xy} P(x,y) \log \frac{P(x|y)}{P(x)} && (8.18)\\ &= \sum_{xy} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} && (8.19) \end{aligned}$$

This expression is symmetric in x and y so

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \quad (8.20)$$

We can prove that mutual information is positive two ways. One is to continue from

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \quad (8.21)$$

which is a relative entropy and use Gibbs' inequality (to be proved imminently), which asserts that relative entropies are ≥ 0, with equality in this case only if $P(x,y) = P(x)P(y)$, that is if X and Y are independent.

The other is to use Jensen's inequality on

$$-\sum_{x,y} P(x,y) \log \frac{P(x)P(y)}{P(x,y)} \geq -\log \sum_{x,y} \frac{P(x,y)}{P(x,y)} P(x)P(y) = \log 1 = 0. \quad (8.22)$$

Solution to exercise 8.1 (p. 156): Let $P(x,y) = P(x)P(y)$. Then

$$\begin{aligned} H(X,Y) &= \sum_{xy} P(x)P(y) \log \frac{1}{P(x)P(y)} && (8.23)\\ &= \sum_{xy} P(x)P(y) \log \frac{1}{P(x)} + \sum_{xy} P(x)P(y) \log \frac{1}{P(y)} && (8.24)\\ &= \sum_{x} P(x) \log \frac{1}{P(x)} + \sum_{y} P(y) \log \frac{1}{P(y)} && (8.25)\\ &= H(X) + H(Y). && (8.26) \end{aligned}$$

Solution to exercise 8.2 (p. 156):

$$H(X,Y) = H(U,V,V,W) = H(U,V,W) = H_u + H_v + H_w. \quad (8.27)$$

$$H(X|Y) = H_u. \quad (8.28)$$

$$I(X;Y) = H_v. \quad (8.29)$$

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solution to exercise 8.3 (p. 156): See exercise 8.7 (p. 156) for an example where $H(X|y)$ exceeds $H(X)$ (set $y = 3$).

We can prove the inequality $H(X|Y) \leq H(X)$ by turning the expression into a relative entropy (using Bayes' theorem) and invoking Gibbs' inequality (exercise 2.21 (p. 39)):

$$\begin{aligned} H(X|Y) &\equiv \sum_{y \in \mathcal{A}_Y} P(y) \left[\sum_{x \in \mathcal{A}_X} P(x|y) \log \frac{1}{P(x|y)} \right] \\ &= \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x|y)} \qquad (8.30) \\ &= \sum_{xy} P(x) P(y|x) \log \frac{P(y)}{P(y|x)P(x)} \qquad (8.31) \\ &= \sum_{x} P(x) \log \frac{1}{P(x)} + \sum_{x} P(x) \sum_{y} P(y|x) \log \frac{P(y)}{P(y|x)}. \qquad (8.32) \\ & \qquad (8.33) \end{aligned}$$

The last expression is a sum of relative entropies between the distributions $P(y|x)$ and $P(y)$. So

$$H(X|Y) \leq \sum_{x} H(X) + 0, \qquad (8.34)$$

with equality only if $P(y|x) = P(y)$ for all x and y (that is, only if X and Y are independent).

Solution to exercise 8.4 (p. 156): The chain rule for entropy follows from the decomposition of a joint probability:

$$\begin{aligned} H(X,Y) &= \sum_{xy} P(x,y) \log \frac{1}{P(x,y)} \qquad (8.35) \\ &= \sum_{xy} P(x) P(y|x) \left[\log \frac{1}{P(x)} + \log \frac{1}{P(y|x)} \right] \qquad (8.36) \\ &= \sum_{x} P(x) \log \frac{1}{P(x)} + \sum_{x} P(x) \sum_{y} P(y|x) \log \frac{1}{P(y|x)} \qquad (8.37) \\ &= H(X) + H(Y|X). \qquad (8.38) \end{aligned}$$

Solution to exercise 8.6 (p. 156): The entropy distance:

$$D_H(X,Y) \equiv H(X,Y) - I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x)P(y)}{P(x,y)^2}. \qquad (8.39)$$

is fairly easily shown to satisfy the first three axioms $D_H(X,Y) \geq 0$, $D_H(X,X) = 0$, $D_H(X,Y)$=$D_H(Y,X)$.

A proof that it obeys the triangle inequality is not so immediate. It helps to know in advance what the difference $D(X,Y) + D(Y,Z) - D(X,Z)$ should add up to; this is most easily seen by first making a picture in which the quantities $H(X), H(Y)$, and $H(Z)$ are represented by overlapping areas, c.f. figure 8.2 and exercise 8.9 (p. 157). Such a picture indicates that $D(X,Y) + D(Y,Z) - D(X,Z) = H(Y|X,Z) + I(X;Z|Y)$.

$$D(X,Y) + D(Y,Z) - D(X,Z)$$

$$= \sum_{x,y,z} P(x,y,z) \log \frac{P(x)P(y)P(y)P(z)P(xz)^2}{P(xy)^2 P(x)P(z)P(y,z)^2} \quad (8.40)$$

$$= 2 \sum_{x,y,z} P(x,y,z) \log \frac{P(x,z)P(x,z|y)}{P(x,y,z)P(x|y)P(z|y)} \quad (8.41)$$

$$= 2 \sum_{x,y,z} P(x,y,z) \left[\log \frac{1}{P(y|xz)} + \log \frac{P(x,z|y)}{P(x|y)P(z|y)} \right] \quad (8.42)$$

$$= 2 \sum_{x,z} P(x,z) \sum_{y} P(y|x,z) \log \frac{1}{P(y|x,z)} + \quad (8.43)$$

$$2 \sum_{y} P(y) \sum_{x,z} P(x,z|y) \log \frac{P(x,z|y)}{P(x|y)P(z|y)} \quad (8.44)$$

$$= 2 \sum_{x,z} P(x,z) H(Y|x,z) + 2 \sum_{y} P(y) I(X;Z|y). \quad (8.45)$$

$$= 2H(Y|X,Z) + 2I(X;Z|Y). \quad (8.46)$$

The quantity $I(X;Z|Y)$ is a conditional mutual information, which like a mutual information is positive. The other term $H(Y|X,Z)$ is also positive, so $D(X,Y) + D(Y,Z) - D(X,Z) \geq 0$.

Solution to exercise 8.7 (p. 156): See chapter 9.

Three term entropies

Solution to exercise 8.9 (p. 157): The depiction of entropies in terms of Venn diagrams is misleading for at least two reasons.

First, one is used to thinking of Venn diagrams as depicting sets; but what are the 'sets' $H(X)$ and $H(Y)$ depicted in figure 8.2, and what are the objects that are members of those sets? I think this diagram encourages the novice student to make inappropriate analogies. For example, some students imagine that the random outcome (x,y) might somehow correspond to a point in the diagram, and thus confuse entropies with probabilities.

Secondly, the depiction in terms of Venn diagrams encourages one to believe that all the areas correspond to positive quantities. In the special case of two random variables it is indeed true that $H(X|Y), I(X;Y)$ and $H(Y|X)$ are positive quantities. But as soon as we progress to three-variable ensembles, we obtain a diagram with positive-looking areas that may actually correspond to negative quantities. Figure 8.3 correctly shows relationships such as

$$H(X) + H(Z|X) + H(Y|X,Z) = H(X,Y,Z). \quad (8.47)$$

But it gives the misleading impression that the conditional mutual information $I(X;Y|Z)$ is *less than* the mutual information $I(X;Y)$. In fact the area labelled A can correspond to a *negative* quantity. Consider the joint ensemble (X,Y,Z) in which $x \in \{0,1\}$ and $y \in \{0,1\}$ are independent binary variables and $z \in \{0,1\}$ is defined to be $z = x + y \bmod 2$. Then clearly $H(X) = H(Y) = 1$ bit. Also $H(Z) = 1$ bit. And $H(Y|X) = H(Y) = 1$ since the two variables are independent. So the mutual information between X and Y is zero. $I(X;Y) = 0$. However, if z is observed, X and Y become correlated — knowing x, given z, tells you what y is — $y = z - x \bmod 2$. So $I(X;Y|Z) = 1$ bit. Thus the area labelled A must correspond to -1 bits for the figure to give the correct answers.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

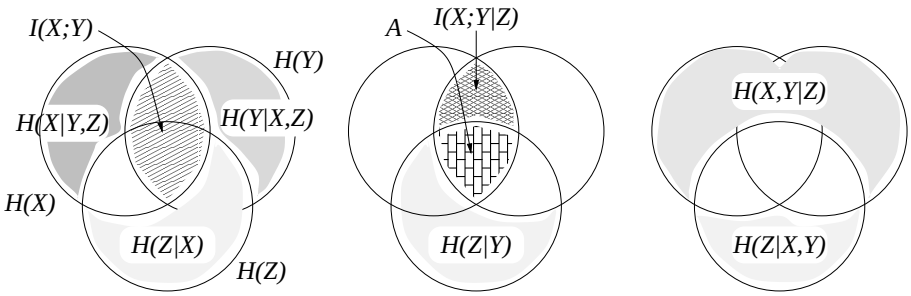


Figure 8.3. A misleading representation of entropies, continued.

The above example is not at all a capricious or exceptional illustration. The classic earthquake-burglar-alarm ensemble (exercise 21.7, p. 279) with earthquake $= X$, burglar $= Y$ and alarm $= Z$, is a perfect example of a situation in which $I(X;Y) = 0$ but $I(X;Y|Z) > 0$.

The Venn diagram representation is therefore valid only if one is aware that positive areas may represent negative quantities. With this proviso kept in mind, the interpretation of entropies in terms of sets can be helpful, as pointed out by Yeung (1991).

Solution to exercise 8.10 (p. 157): For any joint ensemble XYZ, the following chain rule for mutual information holds.

$$I(X;Y,Z) = I(X;Y) + I(X;Z|Y). \tag{8.48}$$

Now, in the case $w \rightarrow d \rightarrow r$, w and r are independent given d, so $I(W;R|D) = 0$. Using the chain rule twice, we have:

$$I(W;D,R) = I(W;D) \tag{8.49}$$

and

$$I(W;D,R) = I(W;R) + I(W;D|R), \tag{8.50}$$

so

$$I(W;R) - I(W;D) \leq 0. \tag{8.51}$$

Solution to exercise 8.8 (p. 157):

$$z = x + y \bmod 2. \tag{8.52}$$

(a) If $q = 1/2$, $\mathcal{P}_Z = \{1/2, 1/2\}$ and $I(Z;X) = H(Z) - H(Z|X) = 1 - 1 = 0$.

(b) For general q and p, $\mathcal{P}_Z = \{pq + (1-p)(1-q), p(1-q) + q(1-p)\}$. The mutual information is $I(Z;X) = H(Z) - H(Z|X) = H_2(pq + (1-p)(1-q)) - H_2(q)$.

About Chapter 9

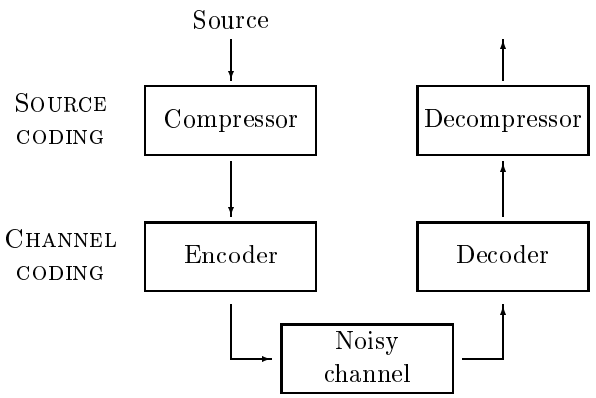
Before reading chapter 9, you should have read chapter 1 and worked on exercises 8.3–8.5 (pp. 156–156), 2.21–8.7 (pp. 39–156), and 8.8 (p. 157).

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

9

Communication over a noisy channel

9.1 The big picture



In chapters 4–6, we have discussed source coding with block codes, symbol codes and stream codes. We implicitly assumed that the channel from the compressor to the decompressor was noise-free. Real channels are noisy. We will now spend two chapters on the subject of noisy channel coding — the fundamental possibilities and limitations of error-free communication through a noisy channel. The aim of channel coding is to make the noisy channel behave like a noiseless channel. We will assume that the data to be transmitted has been through a good compressor, so the bit stream has no obvious redundancy. The channel code will put back into the transmission, redundancy of a special sort, designed to make the noisy received signal decodeable.

Suppose we transmit 1000 bits per second with $p_0 = p_1 = \frac{1}{2}$ over a noisy channel that flips bits with probability $f = 0.1$. What is the rate of transmission of information? We might guess that the rate is 900 bits per second by subtracting the expected number of errors per second. But this is not correct, because the recipient does not know where the errors occurred. Consider the case where the noise is so great that the received symbols are independent of the transmitted symbols. This corresponds to a noise level of $f = 0.5$, since half of the received symbols are correct due to chance alone. But when $f = 0.5$, no information is transmitted at all.

Given what we have learnt about entropy, it seems reasonable that a measure of the information transmitted is given by the mutual information between

the source and the received signal, that is, the entropy of the source minus the *conditional entropy* of the source given the received signal.

We will now review the definition of conditional entropy and mutual information. Then we will examine whether it is possible to use such a noisy channel to communicate *reliably*. We will show that for any channel Q there is a non-zero rate, the capacity $C(Q)$, up to which information can be sent with arbitrarily small probability of error.

9.2 Review of probability and information

As an example, we take the joint distribution XY from exercise 8.7 (p. 156). The marginal distributions $P(x)$ and $P(y)$ are shown in the margins.

$P(x,y)$		x				$P(y)$
		1	2	3	4	
	1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
y	2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
	3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{4}$
	4	$\frac{1}{4}$	0	0	0	$\frac{1}{4}$
$P(x)$		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	

The joint entropy is $H(X,Y) = 27/8$ bits. The marginal entropies are $H(X) = 7/4$ bits and $H(Y) = 2$ bits.

We can compute the conditional distribution of x for each value of y, and the entropy of each of those conditional distributions:

$P(x\|y)$		x				$H(X\|y)$/bits
		1	2	3	4	
	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{7}{4}$
y	2	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{7}{4}$
	3	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	2
	4	1	0	0	0	0
						$H(X\|Y) = \frac{11}{8}$

Note that whereas $H(X|y{=}4) = 0$ is less than $H(X)$, $H(X|y{=}3)$ is greater than $H(X)$. So in some cases, learning y can *increase* our uncertainty about x. Note also that although $P(x|y{=}2)$ is a different distribution from $P(x)$, the conditional entropy $H(X|y{=}2)$ is equal to $H(X)$. So learning that y is 2 changes our knowledge about x but does not reduce the uncertainty of x, as measured by the entropy. On average though, learning y does convey information about x, since $H(X|Y) < H(X)$.

One may also evaluate $H(Y|X) = 13/8$ bits. The mutual information is $I(X;Y) = H(X) - H(X|Y) = 3/8$ bits.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

9.3 Noisy channels

A discrete memoryless channel Q is characterized by an input alphabet $\mathcal{A}_X$, an output alphabet $\mathcal{A}_Y$, and a set of conditional probability distributions $P(y|x)$, one for each $x \in \mathcal{A}_X$.

These *transition probabilities* may be written in a matrix

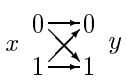
$$Q_{j|i} = P(y{=}b_j|x{=}a_i). \tag{9.1}$$

I usually orient this matrix with the output variable j indexing the rows and the input variable i indexing the columns, so that each column of $\mathbf{Q}$ is a probability vector. With this convention, we can obtain the probability of the output, $\mathbf{p}_Y$, from a probability distribution over the input, $\mathbf{p}_X$, by right-multiplication:

$$\mathbf{p}_Y = \mathbf{Q}\mathbf{p}_X. \tag{9.2}$$

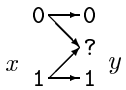
Some useful model channels are:

Binary symmetric channel. $\mathcal{A}_X{=}\{0, 1\}$. $\mathcal{A}_Y{=}\{0, 1\}$.

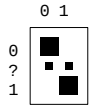


$$P(y{=}0|x{=}0){=}1-f; \quad P(y{=}0|x{=}1){=}f;$$
$$P(y{=}1|x{=}0){=}f; \quad P(y{=}1|x{=}1){=}1-f.$$

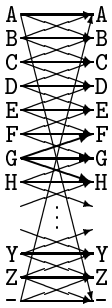
Binary erasure channel. $\mathcal{A}_X{=}\{0, 1\}$. $\mathcal{A}_Y{=}\{0, ?, 1\}$.



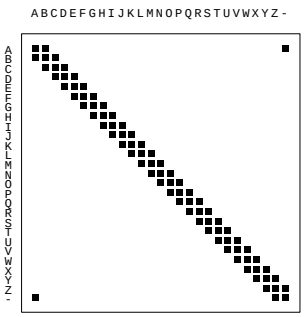
$$P(y{=}0|x{=}0){=}1-f; \quad P(y{=}0|x{=}1){=}0;$$
$$P(y{=}?|x{=}0){=}f; \quad P(y{=}?|x{=}1){=}f;$$
$$P(y{=}1|x{=}0){=}0; \quad P(y{=}1|x{=}1){=}1-f.$$



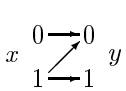
Noisy typewriter. $\mathcal{A}_X = \mathcal{A}_Y =$ the 27 letters {A, B, ..., Z, -}. The letters are arranged in a circle, and when the typist attempts to type B, what comes out is either A, B or C, with probability 1/3 each; when the input is C, the output is B, C or D; and so forth, with the final letter '-' adjacent to the first letter A.



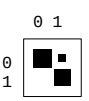
$$\vdots$$
$$P(y{=}\mathtt{F}|x{=}\mathtt{G}){=}1/3;$$
$$P(y{=}\mathtt{G}|x{=}\mathtt{G}){=}1/3;$$
$$P(y{=}\mathtt{H}|x{=}\mathtt{G}){=}1/3;$$
$$\vdots$$



Z channel. $\mathcal{A}_X{=}\{0, 1\}$. $\mathcal{A}_Y{=}\{0, 1\}$.



$$P(y{=}0|x{=}0){=}1; \quad P(y{=}0|x{=}1){=}f;$$
$$P(y{=}1|x{=}0){=}0; \quad P(y{=}1|x{=}1){=}1-f.$$



9.4 Inferring the input given the output

If we assume that the input x comes from an ensemble X, then we obtain a joint ensemble XY in which the random variables x and y have the joint distribution:

$$P(x, y) = P(y|x)P(x). \tag{9.3}$$

Now if we receive a particular symbol y, what was the input symbol x? We typically won't know for certain. We can write down the posterior distribution of the input using Bayes' theorem:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} = \frac{P(y|x)P(x)}{\sum_{x'} P(y|x')P(x')} \tag{9.4}$$

Example 9.1: Consider a binary symmetric channel with probability of error f=0.15. Let the input ensemble be $\mathcal{P}_X : \{p_0$=0.9$, p_1$=0.1$\}$. Assume we observe y=1.

$$\begin{aligned} P(x\text{=}1|y\text{=}1) &= \frac{P(y\text{=}1|x\text{=}1)P(x\text{=}1)}{\sum_{x'} P(y|x')P(x')} \\ &= \frac{0.85 \times 0.1}{0.85 \times 0.1 + 0.15 \times 0.9} \\ &= \frac{0.085}{0.22} = 0.39. \end{aligned} \tag{9.5}$$

Thus 'x=1' is still less probable than 'x=0', although it is not as improbable as it was before.

Exercise 9.2: Now assume we observe y=0. Compute the probability of x=1 given y=0.

Example 9.3: Consider a Z channel with probability of error f=0.15. Let the input ensemble be $\mathcal{P}_X : \{p_0$=0.9$, p_1$=0.1$\}$. Assume we observe y=1.

$$\begin{aligned} P(x\text{=}1|y\text{=}1) &= \frac{0.85 \times 0.1}{0.85 \times 0.1 + 0 \times 0.9} \\ &= \frac{0.085}{0.085} = 1.0. \end{aligned} \tag{9.6}$$

So given the output y=1 we become certain of the input.

Exercise 9.4: Alternatively, assume we observe y=0. Compute $P(x$=1$|y$=0$)$.

9.5 Information conveyed by a channel

We now consider how much information can be communicated through a channel. In operational terms, we are interested in finding ways of using the channel such that all the bits that are communicated are recovered with negligible probability of error. In mathematical terms, assuming a particular input ensemble X, we can measure how much information the output conveys about the input by the mutual information:

$$I(X;Y) \equiv H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{9.7}$$

Our aim is to establish the connection between these two ideas. Let us evaluate $I(X;Y)$ for some of the channels above.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Hint for computing mutual information

We will tend to think of $I(X;Y)$ as $H(X) - H(X|Y)$, *i.e.*, how much the uncertainty of the input X is reduced when we look at the output Y. But for computational purposes it is often handy to evaluate $H(Y) - H(Y|X)$ instead.

Example 9.5: Consider the binary symmetric channel again, with f=0.15 and $\mathcal{P}_X : \{p_0$=0.9$, p_1$=0.1$\}$. We already evaluated the marginal probabilities $P(y)$ implicitly above: $P(y$=0$)$ = 0.78; $P(y$=1$)$ = 0.22. The mutual information is:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(0.22) - H_2(0.15) \\ &= 0.76 - 0.61 \;=\; 0.15 \text{ bits.} \end{aligned} \tag{9.8}$$

This may be contrasted with the entropy of the source $H(X) = H_2(0.1) = 0.47$ bits.

Note: here we have made use of the binary entropy function $H_2(p) \equiv H(p, 1-p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}$.

Example 9.6: And now the Z channel, with $\mathcal{P}_X$ as above. $P(y$=1$)$=0.085.

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(0.085) - [0.9 H_2(0) + 0.1 H_2(0.15)] \\ &= 0.42 - (0.1 \times 0.61) = 0.36 \text{ bits.} \end{aligned} \tag{9.9}$$

The entropy of the source, as above, is $H(X) = 0.47$ bits. Notice that the mutual information $I(X;Y)$ for the Z channel is bigger than the mutual information for the binary symmetric channel with the same f. This fits with our intuition that the Z channel is a more reliable channel.

Exercise 9.7: Compute the mutual information between X and Y for the binary symmetric channel with f=0.15 when the input distribution is $\mathcal{P}_X : \{p_0$=0.5$, p_1$=0.5$\}$.

Exercise 9.8: Compute the mutual information between X and Y for the Z channel with $f = 0.15$ when the input distribution is $\mathcal{P}_X : \{p_0$=0.5$, p_1$=0.5$\}$.

Maximizing the mutual information

We have observed in the above examples that the mutual information between the input and the output depends on the chosen input ensemble.

Let us assume that we wish to maximize the mutual information conveyed by the channel by choosing the best possible input ensemble. We define the *capacity* of the channel to be its maximum mutual information.

The Capacity of a channel Q is:

$$C(Q) = \max_{\mathcal{P}_X} I(X;Y). \tag{9.10}$$

The distribution $\mathcal{P}_X$ that achieves the maximum is called the *optimal input distribution*, denoted by $\mathcal{P}_X^*$. [There may be multiple optimal input distributions achieving the same value of $I(X;Y)$.]

In chapter 10 we will show that the capacity does indeed measure the maximum amount of error-free information that can be transmitted over the channel per unit time.

Example 9.9: Consider the binary symmetric channel with f=0.15. Above we considered $\mathcal{P}_X : \{p_0$=0.9$, p_1$=0.1$\}$, and found $I(X;Y) = 0.15$ bits. How much better can we do? By symmetry, the optimal input distribution is $\{0.5, 0.5\}$ and the capacity is

$$C(Q_{\rm BSC}) = H_2(0.5) - H_2(0.15) = 1.0 - 0.61 = 0.39 \,{\rm bits}. \tag{9.11}$$

We'll justify the symmetry argument later. If there's any doubt about the symmetry argument, we can always resort to explicit maximization of the mutual information $I(X;Y)$,

$$I(X;Y) = H_2((1-f)p_1 + (1-p_1)f) - H_2(f) \text{ (figure 9.1)} . \tag{9.12}$$

$I(X;Y)$

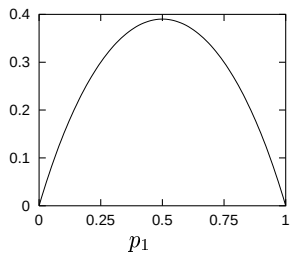


Figure 9.1. The mutual information $I(X;Y)$ for a binary symmetric channel with $f = 0.15$ as a function of the input distribution.

Example 9.10: The noisy typewriter. The optimal input distribution is a uniform distribution over x, and gives $C = \log_2 9$ bits.

Example 9.11: Consider the Z channel with f=0.15. Identifying the optimal input distribution is not so straightforward. We evaluate $I(X;Y)$ explicitly for $\mathcal{P}_X = \{p_0, p_1\}$. First, we need to compute $P(y)$. The probability of y=1 is easiest to write down:

$$P(y{=}1) = p_1(1-f). \tag{9.13}$$

Then the mutual information is:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(p_1(1-f)) - (p_0 H_2(0) + p_1 H_2(f)) \\ &= H_2(p_1(1-f)) - p_1 H_2(f). \end{aligned} \tag{9.14}$$

This is a non-trivial function of p_1, shown in figure 9.2. It is maximized for $f = 0.15$ by $p_1^* = 0.445$. We find $C(Q_{\rm Z}) = 0.685$. Notice that the optimal input distribution is not $\{0.5, 0.5\}$. We can communicate slightly more information by using input symbol 0 more frequently than 1.

$I(X;Y)$

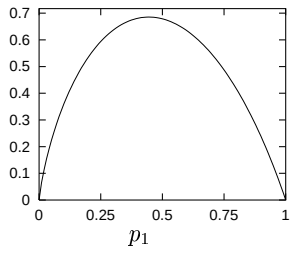


Figure 9.2. The mutual information $I(X;Y)$ for a Z channel with $f = 0.15$ as a function of the input distribution.

Solutions on p. 176

Exercise 9.12: What is the capacity of the binary symmetric channel for general f?

Exercise 9.13: Show that the capacity of the binary erasure channel with $f = 0.15$ is $C_{\rm BEC} = 0.85$. What is its capacity for general f? Comment.

9.6 The noisy channel coding theorem

It seems plausible that the 'capacity' we have defined may be a measure of information conveyed by a channel; what is not obvious, and what we will prove in the next chapter, is that the capacity indeed measures the rate at which bits can be communicated over the channel *with arbitrarily small probability of error.*

We make the following definitions.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

An (N, K) **block code** for a channel Q is a list of $S = 2^K$ codewords

$$\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(2^K)}\},$$

each of length N: $\mathbf{x}^{(s)} \in \mathcal{A}_X^N$. Using this code we can encode a signal $s \in \{1, 2, 3, \ldots, 2^K\}$ as $\mathbf{x}^{(s)}$. [The number of codewords S is an integer, but the number of bits specified by choosing a codeword, $K \equiv \log_2 S$, is not necessarily an integer.]

The *rate* of the code is $R = K/N$ bits.

[This definition holds for any channels, not only binary channels.]

A decoder for an (N, K) block code is a mapping from $\mathcal{A}_Y^N$ to $\hat{s} \in \{0, 1, 2, \ldots, 2^K\}$.

The extra symbol $\hat{s}$=0 can be used to indicate a 'failure'.

The probability of block error of a code and decoder, for a given channel, and for a given probability distribution over the encoded signal $P(s_{\text{in}})$, is:

$$p_{\text{B}} = \sum_{s_{\text{in}}} P(s_{\text{in}}) P(s_{\text{out}} \neq s_{\text{in}} | s_{\text{in}}) \tag{9.15}$$

The maximal probability of block error is

$$p_{\text{BM}} = \max_{s_{\text{in}}} P(s_{\text{out}} \neq s_{\text{in}} | s_{\text{in}}) \tag{9.16}$$

The optimal decoder for a channel code is the one that minimizes the probability of block error. It decodes an output $\mathbf{y}$ as the input m that has maximum posterior probability $P(m|\mathbf{y})$.

$$P(s|\mathbf{y}) = \frac{P(\mathbf{y}|s) P(s)}{\sum_{s'} P(\mathbf{y}|s') P(s')} \tag{9.17}$$

$$\hat{s}_{\text{optimal}} = \text{ argmax } P(s|\mathbf{y}). \tag{9.18}$$

A uniform prior distribution on m is usually assumed, in which case the optimal decoder is also the *maximum likelihood decoder*, *i.e.*, the decoder that maps an output $\mathbf{y}$ to the input s that has maximum *likelihood* $P(\mathbf{y}|s)$.

The probability of bit error p_b is defined assuming that the codeword number s is represented by a binary vector $\mathbf{s}$ of length K bits; it is the average probability that a bit of $\mathbf{s}_{\text{out}}$ is not equal to the corresponding bit of $\mathbf{s}_{\text{in}}$.

Shannon's noisy channel coding theorem (positive part). Associated with each discrete memoryless channel, there is a nonnegative number C (called the channel capacity) with the following property. For any $\epsilon > 0$ and $R < C$, for large enough N, there exists a block code of length N and rate $\geq R$ and a decoding algorithm, such that the maximal probability of block error is $< \epsilon$.

The negative part of the theorem. Error-free communication above capacity is impossible. It is not possible to achieve a rate R with probability of bit error smaller than

$$p_b(R) = H_2^{-1}(1 - C/R). \tag{9.19}$$

This is the equation of the curve separating achievable from non-achievable points (p_b, R).

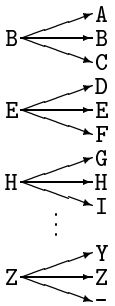
For the binary symmetric channel with cross-over probability f, the curve can be written:

$$R(p_b) = \frac{1 - H_2(f)}{1 - H_2(p_b)}. \tag{9.20}$$

Confirmation of the theorem for the noisy typewriter channel

In the case of the noisy typewriter, we can easily confirm the positive part of the theorem, because we can create a completely error–free communication strategy using a block code of length $N = 1$: we use only the letters `B`, `E`, `H`, ..., `Z`, *i.e.*, every third letter. These letters form a *non-confusable subset* of the input alphabet (see figure 9.3). Any output can be uniquely decoded. The number of inputs in the non-confusable subset is 9, so the error-free information rate of this system is $\log_2 9$ bits, which is equal to C, which we evaluated in example 9.10 (p. 168).

In terms of the theorem, assume we are given an $R < C = \log_2 9$ and an $\epsilon > 0$. No matter what ϵ and R are, we can set the block length N to 1. The block code is $\{\mathtt{B}, \mathtt{E}, \ldots, \mathtt{Z}\}$. The value of K is given by $2^K = 9$, so $K = \log_2 9$, and this code has rate $\log_2 9$, which is greater than R. The decoding algorithm maps the received letter to the nearest letter in the code, and its maximal probability of block error is zero, which, as asserted in the theorem, is less than the given ϵ.



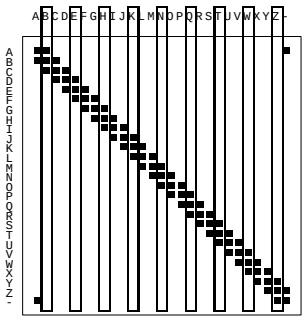


Figure 9.3. A non-confusable subset of inputs for the noisy typewriter.

Intuitive preview of proof

We consider the *extended channel* with $|\mathcal{A}_X|^N$ possible inputs $\mathbf{x}$ and $|\mathcal{A}_Y|^N$ possible outputs. Extended channels obtained from a binary symmetric channel and from a Z channel are shown in figures 9.4 and 9.5, with $N = 2$ and $N = 4$.

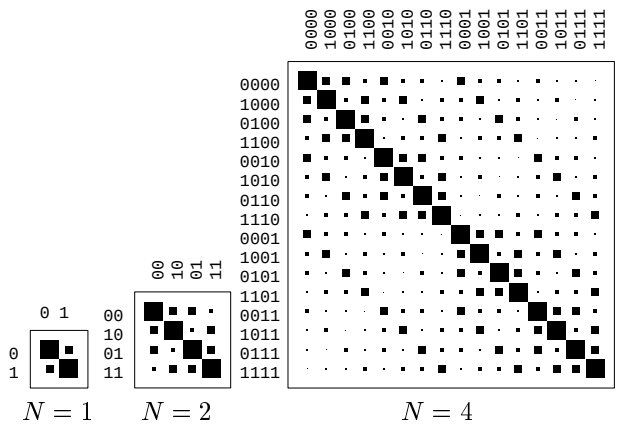


Figure 9.4. Extended channels obtained from a binary symmetric channel with transition probability 0.15.

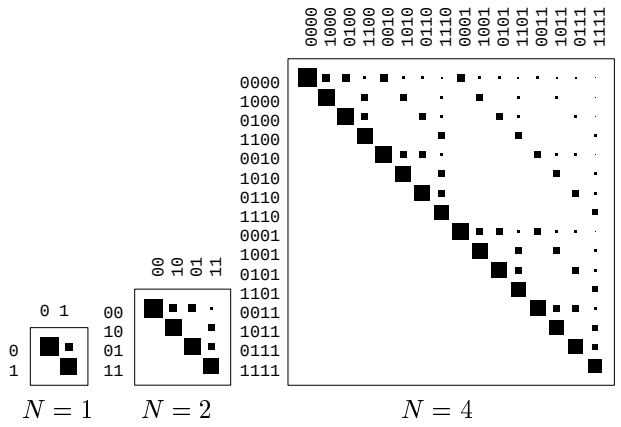


Figure 9.5. Extended channels obtained from a Z channel with transition probability 0.15.

Exercise 9.14:[A1] Find the transition probability matrices $\mathbf{Q}$ for the extended channel with $N = 2$ derived from the binary erasure channel having erasure probability 0.15.

By selecting two columns of this transition probability matrix, we can define a rate 1/2 code for this channel with blocklength $N = 2$. What is the best choice of two columns? What is the decoding algorithm?

To prove the noisy channel coding theorem, we make use of large block lengths N. The intuitive idea is that, if N is large, *an extended channel looks a lot like the noisy typewriter.* Any particular input $\mathbf{x}$ produces, with high probability, an output in a small subspace of the output alphabet. So we can find a non–confusable subset of the inputs that produce essentially disjoint output sequences. For a given N, let us consider a way of generating such a non-confusable subset of the inputs, and count up how many distinct inputs it contains.

Imagine making an input sequence $\mathbf{x}$ for the extended channel by drawing it from an ensemble X^N, where X is an arbitrary ensemble over the input alphabet. Recall the source coding theorem of lecture 2, and consider the number of probable output sequences $\mathbf{y}$. The total number of typical output sequences $\mathbf{y}$, when $\mathbf{x}$ comes from the ensemble X^N, is $2^{NH(Y)}$, all having similar probability. For any particular typical input sequence $\mathbf{x}$, there are about $2^{NH(Y|X)}$ probable sequences. Some of these subsets of $\mathcal{A}_Y^N$ are depicted by circles in figure 9.6a.

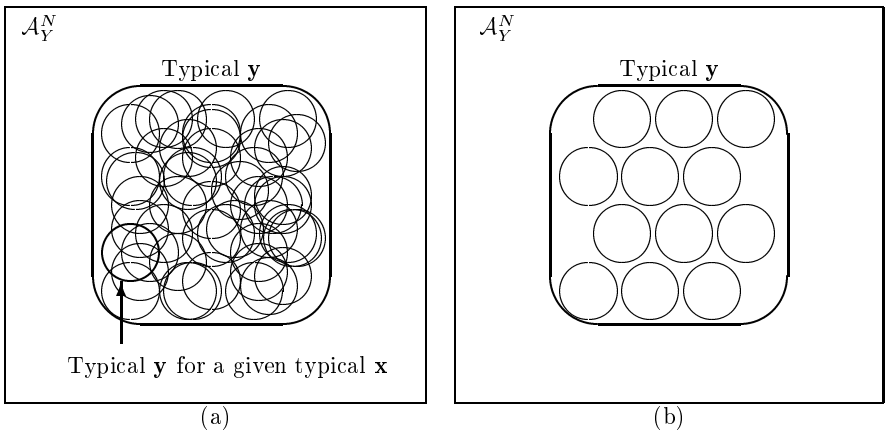


Figure 9.6. (a) Some typical outputs in $\mathcal{A}_Y^N$ corresponding to typical inputs $\mathbf{x}$. (b) A subset of the typical sets shown in (a) that do not overlap each other. This picture can be compared with the solution to the noisy typewriter in figure 9.3.

We now imagine restricting ourselves to a subset of the typical inputs $\mathbf{x}$ such that the corresponding typical output sets do not overlap, as shown in figure 9.6b. We can then bound the number of non-confusable inputs by dividing the size of the typical $\mathbf{y}$ set, $2^{NH(Y)}$, by the size of each typical-$\mathbf{y}$-given-typical-$\mathbf{x}$ set, $2^{NH(Y|X)}$. So the number of non-confusable inputs, if they are selected from the set of typical inputs $\mathbf{x} \sim X^N$, is $\leq 2^{NH(Y)-NH(Y|X)} = 2^{NI(X;Y)}$.

The maximum value of this bound is achieved if X is the ensemble that maximizes $I(X;Y)$, in which case the number of non-confusable inputs is $\leq 2^{NC}$. Thus we anticipate that asymptotically up to C bits per cycle, and no more, can be communicated with vanishing error probability.

This sketch has not rigorously proved that reliable communication really is possible — that's our task for the next chapter.

9.7 Exercises

Solutions on p. 177

Exercise 9.15:[A3] Refer back to the computation of the capacity of the Z channel with $f = 0.15$.

(a) Why is p_1^* less than 0.5?

(b) In the case of general f, show that the optimal input distribution is

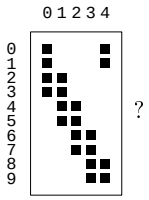
$$p_1^* = \frac{1/(1-f)}{1 + 2^{(H_2(f)/(1-f))}}. \tag{9.21}$$

(c) What happens to p_1^* if the noise level f is very close to 1?

Exercise 9.16:[A2] Sketch graphs of the capacity of the Z channel, the binary symmetric channel and the binary erasure channel as a function of f.

Exercise 9.17:[B2] What is the capacity of the five–input, ten–output channel whose transition probability matrix is

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0.25 & 0 & 0 & 0 \\ 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 \\ 0 & 0 & 0 & 0.25 & 0.25 \end{bmatrix} \tag{9.22}$$



Exercise 9.18:[A2] Consider a Gaussian channel with binary input $x \in \{-1, +1\}$ and *real* output alphabet $\mathcal{A}_Y$, with transition probability density

$$Q(y|x, \alpha, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y - x\alpha)^2}{2\sigma^2}}, \tag{9.23}$$

where α is the signal amplitude.

(a) Compute the posterior probability of x given y, assuming that the two inputs are equiprobable. Put your answer in the form

$$P(x{=}1|y, \alpha, \sigma) = \frac{1}{1 + e^{-a(y)}}. \tag{9.24}$$

Sketch the value of $P(x{=}1|y, \alpha, \sigma)$ as a function of y.

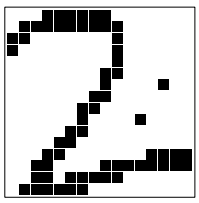
(b) Assume that a single bit is to be transmitted. What is the optimal decoder, and what is its probability of error? Express your answer in terms of the signal to noise ratio α^2/σ^2 and the cumulative probability function of the Gaussian distribution,

$$\Phi(z) \equiv \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \, dz. \tag{9.25}$$

[Note that this definition of the error function $\Phi(z)$ may not correspond to other people's. Some people leave out factors of two in the definition.]

Pattern recognition as a noisy channel

We may think of many pattern recognition problems in terms of communication channels. Consider the case of recognizing handwritten digits (such as postcodes on envelopes). The author of the digit wishes to communicate a message from the set $\mathcal{A}_X = \{0, 1, 2, 3, \ldots, 9\}$; they select one of these as the input to the channel. What comes out of the channel is a pattern of ink on paper which we might represent using 256 binary pixels. The channel Q thus has as its output a random variable $y \in \mathcal{A}_Y = \{0, 1\}^{256}$. An example of an element from this alphabet is shown in the margin.



Exercise 9.19:^{A2} Estimate how many patterns in $\mathcal{A}_Y$ are recognizable as the character '2'. [The aim of this problem is to try to demonstrate the existence of *as many patterns as possible* that are recognizable as '2's.]

Discuss how one might model the channel $P(y|x{=}2)$. Estimate the entropy of the probability distribution $P(y|x{=}2)$.

One strategy for doing pattern recognition is to create a model for $P(y|x)$ for each value of the input $x = \{0, 1, 2, 3, \ldots, 9\}$, then use Bayes' theorem to infer the probability of x given y.

$$P(x|y) = \frac{P(y|x)P(x)}{\sum_{x'} P(y|x')P(x')} \tag{9.26}$$

This strategy is known as *full probabilistic modelling* or *generative modelling*. This is essentially how current speech recognition systems work. In addition to the channel model, $P(y|x)$, one uses a prior probability distribution $P(x)$, which in the case of both character recognition and speech recognition is a language model that specifies the probability of the next character / word given the context and the known grammar and statistics of the language.

Random coding

Exercise 9.20:^{A2} Given twenty–four people in a room, what is the probability that there are two people present who have the same birthday? What is the expected number of pairs of people with the same birthday? Which of these two questions is easiest to solve? Which answer gives most insight? You may find it helpful to solve these problems and those that follow using notation such as A = number of days in year = 365 and q = number of people = 24.

Exercise 9.21:^{B2} The birthday problem may be related to a coding scheme. Assume we wish to convey a message to an outsider identifying one of the twenty–four people. We could simply communicate a number m from $\mathcal{A}_M = \{1, 2, \ldots, 24\}$, having agreed a mapping of people onto numbers; alternatively, we could convey a number from $\mathcal{A}_X = \{1, 2, \ldots, 365\}$, identifying the day of the year that is the selected person's birthday (with apologies to leapyearians). [The receiver is assumed to know all the people's birthdays.] What, roughly, is the probability of error of this communication scheme, assuming it is used for a single transmission? What is the capacity of the communication channel, and what is the rate of communication attempted by this scheme?

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Exercise 9.22:^{B2} Now imagine that there are K rooms in a building, each containing q people. (You might think of $K = 2$ and $q = 24$ as an example.) The aim is to communicate a selection of one person from each room by transmitting an ordered list of K days (from $\mathcal{A}_X$). Compare the probability of error of the following two schemes.

(a) As before, where each room transmits the birthday of the selected person.

(b) To each K–tuple of people, one drawn from each room, an ordered K–tuple of randomly selected days from $\mathcal{A}_X$ is assigned. This list of q^K strings is assumed to be known to the receiver. When the building has selected a particular person from each room, the ordered string of days corresponding to that group is transmitted.

What is the probability of error when $q = 364$ and $K = 1$? What is the probability of error when $q = 364$ and K is large, *e.g.*, $K = 6000$?

Solutions to Chapter 9's exercises

Solution to exercise 9.2 (p. 166): If we assume we observe y=0,

$$P(x{=}1|y{=}0) = \frac{P(y{=}0|x{=}1)P(x{=}1)}{\sum_{x'} P(y|x')P(x')} \quad (9.27)$$

$$= \frac{0.15 \times 0.1}{0.15 \times 0.1 + 0.85 \times 0.9} \quad (9.28)$$

$$= \frac{0.015}{0.78} = 0.019. \quad (9.29)$$

Solution to exercise 9.4 (p. 166): If we observe $y = 0$,

$$P(x{=}1|y{=}0) = \frac{0.15 \times 0.1}{0.15 \times 0.1 + 1.0 \times 0.9} \quad (9.30)$$

$$= \frac{0.015}{0.915} = 0.016. \quad (9.31)$$

Solution to exercise 9.7 (p. 167): The probability that $y = 1$ is 0.5, so the mutual information is:

$$I(X;Y) = H(Y) - H(Y|X) \quad (9.32)$$

$$= H_2(0.5) - H_2(0.15) \quad (9.33)$$

$$= 1 - 0.61 = 0.39 \text{ bits.} \quad (9.34)$$

Solution to exercise 9.8 (p. 167): We again compute the mutual information using $I(X;Y) = H(Y) - H(Y|X)$. The probability that $y = 1$ is 0.575, and $H(Y|X) = \sum_x P(x)H(Y|x) = P(x{=}1)H(Y|x{=}1) + P(x{=}0)H(Y|x{=}0)$ so the mutual information is:

$$I(X;Y) = H(Y) - H(Y|X) \quad (9.35)$$

$$= H_2(0.575) - [0.5 \times H_2(0.15) + 0.5 \times 0] \quad (9.36)$$

$$= 0.98 - 0.30 = 0.679 \text{ bits.} \quad (9.37)$$

Solution to exercise 9.12 (p. 168): By symmetry, the optimal input distribution is $\{0.5, 0.5\}$. Then the capacity is

$$C = I(X;Y) = H(Y) - H(Y|X) \quad (9.38)$$

$$= H_2(0.5) - H_2(f) \quad (9.39)$$

$$= 1 - H_2(f). \quad (9.40)$$

Maybe it's best to get the optimal input distribution without invoking symmetry. We can do this by computing the mutual information in the general case where the input ensemble is $\{p_0, p_1\}$:

$$I(X;Y) = H(Y) - H(Y|X) \quad (9.41)$$

$$= H_2(p_0 f + p_1(1-f)) - H_2(f). \quad (9.42)$$

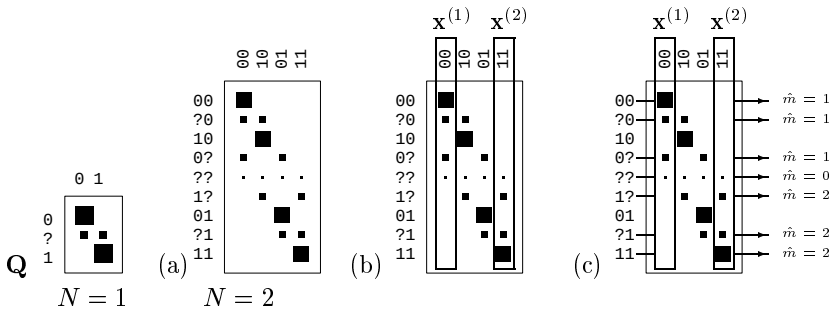


Figure 9.7. (a) The extended channel ($N = 2$) obtained from a binary erasure channel with erasure probability 0.15. (b) A block code consisting of the two codewords 00 and 11. (c) The optimal decoder for this code.

The only p–dependence is in the first term $H_2(p_0 f + p_1(1-f))$, which is maximized by setting the argument to 0.5. This value is given by setting $p_0 = 1/2$.

Solution to exercise 9.13 (p. 168): By symmetry, the optimal input distribution is $\{0.5, 0.5\}$. The capacity is most easily evaluated by writing the mutual information as $I(X;Y) = H(X) - H(X|Y)$. The conditional entropy $H(X|Y)$ is $\sum_y P(y)H(X|y)$; when y is known, x is only uncertain if $y = ?$, which occurs with probability $f/2 + f/2$, so the conditional entropy $H(X|Y)$ is $fH_2(0.5)$.

$$\begin{aligned} C = I(X;Y) &= H(X) - H(X|Y) && (9.43)\\ &= H_2(0.5) - fH_2(0.5) && (9.44)\\ &= 1 - f. && (9.45)\end{aligned}$$

The binary erasure channel 'fails' a fraction f of the time. Its capacity is precisely $1-f$, which is the fraction of the time that the channel is functional.

Alternatively, without invoking the symmetry assumed above, we can start from the input ensemble $\{p_0, p_1\}$. The probability that $y = ?$ is $p_0 f + p_1 f = f$, and when we receive $y = ?$, the posterior probability of x is the same as the prior probability, so:

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) && (9.46)\\ &= H_2(p_1) - fH_2(p_1) && (9.47)\\ &= (1-f)H_2(p_1). && (9.48)\end{aligned}$$

This mutual information achieves its maximum value of $(1-f)$ when $p_1 = 1/2$.

Solution to exercise 9.14 (p. 172): The extended channel is shown in figure 9.7. The best code for this channel with $N = 2$ is obtained by choosing two columns which have minimal overlap, for example, columns 00 and 11. The decoding algorithm returns '00' if the extended channel output is among the top four and '11' if it's among the bottom four, and gives up if the output is ??.

Solution to exercise 9.15 (p. 173): In example 9.11 (p. 168) we showed that the mutual information between input and output of the Z channel is

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X)\\ &= H_2(p_1(1-f)) - p_1 H_2(f). && (9.49)\end{aligned}$$

We differentiate this expression with respect to p_1, taking care not to confuse $\log_2$ with $\log_e$:

$$\frac{d}{dp_1} I(X;Y) = (1-f)\log_2 \frac{1-p_1(1-f)}{p_1(1-f)} - H_2(f). \tag{9.50}$$

Setting this derivative to zero and rearranging using skills developed in exercise 2.25 (p. 40), we obtain:

$$p_1^*(1-f) = \frac{1}{1+2^{H_2(f)/(1-f)}}, \tag{9.51}$$

so the optimal input distribution is

$$p_1^* = \frac{1/(1-f)}{1+2^{(H_2(f)/(1-f))}}. \tag{9.52}$$

As the noise level f tends to 1, this expression tends to $1/e$.

For all values of f, p_1^* is smaller than $1/2$. A rough intuition for why input 1 is used less than input 0 is that when input 1 is used, the noisy channel injects entropy into the received string; whereas when input 0 is used, the noise has zero entropy. Thus starting from $p_1 = 1/2$, a perturbation towards smaller p_1 will reduce the conditional entropy $H(Y|X)$ linearly while leaving $H(Y)$ unchanged, to first order. $H(Y)$ only decreases quadratically in $(p_1 - 1/2)$.

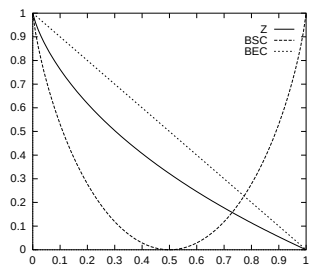


Figure 9.8. Capacities of the Z channel, binary symmetric channel, and binary erasure channel.

Solution to exercise 9.16 (p. 173): The capacities of the three channels are shown in figure 9.8. For any $f < 0.5$, the channels can be ordered with the BEC being the channel with highest capacity and the BSC the lowest.

Solution to exercise 9.17 (p. 173): The conditional entropy of Y given X is $H(Y|X) = \log 4$. The entropy of Y is at most $H(Y) = \log 10$, which is achieved by using a uniform input distribution. The capacity is therefore

$$C = \max_{\mathcal{P}_X} H(Y) - H(Y|X) = \log \frac{10}{4} = \log 5 \text{ bits}. \tag{9.53}$$

Solution to exercise 9.18 (p. 173): The logarithm of the posterior probability ratio, given y, is

$$a(y) = \log \frac{P(x=1|y,\alpha,\sigma)}{Q(x=-1|y,\alpha,\sigma)} = \log \frac{Q(y|x=1,\alpha,\sigma)}{Q(y|x=-1,\alpha,\sigma)} = 2\frac{\alpha y}{\sigma^2}. \tag{9.54}$$

Using our skills picked up from exercise 2.25 (p. 40), we rewrite this in the form

$$P(x=1|y,\alpha,\sigma) = \frac{1}{1+e^{-a(y)}}. \tag{9.55}$$

The optimal decoder selects the most probable hypothesis; this can be done simply by looking at the sign of $a(y)$. If $a(y) > 0$ then decode as $\hat{x} = 1$.

The probability of error is

$$p_b = \int_{-\infty}^{0} dy\, Q(y|x=1,\alpha,\sigma) = \int_{-\infty}^{-x\alpha} dy\, \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} = \Phi\left(-\frac{x\alpha}{\sigma^2}\right) \tag{9.56}$$

where

$$\Phi(z) \equiv \int_{-\infty}^{z} \tfrac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}. \tag{9.57}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solution to exercise 9.19 (p. 174): The number of recognizable '2's is best estimated by concentrating on the type of patterns that make the greatest contribution to this sum. These are patterns in which just a small patch of the pixels make the shape of a 2 and most of the other pixels are set at random. It is unlikely that the random pixels will take on some other recognizable shape, as we will confirm later. A recognizable letter 2 surrounded by a white border can be written in 6×7 pixels. This leaves 214 pixels that can be set arbitrarily, and there are also 12×11 possible locations for the miniature 2 to be placed, and two colourings (white on black / black on white). There are thus about $12 \times 11 \times 2 \times 2^{214} \simeq 2^{219}$ miniature 2 patterns, almost all of which are recognizable only as the character 2. This claim that the noise pattern will not look like some other character is confirmed by noting what a small fraction of all possible patterns the above number of 2s is. Let's assume there are 127 other characters to worry about. Only a fraction 2^{-37} of the 2^{256} random patterns are recognizable 2s, so similarly, of the 2^{219} miniature 2 patterns identified above, only a fraction of about 127×2^{-37} of them also contain another recognizable character. These double–hits decrease undetectably the above answer, 2^{219}.

Figure 9.9. Four random samples from the set of 2^{219} 'miniature 2s' defined in the text.

Another way of estimating the entropy of a 2, this time banning the option of including background noise, is to consider the number of *decisions* that are made in the construction of a font. A font may be **bold (2)** or not bold; *italic (2)* or not; sans–serif (2) or not. It may be normal size, small (2) or tiny (2). It may be calligraphic, futuristic, modern, or gothic. Most of these choices are independent. So we have at least $2^4 \times 3^2$ distinct fonts. I believe that Donald Knuth's METAFONT, with the aid of which this document was produced, turns each of these axes of variation into a continuum so that arbitrary intermediate fonts can also be created. If we can distinguish, say, five degrees of boldness, ten degrees of italicity, and so forth, then we can imagine creating perhaps $10^6 \simeq 2^{20}$ distinguishable fonts, each with a distinct 2. Extra parameters such as loopiness and spikiness could further increase this number. It would be interesting to know how many distinct 2s METAFONT can actually produce in a 16×16 box.

The entropy of the probability distribution $P(y|x{=}2)$ depends on the assumptions about noise and character size. If we assume that noise is unlikely, then the entropy may be roughly equal to the number of bits to make a clean 2 as discussed above. The possibility of noise increases the entropy. The largest it could plausibly be is the logarithm of the number derived above for the number of patterns that are recognizable as a 2, though I suppose one could argue that when someone writes a 2, they may end up producing a pattern $\mathbf{y}$ that is not recognizable as a 2. So the entropy could be even larger than 220 bits. It should be noted however, that if there is a 90% chance that the 2 is a clean 2, with entropy 20 bits, and only a 10% chance that it is a miniature 2 with noise, with entropy 220 bits, then the entropy of y is $H_2(0.1) + 0.1 \times 220 + 0.9 \times 20 \simeq 40$ bits, so in this case the entropy would be much smaller than 220 bits. Notice also, in this case, that the probability distribution $P(y|x{=}2)$ would not be at all uniform over this set of recognizable 2s. The probability of a y in the 'clean' set would be about 2^{-20}, and the probability of a noisy miniature 2 pattern would be much smaller, about 2^{-220}.

Random coding

Solution to exercise 9.20 (p. 174): The probability that $q = 24$ people whose birthdays are drawn at random from $A = 365$ days all have *distinct* birthdays is

$$\frac{A(A-1)(A-2)\ldots(A-q+1)}{A^q}. \tag{9.58}$$

The probability that two (or more) people share a birthday is one minus this quantity, which is about 0.5. This exact answer is not very informative since it is not clear for what value of q the probability changes from being close to 0 to being close to 1.

The number of pairs is $q(q-1)/2$, and the probability that a particular pair shares a birthday is $1/A$, so the expected number of collisions is

$$\frac{q(q-1)}{2}\frac{1}{A}. \tag{9.59}$$

This answer is more informative. The expected number of collisions is tiny if $q \ll \sqrt{A}$ and big if $q \gg \sqrt{A}$.

Solution to exercise 9.21 (p. 174): The probability of error is the probability that the selected message is not uniquely decodeable by the receiver, *i.e.*, it is the probability that one or more of the q–1 other people has the same birthday as our selected person, which is

$$1 - \left(\frac{A-1}{A}\right)^{q-1} = 1 - 0.939 = 0.061. \tag{9.60}$$

The capacity of the communication channel is $\log 365 \simeq 8.5$ bits. The rate of communication attempted is $\log 24 \simeq 4.6$ bits.

So we are transmitting substantially below the capacity of this noiseless channel, and our communication scheme has an appreciable probability of error (6%). Random coding looks a rather silly idea.

Solution to exercise 9.22 (p. 175): The number of possible K–tuples is A^K, and we select q^K such K–tuples, where q is the number of people in each of the K rooms. The probability of error is the probability that the selected message is not uniquely decodeable by the receiver,

$$1 - \left(\frac{A^K - 1}{A^K}\right)^{q^K - 1}. \tag{9.61}$$

In the case $q = 364$ and $K = 1$ this probability of error is

$$1 - \left(1 - \frac{1}{A}\right)^{q-1} \simeq 1 - e^{-(q-1)/A} \simeq 1 - e = 0.63. \tag{9.62}$$

[The exact answer found from equation (9.61) is 0.631.] Thus random coding is highly likely to lead to a communication failure.

As K gets large, however, we can approximate

$$1 - \left(\frac{A^K - 1}{A^K}\right)^{q^K - 1} = 1 - \left(1 - \frac{1}{A^K}\right)^{q^K - 1} \simeq \frac{q^K - 1}{A^K} \simeq \left(\frac{q}{A}\right)^K. \tag{9.63}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

In the example $q = 364$ and $A = 365$, this probability of error decreases as $10^{-0.0012K}$, so, for example, if $K \simeq 6000$ then the probability of error is smaller than 10^{-6}.

For sufficiently large blocklength K, random coding becomes a reasonable strategy.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

About Chapter 10

Before reading chapter 10, you should have read chapters 4 and 9 and worked on exercise 8.10 (p. 157).

This exposition is based on that of Cover and Thomas (1991).

Cast of characters

Q	the noisy channel
C	the capacity of the channel
X^N	an ensemble used to create a random code
$\mathcal{C}$	a random code
N	the length of the codewords
$\mathbf{x}^{(s)}$	a codeword, the sth in the code
s	the number of a chosen codeword (mnemonic: the *source* selects s)
S	the total number of codewords in the code
$K = \log_2 S$	the number of bits conveyed by the choice of one codeword from S
$\mathbf{s}$	a binary representation of the number s
$R = K/N$	the rate of the code, in bits per channel use (sometimes called R' instead)
$\hat{s}$	the decoder's guess of s

Typo Warning: the letter m may turn up where it should read s.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

10

The Noisy Channel Coding Theorem

10.1 The Theorem

- For every discrete memoryless channel, the channel capacity

$$C = \max_{\mathcal{P}_X} I(X;Y) \tag{10.1}$$

  has the following property. For any $\epsilon > 0$ and $R < C$, for large enough N, there exists a code of length N and rate $\geq R$ and a decoding algorithm, such that the maximal probability of block error is $< \epsilon$.

- If a probability of bit error p_b is acceptable, rates up to $R(p_b)$ are achievable, where

$$R(p_b) = \frac{C}{1 - H_2(p_b)}. \tag{10.2}$$

- For given p_b, rates greater than $R(p_b)$ are not achievable.

10.2 Jointly typical sequences

We formalize the intuitive preview of the last chapter.

We will define codewords $\mathbf{x}^{(s)}$ as coming from an ensemble X^N, and consider the random selection of one codeword and a corresponding channel output $\mathbf{y}$, thus defining a joint ensemble $(XY)^N$. We will use a *typical set decoder*, which decodes a received signal $\mathbf{y}$ as s if $\mathbf{x}^{(s)}, \mathbf{y}$ are *jointly typical*, a term to be defined shortly.

The proof will then centre on determining the probabilities (a) that the true input codeword is *not* jointly typical with the output sequence; and (b) that a *false* input codeword is jointly typical with the output. We will show that, for large N, both probabilities go to zero as long as there are fewer than 2^{NC} codewords.

Joint typicality. A pair of sequences $\mathbf{x}, \mathbf{y}$ of length N are jointly typical (to tolerance β) with respect to the distribution $P(x, y)$ if

$$\begin{aligned}
\mathbf{x} \text{ is typical of } P(\mathbf{x}), \quad & i.e., \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{x})} - H(X) \right| < \beta, \\
\mathbf{y} \text{ is typical of } P(\mathbf{y}), \quad & i.e., \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{y})} - H(Y) \right| < \beta, \\
\text{and } \mathbf{x}, \mathbf{y} \text{ is typical of } P(\mathbf{x}, \mathbf{y}), \quad & i.e., \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{x}, \mathbf{y})} - H(X,Y) \right| < \beta.
\end{aligned} \tag{10.3}$$

The jointly typical set $J_{N\beta}$ is the set of all jointly typical sequence pairs of length N.

Joint typicality theorem. Let $\mathbf{x}, \mathbf{y}$ be drawn from the ensemble $(XY)^N$ defined by

$$P(\mathbf{x}, \mathbf{y}) = \prod_{n=1}^{N} P(x_n, y_n).$$

Then

1. the probability that $\mathbf{x}, \mathbf{y}$ are jointly typical tends to 1 as $N \to \infty$.
2. the number of jointly typical sequences $|J_{N\beta}|$ is close to $2^{NH(X,Y)}$. To be precise,
$$|J_{N\beta}| \leq 2^{N(H(X,Y)+\beta)}. \tag{10.4}$$
3. if $\mathbf{x}' \sim X^N$ and $\mathbf{y}' \sim Y^N$, *i.e.*, $\mathbf{x}'$ and $\mathbf{y}'$ are independent samples with the same marginal distribution as $P(\mathbf{x}, \mathbf{y})$, then the probability that $(\mathbf{x}', \mathbf{y}')$ lands in the jointly typical set is about $2^{-NI(X;Y)}$. To be precise,
$$P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) \leq 2^{-N(I(X;Y)-3\beta)}. \tag{10.5}$$

Proof. The proof of parts 1 and 2 by the law of large numbers follows that of the source coding theorem in chapter 4. For the third part,

$$\begin{aligned} P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) &= \sum_{(\mathbf{x},\mathbf{y}) \in J_{N\beta}} P(\mathbf{x}) P(\mathbf{y}) && (10.6) \\ &\leq |J_{N\beta}| 2^{-N(H(X)-\beta)} 2^{-N(H(Y)-\beta)} && (10.7) \\ &\leq 2^{N(H(X,Y)+\beta)-N(H(X)+H(Y)-2\beta)} && (10.8) \\ &= 2^{-N(I(X;Y)-3\beta)} && (10.9) \end{aligned}$$

A cartoon of the jointly typical set is shown in figure 10.1. Two independent typical vectors are jointly typical with probability

$$P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) \simeq 2^{-N(I(X;Y))} \tag{10.10}$$

because the *total* number of independent typical pairs is the area of the rectangle, $2^{NH(X)}2^{NH(Y)}$, and the number of jointly typical pairs is roughly $2^{NH(X,Y)}$, so the probability of hitting a typical pair is roughly

$$2^{NH(X,Y)}/2^{NH(X)+NH(Y)} = 2^{-NI(X;Y)}. \tag{10.11}$$

10.3 Proof of the channel coding theorem

Sketch

We wish to show that there exists a code and a decoder having small probability of error. Evaluating the probability of error of any particular coding and decoding system is not easy. Shannon's innovation was, instead of constructing a good system and evaluating its error probability, to calculate the average probability of block error of *all* codes, and prove that this average is

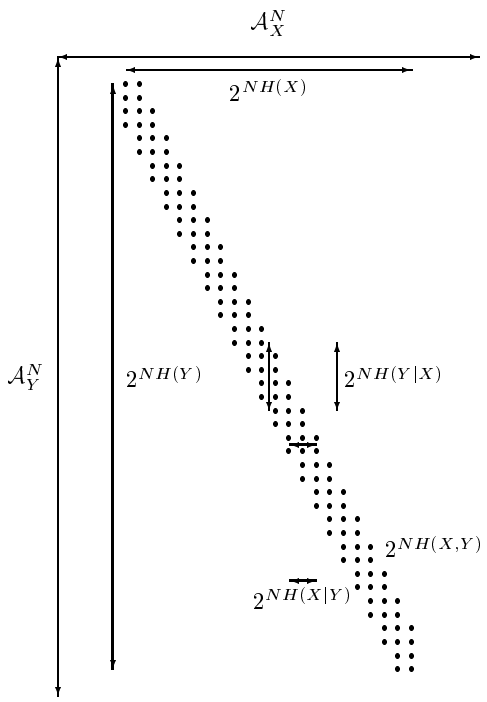


Figure 10.1. The jointly typical set. The horizontal direction represents $\mathcal{A}_X^N$, the set of all input strings of length N. The vertical direction represents $\mathcal{A}_Y^N$, the set of all output strings of length N. Each dot represents a jointly typical pair of sequences $(\mathbf{x}, \mathbf{y})$. The total number of jointly typical sequences is about $2^{NH(X,Y)}$. [Compare with figure 9.7a, page 177.]

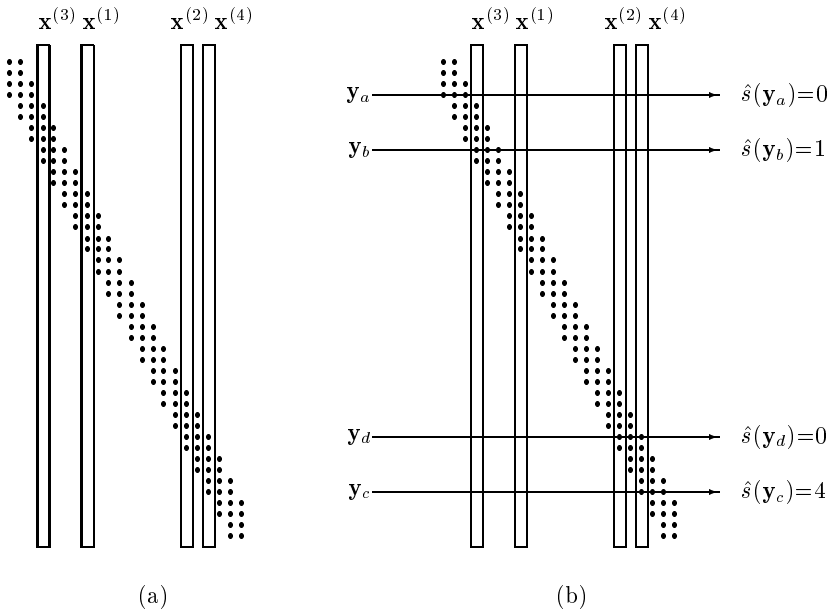


Figure 10.2. (a) A random code. (b) Example decodings by the typical set decoder. A sequence that is not jointly typical with any of the codewords, such as $\mathbf{y}_a$, is decoded as $\hat{s} = 0$. A sequence that is jointly typical with codeword $\mathbf{x}^{(1)}$ only, $\mathbf{y}_b$, is decoded as $\hat{s} = 1$. Similarly, $\mathbf{y}_c$ is decoded as $\hat{s} = 4$. A sequence that is jointly typical with more than one codeword, such as $\mathbf{y}_d$, is decoded as $\hat{s} = 0$.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

small. There must then exist individual codes that have small probability of block error. Finally to prove that the *maximal* probability of error is small too, we modify one of these good codes by throwing away the worst 50% of its codewords.

Random coding and typical set decoding

Consider the following encoding–decoding system, whose rate is R'.

1. We fix $P(x)$ and generate the $S = 2^{NR'}$ codewords of a $(N, NR') = (N, K)$ code $\mathcal{C}$ at random according to

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n). \tag{10.12}$$

   A random code is shown schematically in figure 10.2.

2. The code is known to both sender and receiver.

3. A message s is chosen at random from $\{1, 2, \ldots, 2^{NR'}\}$, and $\mathbf{x}^{(s)}$ is transmitted. The received signal is $\mathbf{y}$, with

$$P(\mathbf{y}|\mathbf{x}^{(s)}) = \prod_{n=1}^{N} P(y_n|x_n^{(s)}). \tag{10.13}$$

4. The signal is decoded by *typical set decoding*.

   Typical set decoding. Decode $\mathbf{y}$ as $\hat{s}$ if $(\mathbf{x}^{(\hat{s})}, \mathbf{y})$ are jointly typical *and* there is no other s' such that $(\mathbf{x}^{(s')}, \mathbf{y})$ are jointly typical; otherwise declare a failure ($\hat{s}$=0).

   This is not the optimal decoding algorithm, but it will be good enough, and easier to analyse. The typical set decoder is illustrated in figure 10.2.

5. A decoding error occurs if $\hat{s} \neq s$.

There are three probabilities of error that we can distinguish. First, there is the probability of block error for a particular code $\mathcal{C}$, that is,

$$p_{\rm B}(\mathcal{C}) \equiv P(\hat{s} \neq s|\mathcal{C}). \tag{10.14}$$

This is a difficult quantity to evaluate for any given code.

Second, there is the average over all codes of this block error probability,

$$\langle p_{\rm B} \rangle \equiv \sum_{\mathcal{C}} P(\hat{s} \neq s|\mathcal{C}) P(\mathcal{C}). \tag{10.15}$$

Fortunately, this quantity is much easier to evaluate than the first quantity $P(\hat{s} \neq s|\mathcal{C})$.

Third, the maximal block error probability of a code $\mathcal{C}$,

$$p_{\rm BM}(\mathcal{C}) \equiv \max_{s} P(\hat{s} \neq s|s\mathcal{C}), \tag{10.16}$$

is the quantity we are most interested in: we wish to show that there exists a code $\mathcal{C}$ with the required rate whose maximal block error probability is small.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

We will get to this result by first finding the average block error probability, $\langle p_{\rm B} \rangle$. Once we have shown that this can be made smaller than a desired small number, we deduce that there must exist *at least one* code $\mathcal{C}$ whose block error probability is also less than this small number. Finally, we show that this code, whose maximal block error probability is unknown, can be modified to make a code whose maximal block error probability is guaranteed to be small.

We therefore embark now on finding the average probability of block error.

Probability of error of typical set decoder

There are two sources of error when we use typical set decoding. Either (a) the output $\mathbf{y}$ is not jointly typical with the transmitted codeword $\mathbf{x}^{(s)}$, or (b) there is some other codeword that is jointly typical with $\mathbf{y}$.

By the symmetry of the code construction, the average probability of error averaged over all codes does not depend on the selected value of s; we can assume without loss of generality that $s = 1$.

(a) The probability that $(\mathbf{x}^{(1)}, \mathbf{y})$ are not jointly typical vanishes by the joint typicality theorem's first part (page 184). We give a name, δ, to the upper bound on this probability, satisfying $\delta \rightarrow 0$ as $N \rightarrow \infty$; for any desired δ, we can find a value of $N(\delta)$ such that the $P((\mathbf{x}^{(1)}, \mathbf{y}) \notin J_{N\beta}) \leq \delta$.

(b) The probability that $(\mathbf{x}^{(s)}, \mathbf{y})$ are jointly typical, for a *given* $s \neq 1$ is $\leq 2^{-N(I(X;Y)-3\beta)}$, by part 3.

Thus the average probability of error $\langle p_{\rm B} \rangle$ satisfies:

$$\begin{aligned} \langle p_{\rm B} \rangle &\leq \delta + \sum_{s=2}^{2^{NR'}} 2^{-N(I(X;Y)-3\beta)} \\ &\leq \delta + 2^{-N(I(X;Y)-R'-3\beta)} \end{aligned}$$

This can be made $< 2\delta$ for sufficiently large N if

$$R' < I(X;Y) - 3\beta. \tag{10.17}$$

We are almost there. We make three modifications:

1. We choose $P(x)$ in the proof to be the optimal input distribution of the channel. Then the condition $R' < I(X;Y) - 3\beta$ becomes $R' < C - 3\beta$.

2. Since the average probability of error over all codes is $< 2\delta$, there must exist a code with mean probability of block error $p_{\rm B}(\mathcal{C}) < 2\delta$.

3. To show that not only the average but also the maximal probability of error, $p_{\rm BM}$, can be made small, we modify this code by throwing away the worst half of the codewords — the ones most likely to produce errors. Those that remain must all have *conditional* probability of error less than 4δ. This new code has $2^{NR'-1}$ codewords, *i.e.*, we have reduced the rate negligibly from R' to $R' - \frac{1}{N}$, and achieved $p_{\rm BM} < 4\delta$. This trick is called *expurgation*.

In conclusion, we have 'constructed' a code of rate $R' - \frac{1}{N}$, where $R' < C - 3\beta$, with maximal probability of error $< 4\delta$. We obtain the theorem as stated by setting $R' = (R + C)/2$, $\delta = \epsilon/4$, $\beta < (C - R')/3$, and N sufficiently large for the remaining conditions to hold. The theorem's first part is thus proved.

10.4 Communication (with errors) above capacity

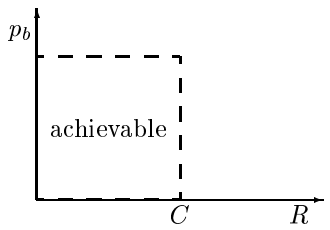


Figure 10.3. Portion of the R, p_b plane proved achievable.

We have proved, for any discrete memoryless channel, the achievability of a portion of the R, p_b plane shown in figure 10.3. We have shown that we can turn any noisy channel into an essentially noiseless binary channel with rate up to C bits per cycle. We now extend the right-hand boundary of the region of achievability at non-zero error rates.

We do this with a new trick. Since we know we can make the noisy channel into a perfect channel with a smaller rate, it is sufficient to consider communication with errors over a *noiseless* channel. How fast can we communicate over a noiseless channel, if we are allowed to make errors?

Consider a noiseless binary channel, and assume that we force communication at a rate greater than its capacity of 1 bit. For example, if we require the sender to attempt to communicate at R=2 bits per cycle then he must effectively throw away half of the information. What is the best way to do this if the aim is to achieve the smallest possible probability of bit error? One simple strategy is to communicate a fraction $1/R$ of the source bits, and ignore the rest. The receiver guesses the missing fraction $1-1/R$ at random, and the average probability of bit error is $p_b = \frac{1}{2}(1-1/R)$. The curve $p_b = \frac{1}{2}(1-1/R)$ is shown by the dashed line in figure 10.4.

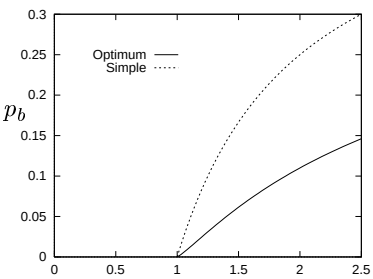


Figure 10.4. A simple bound on achievable points (R, p_b), and Shannon's bound.

We can do better than this (in terms of minimizing p_b) by spreading out the risk of corruption evenly among all the bits. In fact, we can achieve $p_b = H_2^{-1}(1-1/R)$, which is shown by the solid curve in figure 10.4. So, how can this optimum be achieved?

We reuse a tool that we just developed, namely the (N, K) code for a noisy binary symmetric channel, and we turn it on its head, using the *decoder* to define a lossy compressor. Assume that such a code has a rate $R' = K/N$, and that it is capable of correcting errors introduced by a binary symmetric channel whose transition probability is q. Asymptotically, rate R' codes exist that have $R' \simeq 1 - H_2(q)$. Recall that, if we attach one of these capacity–achieving codes of length N to a binary symmetric channel then (a) the probability distribution over the outputs is close to uniform, since the entropy of the output is equal to the entropy of the source (NR') plus the entropy of the noise ($NH_2(q)$), and (b) the optimal decoder of the code, in this situation, typically maps a received vector of length N to a transmitted vector differing in qN bits from the received vector.

We take the signal that we wish to send, and chop it into blocks of length N (not K, note). We pass each block through the *decoder*, and obtain a shorter signal of length K bits, which we communicate over the noiseless channel. To decode the transmission, we pass the K bit message to the *encoder* of the original code. The reconstituted message will now differ from the original message in some of its bits — typically qN of them. So the probability of bit error will be $p_b = q$. The rate of this lossy compressor is $R = N/K = 1/R' = 1/(1 - H_2(p_b))$.

Now, attaching this lossy compressor to our capacity C error-free communicator, we have proved the achievability of communication up to the curve (p_b, R) defined by:

$$R = \frac{C}{1 - H_2(p_b)}. \tag{10.18}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

The non-achievable region

The source, encoder, noisy channel and decoder define a Markov chain:

$$P(s, \mathbf{x}, \mathbf{y}, \hat{s}) = P(s)P(\mathbf{x}|s)P(\mathbf{y}|\mathbf{x})P(\hat{s}|\mathbf{y}). \tag{10.19}$$

The data processing inequality (exercise 8.10, p. 157) must apply to this chain: $I(s;\hat{s}) \leq I(\mathbf{x};\mathbf{y})$. Furthermore, by the definition of channel capacity, $I(\mathbf{x};\mathbf{y}) \leq NC$, so $I(s;\hat{s}) \leq NC$.

Assume that a system achieves a rate R and a bit error probability p_b; then the mutual information $I(s;\hat{s})$ is $\geq NR(1-H_2(p_b))$. But $I(s;\hat{s}) > NC$ is not achievable, so $R > \frac{C}{1-H_2(p_b)}$ is not achievable.

Exercise 10.1:[C2] Fill in the details in the preceding argument. If the bit errors between $\hat{s}$ and s are independent then we have $I(s;\hat{s}) = NR(1-H_2(p_b))$. What if we have complex correlations among those bit errors? Why does the inequality $I(s;\hat{s}) \geq NR(1-H_2(p_b))$ hold?

10.5 Computing capacity*

We have proved that the capacity of a channel is the maximum rate at which reliable communication can be achieved. How can we compute the capacity of a given discrete memoryless channel? We need to find its optimal input distribution. In general we can find the optimal input distribution by a computer search, making use of the derivative of the mutual information with respect to the input probabilities.

Solutions on p. 193

Exercise 10.2:[A2] Find the derivative of $I(X;Y)$ with respect to the input probability p_i, $\partial I(X;Y)/\partial p_i$, for a channel with conditional probabilities $Q_{j|i}$.

Solutions on p. 193

Exercise 10.3:[A2] Show that $I(X;Y)$ is a concave$\cap$ function of the input probability vector $\mathbf{p}$.

Since $I(X;Y)$ is concave$\cap$ in the input distribution $\mathbf{p}$, any probability distribution $\mathbf{p}$ at which $I(X;Y)$ is stationary must be a global maximum of $I(X;Y)$. So it is tempting to put the derivative of $I(X;Y)$ into a routine that finds a local maximum of $I(X;Y)$, that is, an input distribution $P(x)$ such that

$$\frac{\partial I(X;Y)}{\partial p_i} = \lambda \quad \text{for all } i, \tag{10.20}$$

where λ is a Lagrange multiplier associated with the constraint $\sum_i p_i = 1$. However, this approach may fail to find the right answer, because $I(X;Y)$ might be maximized by a distribution that has $p_i = 0$ for some inputs. A simple example is given by the ternary confusion channel.

Ternary confusion channel. $\mathcal{A}_X = \{0, ?, 1\}$. $\mathcal{A}_Y = \{0, 1\}$.

$P(y{=}0|x{=}0){=}1$; $P(y{=}0|x{=}?){=}1/2$; $P(y{=}0|x{=}1){=}0$;
$P(y{=}1|x{=}0){=}0$; $P(y{=}1|x{=}?){=}1/2$; $P(y{=}1|x{=}1){=}1$.

Whenever the input ? is used, the output is random; the other inputs are reliable inputs. The maximum information rate of 1 bit is achieved by making no use of the input ?.

Exercise 10.4:^{B2} Sketch the mutual information for this channel as a function of the input distribution $\mathbf{p}$. Pick a convenient two–dimensional representation of $\mathbf{p}$.

The optimization routine must therefore take account of the possibility that, as we go up hill on $I(X;Y)$, we may run into the inequality constraints $p_i \geq 0$.

Solutions on p. 193

Exercise 10.5:^{B2} Describe the condition, similar to equation (10.20), that is satisfied at a point where $I(X;Y)$ is maximized, and describe a computer program for finding the capacity of a channel.

Results that may help in finding the optimal input distribution

1. All outputs must be used
2. $I(X,Y)$ is a convex$\cup$ function of the channel parameters.
3. There may be several optimal input distributions, but they all look the same at the output.

Solutions on p. 193

Exercise 10.6:^{B2} Prove that no output y is unused by an optimal input distribution, unless it is unreachable, that is, has $Q(y|x) = 0$ for all x.

Exercise 10.7:^{C2} Prove that $I(X,Y)$ is a convex$\cup$ function of $Q(y|x)$.

Solutions on p. 193

Exercise 10.8:^{C2} Prove that all optimal input distributions of a channel have the same output probability distribution $P(y) = \sum_x P(x)Q(y|x)$.

These results, along with the fact that $I(X;Y)$ is a concave$\cap$ function of the input probability vector $\mathbf{p}$, prove the validity of the symmetry argument that we have used when finding the capacity of symmetric channels. If a channel is invariant under a group of symmetry operations — for example, interchanging the input symbols and interchanging the output symbols — then, given any optimal input distribution that is not symmetric, *i.e.*, is not invariant under these operations, we can create another input distribution by averaging together this optimal input distribution and all its permuted forms that we can make by applying the symmetry operations to the original optimal input distribution. The permuted distributions must have the same $I(X;Y)$ as the original, by symmetry, so the new input distribution created by averaging must have $I(X;Y)$ bigger than or equal to that of the original distribution, because of the concavity of I.

Symmetric channels

In order to use symmetry arguments, it will help to have a definition of a symmetric channel. I like Gallager's definition.

A discrete memoryless channel is a symmetric channel if the set of outputs can be partitioned into subsets in such a way that for each subset the matrix of transition probabilities has the property that each row (if more than 1) is a permutation of each other row and each column is a permutation of each other column.

Example 10.9: This channel

$$\begin{array}{ll} P(y{=}0|x{=}0){=}0.7; & P(y{=}0|x{=}1){=}0.1; \\ P(y{=}?|x{=}0){=}0.2; & P(y{=}?|x{=}1){=}0.2; \\ P(y{=}1|x{=}0){=}0.1; & P(y{=}1|x{=}1){=}0.7. \end{array} \tag{10.21}$$

is a symmetric channel because its outputs can be partitioned into $(0, 1)$ and ?, so that the matrix can be rewritten:

$P(y{=}0\|x{=}0){=}0.7;$	$P(y{=}0\|x{=}1){=}0.1;$
$P(y{=}1\|x{=}0){=}0.1;$	$P(y{=}1\|x{=}1){=}0.7;$
$P(y{=}?\|x{=}0){=}0.2;$	$P(y{=}?\|x{=}1){=}0.2.$

(10.22)

Symmetry is a useful property because, as we will see in a later chapter, communication at capacity can be achieved over symmetric channels by *linear* codes.

Solutions on p. 193

Exercise 10.10:^{C2} Prove that for a symmetric channel, the uniform distribution is an optimal input distribution.

Solutions on p. 193

Exercise 10.11:^{B2} Are there channels that are not symmetric whose optimal input distributions are uniform? Find one, or prove there are none.

10.6 Other coding theorems*

The noisy channel coding theorem that we proved in this chapter is quite general, applying to any discrete memoryless channel; but it is not very specific. The theorem only says that reliable communication with error probability ϵ and rate R can be achieved by using codes with *sufficiently large* blocklength N. The theorem does not say how large N needs to be to achieve given values of R and ϵ.

Presumably, the smaller ϵ is and the closer R is to C, the larger N has to be.

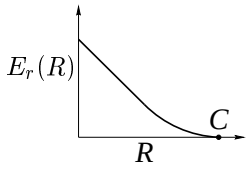


Figure 10.5. A typical random-coding exponent

Noisy channel coding theorem — version with explicit N–dependence

For a discrete memoryless channel, a blocklength N and a rate R, there exist block codes of length N whose average probability of error satisfies:

$$p_{\rm B} \leq \exp\left[-NE_r(R)\right] \tag{10.23}$$

where $E_r(R)$ is the *random–coding exponent* of the channel, a convex $\cup$, decreasing, positive function of R for $0 \leq R < C$.

By an expurgation argument it can also be shown that there exist block codes for which the maximal probability of error $p_{\rm BM}$, like $p_{\rm B}$ in equation (10.23), is exponentially small in N.

The definition of $E_r(R)$ is given in Gallager (1968), p. 139. $E_r(R)$ approaches zero as $R \to C$; the typical behaviour of this function is illustrated in figure 10.5. The computation of the random–coding exponent for interesting channels is a challenging task on which much effort has been expended. Even for simple channels like the binary symmetric channel, there is no simple expression for $E_r(R)$.

Lower bounds on the error probability as a function of blocklength

The theorem stated above asserts that there are codes with $p_{\rm B}$ smaller than $\exp\left[-NE_r(R)\right]$. But how small can the error probability be? Could it be much smaller?

> For any code with blocklength N on a discrete memoryless channel, the probability of error assuming all source messages are used with equal probability satisfies
>
> $$p_B \gtrsim \exp[-NE_{\rm sp}(R)], \tag{10.24}$$
>
> where the function $E_{\rm sp}(R)$, the *sphere–packing exponent*, is a convex $\cup$, decreasing, positive function of R for $0 \leq R < C$.

For a precise statement of this result and further references, see Gallager (1968), p. 157.

Noisy channel coding theorems and coding practice

Imagine a customer who wants to buy an error correcting code and decoder for a noisy channel. The results described above allow us to offer the following service: if he tells us the properties of his channel, the desired rate R and the desired error probability $p_{\rm B}$, we can, after working out the relevant functions C, $E_{\rm r}(R)$, and $E_{\rm sp}(R)$, advise him that there exists a solution to his problem using a particular block length N; indeed that almost any randomly chosen code with that N should do the job. Unfortunately we have not found out how to implement these encoders and decoders in practice, however; the cost of implementing the encoder and decoder for a random code with large N would be exponentially large in N.

Furthermore, for practical purposes, the customer is unlikely to know exactly what channel he is dealing with, so Berlekamp (1980) suggests that the sensible way to approach error-correction is to design encoding-decoding systems and plot their performance on a *variety* of idealised channels as a function of the channel's noise level. These charts can then be shown to the customer, who can choose among the systems on offer without having to specify what he really thinks his channel is like. With this attitude to the practical problem, the importance of the functions $E_{\rm r}(R)$ and $E_{\rm sp}(R)$ is diminished.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 10's exercises

Solution to exercise 10.1 (p. 189): *[More work needed here.]*

Solution to exercise 10.2 (p. 189): Get a specific mutual information like object minus $\log e$.

Solution to exercise 10.3 (p. 189): $I(X,Y) = \sum_{x,y} P(x)Q(y|x) \log \frac{Q(y|x)}{P(x)Q(y|x)}$ is a concave$\cap$ function of $P(x)$. Easy Proof in Gallager p.90, using `z->x->y`, where z chooses between the two things we are mixing. This satisfies $I(X;Y|Z) = 0$ (data processing inequality).

Solution to exercise 10.5 (p. 190): Necessary and sufficient conditions for $\mathbf{p}$ to maximize $I(X;Y)$ are

$$\left.\begin{array}{rcl} \frac{\partial I(X;Y)}{p_i} & = & \lambda \quad \text{and} \quad p_i > 0 \\ \frac{\partial I(X;Y)}{p_i} & \leq & \lambda \quad \text{and} \quad p_i = 0 \end{array}\right\} \quad \text{for all } i, \tag{10.25}$$

where λ is a constant related to the capacity by $C = \lambda + \log_2 e$.

This result can be used in a computer program that evaluates the derivatives and increments and decrements the probabilities p_i in proportion to the differences between those derivatives.

This result is also useful for lazy human capacity–finders who are good guessers. Having guessed the optimal input distribution, one can simply confirm that equation (10.25) holds.

Solution to exercise 10.6 (p. 190): coming

Solution to exercise 10.7 (p. 190): Easy Proof, using `(x,z)->y`.

Solution to exercise 10.8 (p. 190): coming

Solution to exercise 10.10 (p. 191): This can be proved by the symmetry argument given in the chapter.

Alternatively see p.94 of Gallager.

Solution to exercise 10.11 (p. 191): We certainly expect nonsymmetric channels with uniform optimal input distributions to exist, since when inventing a channel we have $I(J-1)$ degrees of freedom and the optimal input distribution is just $(I-1)$–dimensional, so starting from a symmetric channel, in the $I(J-1)$–dimensional space of perturbations, we expect there to be an subspace of perturbations of dimension $I(J-1) - (I-1) = I(J-2) + 1$ that leave the optimal input distribution unchanged.

Here is an explicit example, a bit like a Z channel.

$$\mathbf{Q} = \begin{bmatrix} 0.9585 & 0.0415 & 0.35 & 0.0 \\ 0.0415 & 0.9585 & 0.0 & 0.35 \\ 0 & 0 & 0.65 & 0 \\ 0 & 0 & 0 & 0.65 \end{bmatrix} \tag{10.26}$$

About Chapter 11

Before reading chapter 11, you should have read chapters 9 and 10.

You will also need to be familiar with the *Gaussian distribution*.

One-dimensional Gaussian distribution. If a random variable y is Gaussian and has mean μ and variance σ^2, which we write:

$$y \sim \text{Normal}(\mu, \sigma^2), \text{ or } P(y) = \text{Normal}(y; \mu, \sigma^2), \tag{10.27}$$

then the distribution of y is:

$$P(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-(y-\mu)^2/2\sigma^2\right]. \tag{10.28}$$

[I will use the symbol P for both probability densities and probabilities.]

The inverse-variance $\tau \equiv \frac{1}{\sigma^2}$ is sometimes called the *precision* of the Gaussian distribution.

Multi-dimensional Gaussian distribution. If $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ has a multivariate Gaussian distribution, then

$$P(\mathbf{y}|\mathbf{x}, \mathbf{A}) = \sqrt{\det\left(\frac{\mathbf{A}}{2\pi}\right)} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x})^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x})\right), \tag{10.29}$$

where $\mathbf{x}$ is the mean of the distribution, and $\mathbf{A}$ is the inverse of the variance-covariance matrix.

This distribution has the property that the variance Σ_{ii} of y_i, and the covariance Σ_{ij} of y_i and y_j are given by

$$\Sigma_{ij} \equiv \mathcal{E}\left[(y_i - \bar{y}_i)(y_j - \bar{y}_j)\right] = A^{-1}_{ij}, \tag{10.30}$$

where $\mathbf{A}^{-1}$ is the inverse of the matrix $\mathbf{A}$.

The marginal distribution $P(y_i)$ of one component y_i is Gaussian; the joint marginal distribution of any subset of the components is multivariate-Gaussian; and the conditional density of any subset, given the values of another subset, for example, $P(y_i|y_j)$, is also Gaussian.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

11

Error Correcting Codes & Real Channels

The noisy channel coding theorem that we have proved shows that there exist reliable error correcting codes for any noisy channel. In this chapter we address two questions.

First, many practical channels have real, rather than discrete, inputs and outputs. How should digital signals be mapped into analogue waveforms, and *vice versa*?

Second, how are practical error correcting codes made, and what is achieved in practice, relative to the possibilities proved by Shannon?

11.1 The Gaussian Channel

The most popular continuous channel model is the Gaussian channel.

The Gaussian Channel has a real input x and a real output y. The conditional distribution of y given x is a Gaussian distribution:

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-(y-x)^2/2\sigma^2\right]. \tag{11.1}$$

This channel has a continuous input and output but is discrete in time. We will show below that certain continuous-time channels are in fact equivalent to the discrete time Gaussian channel.

This channel is sometimes called the additive white Gaussian noise (AWGN) channel.

As with discrete channels, we will discuss what rate of error-free information communication can be achieved over this channel.

Motivation for Gaussian channel

Consider a physical (electrical, say) channel with inputs and outputs that are continuous in time. We put in $x(t)$, which is some sort of band-limited signal, and out comes $y(t) = x(t) + n(t)$.

Our transmission has a power cost. The average power of a transmission of length T may be constrained thus:

$$\int_0^T dt\, [x(t)]^2/T \leq P. \tag{11.2}$$

The received signal is assumed to differ from $x(t)$ by additive noise $n(t)$ (for example Johnson noise), which we will model as white Gaussian noise. The magnitude of this noise is quantified by the *noise spectral density* N_0, which might depend on the effective temperature of the system.

How could such a channel be used to communicate information? Consider transmitting a set of N real numbers $\{x_n\}_{n=1}^N$ in a signal of duration T made up of a weighted combination of orthonormal basis functions $\phi_n(t)$,

$$x(t) = \sum_{n=1}^{N} x_n \phi_n(t), \tag{11.3}$$

where $\int_0^T dt\, \phi_n(t)\phi_m(t) = \delta_{nm}$. The receiver can then compute:

$$y_n \equiv \int_0^T dt\, \phi_n(t) y(t) = x_n + \int_0^T dt\, \phi_n(t) n(t) \tag{11.4}$$

$$\equiv x_n + n_n. \tag{11.5}$$

If there were no noise, then y_n would equal x_n. The white Gaussian noise $n(t)$ adds scalar noise n_n to the estimate y_n. This noise is Gaussian: $n_n \sim \text{Normal}(0, N_0/2)$, where N_0 is the spectral density introduced above. Thus a continuous channel used in this way is equivalent to the Gaussian channel defined above. The power constraint $\int_0^T dt\, [x(t)]^2 \leq PT$ defines a constraint on the numbers x_n,

$$\sum_n x_n^2 \leq PT \qquad \Rightarrow \qquad \overline{x_n^2} \leq \frac{PT}{N} \tag{11.6}$$

Before returning to the Gaussian channel, let us define the *bandwidth* (measured in Hertz) of the continuous channel to be:

$$W = \frac{N^{\max}}{2T}, \tag{11.7}$$

where $N^{\max}$ is the maximum number of orthonormal functions that can be produced in an interval of length T. This definition can be justified by imagining creating a band-limited signal of duration T from orthonormal cosine and sine curves of maximum frequency W. The number of orthonormal functions is $N^{\max} = 2WT$. This definition relates to the Nyquist sampling theorem: if the highest frequency present in a signal is W, then the signal can be fully determined from its values at a series of discrete sample points separated by the Nyquist interval $\Delta t = 1/2W$ seconds.

So the use of a real continuous channel with bandwidth W, noise spectral density N_0 and power P is equivalent to $N/T = 2W$ uses per second of a Gaussian channel with noise level $\sigma^2 = N_0/2$ and subject to the signal power constraint $\overline{x_n^2} \leq P/2W$.

Definition of E_b/N_0

Imagine that the Gaussian channel $y_n = x_n + n_n$ is used with an encoding system to transmit *binary* source bits at a rate R bits per channel use. How can we compare two encoding systems that have different rates of communication R and that use different powers $\overline{x_n^2}$? Transmitting at a large rate R is good; using small power is good too.

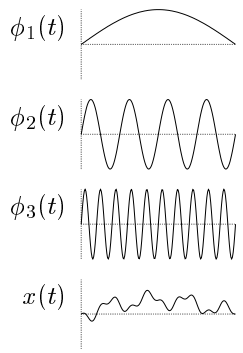


Figure 11.1. Three basis functions, and a weighted combination of them, $x(t) = \sum_{n=1}^{N} x_n \phi_n(t)$, with $x_1 = 0.4$, $x_2 = -0.2$, and $x_3 = 0.1$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

It is conventional to measure the signal to noise ratio by the ratio of the power per source bit $E_b = \overline{x_n^2}/R$ to the noise spectral density N_0:

$$E_b/N_0 = \frac{\overline{x_n^2}}{2\sigma^2 R}. \tag{11.8}$$

This is not the only measure used to compare coding schemes for Gaussian channels.

11.2 Inferring the input to a real channel

'The best detection of pulses'

In 1944 Shannon wrote a memorandum (Shannon, 1993a) on the problem of best differentiating between two types of pulses of known shape, represented by vectors $\mathbf{x}_0$ and $\mathbf{x}_1$, given that one of them has been transmitted over a noisy channel. This is a pattern recognition problem. It is assumed that the noise is Gaussian with probability density

$$P(\mathbf{n}) = \left[\det\left(\frac{\mathbf{A}}{2\pi}\right)\right]^{1/2} \exp\left(-\frac{1}{2}\mathbf{n}^{\mathsf{T}}\mathbf{A}\mathbf{n}\right), \tag{11.9}$$

where $\mathbf{A}$ is the inverse of the variance-covariance matrix of the noise. (This form of Gaussian allows the noise to be coloured.) The probability of the received vector $\mathbf{y}$ given that the source signal was s (either zero or one) is then

$$P(\mathbf{y}|s) = \sqrt{\det\left(\frac{\mathbf{A}}{2\pi}\right)} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x}_s)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_s)\right). \tag{11.10}$$

The optimal detector is based on the posterior probability ratio:

$$\frac{P(s=1|\mathbf{y})}{P(s=0|\mathbf{y})} = \frac{P(\mathbf{y}|s=1)}{P(\mathbf{y}|s=0)}\frac{P(s=1)}{P(s=0)} \tag{11.11}$$

$$= \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x}_1)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_1) + \frac{1}{2}(\mathbf{y}-\mathbf{x}_0)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_0) + \log\frac{P(s=1)}{P(s=0)}\right)$$

$$= \exp\left(\mathbf{y}^{\mathsf{T}}\mathbf{A}(\mathbf{x}_1-\mathbf{x}_0) + \theta\right), \tag{11.12}$$

where θ is a constant

$$\theta = -\frac{1}{2}\mathbf{x}_1^{\mathsf{T}}\mathbf{A}\mathbf{x}_1 + \frac{1}{2}\mathbf{x}_0^{\mathsf{T}}\mathbf{A}\mathbf{x}_0 + \log\frac{P(s=1)}{P(s=0)}. \tag{11.13}$$

If the detector is forced to make a decision (*i.e.*, guess either $s=1$ or $s=0$) then the decision that minimizes the probability of error is to guess the most probable hypothesis. We can write the optimal decision in terms of a *discriminant function*:

$$a(\mathbf{y}) \equiv \mathbf{y}^{\mathsf{T}}\mathbf{A}(\mathbf{x}_1-\mathbf{x}_0) + \theta \tag{11.14}$$

with the decisions

$$\begin{array}{lcl} a(\mathbf{y}) > 0 & \rightarrow & \text{guess } s=1 \\ a(\mathbf{y}) < 0 & \rightarrow & \text{guess } s=0 \\ a(\mathbf{y}) = 0 & \rightarrow & \text{guess either.} \end{array} \tag{11.15}$$

Notice that $a(\mathbf{y})$ is a linear function of the received vector,

$$a(\mathbf{y}) = \mathbf{w}^{\mathsf{T}}\mathbf{y} + \theta, \tag{11.16}$$

where $\mathbf{w} \equiv \mathbf{A}(\mathbf{x}_1 - \mathbf{x}_0)$.

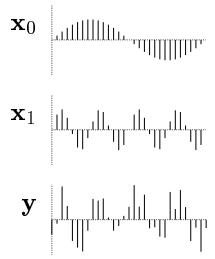


Figure 11.2. Two pulses $\mathbf{x}_0$ and $\mathbf{x}_1$, and a noisy version of one of them, $\mathbf{y}$.

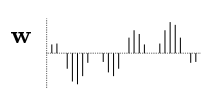


Figure 11.3. The weight vector $\mathbf{w} \propto \mathbf{x}_1 - \mathbf{x}_0$ that is used to discriminate between $\mathbf{x}_0$ and $\mathbf{x}_1$.

11.3 Capacity of Gaussian channel

Until now we have only measured the joint, marginal, and conditional entropy of discrete variables. In order to define the information conveyed by continuous variables, there are two issues we must address — the infinite length of the real line, and the infinite precision of real numbers.

Infinite inputs

How much information can we convey in one use of a Gaussian channel? If we are allowed to put *any* real number x into the Gaussian channel, we could communicate an enormous string of N digits $d_1 d_2 d_3 \ldots d_N$ by setting $x = d_1 d_2 d_3 \ldots d_N 000 \ldots 000$. The amount of error-free information conveyed in just a single transmission could be arbitrarily large by increasing N, and the communication could be made arbitrarily reliable by increasing the number of zeroes at the end of x. There is usually some power cost associated with large inputs, however. It is therefore conventional to introduce a *cost function* $v(x)$ for every input x, and constrain codes to have an average cost $\bar{v}$ less than or equal to some maximum value. A generalized channel coding theorem, including a cost function for the inputs, can be proved for the discrete channels discussed previously — see, *e.g.*, McEliece (1977). The result is a channel capacity $C(\bar{v})$ that is a function of the permitted cost. For the Gaussian channel we will assume a cost

$$v(x) = x^2 \tag{11.17}$$

such that the 'average power' $\overline{x^2}$ of the input is constrained. We have already motivated this cost function above in the case of real electrical channels in which the physical power consumption is indeed quadratic in x. The constraint $\overline{x^2} = \bar{v}$ makes it impossible to communicate infinite information in one use of the Gaussian channel.

Infinite precision

It is tempting to define joint, marginal, and conditional entropies for real variables simply by replacing summations by integrals, but this is not a well defined operation. As we discretize an interval into smaller and smaller divisions, the entropy of the discrete distribution diverges (as the logarithm of the granularity). Also, it is not permissible to take the logarithm of a dimensional quantity such as a probability density $P(x)$ (whose dimensions are $[x]^{-1}$).

$\int P(x) \log \frac{1}{P(x)} \, \mathrm{d}x$ is an illegal integral.

There is one information measure, however, that has a well-behaved limit, namely the mutual information — and this is the one that really matters, since it measures how much information one variable conveys about another. In the discrete case,

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}. \tag{11.18}$$

Now because the argument of the log is a ratio of two probabilities over the same space, it is OK to have $P(x,y)$, $P(x)$ and $P(y)$ be probability densities

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

(as long as they are not pathological) and replace the sum by an integral:

$$I(X;Y) = \int dx\, dy\, P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{11.19}$$

$$= \int dx\, dy\, P(x)P(y|x) \log \frac{P(y|x)}{P(y)}. \tag{11.20}$$

We can now ask these questions for the Gaussian channel: (a) what probability distribution $P(x)$ maximizes the mutual information (subject to the constraint $\overline{x^2} = v$)? and (b) does the maximal mutual information, as with the discrete channel, measure the maximum error free communication rate of this real channel?

Exercise 11.1:^{C3} Prove that the probability distribution $P(x)$ that maximizes the mutual information (subject to the constraint $\overline{x^2} = v$) is a Gaussian distribution of mean zero and variance v.

Exercise 11.2:^{A2} Show that the mutual information $I(X;Y)$, in the case of this optimized distribution, is

$$C = \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2} \right). \tag{11.21}$$

This is an important result. We see that the capacity of the Gaussian channel is a function of the *signal to noise ratio* v/σ^2.

Inferences given a Gaussian input distribution

If $P(x) = \text{Normal}(x; 0, v)$ and $P(y|x) = \text{Normal}(y; x, \sigma^2)$ then the marginal distribution of y is $P(y) = \text{Normal}(y; 0, v + \sigma^2)$ and the posterior distribution of the input given that the output is y is:

$$P(x|y) \propto P(y|x)P(x) \tag{11.22}$$

$$\propto \exp(-(y-x)^2/2\sigma^2) \exp(-x^2/2v) \tag{11.23}$$

$$= \text{Normal}\left(x; \frac{v}{v+\sigma^2}\, y, \left(\frac{1}{v} + \frac{1}{\sigma^2} \right)^{-1} \right). \tag{11.24}$$

This formula deserves careful study. The mean of the posterior distribution, $\frac{v}{v+\sigma^2}\, y$, can be viewed as a weighted combination of the value that best fits the output, y, and the value that best fits the prior, $x = 0$:

$$\frac{v}{v+\sigma^2}\, y = \frac{1/\sigma^2}{1/v + 1/\sigma^2}\, y + \frac{1/v}{1/v + 1/\sigma^2}\, 0. \tag{11.25}$$

The weights $1/\sigma^2$ and $1/v$ are the *precisions* of the two Gaussians that we multiplied together in equation (11.23): the prior and the likelihood.

The precision of the posterior distribution is given by adding these two precisions. This is a general property: whenever two independent sources contribute information, via Gaussian distributions, about an unknown variable, the precisions add. [This is the flip-side to the better known relationship 'when independent variables are added, their variances add'.]

Noisy channel coding theorem for the Gaussian channel

We have evaluated a maximal mutual information. Does it correspond to a maximum possible rate of error-free information transmission? One way of proving that this is so is simply to define a sequence of discrete channels, all based on the Gaussian channel, with increasing numbers of inputs and outputs, and prove that the maximum mutual information of these channels tends to C. Alternatively, we can make an intuitive argument for the coding theorem specific for the Gaussian channel.

Geometrical view of the noisy channel coding theorem: sphere packing

Consider a sequence $\mathbf{x} = (x_1, \ldots, x_N)$ of inputs, and the corresponding output $\mathbf{y}$, as defining two points in an N dimensional space. For large N, the noise power is very likely to be close (fractionally) to $N\sigma^2$. The output $\mathbf{y}$ is therefore very likely to be close to the surface of a sphere of radius $\sqrt{N\sigma^2}$ centred on $\mathbf{x}$. Similarly, if the original signal $\mathbf{x}$ is generated at random subject to an average power constraint $\overline{x^2} = v$, then $\mathbf{x}$ is likely to lie close to a sphere, centred on the origin, of radius $\sqrt{Nv}$; and because the total average power of $\mathbf{y}$ is $v + \sigma^2$, the received signal $\mathbf{y}$ is likely to lie on the surface of a sphere of radius $\sqrt{N(v+\sigma^2)}$, centred on the origin.

The volume of an N-dimensional sphere of radius r is

$$V(r, N) = \frac{\pi^{N/2}}{\Gamma(N/2+1)} r^N. \tag{11.26}$$

Now consider making a communication system based on non-confusable inputs $\mathbf{x}$, that is, inputs whose spheres do not overlap. The maximum number M of non-confusable inputs is given by dividing the volume of the sphere of probable $\mathbf{y}$s by the volume of the sphere for $\mathbf{y}$ given $\mathbf{x}$:

$$M \leq \left(\frac{\sqrt{N(v+\sigma^2)}}{\sqrt{N\sigma^2}} \right)^N \tag{11.27}$$

Thus the capacity is bounded by:

$$C = \frac{1}{N} \log M \leq \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2} \right). \tag{11.28}$$

A more detailed argument like the one used in the previous chapter can establish equality.

Back to the continuous channel

Recall that the use of a real continuous channel with bandwidth W, noise spectral density N_0 and power P is equivalent to $N/T = 2W$ uses per second of a Gaussian channel with $\sigma^2 = N_0/2$ and subject to the constraint $\overline{x_n^2} \leq P/2W$. Substituting the result for the capacity of the Gaussian channel, we find the capacity of the continuous channel to be:

$$C = W \log \left(1 + \frac{P}{N_0 W} \right) \quad \text{bits per second.} \tag{11.29}$$

This formula gives insight into the tradeoffs of practical communication. Imagine that we have a fixed power constraint. What is the best bandwidth to make

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

use of that power? Introducing $W_0 = P/N_0$, *i.e.*, the bandwidth for which the signal to noise ratio is 1, figure 11.4 shows $C/W_0 = W/W_0 \log(1 + W_0/W)$ as a function of W/W_0. The capacity increases to an asymptote of $W_0 \log e$. It is dramatically better (in terms of capacity for fixed power) to transmit at a low signal to noise ratio over a large bandwidth, than with high signal to noise in a narrow bandwidth. Of course, you are not alone, and your electromagnetic neighbours may not be pleased if you use a large bandwidth, so for social reasons, engineers often have to make do with high-power, narrow-bandwidth transmitters.

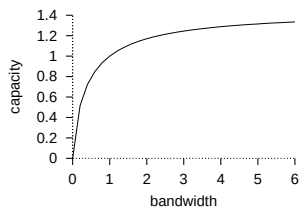


Figure 11.4. Capacity versus bandwidth for a real channel: $C/W_0 = W/W_0 \log(1 + W_0/W)$ as a function of W/W_0.

11.4 What are the capabilities of practical error correcting codes?

Nearly all codes are good, but nearly all codes require exponential look-up tables for practical implementation of the encoder and decoder — exponential in the block length N. And the coding theorem required N to be large.

By a *practical* error-correcting code, we mean one that can be encoded and decoded in a reasonable amount of time, for example, a time that scales as a polynomial function of the block length N — preferably linearly.

The Shannon limit is not achieved in practice

The non-constructive proof of the noisy channel coding theorem showed that good block codes exist for any noisy channel, and indeed that nearly all block codes are good. But writing down an explicit and practical encoder and decoder that is as good as promised by Shannon is still an unsolved problem.

Very good codes. Given a channel, a family of block codes that achieve arbitrarily small probability of error at any communication rate up to the capacity of the channel are called 'very good' codes for that channel.

Good codes are code families that achieve arbitrarily small probability of error at non-zero communication rates up to some maximum rate that may be *less than* the capacity of the given channel.

Bad codes are code families that can only achieve arbitrarily small probability of error by decreasing the information rate to zero. (Bad codes are not necessarily useless for practical purposes.)

Practical codes are code families that can be encoded and decoded in time and space polynomial in the block length.

Most established codes are linear codes

Let us review the definition of a block code, and then add the definition of a linear block code.

An (N, K) Block Code for a channel Q is a list of $M = 2^K$ codewords $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(2^K)}\}$, each of length N: $\mathbf{x}^{(m)} \in \mathcal{A}_X^N$. The signal to be encoded is assumed to come from an alphabet of size 2^K; signal m is encoded as $\mathbf{x}^{(m)}$.

The *rate* of the code is $R = K/N$ bits.

[This definition holds for any channels, not only binary channels.]

A linear (N, K) **block code** is a block code in which all the codewords $\{\mathbf{x}^{(m)}\}$ lie in a K-dimensional subspace of $\mathcal{A}_X^N$. The encoding operation can be represented by an $N \times K$ binary matrix $\mathbf{G}^{\mathsf{T}}$ such that if the signal to be encoded, in binary notation, is $\mathbf{s}$ (a vector of length K bits), then the encoded signal is $\mathbf{G}^{\mathsf{T}}\mathbf{s}$ modulo 2.

For example the (7,4) Hamming code of section 1.2 takes $K = 4$ signal bits, $\mathbf{s}$, and transmits them followed by three parity check bits. The $N = 7$ transmitted symbols are given by $\mathbf{G}^{\mathsf{T}}\mathbf{s} \bmod 2$.

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \tag{11.30}$$

Coding theory was born with the work of Hamming, who wrote down a family of error correcting codes, each able to correct one error in a block of length N, of which the repetition code R_3 and the (7,4) code above are the simplest. Since then most established codes have been generalizations of Hamming's codes: Bose-Chaudhury-Hocquenhem codes, Reed-Müller codes, Reed-Solomon codes, and Goppa codes, to name a few.

Convolutional codes

Another family of linear codes are *convolutional codes*, which do not divide the source stream into blocks, but, instead, read and transmit bits continuously. The transmitted bits are a linear function of the past source bits. Usually the rule for generating the transmitted bits involves feeding the present source bit into a linear feedback shift register of length k, and transmitting one or more linear functions of the state of the shift register at each iteration. The transmitted bit stream is then the convolution of the source stream with some fixed linear filter. The impulse response function of this filter may have finite or infinite duration, depending on the choice of feedback shift register.

We will discuss convolutional codes in chapter 52.

Are linear codes 'good'?

One might ask, is the reason that the Shannon limit is not achieved because linear codes are inherently not as good as random codes? The answer is no, the noisy channel coding theorem can still be proved for linear codes, at least for some channels (see chapter 12), though the proofs, like Shannon's proof for random codes, are non-constructive.

Linear codes are easy to implement at the encoding end. Is decoding a linear code also easy? Not necessarily. The general decoding problem (find the maximum likelihood $\mathbf{s}$ in the equation $\mathbf{G}\mathbf{s}+\mathbf{n} = \mathbf{r}$) is in fact NP-complete. NP-complete problems are computational problems that are all equally difficult and which are widely believed to require exponential computer time to solve in general. So attention focuses on families of codes (such as those listed above) for which there is a fast decoding algorithm.

Concatenation

One trick for building codes with practical decoders is the idea of concatenation.

An encoder-channel-decoder system $\mathcal{C} \rightarrow Q \rightarrow \mathcal{D}$ can be viewed as defining a super-channel Q' with a smaller probability of error, and with complex correlations among its errors. We can create an encoder $\mathcal{C}'$ and decoder $\mathcal{D}'$ for

this super-channel Q'. The code consisting of the outer code $\mathcal{C}'$ followed by the inner code $\mathcal{C}$ is known as a *concatenated code*.

Some concatenated codes make use of the idea of *interleaving*. We read the data in blocks, the size of each block being larger than the block lengths of the constituent codes $\mathcal{C}$ and $\mathcal{C}'$. After encoding the data of one block using code $\mathcal{C}'$, the bits are reordered within the block in such a way that nearby bits are separated from each other once the block is fed to the second code $\mathcal{C}$. A simple example of an interleaver is a *rectangular code* or *product code* in which the data is arranged in a $K_2 \times K_1$ block, and encoded horizontally using an (N_1, K_1) linear code, then vertically using a (N_2, K_2) linear code. Either of the two codes can be viewed as the inner code or the outer code.

The motivation for interleaving is that by spreading out bits that are nearby in one code, we make it possible to ignore the complex correlations among the errors that are produced by the inner code. Maybe the inner code will mess up an entire codeword; but that codeword is spread out one bit at a time over several codewords of the outer code. So we can treat the errors introduced by the inner code as if they are independent.

Other channel models

In addition to the binary symmetric channel and the Gaussian channel, coding theorists keep more complex channels in mind also.

Burst error channels are important models in practice. Reed-Solomon codes use Galois fields (see appendix A.3) with large numbers of elements (*e.g.*, 2^{16}), and thereby automatically achieve a degree of burst error tolerance in that even if 17 successive bits are corrupted, only 2 successive symbols in the Galois field representation are corrupted. Concatenation and interleaving can give further protection against burst errors. The concatenated Reed-Solomon codes used on digital compact discs are able to correct bursts of errors of length 4000 bits.

Exercise 11.3:^{B2} The technique of interleaving, which allows bursts of errors to be treated as independent, is widely used, but is theoretically a poor way to protect data against burst errors, in terms of the amount of redundancy required. Explain why interleaving is a poor method, using the following burst error channel as an example. Time is divided into chunks of length $N = 100$ clock cycles; during each chunk, there is a burst with probability $b = 0.2$; during a burst, the channel is a binary symmetric channel with $f = 0.5$. If there is no burst, the channel is an error-free binary channel. Compute the capacity of this channel and compare it with the maximum communication rate that could conceivably be achieved using interleaving and treating the errors as independent.

Fading channels are real channels like Gaussian channels except that the received power is assumed to vary with time. A moving mobile phone is an important example. The incoming radio signal is reflected off nearby objects so that there are interference patterns and the intensity of the signal received by the phone varies with its location. The received power can easily vary by 10 decibels (a factor of ten) as the phone's antenna moves through a distance similar to the wavelength of the radio signal (a few centimetres).

11.5 The state of the art

What are the best known codes for communicating over Gaussian channels? All the practical codes are linear codes, and can be divided into codes based on convolutional codes and codes based on block codes.

Convolutional codes

Textbook convolutional codes. The 'de facto standard' error correcting code for satellite communications is a convolutional code with constraint length 7.

Concatenated convolutional codes. The above convolutional code can be used as the inner code of a concatenated code whose outer code is a Reed-Solomon code with eight-bit symbols. This code is used in deep space communication systems such as the Voyager spacecraft.

The code for Galileo. A code using the same format but using a longer constraint length for its convolutional code and a larger Reed-Solomon code has been developed by the Jet Propulsion Laboratory. The details of this code are unpublished outside JPL, and the decoding is only possible using special purpose hardware.

Turbo codes. In 1993, Berrou, Glavieux and Thitimajshima reported work on *turbo codes*. The encoder of a turbo code is based on the encoders of two or more constituent codes. In the original paper the constituent codes were two convolutional codes. The source bits are fed into each encoder, the order of the source bits being permuted in a random way, and the resulting parity bits from each constituent code are transmitted. [The random permutation is chosen when the code is designed, and fixed thereafter.]

The decoding algorithm invented by Berrou *et al* involves iteratively decoding each constituent code using its standard decoding algorithm, then using the output of the decoder as the input to the other decoder. This decoding algorithm is an instance of a *probability propagation* algorithm called the *sum-product algorithm*.

Block codes

Gallager's low density parity check codes. The best block codes for Gaussian channels that are currently known were invented by Gallager in 1962 but were promptly forgotten by most of the coding theory community. They were rediscovered in 1995 by MacKay and Neal, and shown to have outstanding theoretical and practical properties.

Recall that a linear block code may be defined in terms of a generator matrix or in terms of a parity check matrix. Gallager's *low density parity check codes* are defined in terms of a parity check matrix that has a small number of non-zero entries in each column, say three (figure 11.5). The number of non-zero entries in each column exceeds one, so this parity check matrix is in 'non-systematic' form. In a 'regular' low density parity check code, the non-zero entries are arranged at random in the matrix subject to

$$\mathbf{H} = \begin{bmatrix}
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} \\
0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Figure 11.5. Example of a low density parity check matrix with weight per column equal to 3. The parity matrix has $N = 16$ columns and $M = 12$ rows so this code is a (16,4) code. Useful performance is obtained when the block length is increased to $N \simeq 10000$.

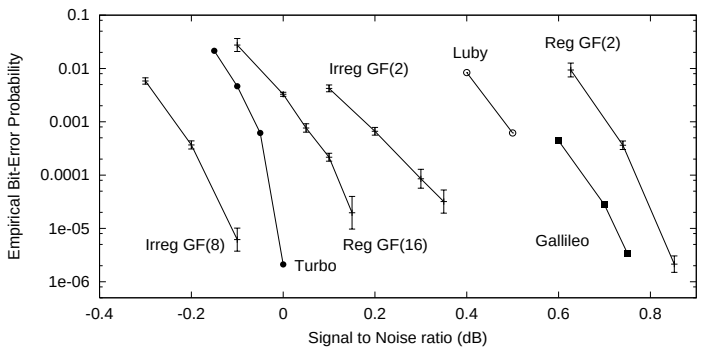


Figure 11.6. State-of-the-art codes. The figure shows the performance of various codes with rate 1/4 over the Gaussian Channel. From left to right: Irregular LDPC over $GF(8)$, blocklength 48000 bits (Davey, 1999); JPL turbo code (JPL, 1996) blocklength 65536; Regular LDPC over $GF(16)$, blocklength 24448 bits (Davey and MacKay, 1998a); Irregular binary LDPC, blocklength 16000 bits (Davey, 1999); Luby *et al.* (Luby *et al.*, 1998)irregular binary LDPC, blocklength 64000 bits; JPL concatenated code based on constraint length 15, rate 1/4 convolutional code (in 1992, this was the best known code of rate 1/4); Regular binary LDPC: blocklength 40000 bits (MacKay, 1999). The Shannon limit is at about -0.79dB.

the constraint that each row should have roughly the same number of non-zero entries, and the number in each column likewise should be roughly uniform.

This simple code construction leads (with a probability that increases to 1 as the block length N increases) to a good code. The decoding of this code is not trivial, and indeed, for sufficiently large noise levels close to the Shannon limit, there is no known algorithm that succeeds. But at highly competitive noise levels a *probability propagation* algorithm called the *sum-product algorithm* succeeds and outperforms the textbook standards by a large margin. We will discuss probability propagation and Gallager codes further in chapters 48 and 51.

Only the Galileo code and turbo codes outperform the original 'regular, binary' Gallager codes. The best known Gallager codes, which are irregular and non-binary, outperform the Galileo code and turbo codes too (Davey and MacKay, 1998a). The performance of the above codes are compared for Gaussian channels in figure 11.6.

11.6 Summary

Random codes are good, but they require exponential resources to encode and decode them.

Non-random codes tend for the most part not to be as good as random codes; encoding them may be easy, but even for simply defined linear codes, the decoding problem remains very difficult.

The best practical codes (a) employ very large block sizes; (b) are based on semi-random code constructions; and (c) make use of probability-based decoding algorithms.

11.7 Nonlinear codes

Most practically used codes are linear, but not all. When a digital soundtrack is written onto cinema film, the amplitude is encoded as a binary digital pattern. The likely errors affecting the film involve dirt and scratches, which produce large numbers of `1`s and `0`s respectively. We therefore want none of the codewords to look like all-`1`s or all-`0`s, so that it will be easy to detect errors caused by dirt and scratches. One of the codes used in digital cinema sound systems is a nonlinear (8,6) code consisting of 64 of the $\binom{8}{4}$ binary patterns of weight 4.

11.8 Errors other than noise

Another source of uncertainty for the receiver is uncertainty about the timing of the transmitted signal $x(t)$. In ordinary coding theory and information theory, the transmitter's time t and the receiver's time u are assumed to be perfectly synchronised. But if the receiver receives a signal $y(u)$, where u, the receiver's time, is an imperfectly known function of t, then the capacity of this channel for communication must be reduced. The theory of such channels is incomplete, compared with the ordinary synchronised channels we have discussed thus far (Levenshtein, 1966; Ferreira *et al.*, 1997); codes for reliable communication over channels with synchronisation errors are an exciting research area (Davey and MacKay, 1999).

11.9 Exercises

The Gaussian channel

Exercise 11.4:^{B2} Consider a Gaussian channel with a real input x, and signal to noise ratio v/σ^2.

(a) What is its capacity C?

(b) If the input is constrained to be binary, $x \in \{\pm\sqrt{v}\}$, what is the capacity C' of this constrained channel?

(c) If in addition the output of the channel is thresholded using the mapping

$$y \to y' = \begin{cases} 1 & y > 0 \\ 0 & y \leq 0 \end{cases} , \tag{11.31}$$

what is the capacity C'' of the resulting channel?

(d) Plot the three capacities above as a function of v/σ^2 from 0.1 to 2. [You'll need to do a numerical integral to evaluate C'.]

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Further reading

Exercise 11.5:^{C2} How do redundant arrays of independent discs (RAID) work? These are information storage systems consisting of about ten disc drives, of which any two or three can be disabled and the others are able to still able to reconstruct any requested file. What codes are used, and how far are these systems from the Shannon limit for the problem they are solving? How would *you* design a better RAID system? See `http://www.acnc.com/raid2.html` and `http://www.dfountain.com/`.

Solutions to Chapter 11's exercises

Solution to exercise 11.1 (p. 199): Introduce a Lagrange multiplier λ for the power constraint and another, μ, for the constraint of normalization of $P(x)$.

$$F = H(X;Y) - \lambda \int dx P(x) x^2 - \mu \int dx P(x) \tag{11.32}$$

$$= \int dx\, P(x) \left[\int dy\, P(y|x) \log \frac{P(y|x)}{P(y)} - \lambda x^2 - \mu \right] \tag{11.33}$$

We make the functional derivative with respect to $P(x^*)$.

$$\frac{\delta F}{\delta P(x^*)} = \int dy\, P(y|x^*) \log \frac{P(y|x^*)}{P(y)} - \lambda x^{*2} - \mu - \int dx\, P(x) \int dy\, P(y|x) \frac{1}{P(y)} \frac{\delta P(y)}{\delta P(x^*)}. \tag{11.34}$$

The final factor $\delta P(y)/\delta P(x^*)$ is found using $P(y) = \int dx\, P(x) P(y|x)$ to be $P(y|x^*)$, and the whole of the last term collapses in a puff of smoke to 1, which can be absorbed into the μ term.

We now substitute $P(y|x) = \exp(-(y-x)^2/2\sigma^2)/\sqrt{2\pi\sigma^2}$ and set the derivative to zero:

$$\int dy P(y|x) \log \frac{P(y|x)}{P(y)} - \lambda x^2 - \mu' = 0 \tag{11.35}$$

$$\Rightarrow \int dy \frac{\exp(-(y-x)^2/2\sigma^2)}{\sqrt{2\pi\sigma^2}} \log [P(y)\sigma] = -\lambda x^2 - \mu' - \frac{1}{2} \tag{11.36}$$

This condition must be satisfied by $\log [P(y)\sigma]$ for all x.

Writing a Taylor expansion of $\log [P(y)\sigma] = a + by + cy^2 + \ldots$, it is evident that it can only be a quadratic function $\log [P(y)\sigma] = a + cy^2$. (Any higher order terms y^p, $p > 2$, would produce terms in x^p that are not present on the right hand side.) Therefore $P(y)$ is Gaussian. Working backwards we observe that we can obtain this optimal output distribution by using a Gaussian input distribution $P(x)$.

Solution to exercise 11.2 (p. 199): Given a Gaussian input distribution of variance v, the output distribution is Normal$(0, v + \sigma^2)$, since x and the noise are independent random variables, and variances add for independent random variables. The mutual information is:

$$H(X;Y) = \int dx\, dy\, P(x) P(y|x) \log P(y|x) - \int dy\, P(y) \log P(y) \tag{11.37}$$

$$= \frac{1}{2} \log \frac{1}{\sigma^2} - \frac{1}{2} \log \frac{1}{v + \sigma^2} \tag{11.38}$$

$$= \frac{1}{2} \log \frac{v + \sigma^2}{\sigma^2} = \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2} \right). \tag{11.39}$$

Solution to exercise 11.3 (p. 203): The capacity of the channel is one minus the information content of the noise that it adds. That information content is, per chunk, the entropy of the selection of whether the chunk is bursty, $H_2(b)$, plus, with probability b, the entropy of the flipped bits, N, which adds up to $H_2(b) + Nb$ per chunk. So, per bit, the capacity is

$$C = 1 - \left(\frac{1}{N} H_2(b) + b \right) = 1 - 0.207 = 0.793. \tag{11.40}$$

In contrast, interleaving, which treats bursts of errors as independent, causes the same channel to be treated as a binary symmetric channel with $f = 0.1$, whose capacity is about 0.53.

Interleaving throws away the useful information about the correlatedness of the errors. Theoretically, we should be able to communicate about $(0.79/0.53) \simeq 1.6$ times faster using a code and decoder that explicitly treat bursts as bursts.

Solution to exercise 11.4 (p. 206):

(a) Putting together the results of exercises 11.1 and 11.2, we deduce that a Gaussian channel with real input x, and signal to noise ratio v/σ^2 has capacity

$$C = \frac{1}{2} \log \left(1 + \frac{v}{\sigma^2} \right). \tag{11.41}$$

(b) If the input is constrained to be binary, $x \in \{\pm\sqrt{v}\}$, the capacity is achieved by using these two inputs with equal probability. The capacity is reduced to a somewhat messy integral,

$$C'' = \int_{-\infty}^{\infty} \mathrm{d}y \, N(y;0) \log N(y;0) - \int_{-\infty}^{\infty} \mathrm{d}y \, P(y) \log P(y) \tag{11.42}$$

where $N(y;x) \equiv (1/\sqrt{2\pi}) \exp[(y-x)^2/2]$, $x \equiv \sqrt{v}/\sigma$, and $P(y) \equiv [N(y;x) + N(y;-x)]/2$. This capacity is smaller than the unconstrained capacity (11.41), but for small signal to noise ratio, the two capacities are close in value.

(c) If the output is thresholded, then the Gaussian channel is turned into a binary symmetric channel whose transition probability is given by the error function Φ defined in section 2b. The capacity is

$$C'' = 1 - H_2(f), \text{ where } f = \Phi(\sqrt{v}/\sigma). \tag{11.43}$$

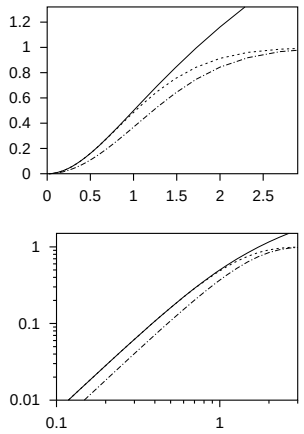


Figure 11.7. Capacities (from top to bottom in each graph) C, C', and C'', versus the signal to noise ratio $(\sqrt{v}/\sigma)$. The lower graph is a log-log plot.

Solution to exercise 11.5 (p. 207): There are several RAID systems. One of the easiest to understand consists of 7 disc drives which store data at rate 4/7 using a (7,4) Hamming code: each successive four bits are encoded with the code and the seven codeword bits are written one to each disc. Two or perhaps three disc drives can go down and the others can recover the data. The effective channel model here is a binary erasure channel, because it is assumed that we can tell when a disc is dead.

It is not possible to recover the data for *some* choices of the three dead disc drives; can you see why?

Exercise 11.6: Give an example of three disc drives which, if lost, lead to failure of the above RAID system, and three which can be lost without failure.

Solution to exercise 11.6 (p. 210): The Hamming code has codewords of weight 3. If any set of three disc drives corresponding to one of those codewords is lost, then the other four discs can only recover 3 bits of information about the four source bits; a fourth bit is lost.

Any other set of three disc drives can be lost without problems because the corresponding four by four submatrix of the generator matrix is invertible.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

12

Linear Error Correcting Codes and Perfect Codes

Include erasure-correcting codes in this chapter, and see xword.tex for some word pairs. Could mention idea of phone numbers, and privileged users whose numbers are hard to misdial, or whose numbers are unlikely to be reached in error. Discuss perfect codes for noisy channels and for erasure channels.

1. Define perfect codes.
2. Prove that there are no big perfect codes because Shannon shows that communication is possible at rates where the error probability must be non–zero.
3. Define weight enumerator function.
4. Evaluate expected weight enumerator function of random code.
5. Define Gilbert bound. Define the Gilbert distance and the Gilbert rate.
6. Use random linear codes to prove coding theorem for symmetric channels, and symmetric channels with memory.
7. Concatenation
8. Discuss Berlekamp's bat in a cave analogy.
9. Finding the distance of a code is NP-complete. Using hash codes to find the distance of a code.

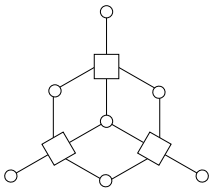
12.1 Weight enumerator function

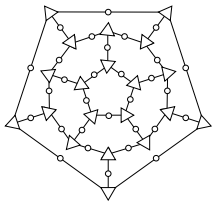
12.2 Concatenation of Hamming codes

Consider a binary symmetric channel with noise density f. The Hamming codes are all single–error correcting codes and form a sequence as follows

M		number of parity check constraints
N	$N = 2^M - 1$	block length
K	$K = N - M$	number of source bits
d	3	minimum distance
p_B	$p_B = \frac{3}{N}\binom{N}{2} f^2$	probability of block error to leading order

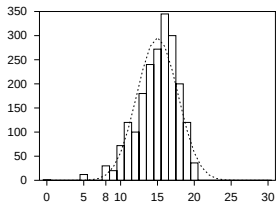
© David J.C. MacKay: Draft 2.0.7. February 14, 2000

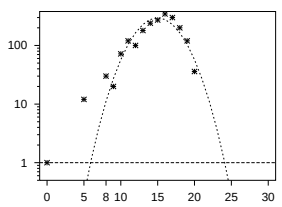


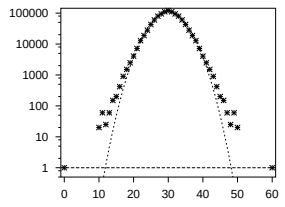


w	$A(w)$
0	1
3	7
4	7
7	1
Total	16

w	$A(w)$
0	1
5	12
8	30
9	20
10	72
11	120
12	100
13	180
14	240
15	272
16	345
17	300
18	200
19	120
20	36
Total	2048







a (60,20) code:

Figure 12.1. The graph of the (7,4) Hamming code. (a) The graph defining the (30,11) dodecahedron code. The circles are the 30 bits and the triangles and the 20 parity checks, one of which is redundant. (b) The weight enumerator function of the code (solid lines), and the average weight enumerator function of a random linear code with the same size of generator matrix (dotted lines). (c) Same with log scale.

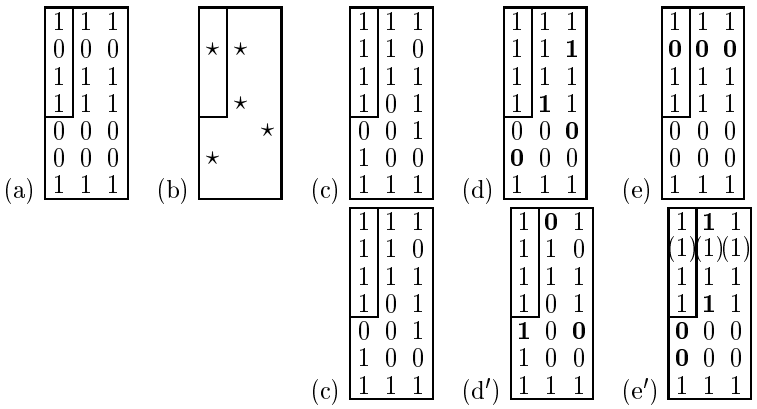


Figure 12.2. (a) A string (1,0,1,1) encoded using a concatenated code consisting of two Hamming codes, $H(3,1)$ and $H(7,4)$. (b) a noise pattern that flips 5 bits. (c) The received vector. (d) After decoding using the (3,1) horizontal decoder, and (e) after using the (7,4) vertical decoder. The decoded vector matches the original.
(d′, e′) After decoding in the other order, three errors still remain.

The first few Hamming codes have the following rates:

checks M	(block length, source bits) $(N = 2^M - 1\ ,\ K = N - M)$	rate $R = K/N$	
2	(3,1)	1/3	repetition code R_3
3	(7,4)	4/7	(7,4) Hamming code
4	(15,11)	11/15	
5	(31,26)	26/31	
6	(63,57)	57/63	

We can create a concatenated code by encoding with several Hamming codes in succession. Figure 12.2 shows a rectangular scheme in which we encode first with $H(3,1)$ horizontally then with $H(7,4)$ vertically. The block length of the concatenated code is 27. The number of source bits per codeword is four, shown by the heavy rectangle. The code would be equivalent if we encoded first with $H(7,4)$ and second with $H(3,1)$.

We can decode conveniently (though not optimally) by using the individual decoders for each of the subcodes in some sequence. It makes most sense to first decode the code which has the lowest rate and hence the greatest error–correcting ability.

Figure **??** shows what happens if we receive the codeword of figure 12.2 with some errors (five bits flipped, as shown) and apply the decoder for $H(3,1)$ first, and then the decoder for $H(7,4)$. The first decoder corrects three of the errors, but erroneously modifies the third bit in the second row where there are two bit errors. The (7,4) decoder can then correct all three of these errors.

Figure **??** shows what happens if we decode the two codes in the other order. In columns one and two there are two errors, so the (7,4) decoder introduces two extra errors. It corrects the one error in column 3. The (3,1) decoder then cleans up four of the errors, but erroneously infers the second bit.

Proving that good codes can be made by concatenation

If we make a product code by concatenating a sequence of C Hamming codes with increasing M, we can choose those $\{M_c\}_{c=1}^{C}$ in such a way that the rate

of the product code R_C

$$R_C = \prod_{c=1}^{C} \frac{N_c - M_c}{N_c} \tag{12.1}$$

tends to a non–zero limit as C increases. For example, if we simply set $M_1 = 2$, $M_2 = 3$, $M_3 = 4$, etc., then the rate is..... *FIGURES HERE.*

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part III

Further Topics in Information Theory

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

13

*Units of information content**

The information content of an outcome, x, whose probability is $P(x)$, is defined to be

$$h(x) = \log \frac{1}{P(x)}. \tag{13.1}$$

The entropy of an ensemble is an average information content,

$$H(X) = \sum_x P(x) \log \frac{1}{P(x)}. \tag{13.2}$$

When we compare hypotheses with each other in the light of data, it is often convenient to compare the log of the probability of the data under the alternative hypotheses,

$$\text{`log evidence for } \mathcal{H}_i\text{'} = \log P(D|\mathcal{H}_i), \tag{13.3}$$

or, in the case where just two hypotheses are being compared, we evaluate the 'log odds',

$$\log \frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)}, \tag{13.4}$$

which has also been called the 'weight of evidence in favour of $\mathcal{H}_1$'. The log evidence for a hypothesis, $\log P(D|\mathcal{H}_i)$ is the negative of the information content of the data D: if the data have large information content, given a hypotheses, then they are surprising; if some other hypothesis is not so surprised by the data, then that hypothesis becomes more probable.

All these quantities are logarithms of probabilities, or weighted sums of logarithms of probabilities, so they can all be measured in the same units. The units depend on the choice of the base of the logarithm.

This chapter is a brief aside to mention the names that have been given to these units.

Unit	Expression that has those units
bit	$\log_2 p$
nat	$\log_e p$
ban	$\log_{10} p$
deciban (db)	$10 \log_{10} p$

Figure 13.1. Units of measurement of information content

The *bit* is the unit that we use most in this book. Because the word 'bit' has other meanings, a backup name for this unit is the *shannon*. A *byte* is 8 bits. A megabyte is $2^{20} \simeq 10^6$ bytes. If one works in natural logarithms, information contents and weights of evidence are measured in *nats*. The most interesting units are the *ban* and the *deciban*. Let me tell you why a factor of ten in probability is called a ban. When Turing and the other codebreakers at Bletchley Park were breaking each new day's Enigma code, their task was a huge inference problem: to infer, given the day's cypher text, which three wheels were in the Enigma machines that day; what their starting positions were; what further letter substitutions were in use on the steckerboard; and, not least, what the original German messages were. These inferences were conducted using Bayesian methods (of course!), and the chosen units were decibans or half-decibans, the deciban being judged the smallest weight of evidence discernible to a human. The evidence in favour of particular hypotheses was tallied using sheets of paper, which were specially printed in Banbury, a town about 30 miles from Bletchley. The inference task was known as Banburismus, and the units in which the game was played were called bans, after that town.

13.1 A taste of Banburismus

The details of the code-breaking methods of Bletchley Park were kept secret for a long time, but some aspects of Banburismus can be pieced together. I hope the following description of a small part of Banburismus is not too inaccurate.[1]

How much information was needed? The number of possible settings of the Enigma machine was about 8×10^{12}. To deduce the state of the machine, it was therefore necessary to find about 129 decibans from somewhere, as Good puts it. Banburismus was aimed not at deducing the entire state of the machine, but only at figuring out which wheels were in use; the logic-based Bombes, fed with guesses of the plaintext (cribs), were then be used to crack the remaining uncertainty.

The Enigma machine, once its wheels and plugs were put in place, implemented a continually-changing permutation cypher that wandered deterministically through a state space of 26^3 possible permutations. Because an enormous number of messages were sent each day, there was a good chance that whatever state one machine was in when sending a message, there would be another machine *in the same state* while sending another message. Because the evolution of the machine's state was deterministic, the two machines would remain in the same state as each other thereafter. The task I will discuss here is the task of detecting that two messages came from machines following the same state sequence.

The hypotheses are the null hypothesis, $\mathcal{H}_0$, which states that the machines are in different states, and the two plain messages are unrelated; and the 'match' hypothesis, $\mathcal{H}_1$, which says that the machines are in the same state, and the two plain messages are unrelated. No attempt is being made here to infer what the state of either machine is. The data provided are the two

[1] I've been most helped by Tony Sale (`http://www.cranfield.ac.uk/ccc/bpark/lectures/`) and by Jack Good, who worked with Turing at Bletchley.

```
u  THEXCODEXBREAKERSXWEREXLOOKINGXFORXINSTANCESXWHEREX
v  TRIGRAMSXFORXTWOXORXMOREXMESSAGESXDIFFEREDXONLYXINX
   *.......*.........................*............*
```

Figure 13.2. Two aligned pieces of English plaintext, **u** and **v**, with matches marked by ∗. Notice that there are four matches, whereas the expected number of matches in two completely random strings of length $T = 51$ would be about 2.

cypherstreams $\mathbf{x}$ and $\mathbf{y}$; let's assume they both have length T and that the alphabet size is A (26 in Enigma). What is the probability of the data, given the two hypotheses?

First, the null hypothesis. This hypothesis asserts that the two cypherstreams are given by

$$\mathbf{x} = x_1 x_2 x_3 \ldots = c_1(u_1)c_2(u_2)c_3(u_3)\ldots \tag{13.5}$$

and

$$\mathbf{y} = y_1 y_2 y_3 \ldots = c'_1(v_1)c'_2(v_2)c'_3(v_3)\ldots, \tag{13.6}$$

where the codes c_t and c'_t are two unrelated time-varying permutations of the alphabet, and $u_1u_2u_3\ldots$ and $v_1v_2v_3\ldots$ are the plaintext messages. An exact computation of the probability of the data $(\mathbf{x}, \mathbf{y})$ would depend on a language model of the plain text, and a model of the Enigma machine's guts, but if we assume that each Enigma machine is an *ideal* random time-varying permutation, then the probability distribution of the two cypherstreams is uniform. All cypherstreams are equally likely.

$$P(\mathbf{x}, \mathbf{y}|\mathcal{H}_0) = \left(\frac{1}{A}\right)^{2T} \quad \text{for all } \mathbf{x}, \mathbf{y} \text{ of length } T. \tag{13.7}$$

What about $\mathcal{H}_1$? This hypothesis asserts that a *single* time-varying permutation c_t underlies both

$$\mathbf{x} = x_1 x_2 x_3 \ldots = c_1(u_1)c_2(u_2)c_3(u_3)\ldots \tag{13.8}$$

and

$$\mathbf{y} = y_1 y_2 y_3 \ldots = c_1(v_1)c_2(v_2)c_3(v_3)\ldots . \tag{13.9}$$

What is the probability of the data $(\mathbf{x}, \mathbf{y})$? We have to make some assumptions about the plaintext language. If it were the case that the plaintext language was completely random, then the probability of $u_1u_2u_3\ldots$ and $v_1v_2v_3\ldots$ would be uniform, and so would that of $\mathbf{x}$ and $\mathbf{y}$, so the two hypotheses would be indistinguishable. We make progress by assuming that the plaintext is not completely random. Particular plaintext letters are used more often than others. So, even though the two plaintext messages are unrelated, they are slightly more likely to use the same letters as each other; so if $\mathcal{H}_1$ is true, two synchronized letters from the two cypherstreams are slightly more likely to be identical. Similarly, if a language uses particular bigrams and trigrams frequently, then the two plaintext messages will occasionally contain the same trigrams at the same time as each other, giving rise, if $\mathcal{H}_1$ is true, to a little burst of 3 identical letters.

The codebreakers hunted among pairs of messages for pairs that were suspiciously similar to each other, counting up the numbers of matching monograms, bigrams, trigrams, etc.. The method was first used by the Polish codebreaker Rejewski.

Let's look at the simple case of a monogram language model and estimate how many messages would be needed, and of what length, to have a good chance of cracking the Enigma. I'll assume the source language is monogram-English, the language in which successive letters are drawn i.i.d. from the probability distribution $\{p_i\}$ of figure 2.1. The probability of $\mathbf{x}$ and $\mathbf{y}$ is non-uniform: consider two single characters, $x_t = c_t(u_t)$ and $y_t = c_t(v_t)$; the probability that they are identical is

$$\sum_{u_t, v_t} P(u_t) P(v_t) \delta(u_t = v_t) = \sum_i p_i^2 \equiv m. \tag{13.10}$$

We give this quantity the name m, for 'match probability'; for both English and German, m is about $2/26$ rather than $1/26$. Assuming that c_t is an ideal random permutation, the probability of x_t and y_t is, by symmetry,

$$P(x_t, y_t|\mathcal{H}_1) = \begin{cases} \frac{m}{A} & \text{if } x_t = y_t \\ \frac{(1-m)}{A(A-1)} & \text{for } x_t \neq y_t \end{cases} \tag{13.11}$$

Given a pair of cypherstrings $\mathbf{x}$ and $\mathbf{y}$ of length T that match in M places and do not match in N places, the log evidence in favour of $\mathcal{H}_1$ is then

$$\log \frac{P(\mathbf{x}, \mathbf{y}|\mathcal{H}_1)}{P(\mathbf{x}, \mathbf{y}|\mathcal{H}_0)} = M \log \frac{m/A}{1/A^2} + N \log \frac{(1-m)/A(A-1)}{1/A^2} \tag{13.12}$$

$$= M \log mA + N \log \frac{(1-m)A}{A-1} \tag{13.13}$$

Every match contributes $\log mA$ in favour of $\mathcal{H}_1$; every non-match contributes $\log \frac{A-1}{(1-m)A}$ in favour of $\mathcal{H}_0$.

Match probability for monogram-English	m	0.076
Coincidental match probability	$1/A$	0.037
log-Evidence for $\mathcal{H}_1$ per match	$10 \log_{10} mA$	3.1 db
log-Evidence for $\mathcal{H}_1$ per non-match	$10 \log_{10} \frac{(1-m)A}{(A-1)}$	-0.18 db

If there were $M = 4$ matches and $N = 47$ non-matches in a pair of length $T = 51$, for example, the weight of evidence in favour of $\mathcal{H}_1$ would be $+4$ decibans, or a likelihood ratio of 2.5 to 1 in favour.

The *expected* weight of evidence from a line of text of length $T = 20$ characters is the expectation of (13.13), which depends on whether $\mathcal{H}_1$ or $\mathcal{H}_0$ is true. If $\mathcal{H}_1$ is true, then matches are expected to turn up at rate m, and the expected weight of evidence is 1.4 decibans per 20 characters. If $\mathcal{H}_0$ is true, then spurious matches are expected to turn up at rate $1/A$, and the expected weight of evidence is -1.1 decibans per 20 characters.

Using better language models, the evidence contributed by runs of matches was more accurately computed. The scoring system was worked out by Turing and refined by Good. According to Good, the longest false-positive that arose in this work was a string of 8 consecutive matches.

The same codebreaking system was implemented on the Colossus computer in the work known as Fish. The computer accumulated weights of evidence and searched for the most probable hypothesis.

14

*Language models and crosswords**

This chapter belongs as close as possible to the compression and noisy channel chapters. But it would also go better after the constrained channel chapter, since language can be viewed as a constrained channel.

Let's draw together the ideas of error correction and data compression by discussing language.

Languages, whether spoken or written, are error–correcting codes. The set of valid English sentences is a tiny subset of the set of all possible strings of text, just as the codewords of a binary error–correcting code of length N are a tiny subset of the set $\{0,1\}^N$. The error-correcting codes that languages are most like are convolutional codes, in which the redundancy is embodied in local grammatical rules.

The redundancy in language allows communication to take place in the presence of noise, and it makes it possible to distinguish a piece of text from other strings. (If there were a language with no redundancy, then any string in another language would be meaningful in that language.)

Examples of redundancy in English that allows error-correction. Hard to find words that have Hamming distance one. Distance 1: `phasers`,`chasers`. `ministries`,`miniseries`. `clatter`,`chatter`. `prance`,`prince`.

Distance 2: `phrase`,`please`. `gratitude`,`platitude`. `grateful`,`plateful`.

If we are blessed with a noise-free channel then there is no need for the redundancy. Predictive encoders for compression. for data-entry. Steganography. Two types. Transmit a genuine message but use degrees of freedom to transmit a second message. (Watermark.) Second type is to use innocent words to encode data. The actual message transmitted is not meaningful. Crosswords. A form of entertainment that relies on redundancy in language.

Figure 14.1. Crosswords of types A (American) and B (British)

14.1 Crosswords

There are two archetypal crossword formats. In a 'type A' crossword, every row and column consists of a succession of words of length 2 or more separated by one or more spaces. In a 'type B' crossword, each row and column consists of a mixture of words and single characters, separated by one or more spaces, and every character lies in at least one word (horizontal or vertical). Whereas in a type A crossword every letter lies in a horizontal word and a vertical word, in a typical type B crossword only about half of the letters do so; the other half lie in one word only. ['A' and 'B' are mnemomic for America and Britain, where these two types of crosswords are respectively more widespread.] Type

A crosswords are harder to create than type B because of the constraint that they are subject to, that no single characters are permitted. Likewise, type B crosswords are generally harder to solve because there are fewer constraints per unit area.

Why are crosswords possible?

If a language has no redundancy, then any letters written on a grid form a valid crossword. Perhaps we should introduce a name like Winglish for 'word-English', the 'language' consisting of strings of words from a dictionary separated by spaces. What is the capacity of Winglish, in bits per character? (relate to telephone rings and chapter on constrained channels). Give an estimate. The fact that many crosswords can be made leads to a lower bound on the entropy of Winglish.

For simplicity, let's assume Winglish consists of W words all of length L. The entropy of such a language, per character, including inter-word spaces, is easy to write down:

$$H_W \equiv \frac{\log_2 W}{L+1} \tag{14.1}$$

We'll find that the conclusions we come to depend on the value of H_W and are not terribly sensitive to the value of L. Consider a large crossword of size S squares in area. Let the number of words be $f_w S$ and let the number of letter-occupied squares be $f_1 S$. For typical crosswords of types A and B made of words of length L, the two fractions f_w and f_1 might have the following values:

	A	B
f_w	$\frac{2}{L+1}$	$\frac{1}{L+1}$
f_1	$\frac{L}{L+1}$	$\frac{3}{4}\frac{L}{L+1}$

We'll now estimate how many crosswords there are of size S using a simple model of Winglish. Let's assume that Winglish is created at random by generating W strings from a monogram (*i.e.*, memoryless) source with entropy H_0. If, for example, the source used all $A = 26$ characters with equal probability then $H_0 = \log_2 A = 4.xxx$ bits. If instead we use chapter 2's distbn then 3.xxx.

Let's now count how many crosswords there are by imagining filling in the squares of a crossword at random using the same distribution that produced the Winglish dictionary and evaluating the probability that random scribbling produces valid words in all rows and columns. The total number of *typical* fillings-in of the $f_1 S$ squares in the crossword that can be made is

$$|T| = 2^{f_1 S H_0}. \tag{14.2}$$

The probability that one word of length L is validly filled-in is

$$\beta = \frac{W}{2^{L H_0}}, \tag{14.3}$$

and the probability that the whole crossword, made of $f_w S$ words, is validly filled-in by a single typical in-filling is

$$\beta^{f_w S}, \tag{14.4}$$

So the log of the number of valid crosswords of size S is estimated to be

$$\log \beta^{f_w S}|T| = S\left[(f_l - f_w L)H_0 + f_w \log W\right] \log \beta^{f_w S}|T| \quad (14.5)$$
$$= S\left[(f_l - f_w L)H_0 + f_w(L+1)H_w\right] \quad (14.6)$$

which is an increasing function of S, indicating that arbitrarily many crosswords can be made, only if

$$(f_1 - f_w L)H_0 + f_w(L+1)H_w > 0 \quad (14.7)$$

i.e., if there's enough words in the Winglish dictionary that

$$H_W > \frac{(f_w L - f_1)}{f_w(L+1)} H_0. \quad (14.8)$$

Plug in the values of f_1 and f_w from previous page.

	A	B
f_w	$\frac{2}{L+1}$	$\frac{1}{L+1}$
$f_w(L+1)$	2	1
f_1	$\frac{L}{L+1}$	$\frac{3}{4}\frac{L}{L+1}$
$-f_1 + f_w L$	$\frac{L}{L+1}$	$\frac{1}{4}\frac{L}{L+1}$
Condition for crosswords	$H_W > \frac{1}{2}\frac{L}{L+1}H_0$	$H_W > \frac{1}{4}\frac{L}{L+1}H_0$

Thus crosswords are possible in English because English has high enough entropy.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

About Chapter 15

In chapters 1–11, we concentrated on two aspects of information theory and coding theory: source coding — the compression of information so as to make efficient use of data transmission and storage channels; and channel coding — the redundant encoding of information so as to be able to detect and correct communication errors.

In both these areas we started by ignoring practical considerations, concentrating on the question of the theoretical limitations and possibilities of coding. We then discussed practical source-coding and channel-coding schemes, shifting the emphasis towards computational feasibility. But the prime criterion for comparing encoding schemes remained the efficiency of the code in terms of the channel resources it required: the best source codes were those that achieved the greatest compression; the best channel codes were those that communicated at the highest rate with a given probability of error.

In this chapter we now shift our viewpoint a little, thinking of *ease of information retrieval* as a primary goal. It turns out that the random codes which were theoretically useful in our study of channel coding are also useful for rapid information retrieval.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

15

Hash Codes: Codes for Efficient Information Retrieval

15.1 The information retrieval problem

A simple example of an information retrieval problem is the task of implementing a phone directory service, which, in response to a person's *name*, returns (a) a confirmation that that person is listed in the directory; and (b) the person's phone number and other details. We could formalize this problem as follows, with S being the number of names that must be stored in the directory.

You are given a list of S binary strings of length N bits, $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(S)}\}$, where S is considerably smaller than the total number of possible strings, 2^N. We will call the superscript 's' in $\mathbf{x}^{(s)}$ the *record number* of the string. The idea is that s runs over customers in the order in which they are added to the directory and $\mathbf{x}^{(s)}$ is the name of customer s. We assume for simplicity that all people have names of the same length. The name length might be, say, $N = 200$ bits, and we might want to store the details of ten million customers, so $S \simeq 10^7 \simeq 2^{23}$. We will ignore the possibility that two customers have identical names.

The task is to construct the inverse of the mapping from s to $\mathbf{x}^{(s)}$, *i.e.*, to make a system which, given a string $\mathbf{x}$, returns the value of s such that $\mathbf{x} = \mathbf{x}^{(s)}$ if one exists, and otherwise reports that no such s exists. (Once we have the record number, we can go and look in memory location s in a separate phone number memory to find the required phone number.) The aim, when solving this task, is to use minimal computational resources in terms of the amount of memory used to store the inverse mapping from $\mathbf{x}$ to s and the amount of time to consult the inverse mapping. And, preferably, the inverse mapping should be implemented such that the addition of further new strings to the directory can be done in a small amount of computer time too.

Some standard solutions

The simplest and dumbest solutions to the information retrieval problem are a look-up table and a raw list.

The look-up table is a piece of memory of size $2^N|S|$, where $|S|$ is the amount of memory required to store an integer between 1 and S. In each of

the 2^N locations, we put a zero, except for the locations $\mathbf{x}$ that correspond to strings $\mathbf{x}^{(s)}$, into which we write the value of s.

The look-up table is a simple and quick solution, if only there is sufficient memory for the table, and the cost of looking up entries in memory is independent of the memory size. But in our definition of the task, we assumed that N is about 200 bits or more, so the amount of memory required would be of size 2^{200}; this solution is completely out of the question. Bear in mind that the number of particles in the solar system is only about 2^{190}.

The raw list is a simple list of ordered pairs $(s, \mathbf{x}^{(s)})$ ordered by the value of s. The mapping from $\mathbf{x}$ to s is achieved by searching through the list of strings, starting from the top, and comparing the incoming string $\mathbf{x}$ with each record $\mathbf{x}^{(s)}$ until a match is found. This system is very easy to maintain, and uses a small amount of memory, about SN bits, but is rather slow to use.

Exercise 15.1:^{B2} Show that the average time taken to find the required string in a raw list, assuming that the original names were chosen at random, is about $S/2 + N$ binary comparisons. (Note that you don't have to compare the whole string of length N; show that you need on average one binary comparison per incorrect string match.) Compare this with the worst case search time — assuming that the devil chooses the set of strings and the search key.

The standard way in which phone directories are made involves an alphabetically ordered list.

Alphabetized list. The strings $\{\mathbf{x}^{(s)}\}$ are sorted into alphabetical order. Searching for an entry now usually takes less time than was needed for the raw list because we can take advantage of the sortedness; for example, we can open the phonebook at its middle page, and compare the name we find there with the target string; if the target is 'greater' than the middle string then we know that the required string, if it exists, will be found in the second half of the alphabetical directory. Otherwise, we look in the first half. By iterating this splitting-in-the-middle procedure, we can identify the target string, or establish that the string is not listed, in $\lceil \log_2 S \rceil$ string comparisons. The expected number of binary comparisons per string comparison will tend to increase as the search progresses, but the total number of binary comparisons required will be no greater than $\lceil \log_2 S \rceil N$.

The amount of memory required is the same as that required for the raw list.

Adding new strings to the database requires that we insert them in the correct location in the list. To find that location takes about $\lceil \log_2 S \rceil$ binary comparisons.

Can we improve on the well-established alphabetized list? Let us consider our task from some new viewpoints.

The task is to construct a mapping from N bits $(\mathbf{x})$ to $\log_2 S$ bits (s). This is a pseudo-invertible mapping, since for any $\mathbf{x}$ that maps to a non-zero s, the

customer database contains the pair $(s, \mathbf{x}^{(s)})$ that takes us back. Where have we come across the idea of mapping from N bits to M bits before?

We encountered this idea twice: first, in source coding, we studied block codes which were mappings from strings of N symbols to a selection of one label in a list. The task of information retrieval is similar to the task (which we never actually solved) of setting up an encoder for a typical-set code. The second time that we mapped bit strings to bit strings of another dimensionality was when we studied channel codes. There, we considered codes which mapped from K bits to N bits, with N greater than K, and we made theoretical progress using *random* codes.

In hash codes, we put together these two notions. We will study random codes that map from N bits to M bits where M is *smaller* than N.

The idea is that we will map the original high-dimensional space down into a lower-dimensional space, one in which it is feasible to implement the dumb look-up table method which we rejected a moment ago.

15.2 A hash code

First we will describe how a hash code works, then we will study the properties of idealized hash codes. A hash code implements a solution to the information retrieval problem, that is, a mapping from $\mathbf{x}$ to s, with the help of a pseudo-random function called a *hash function* that maps the N-bit string $\mathbf{x}$ to an M-bit string $\mathbf{h}(\mathbf{x})$, where M is smaller than N. M is typically chosen such that the 'table size' $T \simeq 2^M$ is a little bigger than S — say, ten times bigger. For example, if we were expecting S to be about a million, we might map $\mathbf{x}$ into a 30-bit hash $\mathbf{h}$. [Regradless of the size N of each item $\mathbf{x}$.] The hash function is some fixed deterministic function which should ideally be indistinguishable from a fixed random code. For practical purposes, the hash function must be quick to compute.

Two simple examples of hash functions are

Division method. The table size T is a prime number, preferably one that is not close to a power of 2. The hash value is the remainder when the integer $\mathbf{x}$ is divided by T.

Variable string addition method. This method assumes that $\mathbf{x}$ is a string of bytes and that the table size T is 256. The characters of $\mathbf{x}$ are added, modulo 256. This hash function has the defect that it maps strings that are anagrams of each other onto the same hash.

It may be improved by putting the running total through a fixed pseudo-random permutation after each character is added.

In the *variable string exclusive-or method* with table size ≤ 65536, the string is hashed twice in this way, with the initial running total being set to 0 and 1 respectively. The result is a 16-bit hash.

Having picked a hash function $\mathbf{h}(\mathbf{x})$, we implement an efficient information retriever as follows.

Encoding. A piece of memory called the *hash table* is created of size $2^M b$, where b is the amount of memory needed to represent an integer between 0 and S. This table is initially set to zero throughout. Each memory

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

```
unsigned char Rand8[256];              /* this array contains a random
                                          permutation from 0..255 to 0..255 */
int Hash(char *str) {
     int h;
     unsigned char h1, h2;

     if (*str == 0) return 0;
     h1 = *str; h2 = *str + 1;         /* initialize two hashes             */
     str++;                            /* Proceed to the next character     */
     while (*str) {
         h1 = Rand8[h1 ^ *str];        /* exclusive-or with the two hashes  */
         h2 = Rand8[h2 ^ *str];        /*    and put through the randomizer */
         str++;
     }
     h = ((int)(h1)<<8) | (int) h2 ;   /* shift h1 left 8 bits and add h2   */
     return h ;
}
```

Figure 15.1. `C` code implementing the variable string exclusive-or method to create from a string `str` a hash `h` in the range $0 \ldots 65535$. Author: Thomas Niemann.

$\mathbf{x}^{(s)}$ is put through the hash function, and at the location in the hash table corresponding to the resulting vector $\mathbf{h}^{(s)} = \mathbf{h}(\mathbf{x}^{(s)})$, the integer s is written — unless that entry in the hash table is already occupied, in which case we have a *collision* of $\mathbf{x}^{(s)}$ and some earlier $\mathbf{x}^{(s')}$ which both happen to have the same hash code. Collisions can be handled in various ways — we will discuss some in a moment — but first let us complete the basic picture.

Decoding. To retrieve a piece of information corresponding to a target vector $\mathbf{x}$, we compute the hash $\mathbf{h}$ of $\mathbf{x}$ and look at the corresponding location in the hash table. If there is a zero, then we know immediately that the string $\mathbf{x}$ is not in the database. The cost of this answer is the cost of one hash function evaluation and one look-up in the table of size 2^M. If, on the other hand, there is a non-zero entry s in the table, there are two possibilities: either the vector $\mathbf{x}$ is indeed equal to $\mathbf{x}^{(s)}$; or the vector $\mathbf{x}^{(s)}$ is another vector that happens to have the same hash code as the target $\mathbf{x}$. (A third possibility is that this non-zero entry might have something to do with our yet-to-be-discussed collision-resolution system.)

To check whether $\mathbf{x}$ is indeed equal to $\mathbf{x}^{(s)}$, we take the tentative answer s, look up $\mathbf{x}^{(s)}$ in the original forward database, and compare it bit by bit with $\mathbf{x}$; if it matches then we report s as the desired answer. This successful retrieval has an overall cost of one hash-function evaluation, one look-up in the table of size 2^M, another look-up in a table of size S, and N binary comparisons.

Exercise 15.2:^{B2} If we have checked the first few bits of $\mathbf{x}^{(s)}$ with $\mathbf{x}$ and found them to be equal, what is the probability that the correct entry has been retrieved, if the alternative hypothesis is that $\mathbf{x}$ is actually not in the database? Assume that the hash function is a random hash function. How many binary evaluations are needed to be sure with odds of a billion to one that the correct entry has been retrieved?

Notice that the hashing method of information retrieval can be used for strings $\mathbf{x}$ of arbitrary length, if the hash function $\mathbf{h}(\mathbf{x})$ can be applied to strings of any length.

15.3 Collision resolution

We will study two ways of resolving collisions: appending in the table, and storing elsewhere.

Appending in table

When encoding, if a collision occurs, we continue down the hash table and write the value of s into the next available location in memory that currently contains a zero. If we reach the bottom of the table before encountering a zero, we continue from the top.

When decoding, if we compute the hash code for $\mathbf{x}$ and find that the s contained in the table doesn't point to an $\mathbf{x}^{(s)}$ that matches the cue $\mathbf{x}$, we continue down the hash table until we either find an s whose $\mathbf{x}^{(s)}$ does match the cue $\mathbf{x}$, in which case we are done, or else encounter a zero, in which case we know that the cue $\mathbf{x}$ is not in the database.

For this method, it is essential that the table be substantially bigger in size than S. If $2^M < S$ then the encoding rule will become stuck with nowhere to put the last strings.

Storing elsewhere

A more robust and flexible method is to use *pointers* to additional pieces of memory in which collided strings are stored. There are many ways of doing this. As an example, we could store in location $\mathbf{h}$ in the hash table a pointer (which must be distinguishable from a valid record number s) to a 'bucket' where all the strings that have hash code $\mathbf{h}$ are stored in a *sorted list*. The encoder sorts the strings in each bucket alphabetically as the hash table and buckets are created.

The decoder simply has to go and look in the relevant bucket and then check the short list of strings that are there by a brief alphabetical search.

This method of storing the strings in buckets allows the option of making the hash table quite small, which may have practical benefits. We may make it so small that almost all strings are involved in collisions, so all buckets contain a small number of strings. It only takes a small number of binary comparisons to identify which of the strings in the bucket matches the cue $\mathbf{x}$.

15.4 Planning for collisions: the birthday problem

Exercise 15.3:[A2] If we wish to store S entries using a hash function whose output has M bits, how many collisions should we expect to happen, assuming that our hash function is an ideal random function? What size M of hash table is needed if we would like the expected number of collisions to be smaller than 1? What size M of hash table is needed if we would like the expected number of collisions to be a small fraction, say 1%, of S?

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

15.5 Other roles for hash codes

Error detection among friends

Are two files the same? If the files are on the same computer, we could just compare them bit by bit. But if they are a long way apart, it would be nice to have a way of confirming that two files are identical without having to transfer one of the files from A to B.

This problem can be solved using hash codes. Let Alice and Bob be the holders of the two files. (Maybe Alice sent the file to Bob, and they wish to confirm it has been received without error.) If Alice computes the hash of her file and sends it to Bob, and Bob computes the hash of his file, using the same S-bit hash function, and the two hashes match, then Bob can deduce that the two files are almost surely the same.

Example 15.4: What is the probability of a false negative, *i.e.*, the probability, given that the two files do differ, that the two hashes are nevertheless identical?

If we assume that the hash function is random and that the process that causes the files to differ knows nothing about the hash function, then the probability of a false negative is 2^{-S}. A 32-bit hash gives a probability of false negative of about 10^{-10}. It is common practice to use a linear hash function called a 32-bit cyclic redundancy check to detect errors in files.

Exercise 15.5:^{B2} Such a simple parity check code only detects errors, it doesn't help correct them. Since error-*correcting* codes exist, why not use one of them to get some error-correcting capability?

Tamper detection

What if the differences between the two files are not simply 'noise', but are introduced by an adversary, a clever *forger* called Fiona, who modifies the original file to make a forgery that purports to be Alice's file. How can Alice make a digital signature for the file so that Bob can confirm that no-one has tampered with the file? And how can we prevent Fiona from listening in on Alice's signature and attaching it to other files?

Let's assume that Alice computes a hash function for the file and sends it securely to Bob, in the same way as for error-detection above. If Alice computes a simple hash function for the file like the linear cyclic redundancy check, and Fiona knows that this is the method of verifying the file's integrity, Fiona can make her chosen modifications to the file and then easily identify (by linear algebra) a further 32-or-so single bits that, when flipped, restore the hash function of the file to its original value.

We must therefore require that the hash function be *hard to invert* so that no-one can construct a tampering that leaves the hash function unaffected. We would still like the hash function to be easy to compute, however, so that Bob doesn't have to do hours of work to verify every files he received. Such a hash function is called a *one-way hash function*. Finding such functions is one of the active research areas of cryptography.

A hash-function that is widely used in the free software community is MD5, which produces a 128 bit hash.[1]

[1] `http://www.freesoft.org/CIE/RFC/1321/3.htm`

Even with a good one-way hash function, the digital signatures described above are still vulnerable to attack, if Fiona has access to the hash function. Fiona could take the tampered file and hunt for a tiny modification to it such that its hash matches the original hash of Alice's file. This would take some time — about 2^{32} attempts, if the hash function has 32 bits — but eventually Fiona would find a tampered file that matches the given hash. To be secure against forgery, digital signatures must either have enough bits for such a random search to take too long, or the hash function itself must itself be kept secret.

Another person who might have a motivation for forgery is Alice herself. For example, she might be making a bet on the outcome of a race, without wishing to broadcast her prediction publicly; a method for placing bets would be for her to send to Bob the bookie the hash of her bet. Later on, she could send Bob the details of her bet. Such a protocol relies on the assumption that Alice cannot change her bet after the event without the hash coming out wrong. How big a hash function do we need to use to ensure that Alice cannot cheat? The answer is different from the size of the hash we needed in order to defeat Fiona above, because Alice is the author of *both* files. Alice could cheat by searching for two files that have identical hashes to each other. So she could make a large number N_1 of versions of bet one, and a large number N_2 of versions of bet two, and hash them all.

Example 15.6: If the hash has M bits, how big do N_1 and N_2 need to be for Alice to have a good chance of finding two different bets with the same hash?

This is a birthday problem like exercise 9.20 (p. 174). If there are N_1 Montagues and N_2 Capulets at a party, and each is assigned a 'birthday' of M bits, the expected number of collisions between a Montague and a Capulet is

$$N_1 N_2 2^{-M}, \tag{15.1}$$

so to minimize the number of files hashed, $N_1 + N_2$, Alice should make N_1 and N_2 equal, and will need to hash about $2^{M/2}$ files until she finds two that match. [This is the square root of the number of hashes Fiona had to make.]

If Alice has the use of $C = 10^6$ computers for $T = 10$ years, each computer taking $t = 1$ ns to evaluate a hash, the bet-communication system is secure against Alice's dishonesty only if $M \gg 2 \log_2 CT/t \simeq 160$ bits.

15.6 Problems

Information theory and the real world

Exercise 15.7:[A1] What is the shortest the address on a typical international letter could be, if it is to get to a unique human recipient? (Assume the permitted characters are `[A-Z,0-9]` How long are typical email addresses?

Exercise 15.8:[A2] How long does a piece of text need to be for you to be pretty sure that no human has written that string of characters before? How many notes are there in a new melody that has not been composed before?

Exercise 15.9:[B2] **Pattern recognition by molecules.**

Some proteins produced in a cell have a regulatory role. A regulatory protein controls the transcription of specific genes in the genome that might code for other proteins or sometimes the protein itself. This control may involve the protein's binding to a particular DNA sequence in the vicinity of the regulated gene. The presence of the bound protein either promotes or inhibits transcription of the gene.

(a) Use information–theoretic arguments to obtain a lower bound on the size of a typical protein that acts as a regulator specific to one gene in the whole human genome. Assume that the genome is a sequence of 3×10^9 nucleotides drawn from a four letter alphabet {A,C,G,T}; a protein is a sequence of amino acids drawn from a twenty letter alphabet. [Hint: establish how long the recognized DNA sequence has to be in order for that sequence to be found in the vicinity of one gene only, treating the rest of the genome as a random sequence. Then discuss how big the protein must be to recognize a sequence of that length uniquely.]

(b) Some of the sequences recognized by DNA–binding regulatory proteins consist of a subsequence that is repeated twice or more, for example the sequence

$$\underline{\texttt{GCCCCC}}\texttt{CACCCCT}\underline{\texttt{GCCCCC}} \tag{15.2}$$

is a binding site found upstream of the alpha–actin gene in humans. Does the fact that some binding sites consist of a repeated subsequence influence your answer to part (a)?

Exercise 15.10:^{B5} **What is the difference (in bits) between an ape and a human?** Let us assume life evolved on earth starting about 10^9 years ago. DNA–binding proteins are just one of the families of sophisticated molecules which the Blind Watchmaker of evolution has created. The entire blueprint of all organisms on the planet has emerged in a teaching process in which the teacher is natural selection: fitter individuals have more progeny, the fitness being defined by the local environment. The teaching signal is only a few bits per individual: an individual simply has a smaller or larger number of grandchildren, depending on their fitness. If the DNA sequence that codes for one of an organism's proteins is defective, that organism might have two grandchildren rather than four. The teaching signal does not communicate to the ecosystem any description of the imperfections in the organism that caused it to have fewer children. And these bits are highly redundant among individuals, because, all through a species, unfit individuals who are similar to each other will be failing to have offspring for similar reasons.

Estimate how many bits it takes to describe the difference between the genome of an ape and of a human.

Estimate how many bits per generation are acquired by the species as a whole by natural selection.

Estimate how many bits natural selection has conveyed to the human branch of the tree since the divergence between Australopithecines and apes 4,000,000 years ago.

Exercise 15.11:^{B2} How good must the error–correcting machinery in DNA replication be, given that mammals have not all died out long ago? Estimate the probability of nucleotide substitution, per cell division.

Exercise 15.12:^{B2} While the genome acquires information through natural selection at a rate of a few bits per generation (exercise 15.10), your brain acquires information at a greater rate.

Estimate at what rate new information can be stored in long term memory by your brain. Think of learning the words of a new language, for example.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 15's exercises

Solution to exercise 15.1 (p. 225): First imagine comparing the string $\mathbf{x}$ with another random string $\mathbf{x}^{(s)}$. The probability that the first bits of the two strings match is 1/2. The probability that the second bits match is 1/2. Assuming we stop comparing once we hit the first mismatch, the expected number of matches is 1 (exercise 2.22, p. 40).

Assuming the correct string is located at random in the raw list, we will have to compare with an average of $S/2$ strings before we find it, which costs $S/2$ binary comparisons; and comparing the correct strings takes N binary comparisons, giving a total expectation of $S/2 + N$ binary comparisons, if the strings are chosen at random.

In the worst case (which may indeed happen in practice), the other strings may be very similar to the search key, so that a lengthy sequence of comparisons is needed to find a mismatch. The worst case is when the correct string is last in the list, and all the other strings differ in the last bit only, giving a requirement of SN binary comparisons.

Solution to exercise 15.2 (p. 227): I'm not completely satisfied with this question. If we include a prior distribution on $\mathbf{x}$ which says, for example, that all strings begin with the sequence `01010101`, then it is obviously dangerous to assert that the two strings are probably the same, on the basis of comparing the first few bits!

But there are some strong results we can get from the assumption that the hash function is random. From a practical point of view it is probably better to rehash the strings with another hash function and compare the first few bits of that second function. Then the argument is safe.

Solution to exercise 15.3 (p. 228): Let the hash function have an output alphabet of size $A = 2^M$. The probability that one pair of entries collide under a random hash function is $1/A$. The number of pairs is $S(S-1)/2$. So the expected number of collisions between pairs is exactly

$$S(S-1)/(2A). \tag{15.3}$$

If we would like this to be smaller than 1, then we need $A > S(S-1)/2$ so

$$M > 2 \log_2 S. \tag{15.4}$$

We need *twice as many* bits as the number of bits, $\log_2 S$, that is sufficient to give each entry a unique name.

If we are happy to have occasional collisions, involving a fraction f of the names S, then we need $A > S/(2f)$ so

$$M > \log_2 S + \log[1/(2f)], \tag{15.5}$$

which means for $f \simeq 0.01$ that we need an extra 6 bits above $\log_2 S$.

The important point to note is the scaling of A with S in the two cases. If we want the hash function to be collision–free, then we must have A greater than $\sim S^2$. If we are happy to have a small frequency of collisions, then A only needs to be of order S.

Solution to exercise 15.5 (p. 229): If you add a tiny $M = 32$ extra bits of hash to a huge N–bit file you get pretty good error detection — the probability that an error is undetected is 2^{-M}, less than a one-in-a-billion. To do error *correction* requires far more check bits, and depends on the expected types of corruption. For example, if just eight random bits in a megabyte file are corrupted, it would takes $8 \times \log_2 8 \times 10^6 \simeq 180$ bits to specify which are the corrupted bits, and the number of parity bits used by a successful error–correcting code would have to exceed this number, by Shannon's noisy channel coding theorem.

Solution to exercise 15.8 (p. 230): We want to know the length L of a string such that it is very improbable that that string matches any part of the entire writings of humanity. Let's estimate that these writings total about one book for each person living, and that each book contains two million characters (200 pages with 10,000 characters per page) — that's 10^{16} characters, drawn from an alphabet of, say, 37 characters.

The probability that a randomly chosen string of length L matches at one point in the collected works of humanity is $1/37^L$. So the expected number of matches is $10^{16}/37^L$, which is vanishingly small if $L \geq 16/\log 37 \simeq 10$. Because of the redundancy and repetition of humanity's writings, it is possible that $L \simeq 10$ is an ovverestimate.

So, if you want to write something unique, sit down and compose a string of ten characters. But don't write `gidnebinzz`, because I already thought of that.

Solution to exercise 15.9 (p. 230):

(a) Let the DNA-binding protein recognize a sequence of length L nucleotides. That is, it binds preferentially to that DNA sequence, and not to any other pieces of DNA in the whole genome. (In reality, the recognized sequence may contain some wildcard characters, *e.g.*, the `*` in `TATAA*A`, which denotes 'any of `A`, `C`, `G` and `T`'; in that case, we are assuming that the recognized sequence contains L non-wildcard characters.)

Assuming the rest of the genome is 'random', *i.e.*, that the sequence consists of random nucleotides `A`, `C`, `G` and `T` with equal probability — which is obviously untrue, but it shouldn't make too much difference to our calculation — the chance of there being no other occurence of the target sequence in the whole genome, of length N nucleotides, is roughly

$$(1 - (1/4)^L)^N \simeq \exp(-N(1/4)^L), \tag{15.6}$$

which is close to one only if

$$N 4^{-L} \ll 1, \tag{15.7}$$

that is,

$$L > \log N / \log 4. \tag{15.8}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Using $N = 3 \times 10^9$, we require the recognized sequence to be longer than $L_{\min} = 16$ nucleotides.

What size of protein does this imply?

i. A weak lower bound can be obtained by assuming that the information content of the protein sequence itself is greater than the information content of the nucleotide sequence the protein prefers to bind to (which we have argued above must be at least 32 bits). This gives a minimum protein length of $32/\log_2(20) \simeq 7$ amino acids.

ii. Thinking realistically, the recognition of the DNA sequence by the protein presumably involves the protein coming into contact with all sixteen nucleotides in the target sequence. If the protein is a monomer, it must be big enough that it can simultaneously make contact with sixteen nucleotides of DNA. One helical turn of DNA containing ten nucleotides has a length of 3.4nm, so a contiguous sequence of sixteen nucleotides has a length of 5.4nm. The diameter of the protein must therefore be about 5.4nm or greater. Egg-white lysozyme is a small globular protein with a length of 129 amino acids and a diameter of about 4nm. Assuming that volume scales as diameter cubed, a protein of diameter 5.4nm must have a sequence of length $2.5 \times 129 \simeq 324$ amino acids.

(b) If, however, a target sequence consists of a twice repeated sub-sequence, we can get by with a much smaller protein which recognizes the sub-sequence only, and which binds to the DNA strongly only if it can form a dimer, both halves of which are bound to the recognized sequence, which must appear twice in succession in the DNA. Halving the diameter of the protein, we need a protein whose length is greater than $324/8 = 40$ amino acids. A protein of length smaller than this cannot serve as a regulatory protein specific to one gene.

Solution to exercise 15.10 (p. 231): Assuming a generation time of 10 years for reproduction, there have been 400,000 generations of human precursors since the divergence from apes. Assuming a population of 10^9 individuals, each receiving a couple of bits of information from natural selection, the total number of bits of information responsible for modifying the genomes of 4 million B.C. into today's human genome is about 8×10^{14} bits. However, natural selection is not smart at collating the information that it dishes out to the population, and there is a great deal of redundancy in that information. If the population size were twice as great, would it evolve twice as fast? No, because natural selection will simply be correcting the same defects twice as often.

John Maynard Smith has suggested that the rate of information acquisition by a species is independent of the population size, and is of order 1 bit per generation. This figure would only allow for 400,000 bits of difference between apes and humans, a number that is much smaller than the total size of the human genome — 6×10^9 bits. It is certainly the case that the genomic overlap between apes and humans is huge, but is the difference that small? (Don't forget that if two bit sequences of length N have 90% overlap, then it takes about $N/2$ bits to describe the differences between them; according to `http://users.ox.ac.uk/~mckee/chimp.html`, we share 98.4% of our DNA

with chimpanzees, which corresponds to a difference of $0.12N$ bits, or 7×10^8 bits. This is considerably larger than the 400,000 bits of difference mentioned above. Of course, the difference between us and chimpanzees could involve neutral changes to the DNA, and if some of the differences are redundant, then we are further overcounting; but are we overcounting by a factor of 1000?)

I have developed a simple model of natural selection[2] which suggests that, *if the species reproduces sexually*, the rate of information acquisition, though independent of the population size, could be as large as $\sqrt{G}$ bits per generation, where G is the size of the genome. Setting $G \simeq 10^4$–10^8, we would then have time to acquire about 4×10^7 or 4×10^9 bits of information from evolution.

Solution to exercise 15.11 (p. 232): See section A.7 for some actual estimates.

[2] `http://wol.ra.phy.cam.ac.uk/mackay/abstracts/gene.html`

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

16

*Cryptography and cryptanalysis: codes for information concealment**

Simple example of encoding using a key. The inference problem this defines, assuming a language model. Bayesian solution by Turing, adding up weights of evidence from multiple sources.

Analogy between decoding an error–correcting code, given noise, and cracking a cipher.

(Key equals source message and language equals noise is one way of thinking about it.)

Find how much text needed for the key to be determined. Distinguish this from the other question, how much time it takes to find the peak of the posterior probability. Most security relies on the latter.

Known plaintext. Often the plaintext is not known, but it can be guessed. During world war II, according to the head German codebreaker, Kapitän H. Bonatz, "The Admiral at Halifax, Nova Scotia, was a big help to us. He sent out a Daily Situation Report which ... always began 'Addressees, Situation, Date', and this repetition of opening style helped us to select very quickly the correct code in use at that time."

16.1 Unicity

unicity is a property of the unknown key's entropy and the language redundancy.

the more bits in the key, the longer till the truth crystallizes out from the alternatives.

17

*Communication over Constrained Noiseless Channels**

In this chapter we will study the task of communicating efficiently over a constrained noiseless channel. A constrained channel is one over which not all strings from the input alphabet can be transmitted.

17.1 Three examples of constrained binary channels

A constrained channel can be defined by rules that define which strings are permitted.

Example 17.1: In Channel A every 1 must be followed by at least one 0. Another way of stating this rule is that the substring 11 is forbidden.

A valid string for this channel is

$$00100101001010100010. \tag{17.1}$$

As a motivation for this model, consider a channel in which 1s are represented by pulses of electromagnetic energy, and the device that produces those pulses requires a recovery time of one clock cycle after generating a pulse before it can create another.

Example 17.2: Channel B has the rule that all 1s must come in groups of two or more, and all 0s must come in groups of two or more. Another way of stating this rule is that the substrings 101 and 010 are forbidden.

A valid string for this channel is

$$00111001110011000011. \tag{17.2}$$

As a motivation for this model, consider a disc drive in which the values 0 and 1 are represented by two opposite magnetic orientiations. Successive bits are written onto neighbouring points in a track along the disc surface. The strings 101 and 010 might be forbidden because a single isolated magnetic domain surrounded by domains having the opposite orientation is unstable, so that 101 might turn into 111, for example.

Example 17.3: Channel C has the rule that the largest permitted runlength is two, that is, each symbol can be repeated at most once; the substrings 111 and 000 are forbidden.

A valid string for this channel is

$$10010011011001101001. \tag{17.3}$$

A physical motivation for this model is a disc drive in which the rate of rotation of the disc is not known accurately, so it is difficult to distinguish between a string of two 1s and a string of three 1s, which are represented by oriented magnetizations of duration 2τ and 3τ respectively, where τ is the time taken for one bit to pass by.

All three of these channels are examples of *runlength–limited channels*. The rules constrain the minimum and maximum numbers of successive 1s and 0s.

Channel	Runlength of 1s		Runlength of 0s	
	min.	max.	min.	max.
unconstrained	1	∞	1	∞
A	1	1	1	∞
B	2	∞	2	∞
C	1	2	1	2

In channel A, runs of 0s may be of any length but runs of 1s are restricted to length 1. In channel B all runs must be of length two or more. In channel C, all runs must be of length one or two.

The capacity of the unconstrained binary channel is one bit per channel use. What are the capacities of the three constrained channels?

Exploring a constrained channel

Let us concentrate for a moment on channel A, in which runs of 0s may be of any length but runs of 1s are restricted to length 1. We would like to communicate a random binary file over this channel as efficiently as possible.

A simple starting point is a (2,1) code that maps each source bit into two transmitted bits:

Code C_1

s	t
0	00
1	10

(17.4)

This is a rate 1/2 code, so the capacity of channel A is at least 0.5. Can we do better?

The code C_1 is redundant because if the first of two received bits is a zero, we know that the second bit will also be a redundant zero. We can achieve a smaller average transmitted length using a code that omits the redundant zeroes in C_1.

$$\text{Code } C_2 \qquad \begin{array}{cc} s & t \\ \hline 0 & 0 \\ 1 & 10 \end{array} \tag{17.5}$$

C_2 is a variable length code; in some applications this variability might be undesirable — for example, a disc drive is easier to control if all blocks of 512 bytes are known to take exactly the same amount of disc area.

If the source symbols are used with equal frequency then the average transmitted length per source bit is

$$L = \frac{1}{2}1 + \frac{1}{2}2 = \frac{3}{2}, \tag{17.6}$$

so the average communication rate is

$$R = 2/3, \tag{17.7}$$

and the capacity of channel A must be at least 2/3.

Can we do better than C_2? There are two ways to argue that the information rate could be increased above $R = 2/3$.

The first argument assumes we are comfortable with the entropy as a measure of information content. The idea is that we can reduce the average message length, without greatly reducing the entropy of the message we send, by decreasing the fraction of 1s that we transmit. Imagine feeding into C_2 a stream of bits in which the frequency of 1s is f. The information rate R achieved is the entropy of the source, $H_2(f)$, divided by the mean transmitted length,

$$L(f) = (1 - f) + 2f = 1 + f. \tag{17.8}$$

Thus

$$R(f) = \frac{H_2(f)}{L(f)} = \frac{H_2(f)}{1 + f}. \tag{17.9}$$

The original code C_2 corresponds to the point $f = 1/2$. What happens if we perturb f a little towards smaller f, setting

$$f = \frac{1}{2} + \delta, \tag{17.10}$$

for small negative δ? In the vicinity of $f = 1/2$, the denominator $L(f)$ varies linearly with δ. In contrast, the numerator $H_2(f)$ only has a second–order dependence on δ.

Exercise 17.4: Find, to order δ^2, the Taylor expansion of $H_2(f)$ as a function of δ.

So, to first order, $R(f)$ increases linearly with decreasing δ. It must be possible to increase R by decreasing f.

Figure 17.1 shows these functions; $R(f)$ does indeed increase as f decreases and has a maximum of about 0.69 bits per channel use at $f \simeq 0.38$.

Exercise 17.5: If a file containing a fraction $f = 0.5$ 1s is transmitted by C_2, what fraction of the transmitted stream is 1s?

What fraction of the transmitted bits are 1s if we drive code C_2 with a sparse source of density $f = 0.38$?

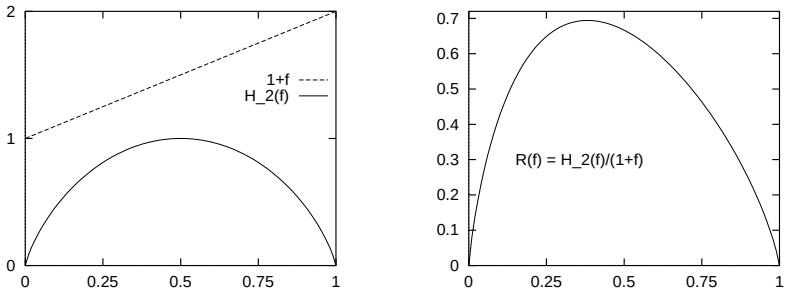


Figure 17.1. (a) The information content per source symbol and mean transmitted length per source symbol as a function of the source density. (b) The information content per transmitted symbol, in bits, as a function of f.

A second, more fundamental approach *counts* how many valid sequences of length N there are, S_N. We can communicate $\log S_N$ bits in N channel cycles by giving one name to each of these valid sequences.

17.2 The capacity of a constrained noiseless channel

How can we define the capacity of a constrained channel? We defined the capacity of a noisy channel in terms of the mutual information between its input and its output, then we proved — with considerable effort — that this number, the capacity, was related to the number of distinguishable messages $S(N)$ that could be reliably conveyed over the channel in N uses of the channel by

$$C = \lim_{N \to \infty} \frac{1}{N} \log S(N). \tag{17.11}$$

In the case of the constrained noiseless channel, we can adopt this identity as our definition of the channel's capacity.

Knowing the capacity of a channel doesn't tell us how practically to achieve that rate of communication, so once we have figured out the capacity of a channel we will have to return the task of making a code for that channel.

17.3 Counting the number of possible messages

First let us introduce some representations of constrained channels. We can often conveniently represent a constrained channel by a state diagram. In a state diagram, states of the transmitter are represented by circles labelled with the name of the state. Directed edges from one state to another indicate that the transmitter is permitted to move from the first state to the second, and a label on that edge indicates the symbol emitted when that transition is made. Figure 17.2(a) shows the state diagram for channel A. It has two states, 0 and 1. When transitions to state 0 are made, a `0` is transmitted; when transitions to state 1 are made, a `1` is transmitted; transitions from state 1 to state 1 are not possible.

We can also represent the state diagram by a *trellis section* which shows two successive states in time at two successive horizontal locations (figure 17.2(b)). The state of the transmitter at time n is called s_n.[1] The set of possible state

[1] Change s_n to i_n?

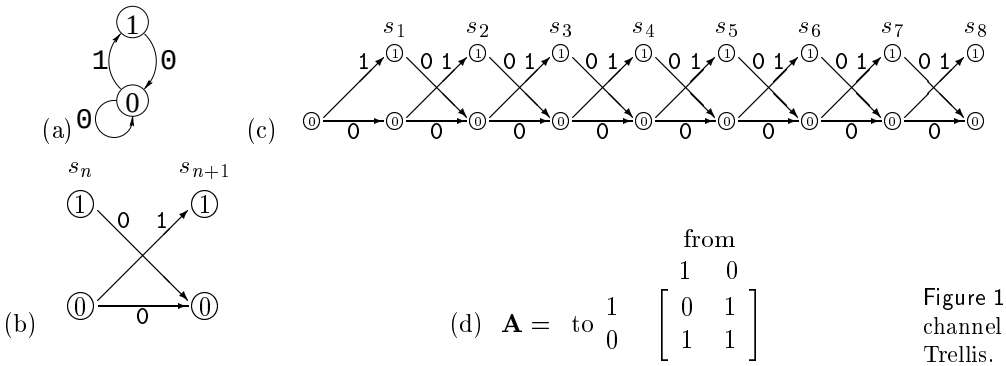


Figure 17.2. (a) State diagram for channel A. (b) Trellis section. (c) Trellis. (d) Connection matrix.

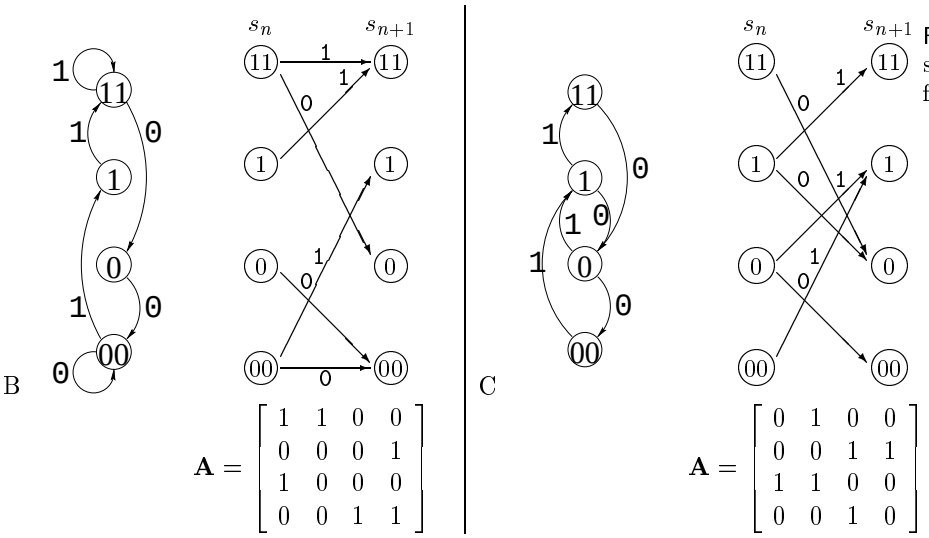


Figure 17.3. State diagrams, trellis sections and connection matrices for channels B and C.

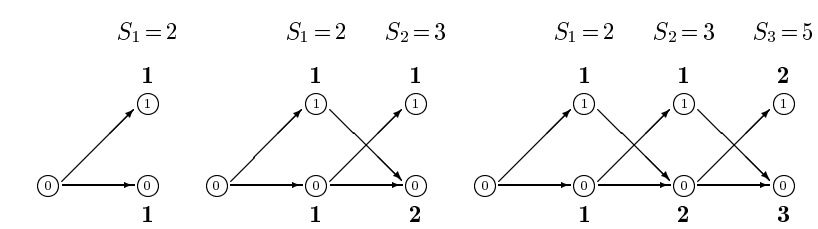


Figure 17.4. Counting number of paths in the trellis of channel A. Global properties of graphs can be computed by a local message–passing algorithm. The counts next to the nodes are accumulated by passing from left to right across the trellises.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

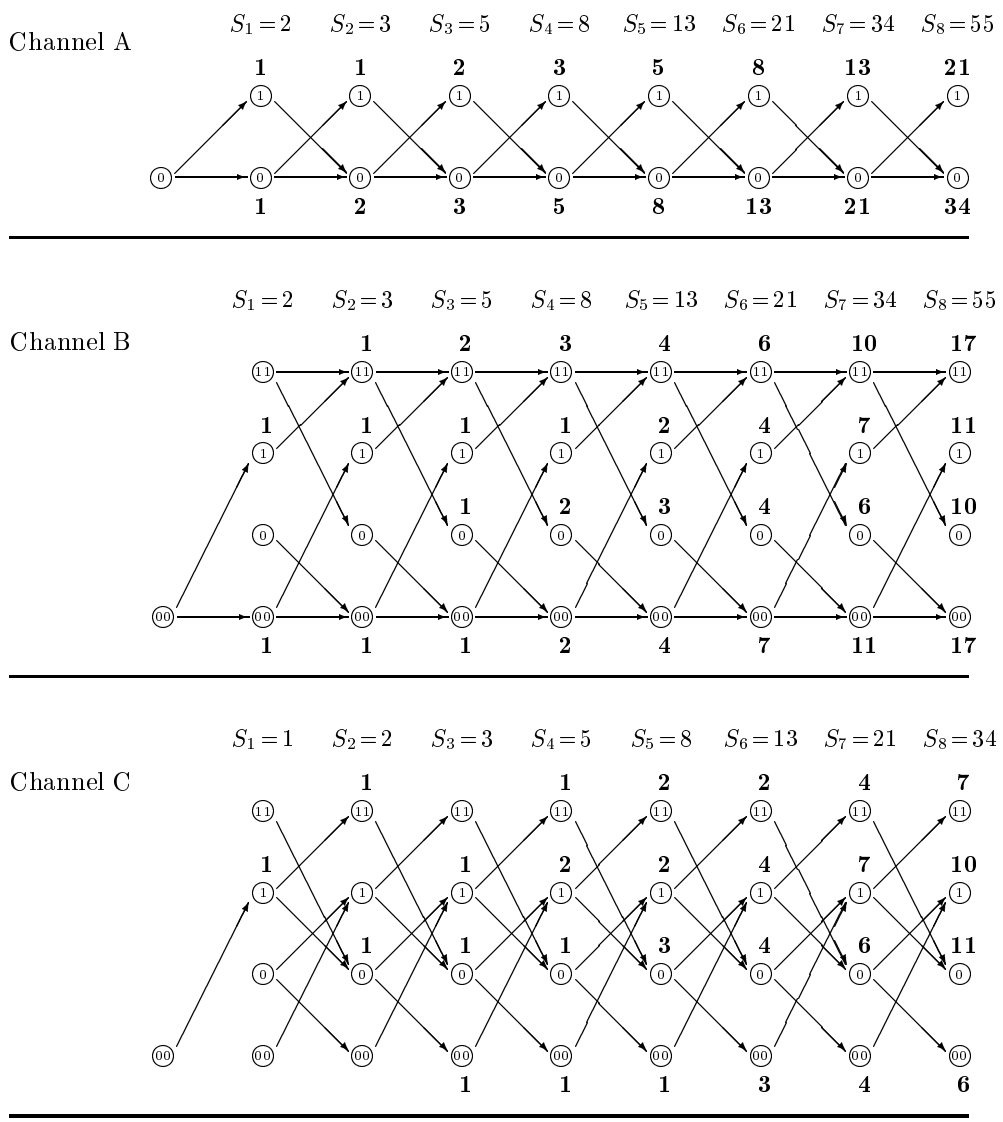


Figure 17.5. Counting number of paths in the trellises of channels A, B, and C. I assume that at the start the first bit is preceded by 00, so that for channels A and B, any initial character is permitted, but for channel C, the first character must be a 1.

n	S_n	S_n/S_{n-1}	$\log_2 S_n$	$\frac{1}{n}\log_2 S_n$
1	2		1.0	1.00
2	3	1.500	1.6	0.79
3	5	1.667	2.3	0.77
4	8	1.600	3.0	0.75
5	13	1.625	3.7	0.74
6	21	1.615	4.4	0.73
7	34	1.619	5.1	0.73
8	55	1.618	5.8	0.72
9	89	1.618	6.5	0.72
10	144	1.618	7.2	0.72
11	233	1.618	7.9	0.71
12	377	1.618	8.6	0.71
100	$9{\cdot}10^{20}$	1.618	69.7	0.70
200	$7{\cdot}10^{41}$	1.618	139.1	0.70
300	$6{\cdot}10^{62}$	1.618	208.5	0.70
400	$5{\cdot}10^{83}$	1.618	277.9	0.69

Figure 17.6. Counting number of paths in the trellis of channel A.

sequences can be represented by a *trellis* as shown in figure 17.2(c). A valid sequence corresponds to a path through the trellis, and the number of valid sequences is the number of paths. For the purpose of counting how many paths there are through the trellis, we can ignore the labels on the edges and summarize the trellis section by the connection matrix $\mathbf{A}$, in which $A_{ss'} = 1$ if there is an edge from state s to s', and $A_{ss'} = 0$ otherwise (figure 17.2(d)).

So, let's count! Figure 17.4 shows the first few steps of this counting process, and figure 17.5(a) shows the number of paths ending in each state after n steps for $n = 1 \ldots 8$. The total number of paths of length n, S_n, is shown along the top. We recognize S_n as the Fibonacci series.

Exercise 17.6: Show that the ratio of successive terms in the Fibonacci series tends to the golden ratio,

$$\gamma \equiv \frac{1+\sqrt{5}}{2} = 1.618. \tag{17.12}$$

Thus, to within a constant factor, M_N scales as $M_N \sim \gamma^N$ as $N \to \infty$, so the capacity of channel A is

$$C = \lim \frac{1}{N} \log_2 \left[\text{constant} \cdot \gamma^N \right] = \log_2 \gamma = \log_2 1.618 = 0.694. \tag{17.13}$$

How can we describe what we just did? The count of the number of paths is a vector $\mathbf{c}^{(n)}$; we can obtain $\mathbf{c}^{(n+1)}$ from $\mathbf{c}^{(n)}$ using:

$$\mathbf{c}^{(n+1)} = \mathbf{A}\mathbf{c}^{(n)}. \tag{17.14}$$

So:

$$\mathbf{c}^{(N)} = \mathbf{A}^N \mathbf{c}^{(0)}, \tag{17.15}$$

where $\mathbf{c}^{(0)}$ is the state count before any symbols are transmitted. In figure 17.5 we assumed $\mathbf{c}^{(0)} = [0, 1]^{\mathsf{T}}$, *i.e.*, that either of the two symbols is permitted at

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

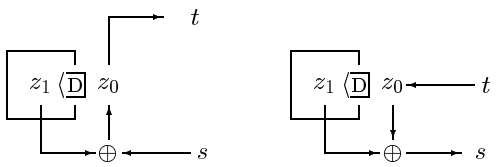


Figure 17.7. An accumulator and a differentiator.

the outset. The total number of paths is $M_n = \sum_s c_s^{(n)} = \mathbf{c}^{(n)} \cdot \mathbf{n}$. In the limit, $\mathbf{c}^{(N)}$ becomes dominated by the principal right–eigenvector of $\mathbf{A}$.

$$\mathbf{c}^{(N)} \rightarrow \text{constant} \cdot \lambda_1^N \mathbf{e}_{\mathrm{R}}^{(0)} \tag{17.16}$$

Here, λ_1 is the principal eigenvalue of $\mathbf{A}$.

So to find the capacity of any constrained channel defined by a connection matrix, all we need to do is find that matrix's principal eigenvalue, λ_1. Then

$$C = \log_2 \lambda_1. \tag{17.17}$$

Comparing figure 17.5(a) and figure 17.5(b,c) it looks as if channels B and C have the same capacity as channel A. The principal eigenvalues of the the three trellises are indeed the same (see section A.6). Indeed the channels are intimately related.

Equivalence of channels A and B

If we take any valid string $\mathbf{s}$ for channel A and pass it through an *accumulator*, obtaining $\mathbf{t}$ defined by:

$$\begin{array}{rcl} t_1 & = & s_1 \\ t_n & = & t_{n-1} + s_n \bmod 2 \quad \text{for } n \geq 2 \end{array} \; , \tag{17.18}$$

then the resulting string is a valid string for channel B, because there are no 11s in $\mathbf{s}$, so there are no isolated digits in $\mathbf{t}$. The accumulator is an invertible operator, so, similarly, any valid string $\mathbf{t}$ for channel B can be mapped onto a valid string $\mathbf{s}$ for channel A through the *binary differentiator*,

$$\begin{array}{rcl} s_1 & = & t_1 \\ s_n & = & t_n - t_{n-1} \bmod 2 \quad \text{for } n \geq 2 \end{array} \; . \tag{17.19}$$

Because + and − are equivalent in modulo 2 arithmetic, the differentiator is also a blurrer, convolving the source stream with the filter $(1, 1)$.

Exercise 17.7: What is the relationship of channel C to channels A and B?

17.4 Practical communication over constrained channels

OK how, to do it in practice? Since all three channels are equivalent, we can concentrate on channel A.

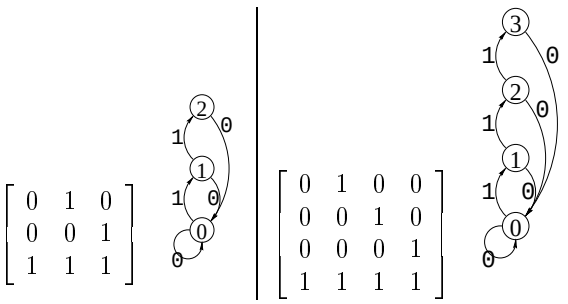


Figure 17.8. State diagrams and connection matrices for channels with maximum runlengths for 1s equal to 2 and 3.

Fixed–length solutions

The code in the margin achieves rate $3/5 = 0.6$.

Similarly, we can enumerate all strings of length 8 that end in the zero state. There are 34 of them. So can map 5 bits (32 source strings) to 8 and achieve rate $5/8 = 0.625$. Crank up to $N = 16$, can have 10 source bits, so rate still 0.625.

s	$c(s)$
1	00000
2	10000
3	01000
4	00100
5	00010
6	10100
7	01010
8	10010

Optimal variable–length solution

It is probably confusing that I have used s to run over source message names, and s to run over states in the trellis. Let's change to u or i for trellis states?

Figure out optimal transition probabilities $Q_{s'|s}$ and use them. Drive with an appropriate arithmetic code. For channel A, we already saw above that with a sparse source with density $f = 0.38$ we can achieve capacity. And we know how to make sparsifiers (chapter 6).

So, how to find these probabilities?

The answer: find the principal right– and left–eigenvectors of $\mathbf{A}$ with eigenvalue λ. Then want a $\mathbf{Q}$ whose invariant distribution is proportional to $e_i^{(R)} e_i^{(L)}$. This is given by

$$Q_{s'|s} = \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}}. \tag{17.20}$$

Exercise 17.8: Show that when sequences are generated using this transitions probability matrix, the entropy of the resulting sequence is indeed asymptotically $\log_2 \lambda$ per symbol. [Hint: it suffices to consider the conditional entropy of just one symbol given the previous one, assuming the previous one's distribution is the invariant distribution.]

Exercise 17.9: How many valid sequences of length 8 ending with a 0 are there for the run–length–limited channels $0^+1^{1,2}$ and $0^+1^{1,3}$ (figure 17.8)?

What are the capacities of these channels?

Using a computer, find the matrix $\mathbf{Q}$ for generating a random path through the trellises of the run–length–limited channels 0^+1^1 (channel A), $0^+1^{1,2}$ and $0^+1^{1,3}$.

Exercise 17.10: Consider the run–length–limited channel in which any length of run of 0s is permitted, and the maximum run length of 1s is a large number L such as nine or ninety.

Estimate the capacity of this channel. (Give the first two terms in a series expansion involving L.)

What, roughly, is the form of the optimal matrix $\mathbf{Q}$ for generating a random path through the trellis of this channel? Focus on the values of the elements $Q_{1|0}$, the probability of a generating 1 given a preceding zero, and $Q_{L|L-1}$, the probability of generating a 1 given a preceding run of $L-1$ 1s. Check your answer by explicit computation for the $0^+1^{1,9}$ channel.

17.5 Variable symbol durations

We encountered a noiseless channel in exercise 7.6 (p. 139). Each symbol had a different duration. Solve that problem, and you've done this topic.

Then there's the general problem of a channel with constraints and with variable duration symbols. [*e.g.*, Morse code, where dots and dashes must be separated by either short or long spaces.]

MORE HERE.

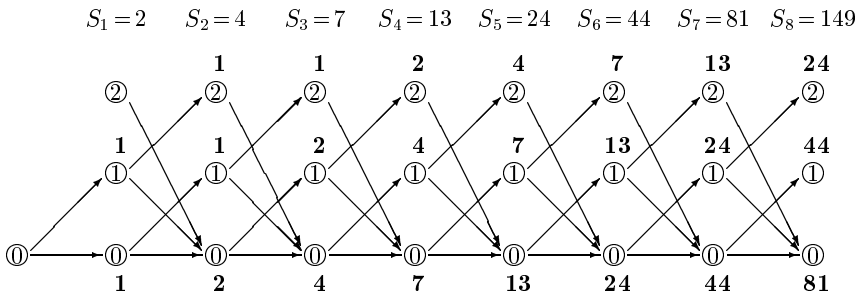
Solutions to Chapter 17's exercises

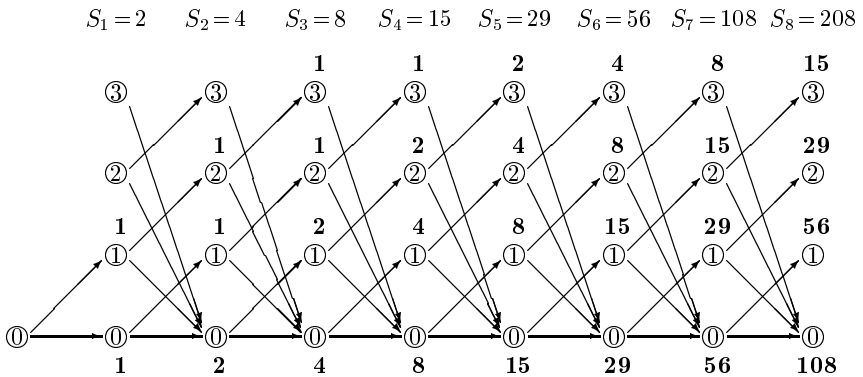
Solution to exercise 17.5 (p. 240): A file transmitted by C_2 contains, on average, one third 1s and two thirds 0s.

If $f = 0.38$, the fraction of 1s is $f/(1+f) = (\gamma - 1.0)/(2\gamma - 1.0) = 0.2764$.

Solution to exercise 17.7 (p. 245): A valid string for channel C can be obtained from a valid string for channel A by first inverting it [1 → 0; 0 → 1], then passing it through an accumulator. These operations are invertible, so any valid string for C can also be mapped onto a valid string for A. The only proviso here comes from the edge effects. If we assume that the first character transmitted over channel C is preceded by a string of zeroes, so that the first character is forced to be a 1 (figure 17.5(c)) then the two channels are only exactly equivalent if we assume that channel A's first character must be a zero.

Solution to exercise 17.9 (p. 246): The principal eigenvalues of the connection matrices of the two channels are 1.839 and 1.928. The capacities ($\log \lambda$) are 0.879 and 0.947 bits. See the eigenvector tables (section A.6) for the matrices $\mathbf{Q}$.





 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solution to exercise 17.8 (p. 246): Let the invariant distribution be

$$P(s) = \alpha e_s^{(L)} e_s^{(R)}, \tag{17.21}$$

where α is a normalization constant. The entropy of S_t given S_{t-1}, assuming S_{t-1} comes from the invariant distribution, is

$$\begin{aligned} H(S_t|S_{t-1}) &= -\sum_{s,s'} P(s)P(s'|s)\log P(s'|s) && (17.22)\\ &= -\sum_{s,s'} \alpha e_s^{(L)} e_s^{(R)} \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}} \log \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}} && (17.23)\\ &= -\sum_{s,s'} \alpha e_s^{(R)} \frac{e_{s'}^{(L)} A_{s's}}{\lambda} \left[\log e_{s'}^{(L)} + \log A_{s's} - \log \lambda - \log e_s^{(L)}\right] && (17.24) \end{aligned}$$

Now, $A_{s's}$ is either 0 or 1, so the contributions from the terms proportional to $A_{s's}\log A_{s's}$ are all zero. So

$$\begin{aligned} H(S_t|S_{t-1}) &= \log\lambda + -\frac{\alpha}{\lambda}\sum_{s'}\left(\sum_s A_{s's} e_s^{(R)}\right) e_{s'}^{(L)} \log e_{s'}^{(L)} + \\ &\quad \frac{\alpha}{\lambda}\sum_s \left(\sum_{s'} e_{s'}^{(L)} A_{s's}\right) e_s^{(R)} \log e_s^{(L)} && (17.25)\\ &= \log\lambda - \frac{\alpha}{\lambda}\sum_{s'} \lambda e_{s'}^{(R)} e_{s'}^{(L)} \log e_{s'}^{(L)} + \frac{\alpha}{\lambda}\sum_s \lambda e_s^{(L)} e_s^{(R)} \log e_s^{(L)} && (17.26)\\ &= \log\lambda. && (17.27) \end{aligned}$$

Solution to exercise 17.10 (p. 247): The channel is similar to the unconstrained binary channel; runs of length greater than L are rare if L is large, so we only expect weak differences from this channel; these differences will show up in contexts where the run length is close to L. The capacity of the channel is very close to one bit.

A lower bound on the capacity is obtained by considering the simple variable–length code for this channel which replaces occurrences of the maximum runlength string `111...1` by `111...10`, and otherwise leaves the source file unchanged. The average rate of this code is $1/(1+2^{-L})$ because the invariant distribution will hit the 'add an extra zero' 2^{-L} of the time.

We can reuse the solution for the variable–length channel. The capacity is the value of β such that the equation

$$Z(\beta) = \sum_{l=1}^{L+1} 2^{-\beta l} = 1 \tag{17.28}$$

is satisfied. The $L+1$ terms in the sum correspond to the $L+1$ possible strings that can be emitted, `0`, `10`, `110`, ..., `11...10`. The sum is exactly given by: ($\sum_{n=0}^{N} ar^n = \frac{a(r^{N+1}-1)}{r-1}$)

$$Z(\beta) = 2^{-\beta}\frac{\left(2^{-\beta}\right)^{L+1} - 1}{2^{-\beta} - 1}. \tag{17.29}$$

We anticipate that β should be a little less than 1 in order for $Z(\beta)$ to equal 1. Rearranging and solving approximately for β, using $\ln(1+x) \simeq x$,

$$Z(\beta) = 1 \tag{17.30}$$
$$\Rightarrow \beta \simeq 1 - 2^{-(L+2)}/\ln 2 \tag{17.31}$$

We evaluated the true capacities for $L = 2$ and $L = 3$ in an earlier exercise. The following table compares the approximate capacity β with the true capacity for a selection of values of L.

L	β	True capacity
2	0.910	0.879
3	0.955	0.947
4	0.977	0.975
5	0.9887	0.9881
6	0.9944	0.9942
9	0.9993	0.9993

The element $Q_{1|0}$ will be close to 1/2 (just a tiny bit larger), since in the unconstrained binary channel $Q_{1|0} = 1/2$. When a run of length $L-1$ has occurred, we effectively have a choice of printing `10` or `0`. Let the probability of selecting `10` be f. Let us estimate the entropy of the *remaining* N characters in the stream as a function of f, assuming the rest of the matrix $\mathbf{Q}$ to have been set to its optimal value. The entropy of the next N characters in the stream is the entropy of the first bit, $H_2(f)$, plus the entropy of the remaining characters, which is roughly $(N-1)$ bits if we select `0` as the first bit and $(N-2)$ bits if `1` is selected. More precisely, if C is the capacity of the channel (which is roughly 1),

$$\begin{aligned} H(\text{the next } N \text{ chars}) &\simeq H_2(f) + [(N-1)(1-f) + (N-2)f]C \\ &= H_2(f) + NC - fC \simeq H_2(f) + N - f. \end{aligned} \tag{17.32}$$

Differentiating and setting to zero to find the optimal f, we obtain:

$$\log_2 \frac{1-f}{f} \simeq 1 \;\Rightarrow \frac{1-f}{f} \simeq 2 \;\Rightarrow f \simeq 1/3. \tag{17.33}$$

Here is the optimal matrix:

$$\begin{bmatrix} 0 & .3334 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .4287 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .4669 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .4841 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .4923 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .4963 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4983 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4993 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4998 \\ 1 & .6666 & .5713 & .5331 & .5159 & .5077 & .5037 & .5017 & .5007 & .5002 \end{bmatrix} \tag{17.34}$$

Our rough theory works.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

18

Further exercises on information theory

I've been asked to include some exercises *without* worked solutions. Here are a few. Numerical solutions to some of them are provided on page 258.

The most exciting exercises, which will introduce you to further ideas in information theory, are towards the end of this chapter.

Refresher exercises on source coding and noisy channels

Solutions on p. 258

Exercise 18.1:^{B2} Let X be an ensemble with $\mathcal{A}_X = \{0, 1\}$ and $\mathcal{P}_X = \{0.995, 0.005\}$. Consider the block coding of X^{100} where every $\mathbf{x} \in X^{100}$ containing 3 or fewer 1s is assigned a distinct codeword, while the other $\mathbf{x}$s are ignored.

(a) If the codewords are all of the same length, find the minimum length required to provide the above set with distinct codewords.

(b) Calculate the probability of getting an $\mathbf{x}$ that will be ignored.

Exercise 18.2:^{B2} Let X be an ensemble with $\mathcal{P}_X = \{0.1, 0.2, 0.3, 0.4\}$. The ensemble is encoded using $\mathcal{C} = \{0001, 001, 01, 1\}$. Consider the codeword corresponding to $\mathbf{x} \in X^N$, where N is large.

(a) Compute the entropy of the fourth bit of transmission.

(b) Compute the conditional entropy of the fourth bit given the third bit.

(c) Estimate the entropy of the hundredth bit.

(d) Estimate the conditional entropy of the hundredth bit given the ninety–ninth bit.

Exercise 18.3:^{A2} Two fair dice are rolled by Alice and the sum is recorded. Bob's task is to ask a sequence of questions with yes/no answers to find out this number. Devise in detail a strategy that achieves the minimum possible average number of questions.

Exercise 18.4:^{B2} Find a probability sequence $\mathbf{p} = (p_1, p_2, \ldots)$ such that $H(\mathbf{p}) = \infty$.

Exercise 18.5:^{B2}
Consider the source $\mathcal{A}_S = \{a, b, c, d, e\}$, $\mathcal{P}_S = \{1/3, 1/3, 1/9, 1/9, 1/9\}$ and

the channel whose transition probability matrix is

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2/3 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1/3 & 0 \end{bmatrix}. \tag{18.1}$$

Note that the source alphabet has five symbols, but the channel alphabet $\mathcal{A}_X = \mathcal{A}_Y = \{0, 1, 2, 3\}$ has only four. Assume that the source produces symbols at exactly $3/4$ the rate that the channel accepts channel symbols. For a given (tiny) $\epsilon > 0$, explain how you would design a system for communicating the source's output over the channel with an average error probability per source symbol less than ϵ. Be as explicit as possible. In particular, *do not* invoke Shannon's noisy channel coding theorem.

Exercise 18.6:^{B2} Consider a discrete memoryless source with $\mathcal{A}_X = \{a, b, c, d\}$ and $\mathcal{P}_X = \{1/2, 1/4, 1/8, 1/8\}$. There are $4^8 = 65536$ eight–letter words that can be formed from the four letters. Find the total number of such words that are in the typical set $T_{N\beta}$ where $N = 8$ and $\beta = 0.1$. The definition of $T_{N\beta}$, from chapter 4, is:

$$T_{N\beta} = \left\{ \mathbf{x} \in \mathcal{A}_X^N : \left| \frac{1}{N} \log_2 \frac{1}{P(\mathbf{x})} - H \right| < \beta \right\}. \tag{18.2}$$

Exercise 18.7:^{B2} Consider a binary symmetric channel and a code $C = \{0000, 0011, 1100, 1111\}$; assume that the four codewords are used with probabilities $\{1/2, 1/8, 1/8, 1/4\}$.

What is the decoding rule that minimizes the probability of decoding error? [The optimal decoding rule depends on the noise level f of the binary symmetric channel. Give the decoding rule for each range of values of f, for f between 0 and 1/2.]

Exercise 18.8:^{A2} Find the capacity and optimizing input distribution for the three–input, three–output channel whose transition probabilities are:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 1/3 & 2/3 \end{bmatrix} \tag{18.3}$$

Exercise 18.9: Describe briefly the encoder for a (7,4) Hamming code.

Assuming that one codeword of this code is sent over a binary symmetric channel, define the *syndrome* $\mathbf{z}$ of the received vector $\mathbf{r}$; state how many different possible syndromes there are; and state the maximum number of channel errors that the optimal decoder can correct.

Define the *capacity* of a channel with input x and output y and transition probability matrix $Q(y|x)$.

The input to a channel Q is a word of 8 bits. The output is also a word of 8 bits. Each time it is used, the channel flips *exactly one* of the transmitted bits, but the receiver does not know which one. The other seven bits are received without error. All 8 bits are equally likely to be the one that is flipped. Derive the capacity of this channel.

Show, by describing an *explicit* encoder *and* decoder that it is possible *reliably* (that is, with *zero* error probability) to communicate 5 bits per cycle over this channel.

Exercise 18.10: The ten–digit number on the cover of a book known as the ISBN incorporates an error–detecting code. The number consists of nine source digits $x_1, x_2, \ldots, x_9$, satisfying $x_n \in \{0, 1, \ldots, 9\}$, and a tenth check digit whose value is given by

$$x_{10} = \left(\sum_{n=1}^{9} n x_n \right) \bmod 11.$$

Here $x_{10} \in \{0, 1, \ldots, 9, 10\}$. If $x_{10} = 10$ then the tenth digit is shown using the roman numeral X. For example, 1-010-00000-4 is a valid ISBN. [The hyphens are included for legibility.]

Show that a valid ISBN satisfies:

$$\left(\sum_{n=1}^{10} n x_n \right) \bmod 11 = 0.$$

Imagine that an ISBN is communicated over an unreliable human channel which sometimes *modifies* digits and sometimes *reorders* digits.

Show that this code can be used to detect (but not correct) all errors in which any one of the ten digits is modified (for example, 1-010-00000-4 $\rightarrow$ 1-010-00080-4).

Show that this code can be used to detect all errors in which any two adjacent digits are transposed (for example, 1-010-00000-4 $\rightarrow$ 1-100-00000-4).

What other transpositions of pairs of *non–adjacent* digits can be detected?

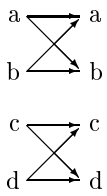
If the tenth digit were defined to be

$$x_{10} = \left(\sum_{n=1}^{9} n x_n \right) \bmod 10,$$

why would the code not work so well? (Discuss the detection of both modifications of single digits and transpositions of digits.)

Exercise 18.11: A channel with input x and output y has transition probability matrix:

$$Q = \begin{bmatrix} 1-f & f & 0 & 0 \\ f & 1-f & 0 & 0 \\ 0 & 0 & 1-g & g \\ 0 & 0 & g & 1-g \end{bmatrix}$$



Assuming an input distribution of the form

$$\mathcal{P}_X = \left\{ \frac{p}{2}, \frac{p}{2}, \frac{1-p}{2}, \frac{1-p}{2} \right\},$$

write down the entropy of the output, $H(Y)$, and the conditional entropy of the output given the input, $H(Y|X)$.

Show that the optimal input distribution is given by

$$p = \frac{1}{1 + 2^{-H_2(g)+H_2(f)}},$$

where $H_2(f) = f \log_2 \frac{1}{f} + (1-f) \log_2 \frac{1}{(1-f)}$.

[You may find the identity $\frac{d}{dp} H_2(p) = \log_2 \frac{1-p}{p}$ helpful.]

Write down the optimal input distribution and the capacity of the channel in the case $f = 1/2$, $g = 0$, and comment on your answer.

Exercise 18.12: What are the differences in the redundancies needed in an error–detecting code (which can reliably detect that a block of data has been corrupted) and an error–correcting code (which can detect and correct errors)?

Further tales from information theory

The following exercises give you the chance to discover for yourself the answers to some more surprising results of information theory.

Exercise 18.13:^{C3} Communication of correlated information. Imagine that we want to communicate data from two data sources $X^{(A)}$ and $X^{(B)}$ to a central location C via noise-free one-way communication channels (figure 18.1a). The signals $x^{(A)}$ and $x^{(B)}$ are strongly correlated such that their joint information content is only a little greater than the marginal information content of either of them. For example, the joint probability of $x^{(A)}$ and $x^{(B)}$ might be

$$P(x^{(A)}, x^{(B)}): \qquad \begin{array}{c|cc} & x^{(A)} & \\ & 0 & 1 \\ \hline x^{(B)} \quad 0 & 0.49 & 0.01 \\ \quad\;\; 1 & 0.01 & 0.49 \end{array} \tag{18.4}$$

The central receiver would like to know the values of $x^{(A)}$ and $x^{(B)}$ exactly. Assuming that variables $x^{(A)}$ and $x^{(B)}$ are generated repeatedly from this distribution, can they be encoded at rates R_A and R_B in such a way that C can reconstruct all the variables, with the sum of information transmission rates on the two lines being less than two bits per cycle? For simplicity, assume that the one-way communication channels are noise-free binary channels.

The answer, which you should demonstrate, is indicated in figure 18.1. In the general case of two correlated sources $X^{(A)}$ and $X^{(B)}$, there exist codes for the two transmitters that can achieve reliable communication of

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

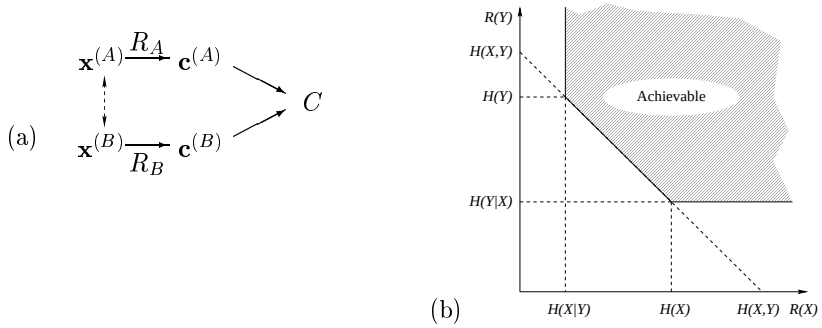


Figure 18.1. Communication of correlated information. (a) The communication situation: $x^{(A)}$ and $x^{(B)}$ are correlated sources (the correlation is represented by the dotted arrow). Strings of values of each variable are encoded using codes of rate R_A and R_B and communicated over a noise free channel to a receiver C. (b) The achievable rate region. Apologies for the AB-XY switch.

both $X^{(A)}$ and $X^{(B)}$ to C, as long as: the information rate from $X^{(A)}$, R_A, exceeds $H(X^{(A)}|X^{(B)})$; the information rate from $X^{(B)}$, R_B, exceeds $H(X^{(B)}|X^{(A)})$; and the total information rate $R_A + R_B$ exceeds the joint information $H(X^{(A)}, X^{(B)})$.

So in the case of $x^{(A)}$ and $x^{(B)}$ above, each transmitter must transmit at a rate greater than $H_2(0.02) = 0.14$ bits, and the total rate $R_A + R_B$ must be greater than 1.14 bits, but that is all. There exist codes that can achieve these rates. Your task is to figure out why this is so.

Try to find an explicit solution in which one of the sources is sent as plain text, $c^{(B)} = x^{(B)}$, and the other is encoded.

Exercise 18.14:^{C3} Multiple access channels. Consider a channel with two sets of inputs and one output — for example, a shared telephone line (figure 18.2a). A simple model system has two binary inputs A and B and a ternary output C equal to the arithmetic sum of the two inputs, that's 0, 1 or 2. There is no noise. A and B cannot communicate with each other, and they cannot hear the output of the channel. If the output is a 0, the receiver can be certain that both inputs were set to 0; and if the output is a 2, the receiver can be certain that both inputs were set to 1. But if the output is 1, then it could be that the input state was (0,1) or (1,0). How should users A and B use this channel so that their messages can be deduced from the received signal C? How fast can A and B communicate?

Clearly the total information rate from A and B to C cannot be two bits. On the other hand it is easy to achieve a total information rate $R_A + R_B$ of one bit. Can reliable communication be achieved at rates (R_A, R_B) such that $R_A + R_B > 1$?

The answer is indicated in figure 18.2. There exist codes for the two transmitters such that the rates $(R(A), R(B))$ can be any point in the convex hull of $\{(1,0), (1,.5), (.5,1), (0,1), (0,0)\}$.

Exercise 18.15:^{C3} Multiterminal information networks are both important practically and intriguing theoretically. Consider the following example of a two-way binary channel (figure 18.3a,b): two people both wish to talk over the channel, and they both want to hear what the other person is saying; but you can only hear the signal transmitted by the other person if you are transmitting a zero. What simultaneous information rate from

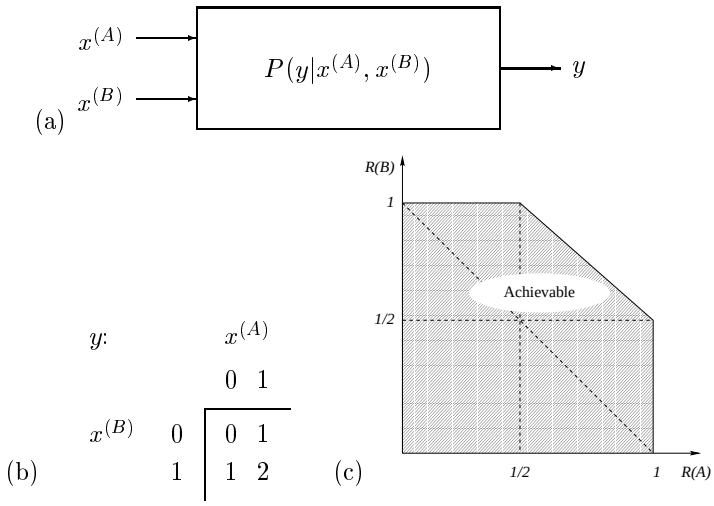


Figure 18.2. Multiple access channels. (a) A general multiple access channel with two transmitters and one receiver. (b) A binary multiple access channel with output given by adding the two inputs. (c) The achievable region.

A to B and from B to A can be achieved, and how? Everyday examples of such networks include the VHF channels used by ships, and computer ethernet networks (in which *all* the devices are unable to hear *anything* if two or more devices are broadcasting simultaneously).

Obviously, we can achieve rates of $1/2$ in both directions by simple time-sharing. But can the two information rates be made larger? Finding the capacity of a general two-way channel is still an open problem. However, we can obtain interesting results concerning achievable points for the simple binary channel discussed above, as indicated in figure 18.3. There exist codes that can achieve rates up to the boundary shown. There may exist better codes too.

Gambling

Exercise 18.16:^{C2} A horse race involving I horses occurs repeatedly, and you are obliged to bet all your money each time. Your bet at time t can be represented by a normalized probability vector $\mathbf{b}$ multiplied by your money $m(t)$. The odds offered by the bookies are such that if horse i wins then your return is $m(t+1) = b_i o_i m(t)$. Assuming the bookies' odds are 'fair', that is,

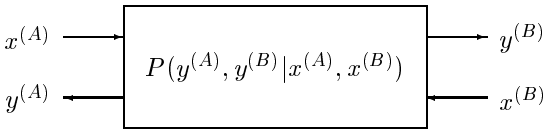
$$\sum_i \frac{1}{o_i} = 1, \tag{18.5}$$

and assuming that the probability of horse i's winning is p_i, work out the optimal betting strategy if your aim is to maximize the expected value of $\log m(T)$. Show that the optimal strategy sets $\mathbf{b}$ equal to $\mathbf{p}$, independent of $\mathbf{o}$. Show that when this strategy is used, the money is expected to grow exponentially as:

$$2^{nW(\mathbf{b},\mathbf{p})} \tag{18.6}$$

where $W = \sum_i p_i \log b_i o_i$.

(a)



(b)

$y^{(A)}$:

$x^{(B)}$ \ $x^{(A)}$	0	1
0	0	0
1	1	0

$y^{(B)}$:

$x^{(B)}$ \ $x^{(A)}$	0	1
0	0	1
1	0	0

(c)

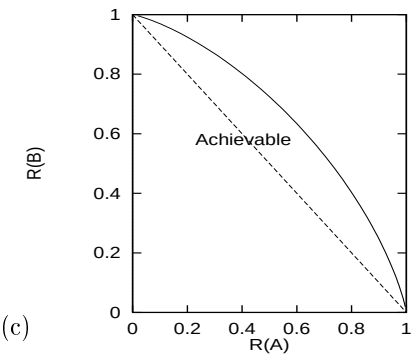


Figure 18.3. (a) A general two-way channel. (b) The rules for a binary two-way channel. The two tables show the outputs $y^{(A)}$ and $y^{(B)}$ that result for each state of the inputs. (c) Achievable region for the two-way binary channel. Rates below the solid line are achievable. The dotted line shows the 'obviously achievable' region which can be attained by simple time-sharing.

Solutions to Chapter 18's exercises

Solution to exercise 18.1 (p. 251):

(a) $\lceil \log_2 166751 \rceil = 18$ bits.

(b) 1.67×10^{-3}

Solution to exercise 18.2 (p. 251):

(a) $H_2(0.4804) = 0.998891$.

(b) $0.496 \times H_2(0.5597) + 0.504 \times H_2(0.6) = 0.9802$ bits

(c) 1 bit.

(d) $H_2(0.6) = 0.9709$ bits.

Solution to exercise 18.3 (p. 251): The optimal symbol code (*i.e.*, questioning strategy) has expected length $3\frac{11}{36}$.

Solution to exercise 18.8 (p. 252): By symmetry, the optimal input distribution for the channel

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-f & f \\ 0 & f & 1-f \end{bmatrix} \tag{18.7}$$

has the form $((1-p), p/2, p/2)$. The optimal input distribution is given by

$$p^* = \frac{1}{1 + 2^{H_2(f)-1}}. \tag{18.8}$$

In the case $f = 1/3$, $p^* = 0.514$ and the capacity is $C = 1.041$ bits.

Solution to exercise 18.6 (p. 252): $|T| = 2716$.

Solution to exercise 18.9 (p. 252): $C(Q) = 5$ bits.
Hint for last part: a solution exists that involves a simple (8,5) code.

Solution to exercise 18.11 (p. 253): The optimal input distribution is $(1/6, 1/6, 1/3, 1/3)$, and the capacity is $\log_2 3$ bits.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part IV

Probabilities and Inference

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

19

Introduction to Part IV

The number of inference problems that can (and perhaps should) be tackled by Bayesian inference methods is enormous. In this book, for example, we will discuss the decoding problem for error correcting codes, the task of infering clusters from data, the task of interpolation through noisy data, and the task of classifying patterns given labelled examples. Most techniques for solving these problems can be categorized as follows.

Exact methods compute the required quantities directly. Only a few interesting problems have a direct solution, but exact methods are important as tools for solving subtasks within larger problems. Methods for the exact solution of inference problems are the subject of chapter 20.

Approximate methods can be subdivided into

1. **Deterministic approximations**, which include Laplace's method (chapter 22) and variational methods (chapter 26); and
2. **Monte Carlo methods**, which will be discussed in chapter 23.

This part of the book does not form a one–dimensional story. Rather, the ideas make up a web of interrelated threads which will recombine in subsequent chapters.

Chapter 3, which is an honorary member of this part of the book, discussed a range of simple examples of inference problems and their Bayesian solution.

Chapter 20 discusses exact inference methods, using the decoding of error–correcting codes as an example. This chapter is a prerequisite for the understanding of advanced error–correcting codes.

Chapter 22 discusses deterministic approximations including Laplace's method. This chapter is a prerequisite for understanding the topic of complexity control in learning algorithms.

Chapter 23 discusses Monte Carlo methods.

Chapter 24 introduces the Ising model as a test–bed for probabilistic methods. An exact method and a Monte Carlo method are demonstrated. A motivation for studying the Ising model is that it is intimately related to several neural network models.

Chapter 26 discusses variational methods and their application to Ising models and to simple statistical inference problems. This chapter will help the reader understand the Hopfield network (chapter 33) and the EM algorithm (chapter 42).

A theme: what inference is about

A widespread misconception is that the aim of inference is to find *the most probable explanation* for some data. While this most probable hypothesis may be of interest, and some inference methods may locate it, this hypothesis is just the peak of a probability distribution, and it is the whole distribution that is of interest.

As we saw in chapter 4, the *most probable* outcome from a source is often not a *typical* outcome from that source. Similarly, the most probable hypothesis given some data may be atypical compared with the whole set of reasonably–plausible hypotheses.

20

Exact Inference Methods

In this chapter we will discuss a small subset of the exact methods that are used in probabilistic modelling. We will do this with the aid of two examples. The first is the task of decoding a linear error-correcting code. The second is the burglar-alarm problem of exercise 21.7 (p. 279). In both examples we will see that inferences can be conducted most efficiently by *message-passing algorithms*, which take advantage of the graphical structure of the problem to avoid unnecessary duplication of computations.

20.1 The decoding problems

A codeword $\mathbf{t}$ is selected from a linear (N, K) code $\mathcal{C}$, and it is transmitted over a noisy channel; the received signal is $\mathbf{y}$. In this chapter we will assume that the channel is a memoryless channel such as a Gaussian channel. Given an assumed channel model $P(\mathbf{y}|\mathbf{t})$, there are two decoding problems.

The codeword decoding problem is the task of inferring which codeword $\mathbf{t}$ was transmitted given the received signal.

The bitwise decoding problem is the task of inferring for each transmitted bit t_n how likely it is that that bit was a one rather than a zero.

As a concrete example, let us take the (7,4) Hamming code. In chapter 1, we partially discussed how to decode that code. We considered the case of a binary symmetric channel, and discussed a method for deducing the *most probable codeword* from the syndrome of the received signal. We didn't discuss the bitwise decoding problem and we didn't discuss how to handle more general channel models such as a Gaussian channel.

Solving the codeword decoding problem

By Bayes' theorem, the posterior probability of the codeword $\mathbf{t}$ is

$$P(\mathbf{t}|\mathbf{y}) = \frac{P(\mathbf{y}|\mathbf{t})P(\mathbf{t})}{P(\mathbf{y})}. \tag{20.1}$$

Likelihood function. The first factor in the numerator, $P(\mathbf{y}|\mathbf{t})$, is the *likelihood* of the codeword, which, for any memoryless channel, is a separable function,

$$P(\mathbf{y}|\mathbf{t}) = \prod_{n=1}^{N} P(y_n|t_n). \tag{20.2}$$

For example, if the channel is a Gaussian channel with transmissions $\pm x$ and additive noise of standard deviation σ, then the probability density of the received signal y_n in the two cases $t_n = 0, 1$ is

$$P(y_n|t_n{=}1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - x)^2}{2\sigma^2}\right) \tag{20.3}$$

$$P(y_n|t_n{=}0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n + x)^2}{2\sigma^2}\right). \tag{20.4}$$

From the point of view of decoding, all that matters is the *likelihood ratio* which for the case of the Gaussian channel is

$$\frac{P(y_n|t_n = 1)}{P(y_n|t_n = 0)} = \exp\left(\frac{2xy_n}{\sigma^2}\right). \tag{20.5}$$

Exercise 20.1:^{A2} Show that from the point of view of decoding, a Gaussian channel is equivalent to a time-varying binary symmetric channel with a known noise level f_n which depends on n.

Prior. The second factor in the numerator is the *prior* probability of the codeword, $P(\mathbf{t})$, which is often assumed to be uniform over all valid codewords.

The denominator is simply the normalizing constant

$$P(\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{y}|\mathbf{t})P(\mathbf{t}). \tag{20.6}$$

The complete solution to the codeword decoding problem is a list of all codewords and their probabilities as given by equation (20.1). Since the number of codewords in a linear code, 2^K, is often very large, and since we are not interested in knowing the detailed probabilities of all the codewords, we often restrict attention to a simplified version of the codeword decoding problem.

The MAP codeword decoding problem is the task of identifying *the most probable codeword* $\mathbf{t}$ given the received signal.

If the prior probability over codewords is uniform then this task is identical to the problem of *maximum likelihood decoding*, that is, identifying the codeword that maximizes $P(\mathbf{y}|\mathbf{t})$.

This problem can obviously be solved in exponential time (of order 2^K) by searching through all codewords for the one that maximizes $P(\mathbf{y}|\mathbf{t})P(\mathbf{t})$. But we are interested in methods that are more efficient than this. In section 20.4, we will discuss an exact method known as the *Viterbi algorithm* or *min-sum algorithm* which may be able to solve the codeword decoding problem more efficiently; how much more efficiently depends on the properties of the code.

It is worth emphasizing that the MAP codeword decoding problem for a *general* linear code is known to be NP-complete, which means in layman's terms that MAP codeword decoding can only be done in general in a time that scales exponentially with the block length, unless there is a revolution in computer science. So restricting attention to the MAP decoding problem hasn't necessarily made the task much less challenging; it simply makes the answer briefer to report.

Solving the bitwise decoding problem

Formally, the exact solution of the bitwise decoding problem is obtained from equation (20.1) by *marginalizing* over the other bits.

$$P(t_n|\mathbf{y}) = \sum_{\{t_{n'}: n' \neq n\}} P(\mathbf{t}|\mathbf{y}) \tag{20.7}$$

We can also write this marginal with the aid of a delta function $\delta(S)$ that is one if the proposition S is true and zero otherwise.

$$P(t_n = 1|\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t}|\mathbf{y})\delta(t_n = 1) \tag{20.8}$$

$$P(t_n = 1|\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t}|\mathbf{y})\delta(t_n = 0) \tag{20.9}$$

Computing these marginal probabilities by an explicit sum over all codewords $\mathbf{t}$ takes exponential time. But, for certain codes, the bitwise decoding problem can be solved much more efficiently using the *forward-backward algorithm* or *sum-product* algorithm. We will describe this algorithm in a moment. Both the min-sum algorithm and the sum-product algorithm have widespread importance, and have been invented many times in many fields.

20.2 Codes and trellises

In chapters 1 and 11, we represented linear (N, K) codes in terms of their generator matrices and their parity check matrices. In the case of a *systematic* block code, the first K transmitted bits in each block of size N are the source bits, and the remaining $M = N - K$ bits were the parity check bits. This means that the generator matrix of the code can be written

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P} \end{bmatrix}, \tag{20.10}$$

and the parity check matrix can be written

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_M \end{bmatrix}, \tag{20.11}$$

where $\mathbf{P}$ is an $M \times K$ matrix.

In this section we will now study another representation of a linear code called a trellis. The codes that these trellises represent will not in general be systematic codes, but they can be mapped onto systematic codes if desired by a reordering of the bits in a block.

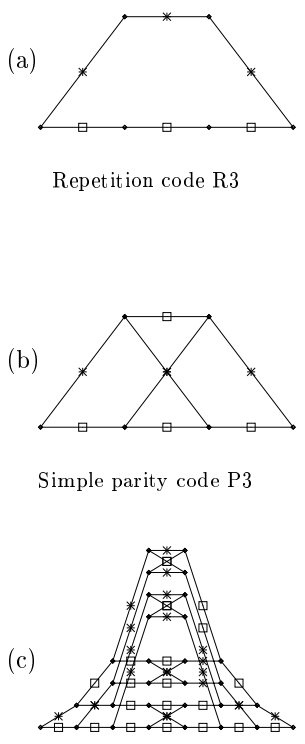


Figure 20.1. Examples of trellises. Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

Definition of a trellis

Our definition of a trellis will be quite narrow. For a more comprehensive view of trellises, the reader should consult Kschischang and Sorokine (1995).

A trellis is a *graph* consisting of *nodes* (also known as vertices) and *edges*. The nodes are grouped into vertical slices called *states*, and the states are ordered such that each edge connects a node in one state to a node in a neighbouring state. Every edge is labelled with a *symbol*. The leftmost and rightmost states contain only one node. Apart from these two extreme nodes, all nodes in the trellis have at least one edge connecting leftwards and at least one connecting rightwards.

A trellis with $N+1$ states defines a code of block length N as follows: the set of codewords in the code is obtained by taking all paths that cross the trellis from left to right and reading out the symbols on the edges that are traversed. Each valid path through the trellis defines a codeword. We will number the leftmost state 'state 0' and the rightmost 'state N'. The nth bit of the codeword is emitted as we move from state $n-1$ to state n.

The *width* of the trellis at a given state is the number of nodes in that state. The *maximal width* of a trellis is what it sounds like.

A trellis is called a *linear trellis* if the code it defines is a linear code. We will solely be concerned with linear trellises from now on, as nonlinear trellises are much more complex beasts (Kschischang and Sorokine, 1995). For brevity, we will only discuss binary trellises, that is, trellises whose edges are labelled with zeroes and ones. It is not hard to generalize the methods that follow to q-ary trellises.

Figures 20.1(a,b) show the trellises corresponding to the repetition code R_3 which has $(N, K) = (3, 1)$ and the parity code P_3 with $(N, K) = (3, 2)$.

Exercise 20.2:^{B2} Confirm that the sixteen codewords listed in figure 1.11 are generated by the trellis shown in figure 20.1(c).

Observations about linear trellises

For any linear code the *minimal trellis* is the one that has the smallest number of nodes. In a minimal trellis, each node has at most two edges entering it and at most two edges leaving it. All nodes in a state have the same left degree as each other and they have the same right degree as each other. The width is always a power of two.

A minimal trellis for a linear (N, K) code cannot have a width greater than 2^K since every node has at least one valid codeword through it, and there are only 2^K codewords. Furthermore, if we define $M = N - K$, the minimal trellis's width is everywhere less than 2^M. This will be proved in section **??**.

Notice that for the linear trellises in figure 20.1, all of which are minimal trellises, K is the number of times a binary branch point is encountered as the trellis is traversed from left to right or from right to left.

We will discuss the construction of trellises more in section 20.6. But we now know enough to discuss the decoding problem.

20.3 Solving the decoding problems on a trellis

We can view the trellis of a linear code as giving a causal description of the probabilistic process that gives rise to a codeword, with time flowing from left to right (or the other way). At each timestep we move one state to the right. Each time a divergence is encountered, a random source (the source of information bits for communication) determines which way we go.

At the receiving end, we receive a noisy version of the sequence of edge-labels, and wish to infer which path was taken, or to be precise, we want (a) to identify the most probable path; (b) to find the probability that the transmitted symbol at time n was a zero or a one.

Example 20.3: Consider the case of a a single transmission from the Hamming (7,4) trellis shown in figure 20.1c.

t	likelihood	posterior probability
0000000	0.0275562	0.2456
0001011	0.0001458	0.0013
0010111	0.0013122	0.0117
0011100	0.0030618	0.0273
0100110	0.0002268	0.0020
0101101	0.0000972	0.0009
0110001	0.0708588	0.6315
0111010	0.0020412	0.0182
1000101	0.0001458	0.0013
1001110	0.0000042	0.0000
1010010	0.0030618	0.0273
1011001	0.0013122	0.0117
1100011	0.0000972	0.0009
1101000	0.0002268	0.0020
1110100	0.0020412	0.0182
1111111	0.0000108	0.0001

Figure 20.2. Posterior probabilities over the sixteen codewords when the received vector $\mathbf{y}$ has normalized likelihoods $(0.1, 0.4, 0.9, 0.1, 0.1, 0.1, 0.3)$.

n	Likelihood		Posterior marginals	
	$P(y_n\|t_n{=}1)$	$P(y_n\|t_n{=}0)$	$P(t_n{=}1\|\mathbf{y})$	$P(t_n{=}0\|\mathbf{y})$
1	0.1	0.9	0.061	0.939
2	0.4	0.6	0.674	0.326
3	0.9	0.1	0.746	0.254
4	0.1	0.9	0.061	0.939
5	0.1	0.9	0.061	0.939
6	0.1	0.9	0.061	0.939
7	0.3	0.7	0.659	0.341

Figure 20.3. Marginal posterior probabilities for the 7 bits under the posterior distribution of figure 20.2.

Let the normalized likelihoods be: $(0.1, 0.4, 0.9, 0.1, 0.1, 0.1, 0.3)$. That is, the ratios of the likelihoods are

$$\frac{P(y_1|x_1{=}1)}{P(y_1|x_1{=}0)} = \frac{0.1}{0.9}, \quad \frac{P(y_2|x_2{=}1)}{P(y_2|x_2{=}0)} = \frac{0.4}{0.6}, \quad \text{etc.} \tag{20.12}$$

How should this received signal be decoded?

(a) If we threshold the likelihoods at 0.5 in order to simplify the signal into a binary received vector, we have $\mathbf{r} = (0, 0, 1, 0, 0, 0, 0)$, which decodes, using the optimal decoder for the binary symmetric channel, into $\hat{\mathbf{t}} = (0, 0, 0, 0, 0, 0, 0)$.

(b) We can find the posterior probability over codewords by explicit enumeration of all sixteen codewords. This is shown in figure 20.2. Of course, we aren't really interested in such brute-force solutions, and the aim of the following sections is to understand algorithms for getting the same information out in less than 2^K computer time.

Examining the posterior probabilities, we notice that the most probable codeword is actually the string $\mathbf{t} = \texttt{0110001}$. This is more than twice as probable as `0000000`.

Using the posterior probabilities shown in figure 20.2, we can also compute the posterior marginal distributions of each of the bits. The result is shown in figure 20.3. Notice that bits 1, 4, 5 and 6 are all quite confidently inferred to be zero. The strengths of the posterior probabilities for bits 2, 3 and 7 are not so great.

In the above example, the MAP codeword is in agreement with the bitwise decoding that is obtained by selecting the most probable state for each bit using the posterior marginal distributions. But this is not always the case, as the following exercise shows.

Exercise 20.4:[A2] Find the most probable codeword in the case where the normalized likelihood is $(0.2, 0.2, 0.9, 0.2, 0.2, 0.2, 0.2)$. Also find or estimate the marginal posterior probability for each of the seven bits, and give the bit-by-bit decoding.

20.4 The min-sum algorithm

UNFINISHED SECTION.

Connect this section to counting paths in the constrained channel, chapter 17.

Consider a node on the most probable path, which has two upstream parents. The most probable way of creating the first n emissions and getting to the present node must be the same as the most probable path, because if it weren't.... To find the most probable path to a node, only need to know the score of the most probable paths to its parents, and the score associated with transitions from those two parents. Then can identify which is the cheaper parent.

20.5 The sum-product algorithm

UNFINISHED SECTION. aka Belief propagation. The forward-backward algorithm.

20.6 Exact inference for other problems

UNFINISHED SECTION. Here we blend seamlessly into other tasks and show how belief propagation is useful for earthquakes too.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

More on trellises

In this appendix we discuss various ways of making the trellis of a code. You may safely jump over this section.

The *span* of a codeword is the set of bits contained between the first bit in the codeword that is non-zero, and the last bit that is non-zero, inclusive. We can indicate the span of a codeword by a binary vector as shown in figure 20.4. A generator matrix is in *trellis-oriented form* if the spans of the rows of the generator matrix all start in different columns and the spans all end in different columns.

Codeword	0000000	0001011	0100110	1100011	0101101
Span	0000000	0001111	0111110	1111111	0111111

Figure 20.4. Some codewords and their spans

How to make a trellis from a generator matrix

First, put the generator matrix into trellis-oriented form by row-manipulations similar to Gaussian elimination. For example, our (7,4) Hamming code can be generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{20.13}$$

but this matrix is not in trellis-oriented form — for example, rows 1, 3 and 4 all have spans that end in the same column. By subtracting lower rows from upper rows, we can obtain an equivalent generator matrix (that is, one that generates the same set of codewords) as follows:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{20.14}$$

Now, each row of the generator matrix can be thought of as defining an $(N, 1)$ subcode of the (N, K) code, that is, in this case, a code with two codewords of length $N = 7$. For the first row, the code consists of the two codewords `1101000` and `0000000`. The subcode defined by the second row consists of `0100110` and `0000000`. It is easy to construct the minimal trellises of these subcodes; they are shown in the left column of figure 20.5.

We build the trellis incrementally. First we make the trellis corresponding to the subcode given by the first row of the generator matrix. Then *more here...*

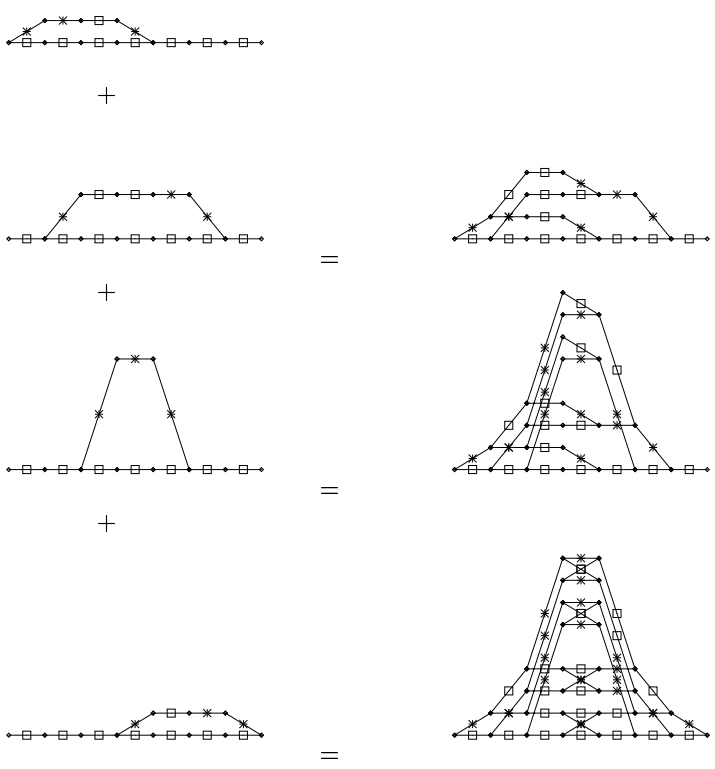


Figure 20.5. Trellises for four subcodes of the (7,4) Hamming code (left column), and the sequence of trellises that are made when constructing the trellis for the (7,4) Hamming code (right column).
Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Another (7,4) Hamming code can be generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{20.15}$$

The (7,4) Hamming code generated by this matrix differs by a permutation of its bits from the code generated by the systematic matrix used in chapter 1 and above. This permutation has been chosen such that the parity check matrix can be written thus:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{20.16}$$

The trellis obtained from the permuted matrix $\mathbf{G}$ given in equation (20.15) is shown in figure 20.6(a). Notice that the number of nodes in this trellis is smaller than the number of nodes in the previous trellis for the Hamming (7,4) code in figure 20.1(c). We thus observe that *rearranging the order of the codeword bits can sometimes lead to smaller, simpler trellises.*

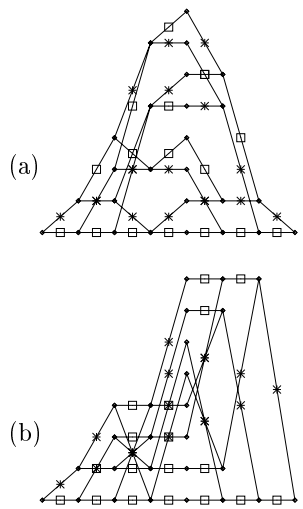


Figure 20.6. Trellises for permuted (7,4) Hamming code generated from (a) the generator matrix; (b) the parity check matrix.
Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

Trellises from parity check matrices

Another way of viewing the trellis is in terms of the syndrome. The syndrome of a vector $\mathbf{r}$ is defined to be $\mathbf{Hr}$, where $\mathbf{H}$ is the parity check matrix. A vector is only a codeword if its syndrome is zero. As we generate a codeword we can describe the current state by the *partial syndrome*, that is, the product of $\mathbf{H}$ with the codeword bits thus far generated. Each state in the trellis is a partial syndrome at one time coordinate. The starting and ending states are both constrained to be the zero syndrome. Each node in a state represents a different possible value for the partial syndrome. Since $\mathbf{H}$ is an $M \times N$ matrix, where $M = N - K$, the syndrome is at most an M-bit vector. So we need at most 2^M nodes in each state. We can construct the trellis of a code from its parity check matrix by walking from each end, generating two trees of possible syndromes sequences. The intersection of these two trees defines the trellis of the code.

In the pictures we obtain from this construction, we can let the vertical coordinate represent the syndrome. Then any horizontal edge is necessarily associated with a zero bit (since any non-zero bit changes the syndrome) and any non-horizontal edge is associated with a one bit. Thus in this representation we no longer need to label the edges in the trellis. Figure 20.6(b) shows the trellis corresponding to the parity check matrix of equation (20.16).

Solutions to Chapter 20's exercises

Solution to exercise 20.4 (p. 267): The posterior probability over codewords is:

t	likelihood	posterior probability
0000000	0.0262144	0.3006
0001011	0.0004096	0.0047
0010111	0.0036864	0.0423
0011100	0.0147456	0.1691
0100110	0.0004096	0.0047
0101101	0.0001024	0.0012
0110001	0.0147456	0.1691
0111010	0.0036864	0.0423
1000101	0.0004096	0.0047
1001110	0.0001024	0.0012
1010010	0.0147456	0.1691
1011001	0.0036864	0.0423
1100011	0.0001024	0.0012
1101000	0.0004096	0.0047
1110100	0.0036864	0.0423
1111111	0.0000576	0.0007

The most probable codeword is 0000000. The marginal posterior probabilities of all seven bits are:

n	Likelihood		Posterior marginals	
	$P(y_n\|t_n{=}1)$	$P(y_n\|t_n{=}0)$	$P(t_n{=}1\|\mathbf{y})$	$P(t_n{=}0\|\mathbf{y})$
1	0.2	0.8	0.266	0.734
2	0.2	0.8	0.266	0.734
3	0.9	0.1	0.677	0.323
4	0.2	0.8	0.266	0.734
5	0.2	0.8	0.266	0.734
6	0.2	0.8	0.266	0.734
7	0.2	0.8	0.266	0.734

So the bitwise decoding is 0010000, which is not actually a codeword.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

21

Intermediate Bayesian Stuff

21.1 Inferring a Gaussian distribution

The one–dimensional Gaussian distribution is parameterized by a mean μ and a standard deviation σ:

$$P(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \equiv \text{Normal}(x;\mu,\sigma^2). \tag{21.1}$$

Let us examine the inference of μ and σ given data points x_n, $n = 1,\ldots,N$, assumed to be drawn independently from this distribution. When inferring these parameters, we must specify their prior distribution. This gives us the opportunity to include specific knowledge that we have about μ and σ (from independent experiments, or on theoretical grounds, for example). If we have no such knowledge, then we can construct an appropriate prior which embodies our supposed ignorance. In either case, it may be convenient to consider *conjugate priors*; these are priors which have a functional form which combines naturally with data measurements such that the inferences have a convenient form. The conjugate prior for a mean μ is a Gaussian, $P(\mu|\mu_0,\sigma_\mu) = \text{Normal}(\mu;\mu_0,\sigma_\mu)$. In the limit $\mu_0 = 0, \sigma_\mu \to \infty$, we obtain the *noninformative prior* for a location parameter, the flat prior. This is *noninformative* because it is *invariant* under the natural reparameterization $\mu' = \mu + c$. The prior $P(\mu) =$ const. is also an *improper* prior, that is, it is not normalizable.

I need to give a better explanation of 'noninformative'.

The conjugate prior for a standard deviation σ is a gamma distribution, conveniently defined in terms of the inverse variance $\beta = 1/\sigma^2$:

$$P(\beta) = \Gamma(\beta; b_\beta, c_\beta) = \frac{1}{\Gamma(c_\beta)}\frac{\beta^{c_\beta-1}}{b_\beta^{c_\beta}}\exp\left(-\frac{\beta}{b_\beta}\right), 0 \le \beta < \infty \tag{21.2}$$

This is a simple peaked distribution with mean $b_\beta c_\beta$ and variance $b_\beta^2 c_\beta$. In the limit $b_\beta c_\beta = 1, c_\beta \to 0$, we obtain the noninformative prior for a scale parameter, the $1/\sigma$ prior. This is 'noninformative' because it is invariant under the reparameterization $\sigma' = c\sigma$. The $1/\sigma$ prior is less strange looking if we examine the resulting density over $\log\sigma$, or $\log\beta$, which is flat. This is the prior that expresses ignorance about σ by saying 'well, it could be 10, or it could be 1, or it could be 0.1, ...' Scale variables such as σ are usually best represented in terms of their logarithm. Again, this noninformative $1/\sigma$ prior is improper.

In the following examples, I will use the improper noninformative priors for μ and σ.

Maximum likelihood and marginalization: σ_N and σ_{N-1}

The task of inferring the mean and standard deviation of a Gaussian distribution from N samples is a familiar one, though maybe not everyone understands the difference between the σ_N and σ_{N-1} buttons on their calculator. Let us recap the formulae, then derive them.

Given data $D = \{x_n\}_{n=1}^N$, an 'estimator' of μ is

$$\bar{x} \equiv \sum_{n=1}^N x_n / N, \tag{21.3}$$

and two estimators of σ are:

$$\sigma_N \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N}} \quad \text{and} \quad \sigma_{N-1} \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N-1}} \tag{21.4}$$

There are two principal paradigms for statistics: sampling theory and Bayesian inference. In sampling theory (also known as 'frequentist' or orthodox statistics), one invents 'estimators' of quantities of interest and then chooses between those estimators using some criterion measuring their sampling properties; there is no clear principle for deciding which criterion to use to measure the performance of an estimator; nor, for most criteria, is there any systematic procedure for the construction of optimal estimators. In Bayesian inference, in contrast, once we have made explicit all our modelling assumptions, our inferences are mechanical. Whatever question we wish to pose, the rules of probability theory give a unique answer which consistently takes into account all the given information. Human–designed estimators and confidence intervals have no role in Bayesian inference; human input only enters into the important tasks of designing the hypothesis space, and figuring out how to do the computations that implement inference in that space. The answers to our questions are probability distributions over the quantities of interest. We often find that the estimators of sampling theory emerge automatically as modes or means of these posterior distributions when we choose a simple hypothesis space and turn the handle of Bayesian inference.

In sampling theory, the estimators above can be motivated as follows. $\bar{x}$ is an unbiased estimator of μ which, out of all the possible unbiased estimators of μ, has smallest variance (where this variance is computed by averaging over an ensemble of imaginary experiments in which the data samples are assumed to come from an unknown Gaussian distribution). The estimator $(\bar{x}, \sigma_N)$ is the maximum likelihood estimator for (μ, σ). The estimator σ_N is *biased*, however: the expectation of σ_N, given σ, averaging over many imagined experiments, is not σ.

Exercise 21.1:[A2] Give an intuitive explanation why the estimator σ_N is biased.

This motivates the invention of σ_{N-1} which can be shown to be an unbiased estimator. Or to be precise, it is σ_{N-1}^2 which is an unbiased estimator of σ^2.

We now look at some Bayesian inferences for this problem, assuming non-informative priors for μ and σ. The emphasis is thus not on the priors, but rather on (a) the likelihood function, and (b) the concept of marginalization. The joint posterior probability of μ and σ is proportional to the likelihood function illustrated by a contour plot in figure 21.1a. The log likelihood is:

$$\log P(\{x_n\}_{n=1}^N | \mu, \sigma) = -N \log(\sqrt{2\pi}\sigma) - \sum_n (x_n - \mu)^2/(2\sigma^2), \tag{21.5}$$

$$= -N \log(\sqrt{2\pi}\sigma) - [N(\mu - \bar{x})^2 + S]/(2\sigma^2), \tag{21.6}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

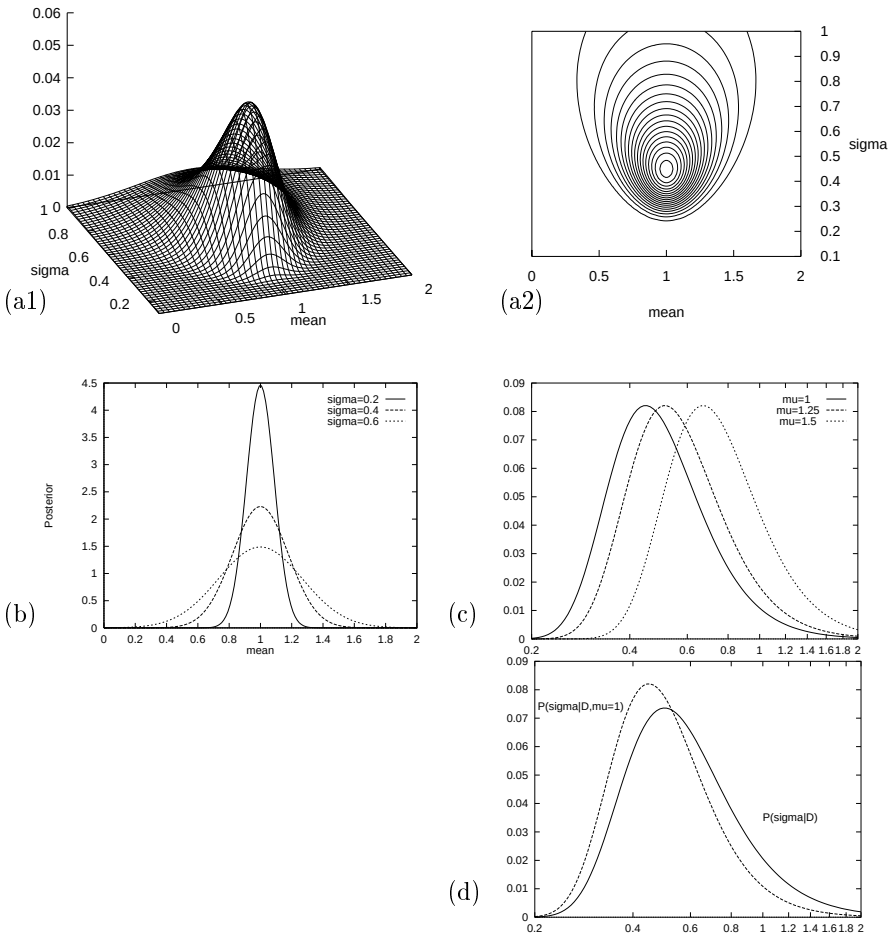


Figure 21.1. The likelihood function for the parameters of a Gaussian distribution.
(a1,a2) Surface plot and contour plot of the log likelihood as a function of μ and σ. The data set of $N = 5$ points had mean $\bar{x} = 1.0$ and $S^2 = \sum(x - \bar{x})^2 = 1.0$. Notice that the maximum is skew in σ. The two estimators of standard deviation have values $\sigma_N = 0.45$ and $\sigma_{N-1} = 0.50$.
(b) The posterior probability of μ for various values of σ.
(c) The posterior probability of σ for various fixed values of μ.
(d) The posterior probability of σ, $P(\sigma|D)$, assuming a flat prior on μ, obtained by projecting the probability mass in (a) onto the σ axis. The maximum of $P(\sigma|D)$ is at σ_{N-1}. By contrast, the maximum of $P(\sigma|D, \mu = \bar{x}, \sigma)$ is at σ_N.

where $S \equiv \sum_n (x_n - \bar{x})^2$. Given the Gaussian model, the likelihood can be expressed in terms of the two functions of the data $\bar{x}$ and S, so these two quantities are known as 'sufficient statistics'. The posterior probability of μ and σ is, using the improper priors:

$$P(\mu, \sigma | \{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu, \sigma)}{P(\{x_n\}_{n=1}^N)} \tag{21.7}$$

$$= \frac{\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N(\mu-\bar{x})^2+S}{2\sigma^2}\right) \frac{1}{\sigma_\mu} \frac{1}{\sigma}}{P(\{x_n\}_{n=1}^N)} \tag{21.8}$$

This function describes the answer to the question, 'given the data, and the noninformative priors, what might μ and σ be?' It may be of interest to find the parameter values that maximize the posterior probability, though it should emphasized that posterior probability maxima have no fundamental status in Bayesian inference, since their location depends on the choice of basis. Here we choose the basis $(\mu, \log \sigma)$, in which our prior is flat, so that the posterior probability maximum coincides with the maximum of the likelihood. Differentiating the log likelihood with respect to μ and $\log \sigma$ we find the maximum likelihood solution: $\{\mu, \sigma\}_{\rm ML} = \left\{\bar{x}, \sigma_N = \sqrt{S/N}\right\}$.

There is more to the posterior distribution than just its mode. As can be seen in figure 21.1a, the likelihood has a skew peak. As we increase σ, the width of the conditional distribution of μ increases (figure 21.1b). And if we fix μ to a sequence of values moving away from the sample mean $\bar{x}$, we obtain a sequence of conditional distributions over σ whose maxima move to increasing values of σ (figure 21.1c).

The posterior probability of μ given σ is

$$P(\mu | \{x_n\}_{n=1}^N, \sigma) = \frac{P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu)}{P(\{x_n\}_{n=1}^N | \sigma)} \tag{21.9}$$

$$\propto \exp(-N(\mu - \bar{x})^2/(2\sigma^2)) \tag{21.10}$$

$$= \text{Normal}(\mu; \bar{x}, \sigma^2/N). \tag{21.11}$$

We note the familiar $\sigma/\sqrt{N}$ scaling of the error bars on μ.

Let us now ask the question 'given the data, and the noninformative priors, what might σ be?' This question differs from the first one we asked in that we are now not interested in μ. This parameter must therefore be *marginalized* over. The posterior probability of σ is:

$$P(\sigma | \{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N | \sigma) P(\sigma)}{P(\{x_n\}_{n=1}^N)}. \tag{21.12}$$

The data–dependent term $P(\{x_n\}_{n=1}^N | \sigma)$ appeared earlier as the normalizing constant in equation (21.9); one name for this quantity is the 'evidence', or marginal likelihood, for σ. We obtain the evidence for σ by integrating out μ; a noninformative prior $P(\mu) =$ constant is assumed; we call this constant $1/\sigma_\mu$, so that we can think of the prior as a top-hat prior of width σ_μ. The Gaussian integral, $P(\{x_n\}_{n=1}^N | \sigma) = \int P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu) \, d\mu$, yields:

$$\log P(\{x_n\}_{n=1}^N | \sigma) = -N \log(\sqrt{2\pi}\sigma) - \frac{S}{2\sigma^2} + \log \frac{\sqrt{2\pi}\sigma/\sqrt{N}}{\sigma_\mu}. \tag{21.13}$$

The first two terms are the best fit log likelihood (*i.e.*, the log likelihood with $\mu = \bar{x}$). The last term is the log of the 'Occam factor' which penalizes smaller values of σ. (We will discuss Occam factors more in a later chapter.) When we differentiate the log evidence with respect to $\log \sigma$, to find the most probable σ, the additional volume factor $(\sigma/\sqrt{N})$ shifts the maximum from σ_N to

$$\sigma_{N-1} = \sqrt{S/(N-1)} \tag{21.14}$$

Intuitively, the denominator $(N-1)$ counts the number of noise measurements contained in the quantity $S = \sum_n (x_n - \bar{x})^2$. The sum contains N residuals squared, but there are only $(N-1)$ effective noise measurements because the determination of one parameter μ from the data causes one dimension of noise to be gobbled up in unavoidable overfitting.

Figure 21.1d shows the posterior probability of σ, which is proportional to the marginal likelihood. This may be contrasted with the posterior probability of σ with μ fixed to its most probable value, $\bar{x} = 1$, which is shown in figure 21.1c and d.

The final inference we might wish to make is 'given the data, what is μ?' To answer this, we marginalize over σ and obtain the posterior marginal distribution of μ, which is a Student t-distribution:

$$P(\mu|D) \propto 1/\left(N(\mu - \bar{x})^2 + S\right)^{N/2}. \tag{21.15}$$

Exercises

Exercise 21.2:^{A2} A data file consisting of two columns of numbers has been printed in such a way that the boundaries between the columns are unclear. Here are the resulting strings.

891.10.0	912.20.0	874.10.0	870.20.0	836.10.0	861.20.0
903.10.0	937.10.0	850.20.0	916.20.0	899.10.0	907.10.0
924.20.0	861.10.0	899.20.0	849.10.0	887.20.0	840.10.0
849.20.0	891.10.0	916.20.0	891.10.0	912.20.0	875.10.0
898.20.0	924.10.0	950.20.0	958.10.0	971.20.0	933.10.0
966.20.0	908.10.0	924.20.0	983.10.0	924.20.0	908.10.0
950.20.0	911.10.0	913.20.0	921.25.0	912.20.0	917.30.0
923.50.0					

Discuss how probable it is, given these data, that the correct parsing of each item is:

(a) 891.10.0 → 891. 10.0, etc.

(b) 891.10.0 → 891.1 0.0, etc.

Exercise 21.3:^{A2} Explain the terms *likelihood function*, *prior probability distribution*, and *posterior probability distribution*, in the context of the inference of parameters θ from data D.

A random variable x is assumed to have a probability distribution that is a *mixture of two Gaussians*,

$$P(x|\mu_1, \mu_2, \sigma) = \left[\sum_{c=1}^{2} p_c \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_c)^2}{2\sigma^2}\right)\right],$$

where the two Gaussians are given the labels $c = 1$ and $c = 2$; the prior probability of the class label c is $\{p_1 = 1/2, p_2 = 1/2\}$; $\{\mu_c\}$ are the means of the two Gaussians; and both have standard deviation σ. For brevity, we will denote these parameters by $\theta \equiv \{\{\mu_c\}, \sigma\}$.

A data set consists of N points $\{x_n\}_{n=1}^N$ which are assumed to be independent samples from this distribution. Let c_n denote the unknown class label of the nth point.

Assuming that $\{\mu_c\}$ and σ are known, show that the posterior probability of the class label c_n of the nth point can be written as

$$\begin{aligned} P(c_n = 1|x_n, \theta) &= \frac{1}{1 + \exp[-(w_1 x + w_0)]} \\ P(c_n = 2|x_n, \theta) &= \frac{1}{1 + \exp[+(w_1 x + w_0)]} \end{aligned} , \tag{21.16}$$

and give expressions for w_1 and w_0.

Assume now that the means $\{\mu_c\}$ are *not* known, and that we wish to infer them from the data $\{x_n\}_{n=1}^N$. (The standard deviation σ is known.) In the remainder of this question we will derive an iterative algorithm for finding values for $\{\mu_c\}$ which maximize the likelihood,

$$P(\{x_n\}_{n=1}^N | \{\mu_c\}, \sigma) = \prod_n P(x_n | \{\mu_c\}, \sigma).$$

Let L denote the log of the likelihood. Show that the derivative of the log likelihood with respect to μ_c is given by

$$\frac{\partial}{\partial \mu_c} L = \sum_n p_{c|n} \frac{(x_n - \mu_c)}{\sigma^2},$$

where $p_{c|n} \equiv P(c_n = c|x_n, \theta)$ appeared above at equation (21.16).

Show, neglecting terms in $\frac{\partial}{\partial \mu_c} P(c_n = c|x_n, \theta)$, that the second derivative is approximately given by

$$\frac{\partial^2}{\partial \mu_c^2} L \simeq -\sum_n p_{c|n} \frac{1}{\sigma^2}.$$

Hence show that from an initial state μ_1, μ_2, an approximate Newton–Raphson step updates these parameters to μ_1', μ_2', where

$$\mu_c' = \frac{\sum_n p_{c|n} x_n}{\sum_n p_{c|n}}.$$

[The Newton–Raphson method for maximizing $L(\mu)$ updates μ to $\mu' = \mu - \left[\frac{\partial L}{\partial \mu} \Big/ \frac{\partial^2 L}{\partial \mu^2} \right]$.]

Assuming that $\sigma = 1$, sketch a contour plot of the likelihood function L as a function of μ_1 and μ_2 for the data set shown below. The data set consists of 32 points. Describe the peaks in your sketch and indicate their widths.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

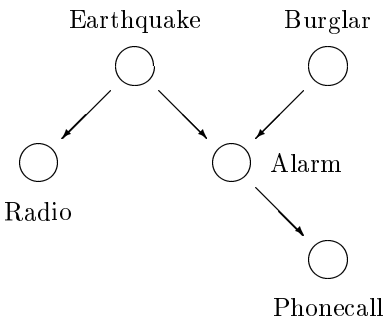


Figure 21.2. Belief network for the burglar alarm problem.

Counting combinations

Solutions on p. 281

Exercise 21.4:[B2] Professor Hinton lives in a city where the streets lie on a Cartesian grid. He walks to work, which is E blocks east and S blocks south of his residence. Assuming that he always takes one of the shortest routes, how many routes to work does he have available?

Exercise 21.5:[B2] Professor Neal likes to choose at random from a greater variety of routes than just the shortest routes. Assuming his workplace is S blocks south and E blocks east and that he selects at random from all walks of length $(S+E+2)$ blocks, how many walks to work does he have? What is the probability that he will in a single walk traverse one block in both directions?

Solutions on p. 282

Exercise 21.6:[C4] Two ordinary dice are thrown repeatedly; the outcome of each throw is the sum of the two numbers. Joe Shark, who says that 6 and 8 are his lucky numbers, bets even money that a 6 will be thrown before the first 7 is thrown. If you were a gambler, would you take the bet? What is your probability of winning? Joe then bets even money that an 8 will be thrown before the first 7 is thrown. Would you take the bet?

Having gained your confidence, Joe suggests combining the two bets into a single bet: he bets a larger sum, still at even odds, that an 8 and a 6 will be thrown before two 7s have been thrown. Would you take the bet? What is your probability of winning?

The burglar alarm

Exercise 21.7:[A1] Fred lives in Los Angeles and commutes 60 miles to work. Whilst at work, he receives a phone–call from his neighbour saying that his home's burglar alarm is ringing. What is the probability that there is a burglar in his house? While driving home to investigate, Fred hears on the radio that there was a small earthquake that day near his home. Given that earthquakes sometimes set off burglar alarms (figure 21.2), *now* what is the probability that there is (or was) a burglar in his house? (After Pearl, 1988).

Solutions to Chapter 21's exercises

Solution to exercise 21.1 (p. 274): The sample mean is unlikely to equal the true mean; the sample mean is the value of μ which minimizes the sum squared deviation of the data points from μ; now the data points are distributed with mean squared deviation σ^2 about the true mean, and their mean squared deviation from the sample mean will necessarily be smaller than this (since the sample mean is the value of μ which minimizes the sum squared deviation of the data points from μ).

I can probably improve on this answer.

Solution to exercise 21.2 (p. 277): Let's discuss the two possible parsings. The first parsing $\mathcal{H}_a$ produces a column of numbers all of which end in a decimal point. This might be viewed as a somewhat improbable parsing. Why is the decimal point there if no decimals follow it? On the other hand, this parsing makes every number four digits long, which has a pleasing and plausible simplicity to it.

However, if we use the second parsing $\mathcal{H}_b$ then the second column of numbers consists almost entirely of the number '0.0'. This also seems odd.

We could assign subjective priors to all these possibilities and suspicious coincidences. The most compelling evidence, however, comes from the fourth column of digits which are either the initial digits of a second list of numbers, or the final, post–decimal digits of the first list of numbers. What is the probability distribution of initial digits, and what is the probability distribution of final, post–decimal digits? It is often our experience that initial digits have a non–uniform distribution, with the digit '1' being much more probable than the digit '9'. Terminal digits often have a uniform distribution, or if they have a non–uniform distribution, it would be expected to be dominated either by '0' and '5' or by '0', '2', '4', '6', '8'. We don't generally expect the distribution of *terminal* digits to be asymmetric about '5', for example, we don't expect '2' and '8' to have very different probabilities.

The empirical distribution seems highly non–uniform and asymmetric, having 20 '1's, 21 '2's, one '3' and one '5'. This fits well with the hypothesis that the digits are initial digits, and does not fit well with any of the terminal digit distributions we thought of.

We can quantify the evidence in favour of the first hypothesis by picking a couple of crude assumptions: First, for initial digits,

$$P(n|\mathcal{H}_a) = \begin{cases} \frac{1}{Z}\frac{1}{n} & n \geq 1 \\ \frac{1}{Z}\frac{1}{10} & n = 0 \end{cases} , \tag{21.17}$$

where $Z = 2.93$, and second, for terminal digits,

$$P(n|\mathcal{H}_b) = \frac{1}{10}. \tag{21.18}$$

Then the probability of the given 43 digits is

$$P(\{n\}|\mathcal{H}_a) = 2.71 \times 10^{-28}. \tag{21.19}$$

$$P(\{n\}|\mathcal{H}_b) = 10^{-43}. \tag{21.20}$$

So the data consisting of the fourth column of digits favour $\mathcal{H}_a$ over $\mathcal{H}_b$ by about 10^{15} to 1.

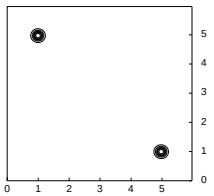
This is an unreasonably extreme conclusion, as is typical of carelessly constructed Bayesian models (Mosteller and Wallace, 1984). But the conclusion is correct; the data are real data that I received from a colleague, and the correct parsing is that of $\mathcal{H}_a$.

Solution to exercise 21.3 (p. 277): (Draft.)

The likelihood function is the probability $P(D|\theta)$ as a function of θ, where D is the observed data. The prior is $P(\theta)$ and it expresses what is known about the parameters before the data were observed. The posterior probability distribution is

$$P(\theta|D) = P(D|\theta)P(\theta)/P(D),$$

and it expresses what we know about the parameters given the data and the prior information.



The model answer at this point simply follows the instructions.

The likelihood function for the 32 data points is shown as a contour plot. The peaks should be pretty near centred on the points (1,5) and (5,1), and they should be pretty near circular in their contours. The width of each of the peaks should be a standard deviation of $\sigma/\sqrt{16} = 1/4$. The peaks are roughly Gaussian in shape.

Move this solution III?

Solution to exercise 21.4 (p. 279): Prof. Hinton can put his $E+S$ blocks in any order. The number of distinguishable walks is $\binom{E+S}{S} = \frac{(E+S)!}{E!S!}$.

Solution to exercise 21.5 (p. 279): Prof. Neal can either traverse $E+1$ blocks east, one west, and S south, in any order, or E east, $S+1$ south and one north. The first option yields $\frac{(E+S+2)!}{(E+1)!S!}$ routes, and the second $\frac{(E+S+2)!}{E!(S+1)!}$, so his number of routes is

$$\begin{aligned}\frac{(E+S+2)!}{(E+1)!S!} + \frac{(E+S+2)!}{E!(S+1)!} &= (E+S+2)\left[\binom{E+S+1}{S} + \binom{E+S+1}{S+1}\right] \\ &= (E+S+2)\binom{E+S+2}{S+1}. \end{aligned} \tag{21.21}$$

The probability that he will traverse one block in both directions is equal to the fraction of the above walks that double back in this way. The number of walks of length $E+S+2$ that double back can be obtained from number of walks Prof. Hinton uses by adding on extra steps. Assuming that a walk doubles back, there must either be one junction from which a sidebranch emerges and returns (there are $2(E+S)+4$ such options) or else there must be one block that is traversed three times (there are $E+S$ such options) so the total number of walks that double back is

$$\binom{E+S}{S}(2(E+S)+4+E+S) \tag{21.22}$$

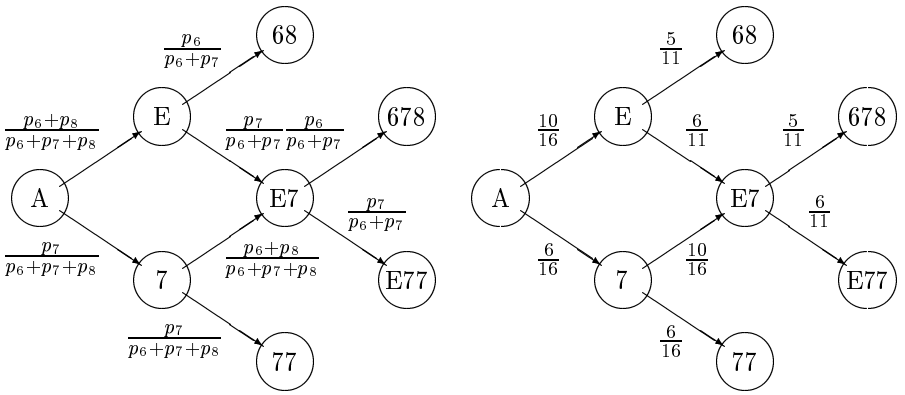


Figure 21.3. Markov process describing the Las Vegas dice game, pruned of all irrelevant outcomes. The end states 68 and 678 are wins for Joe. States $E77$ and 77 are wins for you. Please do not confuse this state diagram, in which arrows indicate which states can follow from each other, with a graphical model, in which each node represents a different variables and arrows indicate causal relationships between them.

so the probability of such a walk is

$$\frac{\binom{E+S}{S}\,(3(E+S)+4)}{(E+S+2)\binom{E+S+2}{S+1}} = \frac{(S+1)(E+1)(3(E+S)+4)}{(E+S+2)^2(E+S+1)}. \tag{21.23}$$

The Las Vegas trickster

Solution to exercise 21.6 (p. 279): On a single throw of the two dice, let the outcomes 6 and 7 have probabilities $P(6) = p_6$ and $P(7) = p_7$. Note $P(8) = p_6$. The values are $p_6 = 5/36$ and $p_7 = 6/36 = 1/6$. For the first bet, we can ignore other outcomes apart from the winning and losing outcomes 7 and 6 and compute the probability that the outcome is a 7, given that the game has terminated,

$$\frac{p_7}{p_6 + p_7} = 6/11 = 0.54545. \tag{21.24}$$

The second bet is identical. Both are favourable bets.

The third bet is the interesting one, because it is not a favourable bet for you, even though it sounds similar to the two bets that have gone before. The essential intuition for why two sevens are less probable than an 8 and a 6 is that the 8 and the 6 can come in either of two orders, so a rough factor of two appears in the probability for 8 and 6.

Computing the probability of winning is quite tricky if a neat route is not found. The probability is most easily computed if, as above, we *discard all the irrelevant events* and just compute the conditional probability of the different ways in which the state of the game can advance by one 'step'. The possible paths taken by this 'pruned' game with their probabilities are shown in the figure as a Markov process. (The original unpruned game is a similar Markov process in which an extra path emerges from each node, giving a transition back to the same node.) The node labelled 'A' denotes the initial state in which no 6s, 7s or 8s have been thrown. From here transitions are possible to state '7' in which exactly one 7 has been thrown, and no 6s or 8s; and to state 'E', in which either [one or more 8s have occurred and no 6s or 7s] or [one or more 6s have occurred and no 6s or 7s]. The probabilities of these transitions are shown. We can progress from state E only if Joe's winning 6 or 8 (whichever it is) is thrown, or if a 7 occurs. These events take us to the states labelled '68' and 'E7' respectively. From state '7' the game advances

when a 6 or 8 is thrown, or when a 7 is thrown, taking us to states 'E7' and '77' respectively. Finally, from state E7, if a 7 is thrown we transfer to state E77, and if Joe's required 6 or 8 is thrown, we move to state 678. States 68 and 678 are wins for Joe; states 77 and E77 are wins for you.

We first need the probability of state E7,

$$(10/16)(6/11) + (6/16)(10/16) = 405/704 = 0.5753 \tag{21.25}$$

The probability that you win is

$$P(77) + P(\mathrm{E77}) = (6/16)^2 + P(E7)(6/11) = 3519/7744 = 0.4544 \tag{21.26}$$

The bet is not favourable. Notice that Joe simply needs to make the third bet with a stake that is bigger than the sum of the first two stakes to have a positive expectation on the sequence of three bets.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

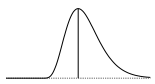
22

Laplace's method

The idea behind the Laplace approximation is very simple. We assume that an unnormalized probability density $P^*(x)$, whose normalizing constant

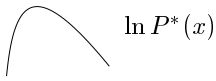
$$Z_P \equiv \int P^*(x)\,\mathrm{d}x \tag{22.1}$$

is of interest, has a peak at a point x_0.



We Taylor-expand the logarithm of $P^*(x)$ around this peak:

$$\ln P^*(x) \simeq \ln P^*(x_0) - \frac{c}{2}(x - x_0)^2 + \ldots, \tag{22.2}$$

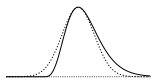


where

$$c = -\left.\frac{\partial^2}{\partial x^2} \ln P^*(x)\right|_{x=x_0}. \tag{22.3}$$

We then approximate $P^*(x)$ by an unnormalized Gaussian,

$$Q^*(x) \equiv P^*(x_0) \exp\left[-\frac{c}{2}(x - x_0)^2\right], \tag{22.4}$$



and we approximate the normalizing constant by that of this Gaussian,

$$Z_Q = P^*(x_0)\sqrt{\frac{2\pi}{c}}. \tag{22.5}$$

The generalization of this integral to a density $P^*(\mathbf{x})$ over a k-dimensional space $\mathbf{x}$ is straightforward. If the matrix of second derivatives of $-\ln P^*(\mathbf{x})$ at the maximum $\mathbf{x}_0$ is $\mathbf{A}$, defined by:

$$A_{ij} = -\left.\frac{\partial^2}{\partial x_i \partial x_j} \ln P^*(\mathbf{x})\right|_{\mathbf{x}=\mathbf{x}_0}, \tag{22.6}$$

then the normalizing constant can be approximated by:

$$Z_P \simeq Z_Q = P^*(x_0)\frac{1}{\sqrt{\det \frac{1}{2\pi}\mathbf{A}}} = P^*(x_0)\sqrt{\frac{(2\pi)^k}{\det \mathbf{A}}}. \tag{22.7}$$

Physicists also call this widely-used approximation the *saddle-point approximation*.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

About Chapter 23

In this chapter I will use the word "sample" in the following sense: a sample from a distribution $P(\mathbf{x})$ is a single realization $\mathbf{x}$ whose probability distribution is $P(\mathbf{x})$. This contrasts with the alternative usage in statistics, where "sample" refers to a collection of realizations $\{\mathbf{x}\}$.

When we discuss transition probability matrices, I will use a right–multiplication convention: I like my matrices to act to the right, preferring

$$\mathbf{u} = \mathbf{M}\mathbf{v} \tag{22.8}$$

to

$$\mathbf{u}^{\mathsf{T}} = \mathbf{v}^{\mathsf{T}}\mathbf{M}^{\mathsf{T}}. \tag{22.9}$$

A transition probability matrix T_{ij} or $T_{i|j}$ specifies the probability, given the current state is j, of making the transition from j to i. The columns of $\mathbf{T}$ are probability vectors. If we write down a transition probability density, we use the same convention for the order of its arguments: $T(x'; x)$ is a transition probability density from x to x'. This unfortunately means that you have to get used to reading from right to left — the sequence xyz has probability $T(z; y)T(y; x)\pi(x)$.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

23

Monte Carlo methods

This chapter describes a sequence of Monte Carlo methods: *importance sampling*, *rejection sampling*, the *Metropolis method*, *Gibbs sampling* and *slice sampling*. For each method, we discuss whether the method is expected to be useful for high-dimensional problems such as arise in inference with graphical models. Along the way, the terminology of Markov chain Monte Carlo methods is presented. The chapter concludes with a discussion of advanced methods, including methods for reducing random walk behaviour.

For details of Monte Carlo methods, theorems and proofs and a full list of references, the reader is directed to Neal (1993b), Gilks *et al.* (1996), and Tanner (1996).

23.1 The problems to be solved

Monte Carlo methods are computational techniques that make use of random numbers. The aims of Monte Carlo methods are to solve one or both of the following problems.

Problem 1: to generate samples $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$ from a given probability distribution $P(\mathbf{x})$.

Problem 2: to estimate expectations of functions under this distribution, for example

$$\Phi = \langle \phi(\mathbf{x}) \rangle \equiv \int d^N\mathbf{x} \; P(\mathbf{x})\phi(\mathbf{x}). \tag{23.1}$$

The probability distribution $P(\mathbf{x})$, which we will call the *target density*, might be a distribution from statistical physics or a conditional distribution arising in data modelling — for example, the posterior probability of a model's parameters given some observed data. We will generally assume that $\mathbf{x}$ is an N-dimensional vector with real components x_n, but we will sometimes consider discrete spaces also.

Simple examples of the sort of functions $\phi(\mathbf{x})$ whose expectations we might be interested include the first and second moments of quantities that we would like to predict; for example if some quantity t depends on $\mathbf{x}$, we can find the mean and variance of t under $P(\mathbf{x})$ by finding the expectations of the functions $\phi_1(\mathbf{x}) = t(\mathbf{x})$ and $\phi_2(\mathbf{x}) = (t(\mathbf{x}))^2$,

$$\Phi_1 \equiv \mathcal{E}[\phi_1(\mathbf{x})] \;\; \text{and} \;\; \Phi_2 \equiv \mathcal{E}[\phi_2(\mathbf{x})], \tag{23.2}$$

then using

$$\bar{t} = \Phi_1 \ \text{ and } \ \text{var}(t) = \Phi_2 - \Phi_1^2. \tag{23.3}$$

It is assumed that $P(\mathbf{x})$ is sufficiently complex that we cannot evaluate these expectations by exact methods, so we are interested in Monte Carlo methods.

We will concentrate on the first problem (sampling), because if we have solved it, then we can solve the second problem by using the random samples $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$ to give the estimator

$$\hat{\Phi} \equiv \frac{1}{R} \sum_r \phi(\mathbf{x}^{(r)}). \tag{23.4}$$

Clearly if the vectors $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$ are generated from $P(\mathbf{x})$ then the expectation of $\hat{\Phi}$ is Φ. Also, as the number of samples R increases, the variance of $\hat{\Phi}$ will decrease as $\frac{\sigma^2}{R}$, where σ^2 is the variance of ϕ,

$$\sigma^2 = \int d^N\mathbf{x} \ P(\mathbf{x})(\phi(\mathbf{x}) - \Phi)^2. \tag{23.5}$$

This is one of the important properties of Monte Carlo methods.

> **The accuracy of the Monte Carlo estimate (equation (23.4)) is independent of the dimensionality of the space sampled.** To be precise, the variance of $\hat{\Phi}$ goes as $\frac{\sigma^2}{R}$. So regardless of the dimensionality of $\mathbf{x}$, it may be that as few as a dozen *independent* samples $\{\mathbf{x}^{(r)}\}$ suffice to estimate Φ satisfactorily.

We will find later, however, that high dimensionality can cause other difficulties for Monte Carlo methods. Obtaining independent samples from a given distribution $P(\mathbf{x})$ is often not easy.

Why is sampling from $P(\mathbf{x})$ hard?

We will assume that the density from which we wish to draw samples, $P(\mathbf{x})$, can be evaluated, at least to within a multiplicative constant; that is, we can evaluate a function $P^*(\mathbf{x})$ such that

$$P(\mathbf{x}) = P^*(\mathbf{x})/Z. \tag{23.6}$$

If we can evaluate $P^*(\mathbf{x})$, why can we not easily solve problem 1? Why is it in general difficult to obtain samples from $P(\mathbf{x})$? There are two difficulties. The first is that we typically do not know the normalizing constant

$$Z = \int d^N\mathbf{x} \ P^*(\mathbf{x}). \tag{23.7}$$

The second is that, even if we did know Z, the problem of drawing samples from $P(\mathbf{x})$ is still a challenging one, especially in high-dimensional spaces, because there is no obvious way to sample from P without enumerating most or all of the possible states. There are only a few high-dimensional densities from which it is easy to draw samples, for example the Gaussian distribution.

> A sample from a univariate Gaussian can be generated by computing $\cos(2\pi u_1)\sqrt{2\ln(1/u_2)}$, where u_1 and u_2 are uniformly distributed in $(0, 1)$.

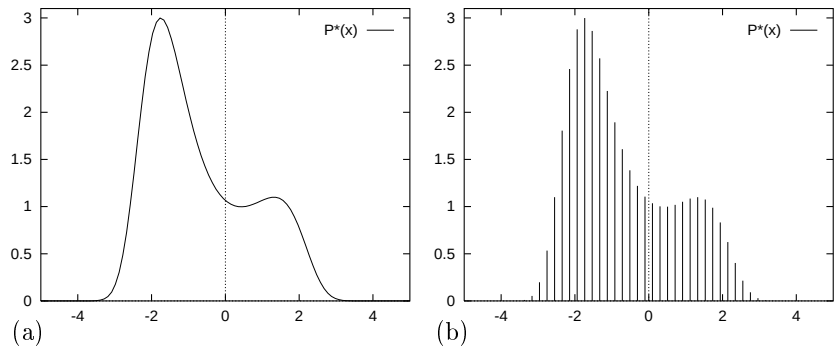


Figure 23.1. (a) The function $P^*(x) = \exp\left[0.4(x-0.4)^2 - 0.08x^4\right]$. How to draw samples from this density? (b) The function $P^*(x)$ evaluated at a discrete set of uniformly spaced points $\{x_i\}$. How to draw samples from this discrete distribution?

Let us start from a simple one-dimensional example. Imagine that we wish to draw samples from the density $P(x) = P^*(x)/Z$ where

$$P^*(x) = \exp\left[0.4(x-0.4)^2 - 0.08x^4\right], \quad x \in (-\infty, \infty). \tag{23.8}$$

We can plot this function (figure 23.1a). But that does not mean we can draw samples from it. To start with, we don't know the normalizing constant Z. To give ourselves a simpler problem, we could discretize the variable x and ask for samples from the discrete probability distribution over a finite set of uniformly spaced points $\{x_i\}$ (figure 23.1b). How could we solve this problem? If we evaluate $p_i^* = P^*(x_i)$ at each point x_i, we can compute

$$Z = \sum_i p_i^* \tag{23.9}$$

and

$$p_i = p_i^*/Z \tag{23.10}$$

and we can then sample from the probability distribution $\{p_i\}$ using various methods based on a source of random bits (see chapter 6). But what is the cost of this procedure, and how does it scale with the dimensionality of the space, N? Let us concentrate on the initial cost of evaluating Z. To compute Z (equation (23.9)) we have to visit every point in the space. In figure 23.1b there are 50 uniformly spaced points in one dimension. If our system had N dimensions, $N = 1000$ say, then the corresponding number of points would be 50^{1000}, an unimaginable number of evaluations of P^*. Even if each component x_n took only two discrete values, the number of evaluations of P^* would be 2^{1000}, a number that is still horribly huge. If every electron in the universe (there's about 2^{266} of them) were a 1000 Gigahertz computer that could evaluate P^* for 2^{40} states every second, and if we ran them for a time equal to the age of the universe (2^{58} seconds), those 2^{266} computers would still only visit 2^{364} states. We'd have to wait for more than $2^{636} \simeq 10^{190}$ universe ages to elapse before all 2^{1000} states had been visited.

Systems with 2^{1000} states are two a penny. One example is a collection of 1000 spins such as a 30×30 fragment of an Ising model whose probability distribution is proportional to

$$P^*(\mathbf{x}) = \exp\left[-\beta E(\mathbf{x})\right] \tag{23.11}$$

Translation for American readers: 'such systems are a dime a dozen'; incidentally, this equivalence (10c=6p) shows that the correct exchange rate is *£1.00 = $1.67*. What's more, in pre-decimal currency, 10c = 6d gives $4 to the pound, which was the (fixed) exchange rate under the Bretton Woods system that was introduced after World War Two, until the devaluations of the 1960s.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

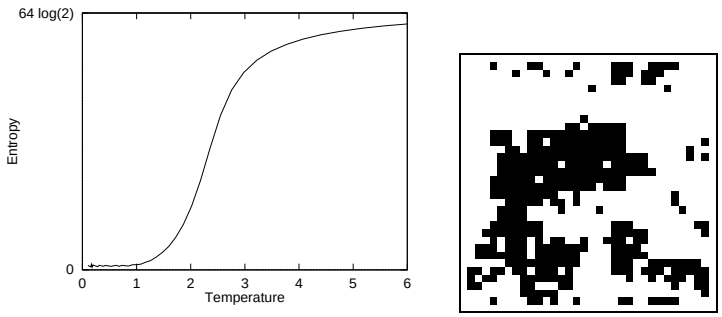


Figure 23.2. (a) Entropy of a 64-spin Ising model as a function of temperature. (b) One state of a 1024-spin Ising model.

where $x_n \in \{\pm 1\}$ and

$$E(\mathbf{x}) = -\left[\frac{1}{2}\sum_{m,n} J_{mn}x_m x_n + \sum_n H_n x_n\right]. \tag{23.12}$$

The energy function $E(\mathbf{x})$ is readily evaluated for any $\mathbf{x}$. But if we wish to evaluate this function at *all* states $\mathbf{x}$, the computer time required would be 2^{1000} function evaluations.

The Ising model is a simple model which has been around for a long time, but the task of generating samples from the distribution $P(\mathbf{x}) = P^*(\mathbf{x})/Z$ is still an active research area as evidenced by the work of Propp and Wilson (1996).

Uniform sampling

Having agreed that we cannot visit every location $\mathbf{x}$ in the state space, we might consider trying to solve the second problem (estimating the expectation of a function $\phi(\mathbf{x})$) by drawing random samples $\{\mathbf{x}^{(r)}\}_{r=1}^R$ *uniformly* from the state space and evaluating $P^*(\mathbf{x})$ at those points. Then we could introduce a normalizing constant Z_R, defined by

$$Z_R = \sum_{r=1}^R P^*(\mathbf{x}^{(r)}), \tag{23.13}$$

and estimate $\Phi = \int d^N\mathbf{x}\ \phi(\mathbf{x})P(\mathbf{x})$ by

$$\hat{\Phi} = \sum_{r=1}^R \phi(\mathbf{x}^{(r)})\frac{P^*(\mathbf{x}^{(r)})}{Z_R}. \tag{23.14}$$

Is anything wrong with this strategy? Well, it depends on the functions $\phi(\mathbf{x})$ and $P^*(\mathbf{x})$. Let us assume that $\phi(\mathbf{x})$ is a benign, smoothly varying function and concentrate on the nature of $P^*(\mathbf{x})$. As we learnt in chapter 4, a high-dimensional distribution is often concentrated in a small region of the state space known as its typical set T, whose volume is given by $|T| \simeq 2^{H(\mathbf{X})}$, where $H(\mathbf{X})$ is the entropy of the probability distribution $P(\mathbf{x})$,

$$H(\mathbf{X}) = \sum_{\mathbf{x}} P(\mathbf{x}) \log_2 \frac{1}{P(\mathbf{x})}. \tag{23.15}$$

If almost all the probability mass is located in the typical set and $\phi(\mathbf{x})$ is a benign function, the value of $\Phi = \int d^N\mathbf{x}\, \phi(\mathbf{x})P(\mathbf{x})$ will be principally determined by the values that $\phi(\mathbf{x})$ takes on in the typical set. So uniform sampling will only stand a chance of giving a good estimate of Φ if we make the number of samples R sufficiently large that we are likely to hit the typical set a number of times. So, how many samples are required? Let us take the case of the Ising model again. (Perhaps this is not a perfect example, since it doesn't necessarily have a typical set, as defined in chapter 4.) The total size of the state space is 2^N states, and the typical set has size 2^H. So each sample has a chance of $2^H/2^N$ of falling in the typical set. The number of samples required to hit the typical set once is thus of order

$$R_{\min} \simeq 2^{N-H}. \tag{23.16}$$

So, what is H? At high temperatures, the probability distribution of an Ising model tends to a uniform distribution and the entropy tends to $H_{\max} = N$ bits, so $R_{\min}$ is of order 1. Under these conditions, uniform sampling may well be a satisfactory technique for estimating Φ. But high temperatures are not of great interest. Considerably more interesting are intermediate temperatures such as the critical temperature at which the Ising model melts from an ordered phase to a disordered phase. The critical temperature of an infinite Ising model, at which it 'melts', is $\theta_c = 2.27$. At this temperature the entropy of an Ising model is roughly $N/2$ bits (figure 23.2). For example, when the entropy of the 64-spin model is $32\log(2)$, the probability mass is concentrated in a typical set of size 2^{32} states; this set is a fraction roughly $1/2^{32}$ of the total size of the state space. For this probability distribution the number of samples required simply to hit the typical set once is of order

$$R_{\min} \simeq 2^{N-N/2} = 2^{N/2}, \tag{23.17}$$

which for $N = 1000$ is about 10^{150}. This is roughly the square of the number of particles in the universe. Thus uniform sampling is utterly useless for the study of Ising models of modest size. And in most high-dimensional problems, if the distribution $P(\mathbf{x})$ is not actually uniform, uniform sampling is unlikely to be useful.

Overview

Having established that drawing samples from a high-dimensional distribution $P(\mathbf{x}) = P^*(\mathbf{x})/Z$ is difficult even if $P^*(\mathbf{x})$ is easy to evaluate, we will now study a sequence of more sophisticated Monte Carlo methods:

- *importance sampling*,
- *rejection sampling*,
- the *Metropolis method*,
- *Gibbs sampling*, and
- *slice sampling*.

23.2 Importance sampling

Importance sampling is not a method for generating samples from $P(\mathbf{x})$ (problem 1); it is just a method for estimating the expectation of a function $\phi(\mathbf{x})$ (problem 2). It can be viewed as a generalization of the uniform sampling method.

For illustrative purposes, let us imagine that the target distribution is a one-dimensional density $P(x)$. Let us assume that we are able to evaluate this density, at least to within a multiplicative constant; thus we can evaluate a function $P^*(x)$ such that

$$P(x) = P^*(x)/Z. \tag{23.18}$$

But $P(x)$ is too complicated a function for us to be able to sample from it directly. We now assume that we have a simpler density $Q(x)$ from which we can generate samples and which we can evaluate to within a multiplicative constant (that is, we can evaluate $Q^*(x)$, where $Q(x) = Q^*(x)/Z_Q$). An example of the functions P^*, Q^* and ϕ is shown in figure 23.3. We call Q the *sampler density*.

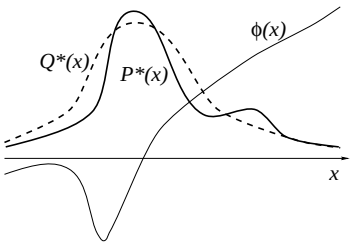


Figure 23.3. Functions involved in importance sampling. We wish to estimate the expectation of $\phi(x)$ under $P(x) \propto P^*(x)$. We can generate samples from the simpler distribution $Q(x) \propto Q^*(x)$. We can evaluate Q^* and P^* at any point.

In importance sampling, we generate R samples $\{x^{(r)}\}_{r=1}^{R}$ from $Q(x)$. If these points were samples from $P(x)$ then we could estimate Φ by equation (23.4). But when we generate samples from Q, values of x where $Q(x)$ is greater than $P(x)$ will be over-represented in this estimator, and points where $Q(x)$ is less than $P(x)$ will be under-represented. To take into account the fact that we have sampled from the wrong distribution, we introduce 'weights'

$$w_r \equiv \frac{P^*(x^{(r)})}{Q^*(x^{(r)})} \tag{23.19}$$

which we use to adjust the 'importance' of each point in our estimator thus:

$$\hat{\Phi} \equiv \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r}. \tag{23.20}$$

Exercise 23.1:^{B2} Prove that, if $Q(x)$ is non-zero for all x where $P(x)$ is non-zero, the estimator $\hat{\Phi}$ converges to Φ, the mean value of $\phi(x)$, as R increases. What is the variance of this estimator, asymptotically? Hint: consider the statistics of the numerator and the denominator separately. Is the estimator $\hat{\Phi}$ an unbiased estimator for small R?

A practical difficulty with importance sampling is that it is hard to estimate how reliable the estimator $\hat{\Phi}$ is. The variance of the estimator is unknown beforehand, because it depends on an integral over x of a function involving $P^*(x)$. And the variance of $\hat{\Phi}$ is hard to estimate, because the empirical variances of the quantities w_r and $w_r\phi(x^{(r)})$ are not necessarily a good guide to the true variances of the numerator and denominator in equation (23.20). If the proposal density $Q(x)$ is small in a region where $|\phi(x)P^*(x)|$ is large then it is quite possible, even after many points $x^{(r)}$ have been generated, that none of them will have fallen in that region. In this case the estimate of Φ would be drastically wrong, and there would be no indication in the empirical variance that the true variance of the estimator $\hat{\Phi}$ is large.

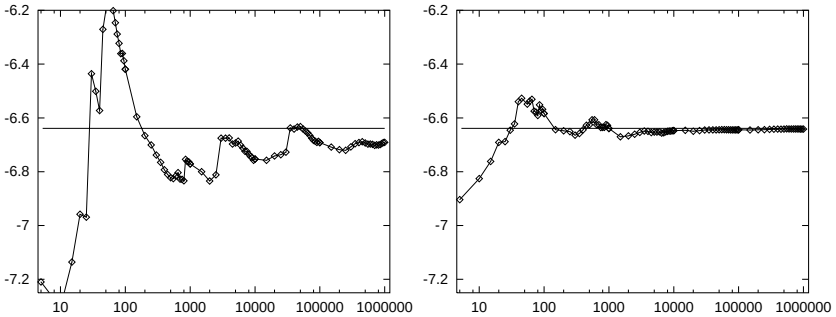
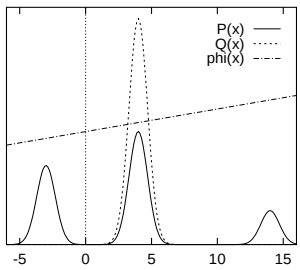


Figure 23.4. Importance sampling in action: (a) using a Gaussian sampler density; (b) using a Cauchy sampler density. Vertical axis shows the estimate $\hat{\Phi}$. The horizontal line indicates the true value of Φ. Horizontal axis shows number of samples on a log scale.

Cautionary illustration of importance sampling

In a toy problem related to the modelling of amino acid probability distributions with a one-dimensional variable x I evaluated a quantity of interest using importance sampling. The results using a Gaussian sampler and a Cauchy sampler are shown in figure 23.4. The horizontal axis shows the number of samples on a log scale. In the case of the Gaussian sampler, after about 500 samples had been evaluated one might be tempted to call a halt; but evidently there are infrequent samples that make a huge contribution to $\hat{\Phi}$, and the value of the estimate at 500 samples is wrong. Even after a million samples have been taken, the estimate has still not settled down close to the true value. In contrast, the Cauchy sampler does not suffer from glitches and converges (on the scale shown here) after about 5000 samples.

This example illustrates the fact that an importance sampler should have **heavy tails**.



Exercise 23.2:^{A2} Consider the situation where $P^*(x)$ is multimodal, consisting of several widely separated peaks. (Probability distributions like this arise frequently in statistical data modelling.) Discuss whether it is a wise strategy to do importance sampling using a sampler $Q(x)$ that is a unimodal distribution fitted to one of these peaks. Assume that the function $\phi(x)$ whose mean Φ is to be estimated is a smoothly varying function of x such as $mx + c$. Describe the typical evolution of the estimator $\hat{\Phi}$ as a function of the number of samples R.

Importance sampling in many dimensions

We have already observed that care is needed in one-dimensional importance sampling problems. Is importance sampling a useful technique in spaces of higher dimensionality, say $N = 1000$?

Consider a simple case-study where the target density $P(\mathbf{x})$ is a uniform distribution inside a sphere,

$$P^*(\mathbf{x}) = \begin{cases} 1 & 0 \leq \rho(\mathbf{x}) \leq R_P \\ 0 & \rho(\mathbf{x}) > R_P \end{cases} , \tag{23.21}$$

where $\rho(\mathbf{x}) \equiv (\sum_i x_i^2)^{1/2}$, and the proposal density is a Gaussian centred on

the origin,

$$Q(\mathbf{x}) = \prod_i \text{Normal}(x_i; 0, \sigma^2). \tag{23.22}$$

An importance sampling method will be in trouble if the estimator $\hat{\Phi}$ is dominated by a few large weights w_r. What will be the typical range of values of the weights w_r? We know from our discussions of typical sequences in part 1 — see exercise 7.1 (p. 138), for example — that if ρ is the distance from the origin of a sample from Q, the quantity ρ^2 has a roughly Gaussian distribution with mean and standard deviation:

$$\rho^2 \sim N\sigma^2 \pm \sqrt{2N}\sigma^2. \tag{23.23}$$

Thus almost all samples from Q lie in a 'typical set' with distance from the origin very close to $\sqrt{N}\sigma$. Let us assume that σ is chosen such that the typical set of Q lies inside the sphere of radius R_P. [If it does not, then the law of large numbers implies that almost all the samples generated from Q will fall outside R_P and will have weight zero.] Then we know that most samples from Q will have a value of Q that lies in the range

$$\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2} \pm \frac{\sqrt{2N}}{2}\right). \tag{23.24}$$

Thus the weights $w_r = P^*/Q$ will typically have values in the range

$$(2\pi\sigma^2)^{N/2} \exp\left(\frac{N}{2} \pm \frac{\sqrt{2N}}{2}\right). \tag{23.25}$$

So if we draw a hundred samples, what will the typical range of weights be? We can roughly estimate the ratio of the largest weight to the median weight by doubling the standard deviation in equation (23.25). The largest weight and the median weight will typically be in the ratio:

$$\frac{w_r^{\max}}{w_r^{\text{med}}} = \exp\left(\sqrt{2N}\right). \tag{23.26}$$

In $N = 1000$ dimensions therefore, the largest weight after one hundred samples is likely to be roughly 10^{19} times greater than the median weight. Thus an importance sampling estimate for a high-dimensional problem will very likely be utterly dominated by a few samples with huge weights.

In conclusion, importance sampling in high dimensions often suffers from two difficulties. First, we clearly need to obtain samples that lie in the typical set of P, and this may take a long time unless Q is a good approximation to P. Second, even if we obtain samples in the typical set, the weights associated with those samples are likely to vary by large factors, because the probabilities of points in a typical set, although similar to each other, still differ by factors of order $\exp(\sqrt{N})$.

23.3 Rejection sampling

We assume again a one-dimensional density $P(x) = P^*(x)/Z$ that is too complicated a function for us to be able to sample from it directly. We assume that we have a simpler *proposal density* $Q(x)$ which we can evaluate (within a

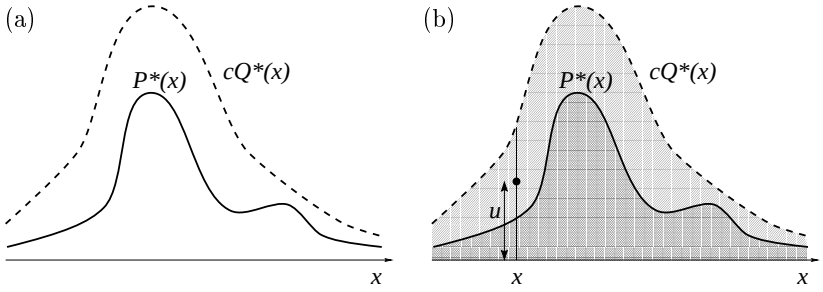


Figure 23.5. Rejection sampling. (a) The functions involved in rejection sampling. We desire samples from $P(x) \propto P^*(x)$. We are able to draw samples from $Q(x) \propto Q^*(x)$, and we know a value c such that $cQ^*(x) > P^*(x)$ for all x. (b) A point (x, u) is generated at random in the lightly shaded area under the curve $cQ^*(x)$. If this point also lies below $P^*(x)$ then it is accepted.

multiplicative factor Z_Q, as before), and from which we can generate samples. We further assume that we know the value of a constant c such that

$$cQ^*(x) > P^*(x), \quad \text{for all } x. \tag{23.27}$$

A schematic picture of the two functions is shown in figure 23.5a.

We generate two random numbers. The first, x, is generated from the proposal density $Q(x)$. We then evaluate $cQ^*(x)$ and generate a uniformly distributed random variable u from the interval $[0, cQ^*(x)]$. These two random numbers can be viewed as selecting a point in the two-dimensional plane as shown in figure 23.5b.

We now evaluate $P^*(x)$ and accept or reject the sample x by comparing the value of u with the value of $P^*(x)$. If $u > P^*(x)$ then x is rejected; otherwise it is accepted, which means that we add x to our set of samples $\{x^{(r)}\}$. The value of u is discarded.

Why does this procedure generate samples from $P(x)$? The proposed point (x, u) comes with uniform probability from the lightly shaded area underneath the curve $cQ^*(x)$ as shown in figure 23.5b. The rejection rule rejects all the points that lie above the curve $P^*(x)$. So the points (x, u) that are accepted are uniformly distributed in the heavily shaded area under $P^*(x)$. This implies that the probability density of the x-coordinates of the accepted points must be proportional to $P^*(x)$, so the samples must be independent samples from $P(x)$.

Rejection sampling will work best if Q is a good approximation to P. If Q is very different from P then, for cQ to exceed P everywhere, c will necessarily have to be large and the frequency of rejection will be large.

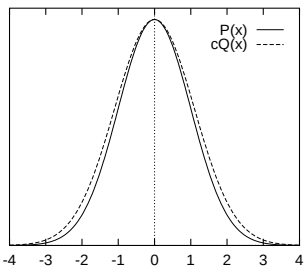


Figure 23.6. A Gaussian $P(x)$ and a slightly broader Gaussian $Q(x)$ scaled up by a factor c such that $cQ(x) \geq P(x)$.

Rejection sampling in many dimensions

In a high-dimensional problem it is very likely that the requirement that cQ^* be an upper bound for P^* will force c to be so huge that acceptances will be very rare indeed. Finding such a value of c may be difficult too, since in many problems we know neither where the modes of P^* are located nor how high they are.

As a case study, consider a pair of N-dimensional Gaussian distributions with mean zero (figure 23.6). Imagine generating samples from one with standard deviation σ_Q and using rejection sampling to obtain samples from the other whose standard deviation is σ_P. Let us assume that these two standard deviations are close in value — say, σ_Q is one percent larger than σ_P. [σ_Q

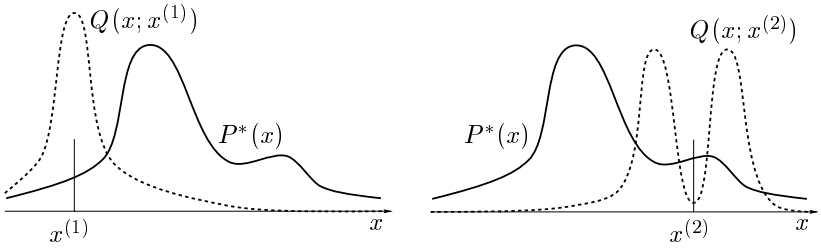


Figure 23.7. Metropolis method in one dimension. The proposal distribution $Q(x'; x)$ is here shown as having a shape that changes as x changes, though this is not typical of the proposal densities used in practice.

must be larger than σ_P because if this is not the case, there is no c such that cQ exceeds P for all $\mathbf{x}$.] So, what value of c is required if the dimensionality is $N = 1000$? The density of $Q(\mathbf{x})$ at the origin is $1/(2\pi\sigma_Q^2)^{N/2}$, so for cQ to exceed P we need to set

$$c = \frac{(2\pi\sigma_Q^2)^{N/2}}{(2\pi\sigma_P^2)^{N/2}} = \exp\left(N \log \frac{\sigma_Q}{\sigma_P}\right). \tag{23.28}$$

With $N = 1000$ and $\frac{\sigma_Q}{\sigma_P} = 1.01$, we find $c = \exp(10) \simeq 20,000$. What will the acceptance rate be for this value of c? The answer is immediate: since the acceptance rate is the ratio of the volume under the curve $P(\mathbf{x})$ to the volume under $cQ(\mathbf{x})$, the fact that P and Q are both normalized here implies that the acceptance rate will be $1/c$. For our case study, this is $1/20,000$. In general, c grows exponentially with the dimensionality N, so the acceptance rate is expected to be exponentially small in N.

Rejection sampling, therefore, whilst a useful method for one-dimensional problems, is not expected to be a practical technique for generating samples from high-dimensional distributions $P(\mathbf{x})$.

23.4 The Metropolis-Hastings method

Importance sampling and rejection sampling only work well if the proposal density $Q(x)$ is similar to $P(x)$. In large and complex problems it is difficult to create a single density $Q(x)$ that has this property.

The Metropolis algorithm instead makes use of a proposal density Q *which depends on the current state* $x^{(t)}$. The density $Q(x'; x^{(t)})$ might be a simple distribution such as a Gaussian centred on the current $x^{(t)}$. The proposal density $Q(x'; x)$ can be *any* fixed density from which we can draw samples. In contrast to importance sampling and rejection sampling, it is not necessary that $Q(x'; x^{(t)})$ look at all similar to $P(x)$ in order for the algorithm to be practically useful. An example of a proposal density is shown in figure 23.7; this figure shows the density $Q(x'; x^{(t)})$ for two different states $x^{(1)}$ and $x^{(2)}$.

As before, we assume that we can evaluate $P^*(x)$ for any x. A tentative new state x' is generated from the proposal density $Q(x'; x^{(t)})$. To decide whether to accept the new state, we compute the quantity

$$a = \frac{P^*(x')}{P^*(x^{(t)})} \frac{Q(x^{(t)}; x')}{Q(x'; x^{(t)})}. \tag{23.29}$$

$$\begin{array}{l} \text{If } a \geq 1 \text{ then the new state is accepted.} \\ \text{Otherwise, the new state is accepted with probability } a. \end{array} \tag{23.30}$$

If the step is accepted, we set $x^{(t+1)} = x'$. If the step is rejected, then we set $x^{(t+1)} = x^{(t)}$. Note the difference from rejection sampling: in rejection sampling, rejected points are discarded and have no influence on the list of samples $\{x^{(r)}\}$ that we collected. Here, a rejection causes the current state to be written onto the list of points another time.

Notation: I have used the superscript $r = 1, \ldots, R$ to label points that are *independent* samples from a distribution, and the superscript $t = 1, \ldots, T$ to label the sequence of states in a Markov chain. It is important to note that a Metropolis simulation of T iterations does not produce T independent samples from the target distribution P. The samples are correlated.

To compute the acceptance probability (23.29) we need to be able to compute the probability ratios $P(x')/P(x^{(t)})$ and $Q(x^{(t)}; x')/Q(x'; x^{(t)})$. If the proposal density is a simple symmetrical density such as a Gaussian centred on the current point, then the latter factor is unity, and the Metropolis-Hastings method simply involves comparing the value of the target density at the two points. This special case is sometimes called the Metropolis method. However, with apologies to Hastings, I will call the general Metropolis-Hastings algorithm for asymmetric Q, the Metropolis algorithm, since I believe important ideas deserve short names.

Convergence of the Metropolis method to the target density

It can be shown that for any positive Q (that is, any Q such that $Q(x'; x) > 0$ for all x, x'), as $t \to \infty$, the probability distribution of $x^{(t)}$ tends to $P(x) = P^*(x)/Z$. [This statement should not be seen as implying that Q *has* to assign positive probability to every point x' — we will discuss examples later where $Q(x'; x) = 0$ for some x, x'; notice also that we have said nothing about how rapidly the convergence to $P(x)$ takes place.]

The Metropolis method is an example of a *Markov chain Monte Carlo* method (abbreviated MCMC). In contrast to rejection sampling, where the accepted points $\{x^{(r)}\}$ are independent samples from the desired distribution, Markov chain Monte Carlo methods involve a Markov process in which a sequence of states $\{x^{(t)}\}$ is generated, each sample $x^{(t)}$ having a probability distribution that depends on the previous value, $x^{(t-1)}$. Since successive samples are correlated with each other, the Markov chain may have to be run for a considerable time in order to generate samples that are effectively independent samples from P.

Just as it was difficult to estimate the variance of an importance sampling estimator, so it is difficult to assess whether a Markov chain Monte Carlo method has 'converged', and to quantify how long one has to wait to obtain samples that are effectively independent samples from P.

Demonstration of the Metropolis method

The Metropolis method is widely used for high-dimensional problems. Many implementations of the Metropolis method employ a proposal distribution with a length scale ϵ that is short relative to the length scale L of the probable region (figure 23.8). A reason for choosing a small length scale is that for most high-dimensional problems, a large random step from a typical point (that is, a sample from $P(\mathbf{x})$) is very likely to end in a state that has very low probability; such steps are unlikely to be accepted. If ϵ is large, movement around the state

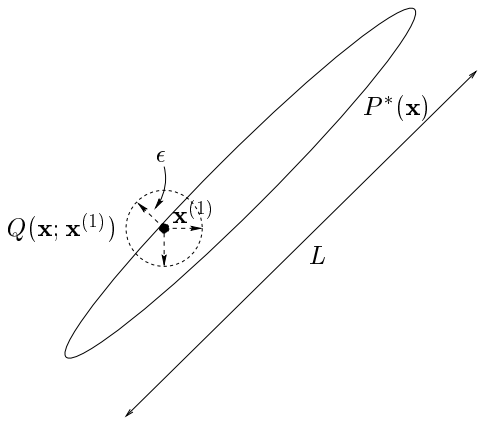


Figure 23.8. Metropolis method in two dimensions, showing a traditional proposal density that has a sufficiently small step size ϵ that the acceptance frequency will be about 0.5.

space will only occur when such a transition is actually accepted, or when a large random step chances to land in another probable state. So the rate of progress will be slow, unless small steps are used.

The disadvantage of small steps, on the other hand, is that the Metropolis method will explore the probability distribution by a *random walk*, and a random walk takes a long time to get anywhere, especially if the walk is made of small steps.

Exercise 23.3:[A1] Consider a one-dimensional random walk, on each step of which the state moves randomly to the left or to the right with equal probability. Show that after T steps of size ϵ, the state is only likely to have moved a distance about $\sqrt{T}\epsilon$. (Compute the root mean square distance travelled.)

Recall that the first aim of Monte Carlo sampling is to generate a number of *independent* samples from the given distribution (a dozen, say). If the largest length scale of the state space is L, then we have to simulate a random-walk Metropolis method for a time $T \simeq (L/\epsilon)^2$ before we can expect to get a sample that is roughly independent of the initial condition — and that's assuming that every step is accepted: if only a fraction f of the steps are accepted on average, then this time is increased by a factor $1/f$.

Rule of thumb: lower bound on number of iterations of a Metropolis method. If the largest length scale of the space of probable states is L, a Metropolis method whose proposal distribution generates a random walk with step size ϵ must be run for at least $T \simeq (L/\epsilon)^2$ iterations to obtain an independent sample.

This rule of thumb only gives a lower bound; the situation may be much worse, if, for example, the probability distribution consists of several islands of high probability separated by regions of low probability.

To illustrate how slowly a random walk explores a state space, figure 23.9 shows a simulation of a Metropolis algorithm for generating samples from the distribution:

$$P(x) = \begin{cases} \frac{1}{21} & x \in \{0, 1, 2, \ldots, 20\} \\ 0 & \text{otherwise} \end{cases} . \tag{23.31}$$

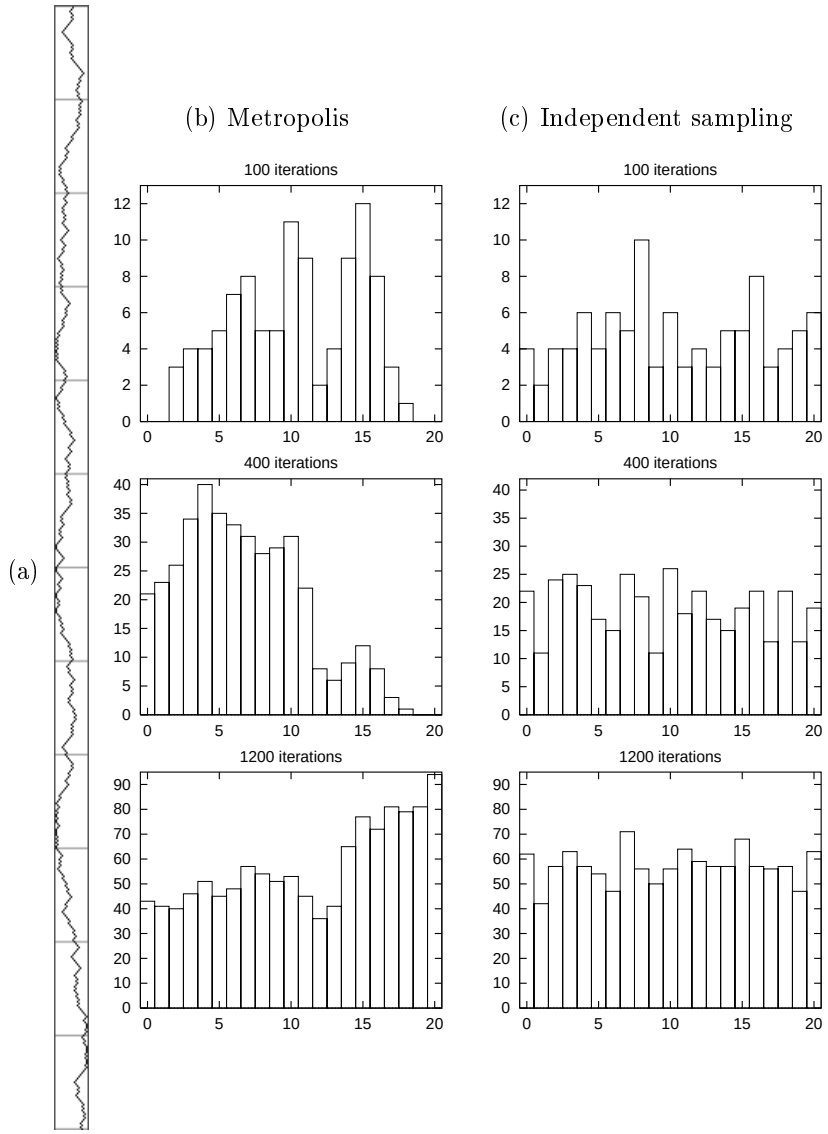


Figure 23.9. Metropolis method for a toy problem. (a) The state sequence for $t = 1, \ldots, 600$. Horizontal direction = states from 0 to 20; vertical direction = time from 1 to 600; the cross bars mark time intervals of duration 50. (b) Histogram of occupancy of the states after 100, 400 and 1200 iterations. (c) For comparison, histograms resulting when successive points are drawn *independently* from the target distribution.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

The proposal distribution is

$$Q(x'; x) = \begin{cases} \frac{1}{2} & x' = x \pm 1 \\ 0 & \text{otherwise} \end{cases} . \tag{23.32}$$

Because the target distribution $P(x)$ is uniform, rejections will occur only when the proposal takes the state to $x' = -1$ or $x' = 21$.

The simulation was started in the state $x_0 = 10$ and its evolution is shown in figure 23.9a. How long does it take to reach one of the end states $x = 0$ and $x = 20$? Since the distance is 10 steps the rule of thumb above predicts that it will typically take a time $T \simeq 100$ iterations to reach an end state. This is confirmed in the present example. The first step into an end state occurs on the 178th iteration. How long does it take to visit *both* end states? The rule of thumb predicts about 400 iterations are required to traverse the whole state space; and indeed the first encounter with the other end state takes place on the 540th iteration. Thus effectively independent samples are only generated by simulating for about four hundred iterations per independent sample.

This simple example shows that it is important to try to abolish random walk behaviour in Monte Carlo methods. A systematic exploration of the toy state space $\{0, 1, 2, \ldots, 20\}$ could get around it, using the same step sizes, in about twenty steps instead of four hundred.

Metropolis method in high dimensions

The rule of thumb that we discussed above, giving a lower bound on the number of iterations of a random walk Metropolis method, also applies to higher dimensional problems. Consider the simple case of a target distribution that is an N-dimensional Gaussian, and a proposal distribution that is a spherical Gaussian of standard deviation in each direction equal to ϵ. Without loss of generality, we can assume that the target distribution is a separable distribution aligned with the axes $\{x_n\}$, and that it has standard deviations $\{\sigma_n\}$ in the different directions n. Let $\sigma^{\max}$ and $\sigma^{\min}$ be the largest and smallest of these standard deviations. Let us assume that ϵ is adjusted such that the acceptance probability is close to 1. Under this assumption, each variable x_n evolves independently of all the others, executing a random walk with step sizes about ϵ. The time taken to generate effectively independent samples from the target distribution will be controlled by the largest lengthscale $\sigma^{\max}$. Just as in the previous section, where we needed at least $T \simeq (L/\epsilon)^2$ iterations to obtain an independent sample, here we need $T \simeq (\sigma^{\max}/\epsilon)^2$.

Now how big can ϵ be? The bigger it is, the smaller this number T becomes, but if ϵ is too big — bigger than $\sigma^{\min}$ — then the acceptance rate will fall sharply. It seems plausible that the optimal ϵ must be similar to $\sigma^{\min}$. Strictly, this may not be true; in special cases where the second smallest σ_n is significantly greater than $\sigma^{\min}$, the optimal ϵ may be closer to that second smallest σ_n. But our rough conclusion is this: where simple spherical proposal distributions are used, we will need at least $T \simeq (\sigma^{\max}/\sigma^{\min})^2$ iterations to obtain an independent sample, where $\sigma^{\max}$ and $\sigma^{\min}$ are the longest and shortest lengthscales of the target distribution.

This is good news and bad news. It is good news because, unlike the cases of rejection sampling and importance sampling, there is no catastrophic dependence on the dimensionality N. We can get answers using a computer

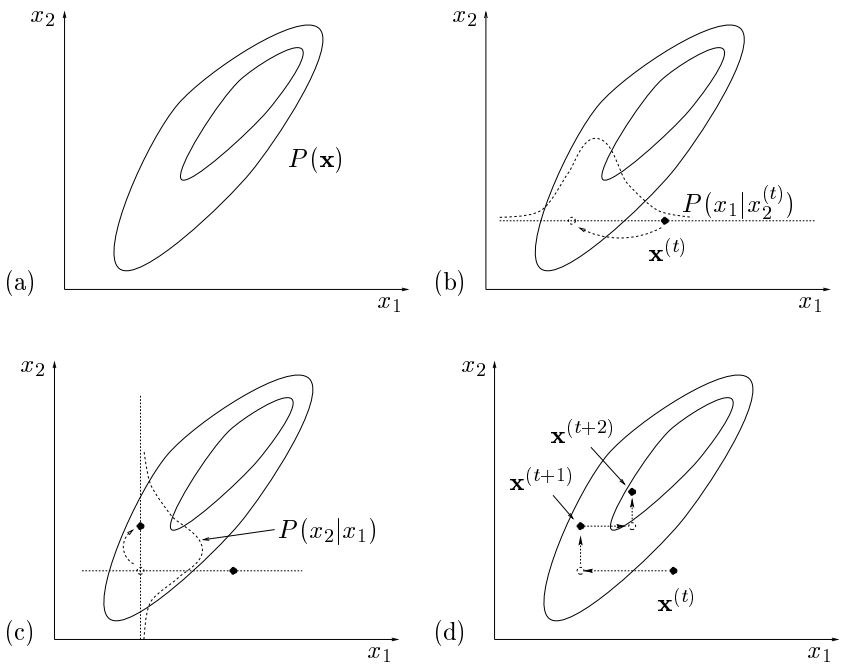


Figure 23.10. Gibbs sampling. (a) The joint density $P(\mathbf{x})$ from which samples are required. (b) Starting from a state $\mathbf{x}^{(t)}$, x_1 is sampled from the conditional density $P(x_1|x_2^{(t)})$. (c) A sample is then made from the conditional density $P(x_2|x_1)$. (d) A couple of iterations of Gibbs sampling.

for a time shorter than the age of the universe. But it is bad news all the same, because this quadratic dependence on the lengthscale ratio may still force us to make very lengthy simulations.

Fortunately, there are methods for suppressing random walks in Monte Carlo simulations, which we will discuss later.

23.5 Gibbs sampling

We introduced importance sampling, rejection sampling and the Metropolis method using one-dimensional examples. Gibbs sampling, also known as the *heat bath method*, is a method for sampling from distributions over at least two dimensions. It can be viewed as a Metropolis method in which the proposal distribution Q is defined in terms of the *conditional* distributions of the joint distribution $P(\mathbf{x})$. It is assumed that, whilst $P(\mathbf{x})$ is too complex to draw samples from directly, its conditional distributions $P(x_i|\{x_j\}_{j\neq i})$ are tractable to work with. For many graphical models (but not all) these one-dimensional conditional distributions are straightforward to sample from. For example, if a Gaussian distribution for some variables $\mathbf{d}$ has an unknown mean $\mathbf{m}$, and the prior distribution of $\mathbf{m}$ is Gaussian, then the conditional distribution of $\mathbf{m}$ given $\mathbf{d}$ is also Gaussian. Conditional distributions that are not of standard form may still be sampled from by *adaptive rejection sampling* if the conditional distribution satisfies certain convexity properties (Gilks and Wild, 1992).

Gibbs sampling is illustrated for a case with two variables $(x_1, x_2) = \mathbf{x}$ in figure 23.10. On each iteration, we start from the current state $\mathbf{x}^{(t)}$, and x_1 is

sampled from the conditional density $P(x_1|x_2)$, with x_2 fixed to $x_2^{(t)}$. A sample x_2 is then made from the conditional density $P(x_2|x_1)$, using the new value of x_1. This brings us to the new state $\mathbf{x}^{(t+1)}$, and completes the iteration.

In the general case of a system with K variables, a single iteration involves sampling one parameter at a time:

$$x_1^{(t+1)} \sim P(x_1|x_2^{(t)}, x_3^{(t)}, \ldots, x_K^{(t)}) \quad (23.33)$$
$$x_2^{(t+1)} \sim P(x_2|x_1^{(t+1)}, x_3^{(t)}, \ldots, x_K^{(t)}) \quad (23.34)$$
$$x_3^{(t+1)} \sim P(x_3|x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_K^{(t)}), \text{ etc.} \quad (23.35)$$

Convergence of Gibbs sampling to the target density

Exercise 23.4:^{A2} Show that Gibbs sampling can be viewed as a Metropolis method with target density $P(\mathbf{x})$, and that this Metropolis method has the property that every proposal is always accepted.

Because Gibbs sampling is a Metropolis method, the probability distribution of $\mathbf{x}^{(t)}$ tends to $P(\mathbf{x})$ as $t \to \infty$, as long as $P(\mathbf{x})$ does not have pathological properties.

Exercise 23.5:^{A2} Discuss whether the syndrome decoding problem for a (7,4) Hamming code can be solved using Gibbs sampling. The syndrome decoding problem, if we are to solve it with a Monte Carlo approach, is to draw samples from the posterior distribution of the noise vector $\mathbf{n} = (n_1, \ldots, n_n, \ldots, n_N)$,

$$P(\mathbf{n}|\mathbf{f}, \mathbf{z}) = \frac{1}{Z} \prod_{n=1}^{N} f_n^{n_n} (1 - f_n)^{(1-n_n)} \delta(\mathbf{Hn} = \mathbf{z}), \quad (23.36)$$

where f_n is the normalized likelihood for the nth transmitted bit and $\mathbf{z}$ is the observed syndrome. The factor $\delta(\mathbf{Hn} = \mathbf{z})$ is 1 if the hypothesis $\mathbf{n}$ has the correct syndrome $\mathbf{z}$ and 0 otherwise.

What about the syndrome decoding problem for any linear code?

Gibbs sampling in high dimensions

Gibbs sampling suffers from the same defect as simple Metropolis algorithms — the state space is explored by a slow random walk, unless a fortuitous parameterization has been chosen which makes the probability distribution $P(\mathbf{x})$ separable. If, say, two variables x_1 and x_2 are strongly correlated, having marginal densities of width L and conditional densities of width ϵ, then it will take at least about $(L/\epsilon)^2$ iterations to generate an independent sample from the target density. Figure 23.14 on page 311 illustrates the slow progress made by Gibbs sampling when $L \gg \epsilon$.

However Gibbs sampling involves no adjustable parameters, so it is an attractive strategy when one wants to get a model running quickly. An excellent software package, `BUGS`, makes it easy to set up almost arbitrary probabilistic models and simulate them by Gibbs sampling (Thomas *et al.*, 1992).[1]

[1] `http://www.mrc-bsu.cam.ac.uk/bugs/`

23.6 Terminology for Markov chain Monte Carlo methods

We now spend a few moments sketching the theory on which the Metropolis method and Gibbs sampling are based. We denote by $p^{(t)}(\mathbf{x})$ the probability distribution of the state of a Markov chain simulator. (To visualize this distribution, imagine running an infinite collection of identical simulators in parallel.) Our aim is to find a Markov chain such that as $t \to \infty$, $p^{(t)}(\mathbf{x})$ tends to the desired distribution $P(\mathbf{x})$.

A *Markov chain* can be specified by an *initial* probability distribution $p^{(0)}(\mathbf{x})$ and a *transition probability* $T(\mathbf{x}'; \mathbf{x})$.

The probability distribution of the state at the $(t+1)$th iteration of the Markov chain, $p^{(t+1)}(\mathbf{x})$, is given by

$$p^{(t+1)}(\mathbf{x}') = \int d^N\mathbf{x}\ T(\mathbf{x}'; \mathbf{x}) p^{(t)}(\mathbf{x}). \tag{23.37}$$

We construct a chain with the following properties:

1. The desired distribution $P(\mathbf{x})$ is the *invariant distribution* of the chain.

   A distribution $\pi(\mathbf{x})$ is an invariant distribution of $T(\mathbf{x}'; \mathbf{x})$ if

   $$\pi(\mathbf{x}') = \int d^N\mathbf{x}\ T(\mathbf{x}'; \mathbf{x}) \pi(\mathbf{x}). \tag{23.38}$$

2. The chain must also be *ergodic*, that is,

   $$p^{(t)}(\mathbf{x}) \to \pi(\mathbf{x}) \text{ as } t \to \infty, \text{ for any } p^{(0)}(\mathbf{x}). \tag{23.39}$$

   A couple of reasons why a chain might not be ergodic are:

   (a) It might contain more than one set of *irreducible* states — a set of states is irreducible if all the states in that set can be reached from each other, and no other states can be reached from them. Such a chain would have many invariant distributions; which one $p^{(t)}(\mathbf{x})$ would tend to as $t \to \infty$ would depend on the initial condition $p^{(0)}(\mathbf{x})$.

   The transition probability matrix of such a chain has more than one eigenvalue equal to 1.

   (b) The chain might have a *periodic* irreducible set, which means that, for some initial conditions, $p^{(t)}(\mathbf{x})$ doesn't tend to an invariant distribution, but instead tends to a periodic limit-cycle.

   A simple Markov chain with this property is the random walk on the N-dimensional hypercube. The chain T takes the state from one corner to a randomly chosen adjacent corner. The unique invariant distribution of this chain is the uniform distribution over all 2^N states, but the chain is not ergodic; it is periodic: if we divide the states into states odd parity and states with even parity, we notice that every odd state is surrounded by even states and vice versa. So if the initial condition at time $t = 0$ is a state with even parity, then at time $t = 1$ — and at all odd times — the state must have odd parity, and at all even times, the state will be of even parity.

   The transition probability matrix of such a chain has more than one eigenvalue with magnitude equal to 1. The random walk on the hypercube has eigenvalues equal to $+1$ and -1.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

It is often convenient to construct T by *mixing* or *concatenating* simple *base transitions* B all of which satisfy

$$P(\mathbf{x}') = \int d^N\mathbf{x}\ B(\mathbf{x}';\mathbf{x})P(\mathbf{x}), \tag{23.40}$$

for the desired density $P(\mathbf{x})$. These base transitions need not be individually ergodic.

T is a *mixture* of several base transitions $B_b(\mathbf{x}',\mathbf{x})$ if we make the transition by picking one of the base transitions at random, and allowing it to determine the transition, *i.e.*,

$$T(\mathbf{x}',\mathbf{x}) = \sum_b \pi_b B_b(\mathbf{x}',\mathbf{x}), \tag{23.41}$$

where $\{\pi_b\}$ is a probability distribution over the base transitions.

T is a concatenation of two base transitions $B_1(\mathbf{x}',\mathbf{x})$ and $B_2(\mathbf{x}',\mathbf{x})$ if we first make a transition to an intermediate state $\mathbf{x}''$ using B_1, and then make a transition from state $\mathbf{x}''$ using B_2.

$$T(\mathbf{x}',\mathbf{x}) = \sum_{\mathbf{x}''} B_2(\mathbf{x}',\mathbf{x}'')B_1(\mathbf{x}'',\mathbf{x}) \tag{23.42}$$

Many useful transition probabilities satisfy the *detailed balance* property:

$$T(\mathbf{x}_a;\mathbf{x}_b)P(\mathbf{x}_b) = T(\mathbf{x}_b;\mathbf{x}_a)P(\mathbf{x}_a), \text{ for all } \mathbf{x}_b \text{ and } \mathbf{x}_a. \tag{23.43}$$

This equation says that if we pick (by magic) a state from the target density P and make a transition under T to another state, it is just as likely that we will pick $\mathbf{x}_b$ and go from $\mathbf{x}_b$ to $\mathbf{x}_a$ as it is that we will pick $\mathbf{x}_a$ and go from $\mathbf{x}_a$ to $\mathbf{x}_b$. Markov chains that satisfy detailed balance are also called *reversible* Markov chains. The reason why the detailed balance property is of interest is that detailed balance implies invariance of the distribution $P(\mathbf{x})$ under the Markov chain T, which is a necessary condition for the key property that we want from our MCMC simulation — that the probability distribution of the chain should converge to $P(\mathbf{x})$.

Exercise 23.6:^{A2} Prove that detailed balance implies invariance of the distribution $P(\mathbf{x})$ under the Markov chain T.

Proving that detailed balance holds is often a key step when proving that a Markov chain Monte Carlo simulation will converge to the desired distribution. The Metropolis method and Gibbs sampling method both satisfy detailed balance, for example. Detailed balance is not an essential condition, however, and we will see later that irreversible Markov chains can be useful in practice, because they may have different random walk properties.

Exercise 23.7:^{B2} Show that, if we concatenate two base transitions B_1 and B_2 which satisfy detailed balance, it is not necessarily the case that the T thus defined (23.42) satisfies detailed balance.

23.7 Slice sampling

Slice sampling (Neal, 1997) is a Markov chain Monte Carlo method which has similarities to rejection sampling, Gibbs sampling and the Metropolis method.

It can be applied to any systems to which the Metropolis method can be applied, that is, any system for which the target density $P^*(\mathbf{x})$ can be evaluated at any point $\mathbf{x}$; it has the advantage over simple Metropolis methods that it is more robust to the choice of parameters like step sizes. It is similar to Gibbs sampling in that a slice sampling simulation consists of transitions each of which makes a one-dimensional move in the state space; however there is no requirement that one-dimensional conditional distributions be easy to evaluate, nor that they have any convexity properties such as are required for adaptive rejection sampling. And slice sampling is similar to rejection sampling in that it is a method that asymptotically draws samples from the volume under the curve described by $P^*(\mathbf{x})$; but there is no requirement for an upper-bounding function.

I will describe slice sampling by giving a sketch of a one-dimensional sampling algorithm, without the details that make the method valid, then giving a pictorial description that includes those details.

The skeleton of slice sampling

Let us assume that we want to draw samples from $P(x) \propto P^*(x)$ where x is a real number. A one-dimensional slice sampling algorithm is a method for making transitions from a two-dimensional point (x, u) lying under the curve $P^*(x)$ to another point (x', u') lying under the same curve, such that the probability distribution of (x, u) tends to a uniform distribution over the area under the curve $P^*(x)$, whatever initial point we start from. A single transition $(x, u) \rightarrow (x', u')$ of a one-dimensional slice sampling algorithm has the following steps, of which steps 3 and 8 will require further elaboration.

1: evaluate $P^*(x)$
2: draw a vertical coordinate $u' \sim \mathsf{Uniform}(0, P^*(x))$
3: create a horizontal interval (x_l, x_r) enclosing x
4: loop {
5: draw $x' \sim \mathsf{Uniform}(x_l, x_r)$
6: evaluate $P^*(x')$
7: if $P^*(x') > u'$ break out of loop 4
8: else modify the interval (x_l, x_r)
9: }

There are several methods for creating the interval (x_l, x_r) in step 3, and several methods for modifying it at step 8. The important point is that these methods must satisfy detailed balance, so that, the uniform distribution for (x, u) under the curve $P^*(x)$ is invariant. Here I will describe methods appropriate for a real variable x.

The 'stepping out' method for step 3

In the 'stepping out' method for creating an interval (x_l, x_r) enclosing x, we make use of a step length w to step out until we find endpoints x_l and x_r at which P^* is smaller than u. The algorithm is shown in figure 23.11.

3a: draw $r \sim \mathsf{Uniform}(0, 1)$
3b: x_l := $x - rw$
3c: x_r := $x + (1 - r)w$

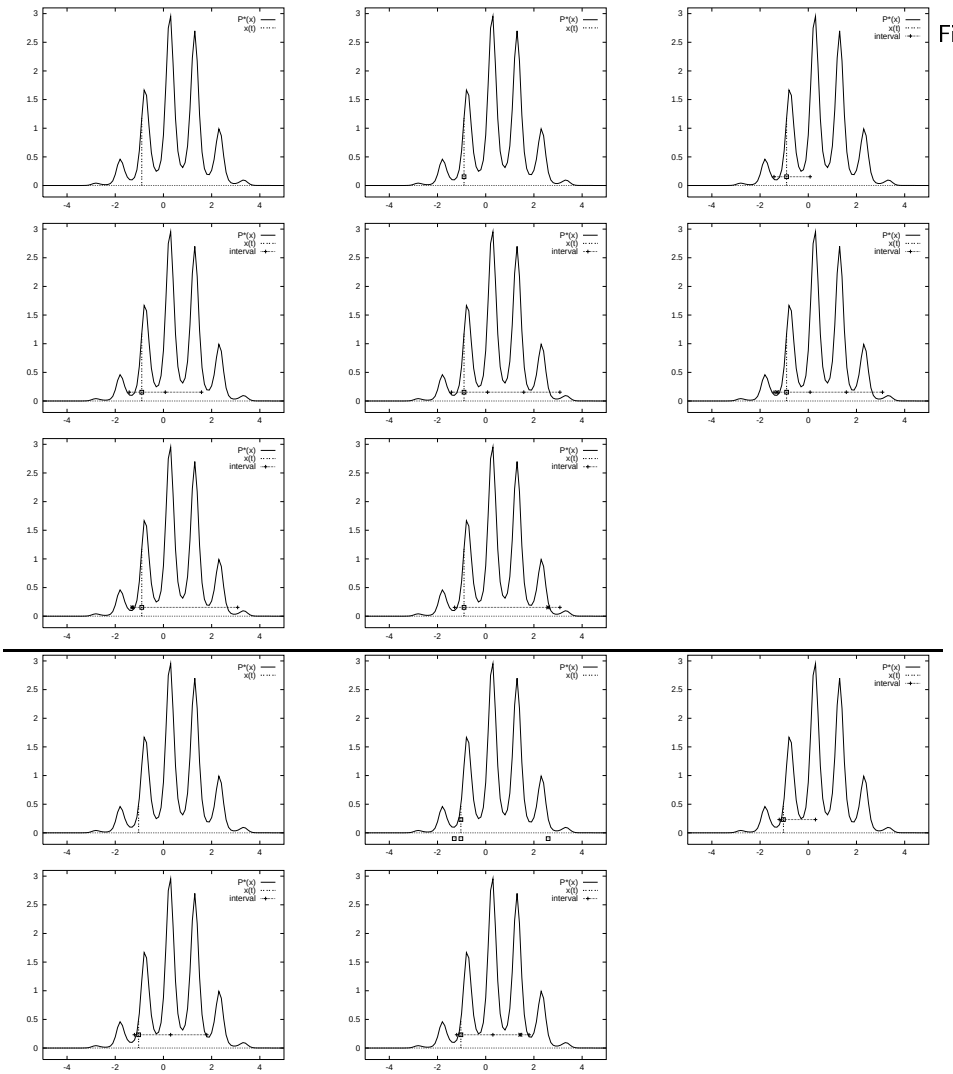


Figure 23.11. Slice sampling

3d: while $(P^*(x_l) > u)$ { $x_l := x_l - w$ }
3e: while $(P^*(x_r) > u)$ { $x_r := x_r + w$ }

The 'shrinking' method for step 8

Whenever a point x' is drawn such that (x', u') lies above the curve $P^*(x)$, we shrink the interval so that one of the end points is x', and such that the original point x is still enclosed in the interval.

8a: if $(x' > x)$
8b: $x_r := x'$
8c: else
8d: $x_l := x'$

Draft of slice sampling discussion

Slice sampling still gets around by a random walk, but there are never any rejections and the step size is self-tuning.

Discuss sensitivity to w. If w too small then waste linear amount of time in stepping out. If too large then waste logarithmic factor shrinking. Contrast with Metropolis, where if w too small get quadratic random walk problem, and if too large, get exponentially bad failure to go places at all.

Slice sampling does not avoid random walk behaviour, but it automatically chooses the largest appropriate step size, thus reducing the bad effects of the random walk compared with, say, a Metropolis method with a tiny step size.

Note that for some distributions it may take a long time to mix, *e.g.*, if there is a peak and a long low tail, then you spend a lot of time in the tail then a lot in the peak. This can in some cases be viewed as beneficial. Skilling has applications where the peak has much more probability mass than the tail, but it is the tail that is of interest, slice sampling is used to explore the tail; transitions between the tail and peak are handled by a separate proposal. Slice sampling is thus one of several base transitions.

23.8 Practicalities

Can we predict how long a Markov chain Monte Carlo simulation will take to equilibrate? By considering the random walks involved in a Markov chain Monte Carlo simulation we can obtain simple *lower bounds* on the time required for convergence. But predicting this time more precisely is a difficult problem, and most of the theoretical results giving upper bounds on the convergence time are of little practical use.

Can we diagnose or detect convergence in a running simulation? This is also a difficult problem. There are a few practical tools available, but none of them is perfect (Cowles and Carlin, 1996).

Can we speed up the convergence time and time between independent samples of a Markov chain Monte Carlo method? Here, there is good news, as described in the following three sections, which describe the hybrid Monte Carlo method, overrelaxation, and simulated annealing.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

```
g = gradE ( x ) ;            # set gradient using initial x
E = findE ( x ) ;            # set objective function too

for l = 1:L                  # loop L times
   p = randn ( size(x) ) ;   # initial momentum is Normal(0,1)
   H = p' * p / 2 + E ;      # evaluate H(x,p)

   xnew = x
   gnew = g ;
   for tau = 1:Tau           # make Tau 'leapfrog' steps

      p = p - epsilon * gnew / 2 ; # make half-step in p
      xnew = xnew + epsilon * p ;  # make step in x

      gnew = gradE ( xnew ) ;      # find new gradient
      p = p - epsilon * gnew / 2 ; # make half-step in p

   endfor

   Enew = findE ( xnew ) ;   # find new value of H
   Hnew = p' * p / 2 + Enew ;
   dH = Hnew - H ;           # Decide whether to accept

   if ( dH < 0 )                accept = 1 ;
   elseif ( rand() < exp(-dH) ) accept = 1 ;
   else                         accept = 0 ;
   endif

   if ( accept )
      g = gnew ;   x = xnew ;    E = Enew ;
   endif
endfor
```

Figure 23.12. `Octave` source code for the hybrid Monte Carlo method.

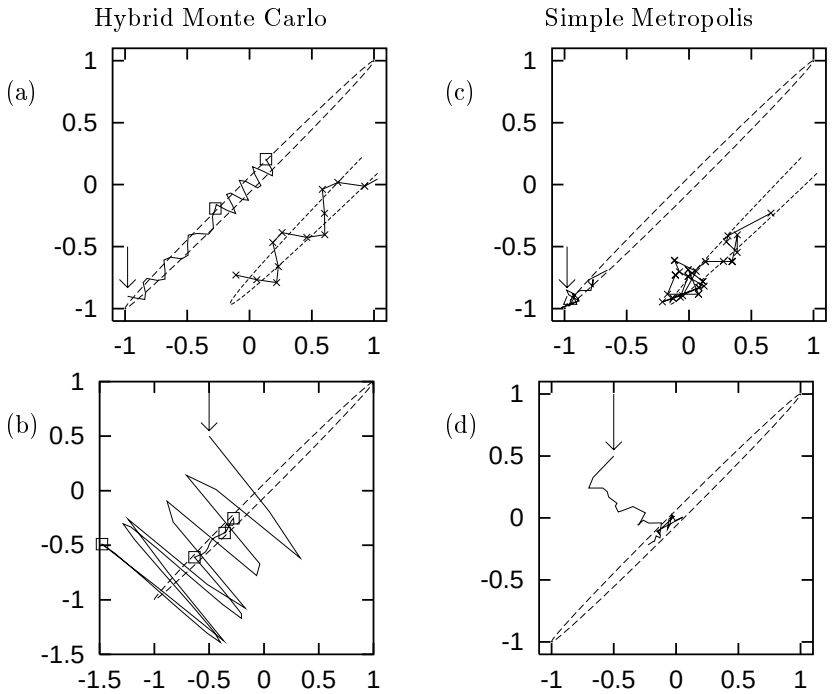


Figure 23.13. (a,b) Hybrid Monte Carlo used to generate samples from a bivariate Gaussian with correlation $\rho = 0.998$. (c,d) A simple random-walk Metropolis method for comparison.

23.9 Reducing random walk behaviour in Metropolis methods

The hybrid Monte Carlo method reviewed in Neal (1993b) is a Metropolis method applicable to continuous state spaces that makes use of gradient information to reduce random walk behaviour.

For many systems, the probability $P(\mathbf{x})$ can be written in the form

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{23.44}$$

where not only $E(\mathbf{x})$, but also its gradient with respect to $\mathbf{x}$ can be readily evaluated. It seems wasteful to use a simple random-walk Metropolis method when this gradient is available — the gradient indicates which direction one should go in to find states with higher probability!

Overview of hybrid Monte Carlo

In the hybrid Monte Carlo method, the state space $\mathbf{x}$ is augmented by *momentum variables* $\mathbf{p}$, and there is an alternation of two types of proposal. The first proposal randomizes the momentum variable, leaving the state $\mathbf{x}$ unchanged. The second proposal changes both $\mathbf{x}$ and $\mathbf{p}$ using simulated Hamiltonian dynamics as defined by the Hamiltonian

$$H(\mathbf{x}, \mathbf{p}) = E(\mathbf{x}) + K(\mathbf{p}), \tag{23.45}$$

where $K(\mathbf{p})$ is a 'kinetic energy' such as $K(\mathbf{p}) = \mathbf{p}^{\mathsf{T}}\mathbf{p}/2$. These two proposals are used to create (asymptotically) samples from the joint density

$$P_H(\mathbf{x}, \mathbf{p}) = \frac{1}{Z_H} \exp[-H(\mathbf{x}, \mathbf{p})] = \frac{1}{Z_H} \exp[-E(\mathbf{x})] \exp[-K(\mathbf{p})]. \tag{23.46}$$

This density is separable, so it is clear that the marginal distribution of $\mathbf{x}$ is the desired distribution $\exp[-E(\mathbf{x})]/Z$. So, simply discarding the momentum variables, we will obtain a sequence of samples $\{\mathbf{x}^{(t)}\}$ which asymptotically come from $P(\mathbf{x})$.

Details of hybrid Monte Carlo

The first proposal draws a new momentum from the Gaussian density $\exp[-K(\mathbf{p})]/Z_K$. During the second, dynamical proposal, the momentum variable determines where the state $\mathbf{x}$ goes, and the *gradient* of $E(\mathbf{x})$ determines how the momentum $\mathbf{p}$ changes, in accordance with the equations

$$\dot{\mathbf{x}} = \mathbf{p} \tag{23.47}$$

$$\dot{\mathbf{p}} = -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}}. \tag{23.48}$$

Because of the persistent motion of $\mathbf{x}$ in the direction of the momentum $\mathbf{p}$ during each dynamical proposal, the state of the system tends to move a distance that goes *linearly* with the computer time, rather than as the square root.

If the simulation of the Hamiltonian dynamics is numerically perfect then the proposals are accepted every time, because the total energy $H(\mathbf{x}, \mathbf{p})$ is a constant of the motion and so a in equation (23.29) is equal to one. If the simulation is imperfect, because of finite step sizes for example, then some of the dynamical proposals will be rejected. The rejection rule makes use of the change in $H(\mathbf{x}, \mathbf{p})$, which is zero if the simulation is perfect. The occasional rejections ensure that, asymptotically, we obtain samples $(\mathbf{x}^{(t)}, \mathbf{p}^{(t)})$ from the required joint density $P_H(\mathbf{x}, \mathbf{p})$.

The source code in figure 23.12 describes a hybrid Monte Carlo method that uses the 'leapfrog' algorithm to simulate the dynamics on the function `findE(x)`, whose gradient is found by the function `gradE(x)`. Figure 23.13 shows this algorithm generating samples from a bivariate Gaussian whose energy function is $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$ with

$$\mathbf{A} = \begin{bmatrix} 250.25 & -249.75 \\ -249.75 & 250.25 \end{bmatrix}. \tag{23.49}$$

In figure 23.13(a), starting from the state marked by the arrow, the solid line represents two successive trajectories generated by the Hamiltonian dynamics. The squares show the endpoints of these two trajectories. Each trajectory consists of `Tau` $= 19$ 'leapfrog' steps with `epsilon` $= 0.055$. These steps are indicated by the crosses on the trajectory in the magnified inset. After each trajectory, the momentum is randomized. Here, both trajectories are accepted; the errors in the Hamiltonian were only $+0.016$ and -0.06 respectively.

Figure 23.13(b) shows how a sequence of four trajectories converges from an initial condition, indicated by the arrow, that is not close to the typical set of the target distribution. The trajectory parameters `Tau` and `epsilon` were randomized for each trajectory using uniform distributions with means 19 and 0.055 respectively. The first trajectory takes us to a new state, $(-1.5, -0.5)$, similar in energy to the first state. The second trajectory happens to end in a state nearer the bottom of the energy landscape. Here, since the potential energy E is smaller, the kinetic energy $K = \mathbf{p}^2/2$ is necessarily larger than it

was at the start of the trajectory. When the momentum is randomized before the third trajectory, its magnitude becomes much smaller. After the fourth trajectory has been simulated, the state appears to have become typical of the target density.

Figures 23.13(c,d) show a random-walk Metropolis method using a Gaussian proposal density to sample from the same Gaussian distribution, starting from the initial conditions of (a) and (b) respectively. In (c) the radius had been adjusted such that the acceptance rate was 58%. The number of proposals was 38 so the total amount of computer time used was similar to that in (a). The distance moved is small because of random walk behaviour. In (d) the random-walk Metropolis method was used and started from the same initial condition as (b) and given a similar amount of computer time.

23.10 Overrelaxation

The method of *overrelaxation* is a similar method for reducing random walk behaviour in Gibbs sampling. Overrelaxation was originally introduced for systems in which all the conditional distributions are Gaussian.

An example of a joint distribution that is *not* Gaussian but whose conditional distributions *are* all Gaussian is $P(x, y) = \exp(-x^2 y^2)/Z$.

Adler's Overrelaxation: Overrelaxation for Gaussian conditional distributions

In ordinary Gibbs sampling, one draws the new value $x_i^{(t+1)}$ of the current variable x_i from its conditional distribution, ignoring the old value $x_i^{(t)}$. This leads to lengthy random walks in cases where the variables are strongly correlated, as illustrated in the left hand panel of figure 23.14. This figure uses a correlated Gaussian distribution as the target density.

In Adler's (1981) overrelaxation method, one instead samples $x_i^{(t+1)}$ from a Gaussian that is biased to the *opposite* side of the conditional distribution. If the conditional distribution of x_i is Normal(μ, σ^2) and the current value of x_i is $x_i^{(t)}$, then Adler's method sets x_i to

$$x_i^{(t+1)} = \mu + \alpha(x_i^{(t)} - \mu) + (1 - \alpha^2)^{1/2}\sigma\nu, \tag{23.50}$$

where $\nu \sim$ Normal$(0, 1)$ and α is a parameter between -1 and 1, usually set to a negative value. (If α is positive, then the method is called under-relaxation.)

Exercise 23.8:[A2] Show that this individual transition leaves invariant the conditional distribution $x_i \sim$ Normal(μ, σ^2).

A single iteration of Adler's overrelaxation, like one of Gibbs sampling, updates each variable in turn as indicated in equation (23.50). The transition matrix $T(\mathbf{x}'; \mathbf{x})$ defined by a complete update of all variables in some fixed order does not satisfy detailed balance. The individual transitions for the individual coordinates just described *do* satisfy detailed balance — so the overall chain gives a valid sampling strategy which converges to the target density $P(\mathbf{x})$ — but when we form a chain by applying the individual transitions in a fixed sequence, the overall chain is not reversible. This temporal asymmetry is the key to why overrelaxation can be beneficial. If, say, two variables are positively

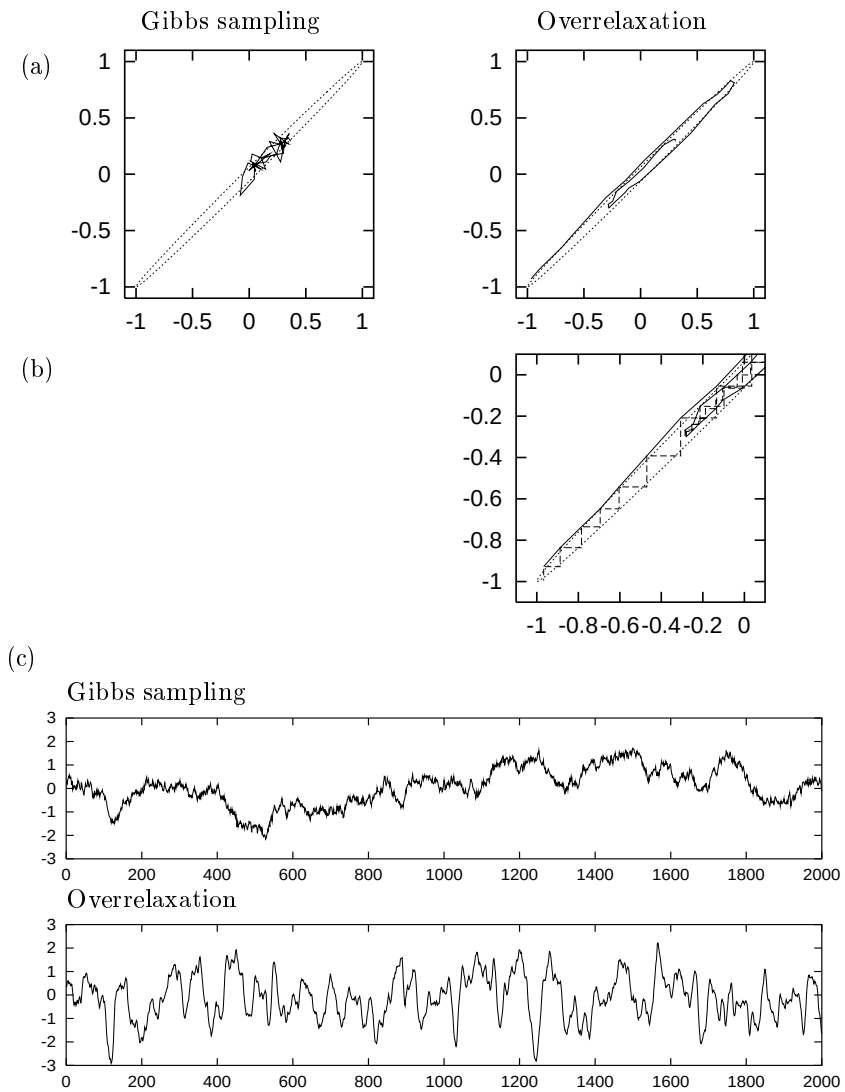


Figure 23.14. Overrelaxation contrasted with Gibbs sampling for a bivariate Gaussian with correlation $\rho = 0.998$. (a) The state sequence for 40 iterations, each iteration involving one update of both variables. The overrelaxation method had $\alpha = -0.98$. (This excessively large value is chosen to make it easy to see how the overrelaxation method reduces random walk behaviour.) The dotted line shows the contour $\mathbf{x}^{\mathrm{T}}\Sigma^{-1}\mathbf{x} = 1$. (b) Detail of (a), showing the two steps making up each iteration. (c) Time-course of the variable x_1 during 2000 iterations of the two methods. The overrelaxation method had $\alpha = -0.89$. (After Neal (1995).)

correlated, then they will (on a short timescale) evolve in a directed manner instead of by random walk, as shown in figure 23.14. This may significantly reduce the time required to obtain effectively independent samples.

Figure 23.14 illustrates the difference between Gibbs sampling and overrelaxation for the case of a bivariate Gaussian distribution. Notice how much more rapidly overrelaxation gets around the distribution.

Ordered Overrelaxation

The overrelaxation method has been generalized by Neal (1995) whose *ordered overrelaxation* method is applicable to *any* system where Gibbs sampling is used. In ordered overrelaxation, instead of taking one sample from the conditional distribution $P(x_i|\{x_j\}_{j\neq i})$, we create K such samples $x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(K)}$, where K might be set to twenty or so. Often generating $K-1$ extra samples adds a negligible computational cost to the initial computations required for making the first sample. The points $\{x_i^{(k)}\}$ are then sorted numerically, and the current value of x_i is inserted into the sorted list, giving a list of $K+1$ points. We give them ranks $0, 1, 2, \ldots, K$. Let κ be the rank of the current value of x_i in the list. We set x_i' to the value that is an equal distance from the other end of the list, that is, the value with rank $K-\kappa$. The role played by Adler's α parameter is here played by the parameter K. When $K=1$, we obtain ordinary Gibbs sampling. Neal recommends using a small value of K, *e.g.*, $K=1$, before 'convergence' of a Gibbs sampler, then a value such as $K=20$ after convergence. For practical purposes Neal estimates that ordered overrelaxation may speed up a simulation by a factor of ten or twenty.

23.11 Simulated annealing

A third technique for speeding convergence is *simulated annealing*. In simulated annealing, a 'temperature' parameter is introduced which, when large, allows the system to make transitions that would be improbable at temperature 1. The temperature may initially be set to a large value and reduced gradually to 1. It is hoped that this procedure reduces the chance of the simulation's becoming stuck in an unrepresentative probability island.

We asssume that we wish to sample from a distribution of the form

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{23.51}$$

where $E(\mathbf{x})$ can be evaluated. In the simplest simulated annealing method, we instead sample from the distribution

$$P_T(\mathbf{x}) = \frac{1}{Z(T)} e^{-\frac{E(\mathbf{x})}{T}} \tag{23.52}$$

and decrease T gradually to 1.

Often the energy function can be separated into two terms,

$$E(\mathbf{x}) = E_0(\mathbf{x}) + E_1(\mathbf{x}), \tag{23.53}$$

of which the first term is 'nice' (for example, a separable function of $\mathbf{x}$) and the second is 'nasty'. In these cases, a better simulated annealing method might

make use of the distribution

$$P'_T(\mathbf{x}) = \frac{1}{Z'(T)} e^{-E_0(\mathbf{x}) - \frac{E_1(\mathbf{x})}{T}} \tag{23.54}$$

with T gradually decreasing to 1. In this way, the distribution at high temperatures reverts to a well-behaved distribution defined by E_0.

Simulated annealing is often used as an optimization method, where the aim is to find an $\mathbf{x}$ that minimizes $E(\mathbf{x})$, in which case the temperature is decreased to zero rather than to 1.

As a Monte Carlo method, simulated annealing as described above doesn't sample exactly from the right distribution, because there is no guarantee that the probability of falling into one basin of the energy is equal to the total probability of all the states in that basin — indeed we would expect this not to be the case in general; simulated annealing is usually a biased sampling method. The closely related 'simulated tempering' methods (Marinari and Parisi, 1992) correct the biases introduced by the annealing process by making the temperature itself a random variable that is updated in Metropolis fashion during the simulation. Neal's (1998) 'annealed importance sampling' method removes the biases introduced by annealing by computing importance weights for each generated point.

23.12 Further practical issues

Can the normalizing constant be evaluated?

If the target density $P(\mathbf{x})$ is given in the form of an unnormalized density $P^*(\mathbf{x})$ with $P(\mathbf{x}) = \frac{1}{Z} P^*(\mathbf{x})$, the value of Z may well be of interest. Monte Carlo methods do not readily yield an estimate of this quantity, and it is an area of active research to find ways of evaluating it. Techniques for evaluating Z include:

1. Importance sampling (reviewed by Neal (1993b)) and annealed importance sampling (Neal, 1998).

2. 'Thermodynamic integration' during simulated annealing, the 'acceptance ratio' method, and 'umbrella sampling' (reviewed by Neal (1993b)).

3. 'Reversible jump Markov chain Monte Carlo' (Green, 1995).

Perhaps the best way of dealing with Z, however, is to find a solution to one's task that does not require that Z be evaluated. In Bayesian data modelling one can avoid the need to evaluate Z — which would be important for model comparison — by not having more than one model. Instead of using several models (differing in complexity, for example) and evaluating their relative posterior probabilities, one can make a single *hierarchical* model having, for example, various continuous hyperparameters which play a role similar to that played by the distinct models (Neal, 1996).

The Metropolis method for big models

Our original description of the Metropolis method involved a joint updating of all the variables using a proposal density $Q(\mathbf{x}'; \mathbf{x})$. For big problems it may be more efficient to use several proposal distributions $Q^{(b)}(\mathbf{x}'; \mathbf{x})$, each of

which updates only some of the components of $\mathbf{x}$. Each proposal is individually accepted or rejected, and the proposal distributions are repeatedly run through in sequence.

Exercise 23.9:[B2] Explain why the rate of movement through the state space will be greater when B proposals $Q^{(1)}, \ldots, Q^{(B)}$ are considered *individually* in sequence, compared with the case of a single proposal Q^* defined by the concatenation of $Q^{(1)}, \ldots, Q^{(B)}$. Assume that each proposal distribution $Q^{(b)}(\mathbf{x}'; \mathbf{x})$ has an acceptance rate f.

In the Metropolis method, the proposal density $Q(\mathbf{x}'; \mathbf{x})$ typically has a number of parameters that control, for example, its 'width'. These parameters are usually set by trial and error with the rule of thumb being that one aims for a rejection frequency of about 0.5. It is *not* valid to have the width parameters be dynamically updated during the simulation in a way that depends on the history of the simulation. Such a modification of the proposal density would violate the detailed balance condition which guarantees that the Markov chain has the correct invariant distribution.

Gibbs sampling in big models

Our description of Gibbs sampling involved sampling one parameter at a time, as described in equations (23.33–23.35). For big problems it may be more efficient to sample *groups* of variables jointly, that is to use several proposal distributions:

$$\begin{aligned} x_1^{(t+1)}, \ldots, x_a^{(t+1)} &\sim P(x_1 \ldots x_a | x_{a+1}^{(t)}, \ldots, x_K^{(t)}) \qquad (23.55) \\ x_{a+1}^{(t+1)}, \ldots, x_b^{(t+1)} &\sim P(x_{a+1}, \ldots, x_b | x_1^{(t+1)}, \ldots, x_a^{(t+1)}, x_{b+1}^{(t)}, \ldots, x_K^{(t)}), \quad \text{etc..} \end{aligned}$$

How many samples are needed?

At the start of this chapter, we observed that the variance of an estimator $\hat{\Phi}$ depends only on the number of independent samples R and the value of

$$\sigma^2 = \int d^N \mathbf{x} \; P(\mathbf{x}) (\phi(\mathbf{x}) - \Phi)^2. \qquad (23.56)$$

We have now discussed a variety of methods for generating samples from $P(\mathbf{x})$. How many independent samples R should we aim for?

In many problems, we really only need about twelve independent samples from $P(\mathbf{x})$. Imagine that $\mathbf{x}$ is an unknown vector such as the amount of corrosion present in each of 10,000 underground pipelines around Cambridge, and $\phi(\mathbf{x})$ is the total cost of repairing those pipelines. The distribution $P(\mathbf{x})$ describes the probability of a state $\mathbf{x}$ given the tests that have been carried out on some pipelines and the assumptions about the physics of corrosion. The quantity Φ is the expected cost of the repairs. The quantity σ^2 is the variance of the cost — σ measures by how much we should expect the actual cost to differ from the expectation Φ.

Now, how accurately would a manager like to know Φ? I would suggest there is little point in knowing Φ to a precision finer than about $\sigma/3$. After all, the true cost is likely to differ by $\pm\sigma$ from Φ. If we obtain $R = 12$ independent samples from $P(\mathbf{x})$, we can estimate Φ to a precision of $\sigma/\sqrt{12}$ — which is smaller than $\sigma/3$. So twelve samples suffice.

Figure 23.15. Three possible Markov Chain Monte Carlo strategies for obtaining twelve samples in a fixed amount of computer time. Time is represented by horizontal lines; samples by white circles. (1) A single run consisting of one long 'burn in' period followed by a sampling period. (2) Four medium-length runs with different initial conditions and a medium-length burn in period. (3) Twelve short runs.

Allocation of resources

Assuming we have decided how many independent samples R are required, an important question is how one should make use of one's limited computer resources to obtain these samples.

A typical Markov chain Monte Carlo experiment involves an initial period in which control parameters of the simulation such as step sizes may be adjusted. This is followed by a 'burn in' period during which we hope the simulation 'converges' to the desired distribution. Finally, as the simulation continues, we record the state vector occasionally so as to create a list of states $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$ that we hope are roughly independent samples from $P(\mathbf{x})$.

There are several possible strategies (figure 23.15).

1. Make one long run, obtaining all R samples from it.
2. Make a few medium length runs with different initial conditions, obtaining some samples from each.
3. Make R short runs, each starting from a different random initial condition, with the only state that is recorded being the final state of each simulation.

The first strategy has the best chance of attaining 'convergence'. The last strategy may have the advantage that the correlations between the recorded samples are smaller. The middle path appears to be popular with Markov chain Monte Carlo experts because it avoids the inefficiency of discarding burn-in iterations in many runs, while still allowing one to detect problems with lack of convergence that would not be apparent from a single run.

Philosophy

One curious defect of these Monte Carlo methods — which are widely used by Bayesian statisticians — is that they are all non-Bayesian. They involve computer experiments from which *estimators* of quantities of interest are derived. These estimators depend on the proposal distributions that were used to generate the samples. In contrast, an alternative Bayesian approach to the problem would use the results of our computer experiments to infer the properties of the target function $P(\mathbf{x})$ and generate predictive distributions for quantities of interest such as Φ. This approach would give answers which would depend only on the computed values of $P^*(\mathbf{x}^{(r)})$ at the points $\{\mathbf{x}^{(r)}\}$; the answers would not depend on how those points were chosen.

It remains an open problem to create a Bayesian version of Monte Carlo methods.

23.13 Summary

- Monte Carlo methods are a powerful tool that allow one to implement any probability distribution that can be expressed in the form $P(\mathbf{x}) = \frac{1}{Z}P^*(\mathbf{x})$.
- Monte Carlo methods can answer virtually any query related to $P(\mathbf{x})$ by putting the query in the form

$$\int \phi(\mathbf{x})P(\mathbf{x}) \simeq \frac{1}{R}\sum_r \phi(\mathbf{x}^{(r)}). \tag{23.57}$$

- In high-dimensional problems the only satisfactory methods are those based on Markov chain Monte Carlo: the Metropolis method, Gibbs sampling and slice sampling. Gibbs sampling is an attractive method because it has no adjustable parameters but its use is restricted to cases where samples can be generated from the conditional distributions. Slice sampling is attractive because, whilst it has step length parameters, its performance is not very sensitive to their values.
- Simple Metropolis algorithms, although widely used, perform poorly because they explore the space by a slow random walk. More sophisticated Metropolis algorithms such as hybrid Monte Carlo make use of proposal densities that give faster movement through the state space.

  The efficiency of Gibbs sampling is also troubled by random walks. The method of ordered overrelaxation is a general purpose technique for suppressing them.

23.14 Exercises

Exercise 23.10:^{A2} Consider the Metropolis algorithm for the one-dimensional toy problem of section 23.4. Whenever the current state is one of the end states, the proposal density given in equation (23.32) will propose with probability 50% a state that will be rejected.

To reduce this 'waste', Fred modifies the software responsible for generating samples from Q so that when $x = 0$, the proposal density is 100% on $x' = 1$, and similarly when $x = 20$, $x' = 19$ is always proposed. Fred sets the software that implements the acceptance rule so that the software accepts all proposed moves. What probability $P'(x)$ will Fred's modified software generate samples from?

What is the correct acceptance rule for Fred's proposal density, in order to obtain samples from $P(x)$?

Exercise 23.11:^{B2} When discussing the time taken by the Metropolis algorithm to generate independent samples we considered a distribution with longest spatial length scale L being explored using a proposal distribution with step size ϵ. Another dimension that a MCMC method must explore is the range of possible values of the log probability $\log P^*(\mathbf{x})$. Assuming that the

state $\mathbf{x}$ contains a number of independent random variables proportional to N, when samples are drawn from $P(\mathbf{x})$, the 'Asymptotic Equipartition' Principle tell us that the value of $-\log P(\mathbf{x})$ is likely to be close to the entropy of $\mathbf{x}$, varying either side with a standard deviation that scales as $\sqrt{N}$. Consider a Metropolis method with a symmetrical proposal density, that is, one which satisfies $Q(\mathbf{x}; \mathbf{x}') = Q(\mathbf{x}'; \mathbf{x})$. Assuming that accepted jumps either increase $\log P^*(\mathbf{x})$ by some amount or decrease it by a *small* amount, *e.g.*, $\log e = 1$ (is this a reasonable assumption?), discuss how long it must take to generate roughly independent samples from $P(\mathbf{x})$. Discuss whether Gibbs sampling has similar properties.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

Solutions to Chapter 23's exercises

Solution to exercise 23.1 (p. 291): We wish to show that

$$\hat{\Phi} \equiv \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r} \tag{23.58}$$

converges to the expectation of Φ under P. We consider the numerator and the denominator separately. First, the denominator. Consider a single importance weight.

$$w_r \equiv \frac{P^*(x^{(r)})}{Q^*(x^{(r)})} \tag{23.59}$$

What is its expectation, averaged under the distribution $Q = Q^*/Z_Q$ of the point $x^{(r)}$?

$$\langle w_r \rangle = \int dx Q(x) \frac{P^*(x)}{Q^*(x)} = \int dx \frac{1}{Z_Q} P^*(x) = \frac{Z_P}{Z_Q}. \tag{23.60}$$

So the expectation of the denominator is

$$\left\langle \sum_r w_r \right\rangle = R \frac{Z_P}{Z_Q}. \tag{23.61}$$

As long as the variance of w_r is finite, the denominator, divided by R, will converge to Z_P/Z_Q as R increases. Similarly, the expectation of one term in the numerator is

$$\langle w_r \phi(x) \rangle = \int dx Q(x) \frac{P^*(x)}{Q^*(x)} \phi(x) = \int dx \frac{1}{Z_Q} P^*(x) \phi(x) = \frac{Z_P}{Z_Q} \Phi, \tag{23.62}$$

where Φ is the expectation of ϕ under P. So the numerator, divided by R, converges to $\frac{Z_P}{Z_Q}\Phi$ with increasing R. Thus $\hat{\Phi}$ converged to Φ.

The numerator and the denominator are unbiased estimators of RZ_P/Z_Q and $RZ_P/Z_Q\Phi$ respectively, but their ratio $\hat{\Phi}$ is not necessarily an unbiased estimator for finite R. More here about variance and bias.

Solution to exercise 23.2 (p. 292): ...

Solution to exercise 23.3 (p. 297): ...

Solution to exercise 23.4 (p. 301): ...

Solution to exercise 23.5 (p. 301): The posterior distribution for the syndrome decoding problem is a pathological distribution from the point of view of Gibbs sampling. The factor $\delta(\mathbf{Hn} = \mathbf{z})$ is only 1 on a small fraction of the space of

possible vectors $\mathbf{n}$, namely the 2^K points that correspond to the valid codewords. No two codewords are adjacent, so similarly, any single bit flip from a viable state $\mathbf{n}$ will take us to a state with zero probability and so the state will never move in Gibbs sampling.

A general code has exactly the same problem. The points corresponding to valid codewords are relatively few in number and they are not adjacent (at least for any useful code). So Gibbs sampling is no use for syndrome decoding for two reasons. First, finding *any* valid hypothesis is difficult, and as long as the state is not near a valid codeword, Gibbs sampling cannot help since none of the conditional distributions is defined; and second, once we are in a valid hypothesis, Gibbs sampling will never take us out of it.

Solution to exercise 23.9 (p. 314): ...

Solution to exercise 23.10 (p. 316): Fred's proposals would be appropriate if the target density $P(x)$ were half as great on the two end states as on all other states. If this were the target density, then the factor of two difference in Q for a transition in or out of an end state would be balanced by the factor of two difference in P, and the acceptance probability would be 1. Fred's algorithm therefore samples from the distribution

$$P'(x) = \begin{cases} \frac{1}{20} & x \in \{1, 2, \ldots, 19\} \\ \frac{1}{40} & x \in \{0, 20\} \\ 0 & \text{otherwise} \end{cases} . \tag{23.63}$$

If Fred wished to retain the new proposal density, he would have to change the acceptance rule such that transitions *out of* the end states would only be accepted with probability 0.5.

Solution to exercise 23.11 (p. 316): Typical samples differ in their value of $\log P(\mathbf{x})$ by a standard deviation of order $\sqrt{N}$, let's say $c\sqrt{N}$. But the value of $\log P(\mathbf{x})$ varies during a Metropolis simulation by a random walk whose steps when negative are roughly of unit size and when positive are of some unknown size. We don't know what fraction of the steps are negative steps or positive steps, but we can say for certain that it will take at least $c\sqrt{N}$ iterations for the state of the system to traverse the typical set once, assuming it makes negative steps nearly all the time and occasional large positive steps. If the random walk of $\log P(\mathbf{x})$ is a more balanced walk with positive steps of similar size to the negative steps then it will take even longer to traverse the typical set. A drunkard's walk takes a time $T \simeq c^2 N$ to go a distance $c\sqrt{N}$ using unit steps.

Gibbs sampling will not necessarily take so long to generate independent samples because in Gibbs sampling it is possible for the value of $\log P(\mathbf{x})$ to change by a large quantity up or down in a single iteration. All the same, in many problems each Gibbs sampling update only changes $\log P(\mathbf{x})$ by a small amount of order 1, so $\log P(\mathbf{x})$ evolves by a random walk which takes a time $T \simeq c^2 N$ to traverse the typical set.

About Chapter 24

Some of the neural network models that we will encounter are related to Ising models, which are idealised magnetic systems. It is not essential to understand the statistical physics of Ising models to understand these neural networks, but I hope you'll find them helpful.

Ising models are also related to several other topics in this book. We will use exact tree-based computation methods like those introduced in chapter 20 to evaluate properties of interest in Ising models. Ising models offer crude models for binary images. And Ising models relate to two-dimensional constrained channels (c.f. chapter 17): a two-dimensional bar-code in which a black dot may not be completely surrounded by black dots and a white dot may not be completely surrounded by white dots is similar to an antiferromagnetic Ising model at low temperature. Evaluating the entropy of this Ising model is equivalent to evaluating the capacity of the constrained channel for conveying bits.

If you would like a refresher course on statistical physics and thermodynamics, you might find appendix A.1 helpful. I also recommend the book by Reif (1965).

24

Ising Models

An Ising model is an array of spins (*e.g.*, atoms that can take states ± 1) that are magnetically coupled to each other. If one spin is, say, in the $+1$ state then it is energetically favourable for its immediate neighbours to be in the same state, in the case of a ferromagnetic model, and in the opposite state, in the case of an antiferromagnet. In this chapter we will discuss two computational techniques for studying Ising models.

Let the state $\mathbf{x}$ of an Ising model with N spins be a vector in which each component x_n takes values -1 or $+1$. If two spins m and n are neighbours we write $(m,n) \in \mathcal{N}$. The energy of a state $\mathbf{x}$ is

$$E(\mathbf{x}; J, H) = -\left[\sum_{(m,n)\in\mathcal{N}} J x_m x_n + \sum_n H x_n \right], \tag{24.1}$$

where J is the coupling and H is the applied field. If $J > 0$ then the model is ferromagnetic, and if $J < 0$ it is antiferromagnetic. Note that each pair (m,n) is counted just once in the first sum; alternatively, we could sum over all m and n, shove in a factor of $\frac{1}{2}$, and define J_{mn} (the coupling between spins m and n) to be zero if $(m,n) \notin \mathcal{N}$. At equilibrium at temperature T, the probability that the state is $\mathbf{x}$ is

$$P(\mathbf{x}|\beta, J, H) = \frac{1}{Z(\beta, J, H)} \exp\left[-\beta E(\mathbf{x}; J, H)\right] \tag{24.2}$$

where $\beta = 1/k_{\mathrm{B}}T$, k_{B} is Boltzmann's constant, and

$$Z(\beta, J, H) \equiv \sum_{\mathbf{x}} \exp\left[-\beta E(\mathbf{x}; J, H)\right]. \tag{24.3}$$

Relevance of Ising models

Ising models are important first as models of magnetism and other systems that show phase transitions. The theory of universality in statistical physics shows that all systems with the same dimension (here, two), and the same symmetries, have equivalent critical properties, *i.e.*, the scaling laws shown by their phase transitions are identical. So by studying Ising models we can find out about many other phase transitions.

Second, if we generalize the energy function to

$$E(\mathbf{x}; \mathbf{J}, \mathbf{h}) = -\left[\sum_{(m,n)\in\mathcal{N}} J_{mn} x_m x_n + \sum_n h_n x_n \right], \tag{24.4}$$

where the couplings J_{mn} and applied fields h_n are not constant, we obtain a family of models known as 'spin glasses' to physicists, and as 'Hopfield networks' or 'Boltzmann machines' to the neural network community. In some of these models, all spins are declared to be neighbours of each other, in which case physicists call the system an 'infinite range' spin glass, and networkers call it a 'fully connected' network.

Third, the Ising model is also useful as a statistical model in its own right.

In sections 24.1 and 24.2 we will study Ising models using two different computational techniques.

Some remarkable relationships in Statistical Physics

We would like to get as much information as possible out of our experiments. Consider for example the heat capacity of a system, which is defined to be

$$C \equiv \frac{\partial}{\partial T} \bar{E}, \tag{24.5}$$

where

$$\bar{E} = \frac{1}{Z} \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})) E(\mathbf{x}). \tag{24.6}$$

Naively, we might guess that to work out the heat capacity of a system at a certain temperature, we have to change the temperature to a higher temperature and measure the energy change. However, there are intimate relationships between the heat capacity of a system at some temperature and the *fluctuations* in its energy at that temperature. Let's start from the partition function,

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})). \tag{24.7}$$

The mean energy is obtained by differentiation with respect to β:

$$\frac{\partial \log Z}{\partial \beta} = \frac{1}{Z} \sum_{\mathbf{x}} -E(\mathbf{x}) \exp(-\beta E(\mathbf{x})) = -\bar{E}. \tag{24.8}$$

A further differentiation spits out the variance of the energy:

$$\frac{\partial^2 \log Z}{\partial \beta^2} = -\frac{1}{Z} \sum_{\mathbf{x}} E(\mathbf{x})^2 \exp(-\beta E(\mathbf{x})) - \bar{E}^2 = \langle E^2 \rangle - \bar{E}^2 = \mathrm{var}(E). \tag{24.9}$$

But the heat capacity is also the derivative of $\bar{E}$ with respect to temperature:

$$\frac{\partial \bar{E}}{\partial T} = -\frac{\partial}{\partial T} \frac{\partial \log Z}{\partial \beta} = -\frac{\partial^2 \log Z}{\partial \beta^2} \frac{\partial \beta}{\partial T} = -\mathrm{var}(E)(-1/k_{\mathrm{B}} T^2). \tag{24.10}$$

So for any system at temperature T,

$$C = \frac{\mathrm{var}(E)}{k_{\mathrm{B}} T^2} = k_{\mathrm{B}} \beta^2 \mathrm{var}(E) \tag{24.11}$$

Thus if we can observe the variance of the energy of a system at equilibrium, this can be a smart way to estimate its heat capacity.

I find this an almost paradoxical relationship. Consider a system with a finite set of states, and imagine heating it up. At high temperature, all states will be equiprobable, so the mean energy will be essentially constant and the

heat capacity will be essentially zero. But on the other hand, with all states being equiprobable, there will certainly be fluctuations in energy. So how can the heat capacity be related to the fluctuations? The answer is in the word 'essentially' above. The heat capacity is not quite zero at high temperature, it just tends to zero. And it tends to zero as $\frac{\mathrm{var}(E)}{k_{\mathrm{B}}T^2}$, with the quantity $\mathrm{var}(E)$ tending to a constant at high temperatures. This $1/T^2$ behaviour of the heat capacity of finite systems at high temperatures is thus completely general.

The $1/T^2$ factor can be viewed as an accident of history. If only temperature scales had been defined using $\beta = \frac{1}{k_B T}$, then the definition of heat capacity would be

$$C^{(\beta)} \equiv \frac{\partial \bar{E}}{\partial \beta} = \mathrm{var}(E). \tag{24.12}$$

So heat capacity and fluctuations would be identical quantities, were it not for this slip by Kelvin, Carnot et al.

Exercise 24.1:^{B2} [We will call the entropy of a physical system S rather than H, while we are in a statistical physics chapter; for convenience we will set $k_{\mathrm{B}} = 1$.]

The entropy of a system whose states are $\mathbf{x}$, at temperature $T = 1/\beta$, is

$$S = \sum p(\mathbf{x}) \left[\log 1/p(\mathbf{x})\right]. \tag{24.13}$$

where

$$p(\mathbf{x}) = \frac{1}{Z(\beta)} \exp\left[-\beta E(\mathbf{x})\right] \tag{24.14}$$

(a) Show that

$$S = \log Z(\beta) + \beta \bar{E}(\beta) \tag{24.15}$$

where $\bar{E}(\beta)$ is the mean energy of the system.

(b) Show that

$$S = -\frac{\partial F}{\partial T}, \tag{24.16}$$

where the free energy $F = -kT \log Z$ and $kT = 1/\beta$.

24.1 Ising Models — Monte Carlo simulation

In this section we will study two-dimensional planar Ising models using a simple Gibbs sampling method. Starting from some initial state, a spin n is selected at random, and the probability that it should be $+1$ given the state of the other spins and the temperature is computed,

$$P(+1|h_n) = \frac{1}{1 + \exp(-2\beta b_n)}, \tag{24.17}$$

where $\beta = 1/k_{\mathrm{B}}T$ and b_n is the local field

$$b_n = \sum_{m:(m,n)\in\mathcal{N}} J x_m + H. \tag{24.18}$$

[The factor of 2 appears in equation (24.17) because the two spin states are $\{+1, -1\}$ rather than $\{+1, 0\}$.] Spin n is set to $+1$ with that probability, and otherwise to -1; then a new spin is selected at random. After sufficiently

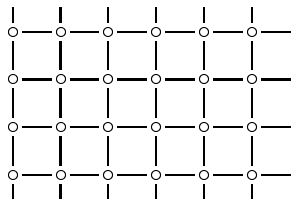
many iterations, this procedure converges to the equilibrium distribution of equation (24.2). An alternative to the Gibbs sampling formula (24.17) is the Metropolis algorithm, in which we consider the change in energy that results from flipping the chosen spin from its current state x_n,

$$\Delta E = 2x_n b_n, \tag{24.19}$$

and adopt this change in configuration with probability

$$P(\text{accept}; \Delta E, \beta) = \begin{cases} 1 & \Delta E \leq 0 \\ \exp(-\beta \Delta E) & \Delta E > 0 \end{cases} . \tag{24.20}$$

This procedure has roughly double the probability of accepting energetically unfavourable moves, so may be a more efficient sampler — but at very low temperatures the relative advantages of Gibbs sampling and the Metropolis algorithm may be subtle.



Rectangular geometry

Let us first write a program that simulates an Ising model with the rectangular geometry shown in the margin, and with periodic boundary conditions. A line between two spins indicates that they are neighbours. I set the external field $H = 0$ and consider the two cases $J = \pm 1$ which are a ferromagnet and antiferromagnet respectively.

I started the system at a large temperature ($T = 33, \beta = 0.03$) and changed the temperature every I iterations, first decreasing it gradually to $T = 0.1, \beta = 10$, then increasing it gradually back to a large temperature again. This procedure gives a crude check on whether 'equilibrium has been reached' at each temperature; if not, we'd expect to see some hysteresis in the graphs we plot. It also gives an idea of the reproducibility of the results, if we assume that the two runs, with decreasing and increasing temperature, are effectively independent of each other.

At each temperature I recorded the mean energy per spin and the standard deviation of the energy, and the mean square value of the magnetization m,

$$m = \frac{1}{N} \sum_n x_n. \tag{24.21}$$

One tricky decision that has to be made is how soon to start taking these measurements after a new temperature has been established; it is virtually impossible to detect 'equilibrium' — or even give a clear definition to a system's being 'at equilibrium'! My crude strategy was to let the number of iterations I be a few hundred times the number of spins N, and to assume equilibrium had been reached after $I/3$ iterations. With $N = 100$, I found I needed more than 100000 iterations to reach equilibrium at any given temperature.

Results for small N with $J = 1$.

I simulated an $l \times l$ grid for $l = 3, 4, 5, \ldots, 10, 40, 64$. Let's have a quick think about what results we expect. At low temperatures the system is expected to be in a ground state. The rectangular Ising model with $J = 1$ has two ground states, the all $+1$ state and the all -1 state. The energy per spin of either ground state is -2. At high temperatures all states are equally probable

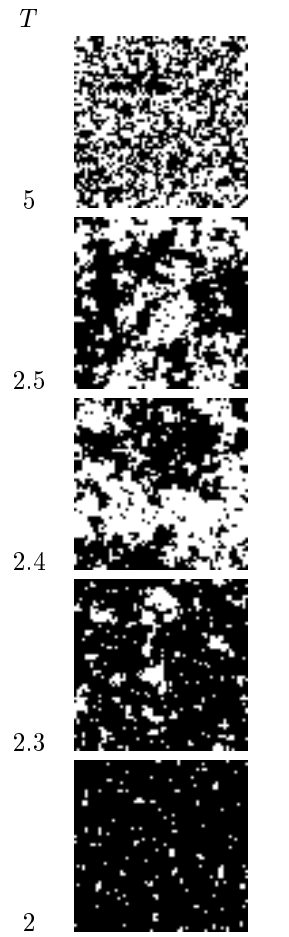


Figure 24.1. Sample states of rectangular Ising models with $J = 1$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

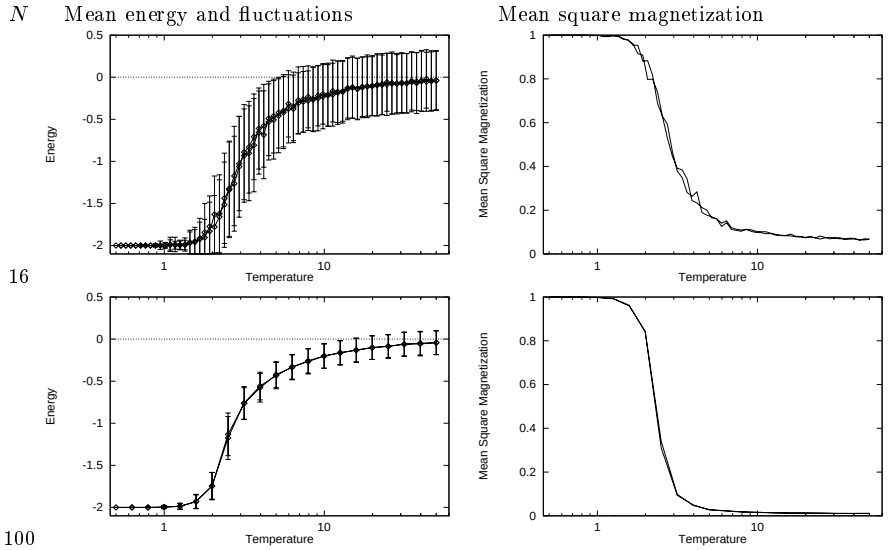


Figure 24.2. Monte Carlo simulations of rectangular Ising models with $J = 1$. Mean energy and fluctuations in energy as a function of temperature (left). Mean square magnetization as a function of temperature (right). In the top row, $N = 16$, and the bottom, $N = 100$. For even larger N, see later figures.

and the energy is expected to fluctuate around a mean of 0 with a standard deviation that goes as $1/\sqrt{N}$. At intermediate temperatures we expect the energy to rise monotonically.

Let's look at some results. In all figures temperature T is shown with $k_{\rm B} = 1$. The basic picture emerges with as few as 16 spins (figure 24.2, top). As we increase the number of spins to 100 (figure 24.2, bottom) some new details emerge. First, as expected, the fluctuations at large temperature decrease as $1/\sqrt{N}$. Second, the fluctuations at intermediate temperature become relatively *bigger*. This is the signature of a 'collective phenomenon', in this case, a phase transition. Only systems with infinite N show true phase transitions, but with $N = 100$ we are getting a hint of the critical fluctuations. Figure 24.5 shows details of the graphs for $N = 100$ and $N = 4096$. Figure 24.10 shows a sequence of typical states from the simulation of $N = 4096$ spins at a sequence of decreasing temperatures.

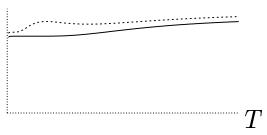


Figure 24.3. Schematic diagram to explain the meaning of a Schottky anomaly. The curve shows the heat capacity of two gases as a function of temperature. The lower curve shows a normal gas whose heat capacity is an increasing function of temperature. The upper curve has a small peak in the heat capacity, which is known as a Schottky anomaly (at least in Cambridge). The peak is produced by the gas having magnetic degrees of freedom with a finite number of accessible states.

Contrast with Schottky anomaly

A peak in the heat capacity, as a function of temperature, occurs in any system that has a finite number of energy levels, and is not in itself evidence of a phase transition. Such peaks were viewed as anomalies in classical thermodynamics, since 'normal' systems with infinite numbers of energy levels have heat capacities that are either constant functions or increasing functions of temperature. In contrast, systems with a finite number of levels produced small blips in the heat capacity graph.

Let us refresh our memory of the simplest such system, a two-level system with states $x = 0$ (energy 0) and $x = 1$ (energy ϵ). The mean energy is

$$E(\beta) = \epsilon \frac{\exp(-\beta\epsilon)}{1 + \exp(-\beta\epsilon)} = \epsilon \frac{1}{1 + \exp(\beta\epsilon)} \tag{24.22}$$

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

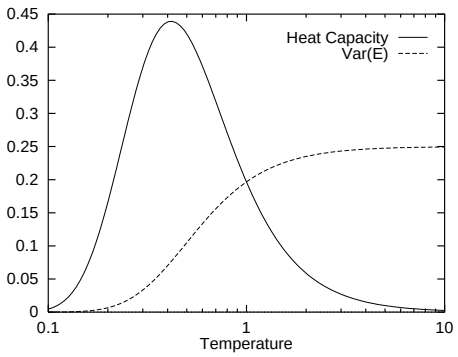


Figure 24.4. 'Schottky anomaly' — Heat capacity and fluctuations in energy as a function of temperature for a two-level system with separation $\epsilon = 1$ and $k_{\rm B} = 1$.

and the derivative with respect to β is

$$dE/d\beta = -\epsilon^2 \frac{\exp(\beta\epsilon)}{[1+\exp(\beta\epsilon)]^2}. \tag{24.23}$$

So the heat capacity is

$$C = dE/dT = -\frac{dE}{d\beta}\frac{1}{k_{\rm B}T^2} = \frac{\epsilon^2}{k_{\rm B}T^2}\frac{\exp(\beta\epsilon)}{[1+\exp(\beta\epsilon)]^2} \tag{24.24}$$

and the fluctuations in energy are given by $\mathrm{var}(E) = Ck_{\rm B}T^2 = -dE/d\beta$, which was already evaluated above (equation (24.23)). The heat capacity and fluctuations are plotted in figure 24.4. The take-home message at this point is that whilst Schottky anomalies do have a peak in the heat capacity, there is *no* peak in their fluctuations; the variance of the energy simply increases monotonically with temperature to a value which scales linearly with the number of independent spins. It is a peak in the fluctuations which is interesting. The Ising model has such a peak in its fluctuations, as can clearly be seen in figure 24.5.

Rectangular Ising model with $J = -1$

What do we expect to happen in the case $J = -1$? The ground states of an infinite system are clearly the two checkerboard patterns, and they have energy per spin -2, like the ground states of the $J = 1$ model. Can this analogy be pressed further? A moment's reflection will confirm that the two systems are equivalent to each other under a checkerboard symmetry operation. If you take an infinite $J = 1$ system in some state and flip all the spins that lie on the black squares of an infinite checkerboard, and set $J = -1$, then the energy is unchanged. (The magnetization changes, of course.) So all thermodynamic properties of the two systems are expected to be identical in the case of zero applied field. But there is a subtlety lurking here. Have you spotted it?

We are simulating finite grids with periodic boundary conditions. If the size of the grid in any direction is odd, then the checkerboard operation is no longer a symmetry operation relating $J = +1$ to $J = -1$, because the checkerboard doesn't match up at the boundaries. This means that for systems of odd size, the ground state of a system with $J = -1$ will have degeneracy greater than 2, and the energy of those ground states will not be as low as -2 per spin. So we

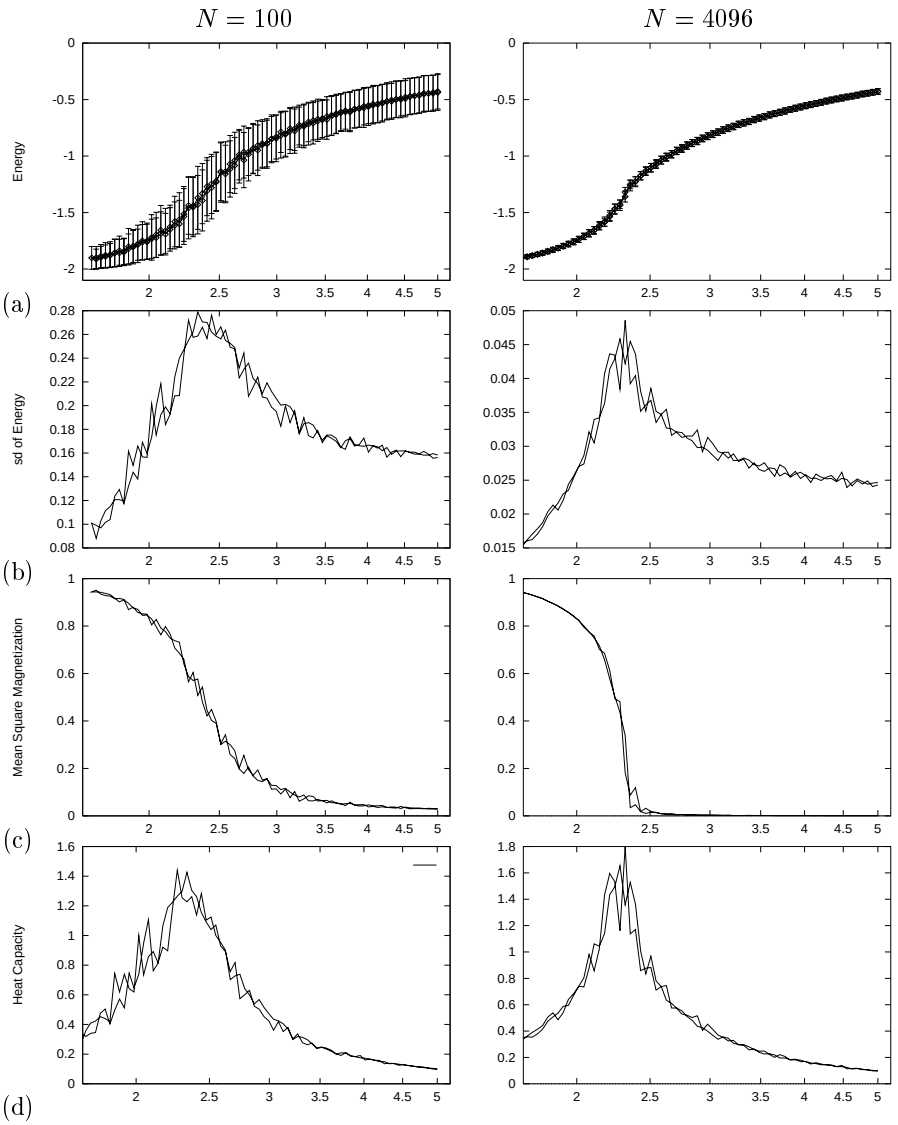


Figure 24.5. Detail of Monte Carlo simulations of rectangular Ising models with $J = 1$. (a) Mean energy and fluctuations in energy as a function of temperature. (b) Fluctuations in energy (standard deviation). (c) Mean square magnetization. (d) Heat capacity.

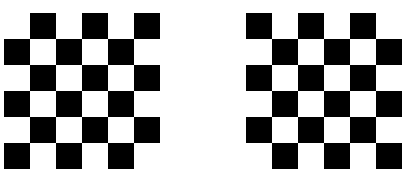


Figure 24.6. The two ground states of a rectangular Ising model with $J = -1$.

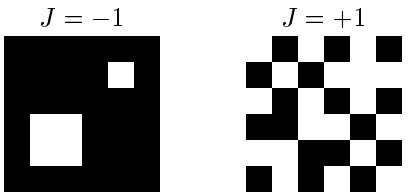


Figure 24.7. Two states of rectangular Ising models with $J = \pm 1$ that have identical energy.

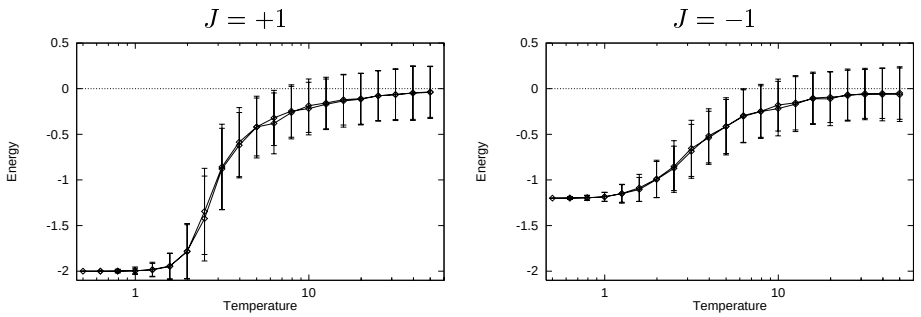


Figure 24.8. Monte Carlo simulations of rectangular Ising models with $J = \pm 1$ and $N = 25$. Mean energy and fluctuations in energy as a function of temperature. (a) $J = 1$. (b) $J = -1$.

expect qualitative differences between the cases $J = \pm 1$ in odd sized systems. These differences are expected to be most prominent for small systems. The frustrations are introduced by the boundaries, and the length of the boundary grows as the square root of the system size, so the fractional influence of this boundary-related frustration on the energy and entropy of the system will decrease as $1/\sqrt{N}$. Figure 24.8 compares the energies of the ferromagnetic and antiferromagnetic models with $N = 25$. Here, the difference is striking.

The graphs for fragments with even size are identical for $J = \pm 1$ as theoretically predicted.

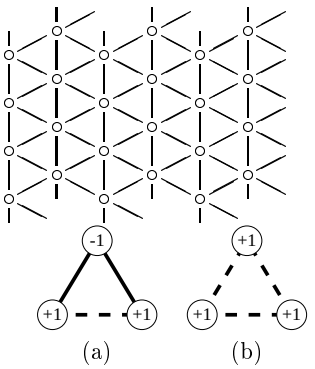
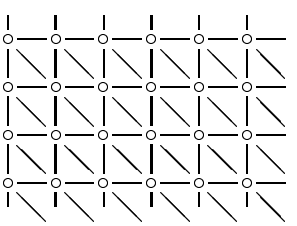


Figure 24.9. In an antiferromagnetic triangular Ising model, any three neighbouring spins are frustrated. Of the eight possible configurations of three spins, six have energy $-|J|$ (a), and two have energy $3|J|$ (b). Solid lines show 'happy' couplings which contribute $-|J|$ to the energy; dashed lines show 'unhappy' couplings which contribute $|J|$.

Triangular Ising Model

We can repeat these computations for a triangular Ising model. Do we expect the triangular Ising model with $J = \pm 1$ to show different physical properties from the rectangular Ising model? Presumably the $J = 1$ model will have broadly similar properties to its rectangular counterpart. But the case $J = -1$ is radically different from what's gone before. Think about it: there is no unfrustrated ground state; in any state, there *must* be frustrations — pairs of neighbours who have the same sign as each other. Unlike the case of the rectangular model with odd size, the frustrations are not introduced by the periodic boundary conditions. *Every set of three mutually neighbouring spins must be in a state of frustration,* as shown in figure 24.9. Thus we certainly expect different behaviour at low temperatures. In fact we might expect this system to have a non-zero entropy at absolute zero. ('Triangular model violates third law of thermodynamics!')

Let's look at some results. There are various ways to implement a periodic triangular grid. I did it as shown in the margin. Figure 24.11 shows the results for $N = 25$. The fluctuations in energy for intermediate temperatures are noticeably smaller when $J = -1$ than for $J = +1$. Figure 24.12 shows



 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

results for $N = 4096$. Note how different the results for $J = \pm 1$ are. There is no peak at all in the standard deviation of the energy in the case $J = -1$. This indicates that the antiferromagnetic system does not have a phase transition to a state with long-range order.

24.2 Direct Computation of Partition Function of Ising Models

We now examine a completely different approach to Ising models. The *transfer matrix method* is an exact and abstract approach which obtains physical properties of the model from the partition function

$$Z(\beta, \mathbf{J}, \mathbf{b}) \equiv \sum_{\mathbf{x}} \exp\left[-\beta E(\mathbf{x}; \mathbf{J}, \mathbf{b})\right] \tag{24.25}$$

where the summation is over all states $\mathbf{x}$, and the inverse temperature is $\beta = 1/T$. [As usual, Let $k_B = 1$.] The free energy is given by $F = -\frac{1}{\beta} \log Z$. The number of states is 2^N, so direct computation of the partition function is not possible for large N. To avoid enumerating all global states explicitly, we can use a clever trick similar to the sum-product algorithm discussed in chapter 20. We concentrate on models that have the form of a long thin strip with periodic boundary conditions in both directions, and we iterate along the length of our model, working out a set of *partial partition functions* at one location l in terms of partial partition functions at the previous location $l-1$. Each iteration only involves a summation over all the states at the boundary. This operation is thus exponential in the width of the strip. The final clever trick is to note that if the system is translation-invariant along its length then we only need to do *one* iteration in order to find the properties of a system of *any* length.

The computational task becomes the evaluation of an $S \times S$ matrix, where S is the number of microstates that need to be considered at the boundary, and the computation of its eigenvalues. The eigenvalue of largest magnitude gives the partition function for an infinite length thin strip.

Here is a more detailed explanation. Label the states of columns of the thin strip $s_1, s_2, \ldots, s_C$, with each s an integer from 0 to $2^W - 1$. The rth bit of s_c indicates whether the spin in row r, column c is up or down. The partition function is

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})) \tag{24.26}$$

$$= \sum_{s_1} \sum_{s_2} \cdots \sum_{s_C} \exp\left(-\beta \sum_{c=1}^{C} \mathcal{E}(s_c, s_{c+1}) \right) \tag{24.27}$$

where $\mathcal{E}(s_c, s_{c+1})$ is an appropriately defined energy, and, if we want periodic boundary conditions, s_{C+1} is defined to be s_1. One definition for $\mathcal{E}$ is:

$$\mathcal{E}(s_c, s_{c+1}) = \sum_{\substack{(m,n)\in\mathcal{N}: \\ m\in c, n\in c+1}} J x_m x_n + \tfrac{1}{2} \sum_{\substack{(m,n)\in\mathcal{N}: \\ m\in c, n\in c}} J x_m x_n + \tfrac{1}{2} \sum_{\substack{(m,n)\in\mathcal{N}: \\ m\in c+1, n\in c+1}} J x_m x_n. \tag{24.28}$$

This function has the nice property that (for the rectangular Ising model) it defines a matrix that is symmetric in its two indices s_c, s_{c+1}. The factors of

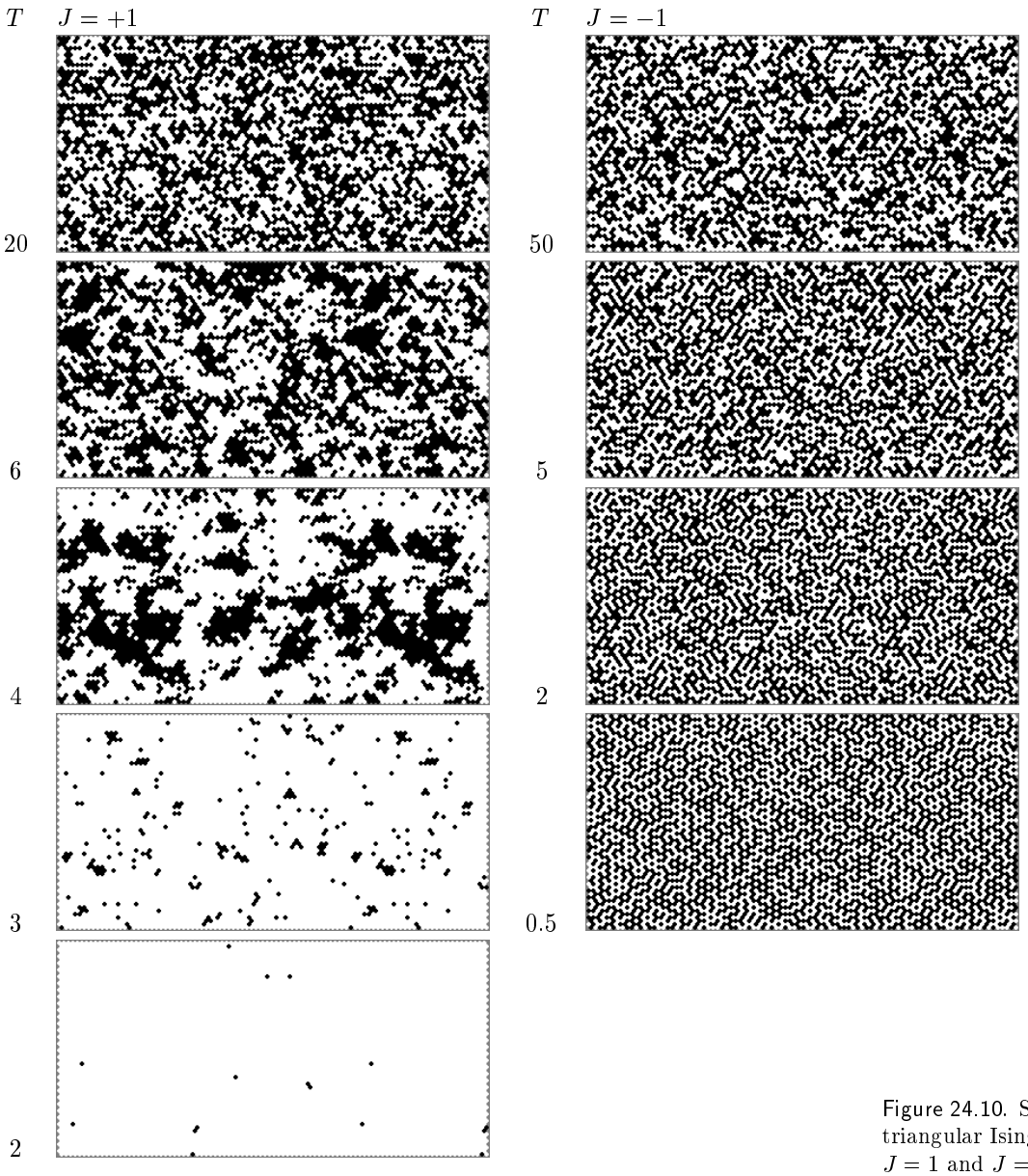


Figure 24.10. Sample states of triangular Ising models with $J = 1$ and $J = -1$.

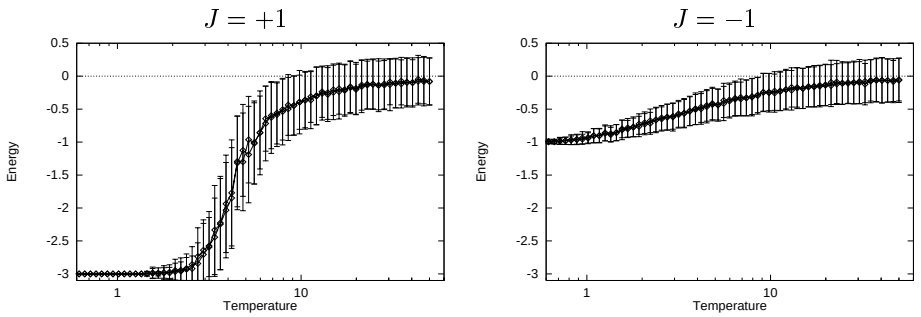


Figure 24.11. Monte Carlo simulations of triangular Ising models with $J = \pm 1$ and $N = 25$. Mean energy and fluctuations in energy as a function of temperature.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

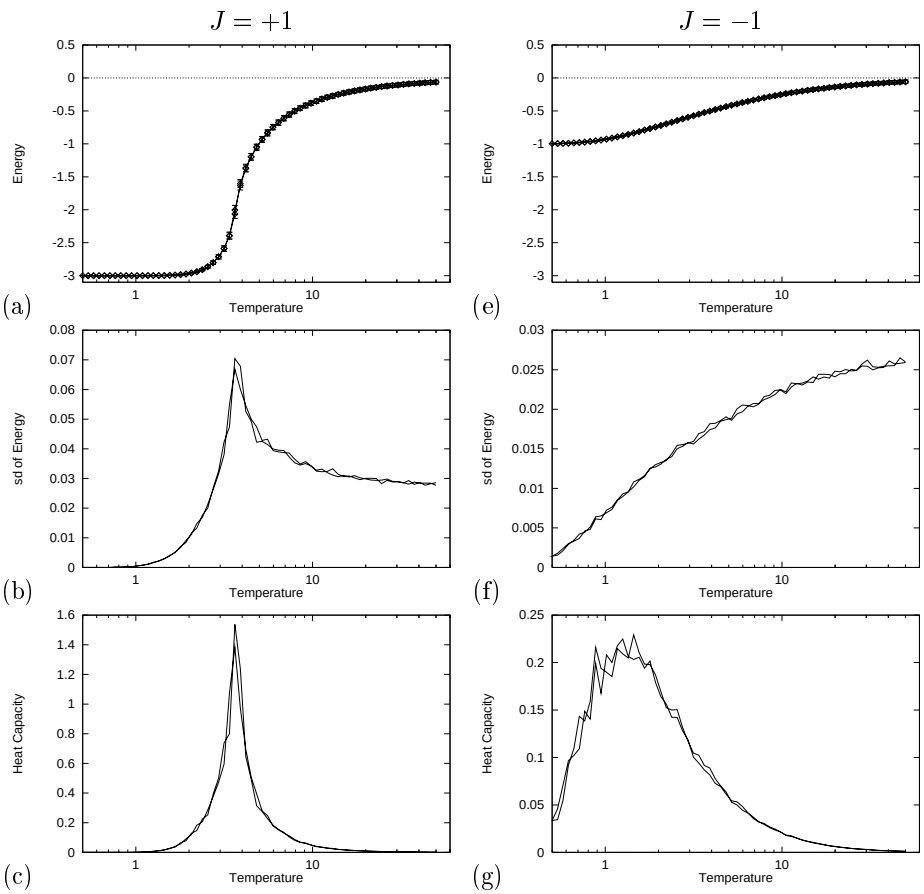


Figure 24.12. Monte Carlo simulations of triangular Ising models with $J = \pm 1$ and $N = 4096$. (a-c) $J = 1$. (e-g) $J = -1$. (a,e) Mean energy and fluctuations in energy as a function of temperature. (b,f) Fluctuations in energy (standard deviation). (c,g) Heat capacity.

1/2 are needed to prevent vertical links from being counted twice. Let us define

$$M_{ss'} = \exp\left(-\beta \mathcal{E}(s,s')\right) . \tag{24.29}$$

Then continuing from equation (24.27),

$$\begin{aligned} Z &= \sum_{s_1}\sum_{s_2}\cdots\sum_{s_C}\left[\prod_{c=1}^{C} M_{s_c,s_{c+1}}\right] && (24.30)\\ &= \text{Trace}\left[\mathbf{M}^C\right] && (24.31)\\ &= \sum_a \mu_a^C , && (24.32) \end{aligned}$$

where $\{\mu_a\}_{a=1}^{2^W}$ are the eigenvalues of $\mathbf{M}$. As the length of the strip C increases, Z becomes dominated by the largest eigenvalue $\mu_{\max}$:

$$Z \to \mu_{\max}^C . \tag{24.33}$$

So the free energy per spin in the limit of an infinite thin strip is given by:

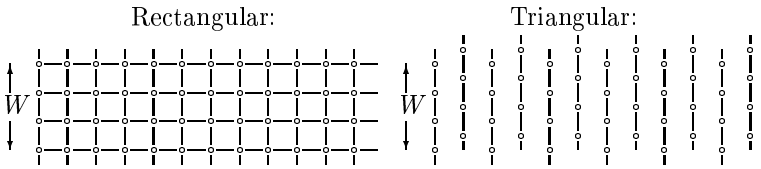
$$f = -kT \log Z/(WC) = -kTC \log \mu_{\max}/(WC) = -kT \log \mu_{\max}/W. \tag{24.34}$$

It's really neat that *all* the thermodynamic properties of a long thin strip can be obtained from just the largest eigenvalue of this matrix $\mathbf{M}$!

One of the interesting properties we can obtain from the free energy is the degeneracy of the ground state. As the temperature goes to zero, the Boltzmann distribution becomes concentrated in the ground state. If the ground state is degenerate (*i.e.*, there are multiple ground states with identical energy) then the entropy as $T \to 0$ is non-zero. We can find the entropy from the free energy using $S = -\partial F/\partial T$.

Computations

I computed the partition functions of *long thin strip* Ising models with the geometries shown below. A line between two spins indicates that they are neighbours. The strips have width W and infinite length.



As in the last section, I set the applied field H to zero and considered the two cases $J = \pm 1$ which are a ferromagnet and antiferromagnet respectively. I computed the free energy per spin, $f(\beta, J, H) = F/N$ for widths from $W = 2$ to 8 as a function of β for $H = 0$.

Computational ideas:

Only the largest eigenvalue is needed. There are several ways of getting this quantity, for example, iterative multiplication of the matrix by an initial vector. Because the matrix is all positive we know that the principal eigenvector

is all positive too (Frobenius-Perron theorem), so a reasonable initial vector is $(1, 1, \ldots, 1)$. This iterative procedure may be faster than explicit computation of all eigenvalues. I computed them all anyway, which has the advantage that we can find the free energy of finite length strips — using equation (24.32) — as well as infinite ones.

Comments on graphs:

For large temperatures all Ising models should show the same behaviour: the free energy is entropy-dominated, and the entropy per spin is $\log(2)$. The mean energy per spin goes to zero. The free energy per spin should tend to $-\log(2)/\beta$.

The entropy of the triangular antiferromagnet at absolute zero appears to be about 0.3, that is, about half its high temperature value.

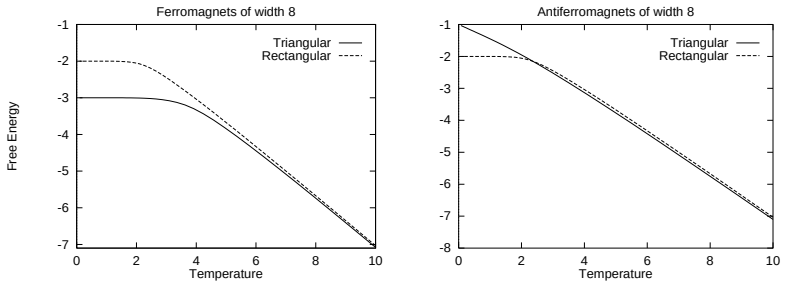


Figure 24.13. Free energy per spin of long-thin-strip Ising models. Note the non-zero gradient at $T = 0$ in the case of the triangular antiferromagnet.

The mean energy as a function of temperature is plotted in figure 24.14. It is evaluated using the identity $\langle E \rangle = -\partial \log Z / \partial \beta$.

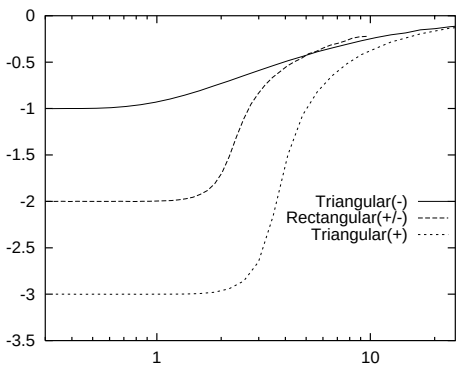


Figure 24.14. Mean energy versus temperature of long thin strip Ising models with width 8. Compare with figure 24.2.

Figure 24.16 shows the estimated heat capacity (taking raw derivatives of the mean energy) as a function of temperature for the triangular models with widths 4 and 8. Figure 24.16 shows the fluctuations in energy as a function of temperature. All of these figures should show smooth graphs; the roughness of the curves is presumably due to inaccurate eigenvalue evaluation. The nature

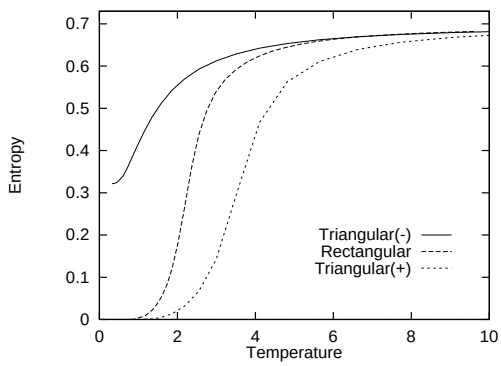


Figure 24.15. Entropies of width 8 Ising systems as a function of temperature, obtained by differentiating the free energy curves in figure 24.13. The rectangular ferromagnet and antiferromagnet have identical thermal properties. For the triangular systems, (+) denotes the ferromagnet and (−) the antiferromagnet.

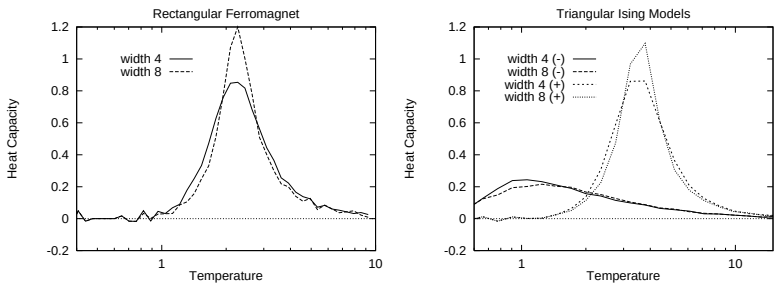


Figure 24.16. Heat Capacities of (a) Rectangular model; (b) Triangular models with different widths, (+) and (−) denoting ferromagnet and antiferromagnet. Compare with figure 24.12.

of any phase transition is not obvious, but the graphs seem compatible with the conjecture that the ferromagnet does, and the antiferromagnet does not show a phase transition.

The pictures of the free energy shown here give some insight into how we could predict the transition temperature. We can see how the two phases of the ferromagnetic systems each have simple free energies: a straight sloping line through $F = 0$, $T = 0$ for the high temperature phase, and a horizontal line for the low temperature phase. (The slope of each line shows what the entropy per spin of that phase is). The phase transition occurs roughly at

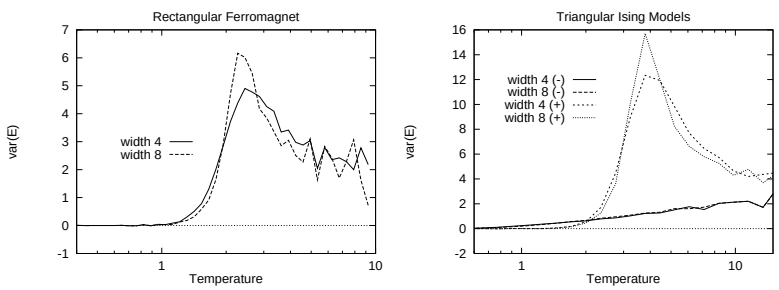


Figure 24.17. Energy variances, per spin, of (a) Rectangular model; (b) Triangular models with different widths, (+) and (−) denoting ferromagnet and antiferromagnet. Compare with figure 24.12.

the intersection of these lines. So we predict the transition temperature to be linearly related to the ground state energy.

Comparison with the Monte Carlo results

The agreement between the results of the two experiments seems very good. The two systems simulated (the long thin strip and the periodic square) are not identical. One could make detailed comparisons by finding all eigenvalues for the strip of width W and computing $\sum \lambda^W$ to get the partitition function of a $W \times W$ patch.

24.3 Problems

Exercise 24.2:^{B4} (Open question) What would be the best way to extract the entropy from the Monte Carlo simulations? What would be the best way to obtain the entropy and the heat capacity from the partition function computation?

Exercise 24.3:^{A3} An Ising model may be generalized to have a coupling J_{mn} between any spins m and n, and the value of J_{mn} could be different for each m and n.

In the special case where all the couplings are positive we know that the system has two ground states, the all up and all down states. For a more general setting of J_{mn} it is conceivable that there could be *many* ground states.

Imagine that it is required to make a spin system whose local minima are a given list of states $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \ldots, \mathbf{x}_{(N)}$. Can you think of a way of setting $\mathbf{J}$ such that the chosen states are likely to be low energy states? You are allowed to adjust all the $\{J_{mn}\}$ to whatever values you wish.

Solutions to Chapter 24's exercises

Solution to exercise 24.1 (p. 323): ...

Solution to exercise 24.3 (p. 335): This is the problem of creating a system whose stable states are a desired set of memories. See later chapters for some ideas.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

25

*Exact Monte Carlo Sampling**

The work of Propp and Wilson (1996) is of great importance for those who want to know for how long to run their Markov chain Monte Carlo simulation.

The exact sampling method depends on three ideas. First, the idea that if several Markov chains starting from different initial conditions share a single random-number generator, then their trajectories in state space may *coalesce*; and, having, coalesced, will not separate again. If all initial conditions lead to trajectories that coalesce into a single trajectory, then we can be sure that the Markov chain has 'forgotten' its initial condition.

How can we use this coalescence property to find an exact sample from the equilibrium distribution of the chain? The state of the system at the moment when complete coalescence occurs is not a valid sample from the equilibrium distribution; in the examples of figure 25.1(d,e,f), final coalescence always occurs when the state is against one of the two walls, because trajectories only merge at the walls. So sampling forward in time until coalescence occurs is not a valid method. The second idea is that we can obtain exact samples by sampling *from a time in the past, up to the present.* If coalescence has occured, the present sample is an unbiased sample from the equilibrium distribution; if not, we restart the simulation from a time further into the past, reusing the same random numbers. (Figures to come here.)

Having established that we can obtain valid samples by simulating forward from times in the past, starting in *all* possible states at those times, the third trick of Propp and Wilson, which makes the exact sampling method useful in practice, is the idea that, for some Markov chains, it may be possible to detect coalescence of all trajectories without simulating all trajectories. This property holds, for example, in the chain of figure 25.1(d,e,f), which has the property that *two trajectories never cross.* If we simply track the two trajectories starting from the leftmost and rightmost states, we will know that coalescence of all trajectories has occurred when those two trajectories coalesce. (More figures here.)

Exercise 25.1:[B2] Is there any relationship between the probability distribution of the time taken for all trajectories to coalesce, and the equilibration time of a Markov chain? Prove that there is a relationship, or find a single chain that can be realised in two different ways that have different coalescence times.

Solution to exercise 25.1 (p. 337): It is perhaps surprising that there is no direct relationship between the equilibration time and the time to coalescence.

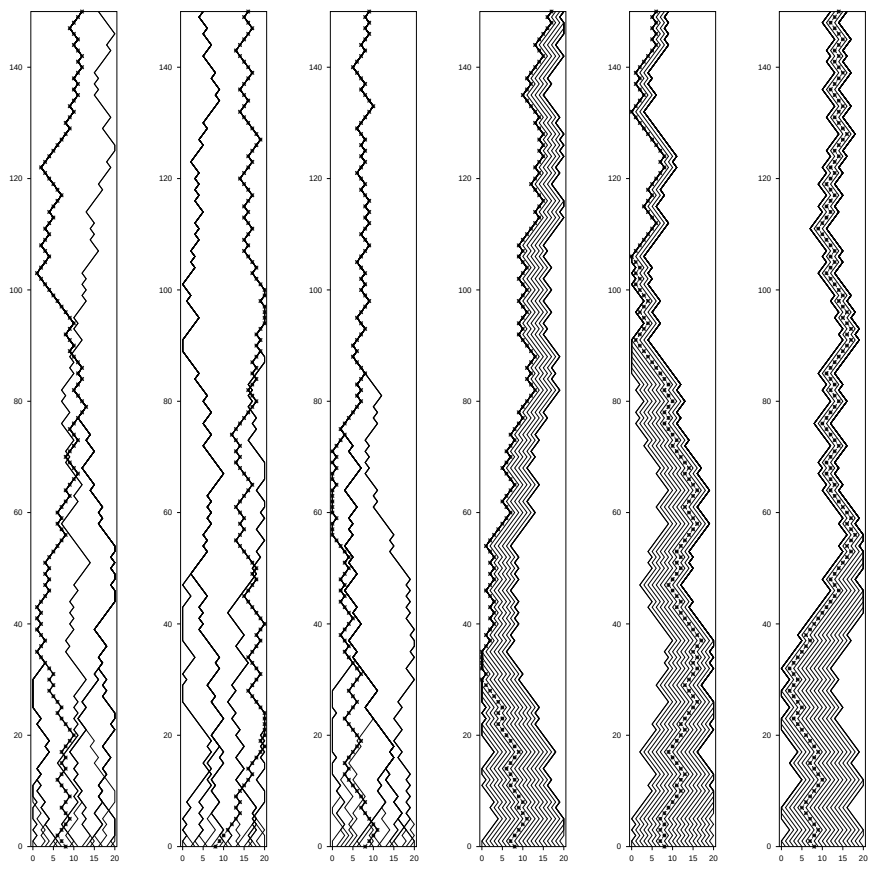


Figure 25.1. Coalescence, the idea behind the exact sampling method. Different coalescence properties are shown depending on the way each state uses the random numbers it is supplied with. (a,b,c): Three examples of coalescence for a Metropolis simulator in which the random bits that determine the proposed step depend on the current state. (d,e,f): In this simulator the random proposal is the same for all states.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

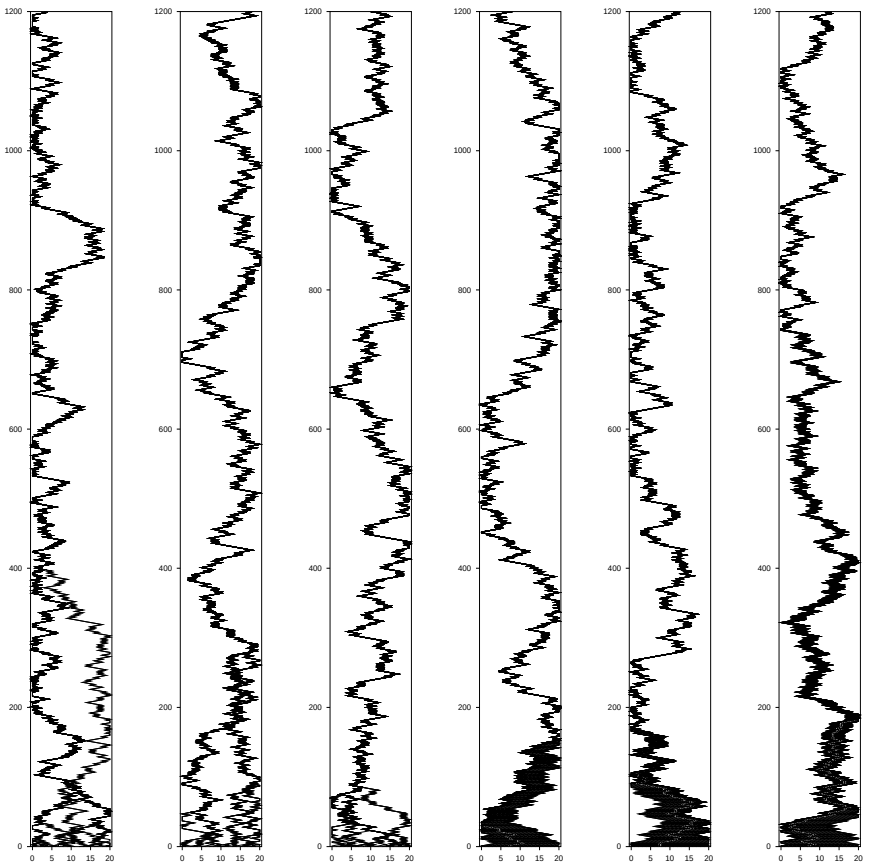


Figure 25.2. Longer time-histories of the coalescences.

A simple example that proves this is the case of the uniform distribution over the integers $\mathcal{A} = \{0, 1, 2, \ldots, 20\}$. A Markov chain that converges to this distribution in exactly one iteration is the chain for which the probability of state s_{t+1} given s_t is the uniform distribution, for all s_t. Such a chain can be coupled to a random number generator in two ways: (a) we could draw a random integer $u \in \mathcal{A}$, and set s_{t+1} equal to u regardless of s_t; or (b) we could draw a random integer $u \in \mathcal{A}$, and set s_{t+1} equal to $(s_t + u) \bmod 21$. Method (b) would produce a cohort of trajectories locked together, similar to the trajectories in figure 25.1, except that no coalescences ever occur. Thus, while the equilibration times of both methods (a) and (b) are one, the coalescence times are respectively one and infinite.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

26

Variational Methods

Variational methods are an important technique for the approximation of complicated probability distributions, having applications in statistical physics, data modelling and neural networks.

26.1 Variational free energy minimization

A well known method for approximating a complex distribution in a physical system is 'mean field theory'. Mean field theory is a special case of a general *variational free energy* approach of Feynman and Bogoliubov which we will now study. The key piece of mathematics needed to understand this method is Gibbs' inequality, which we repeat here.

Gibbs' inequality first appeared in equation (1.24); see also exercise 2.21 (p. 39).

The relative entropy *or* **Kullback-Leibler divergence** between two probability distributions $Q(x)$ and $P(x)$ that are defined over the same alphabet $\mathcal{A}_X$ is

$$D_{\mathrm{KL}}(Q||P) = \sum_x Q(x) \log \frac{Q(x)}{P(x)}. \tag{26.1}$$

The relative entropy satisfies $D_{\mathrm{KL}}(Q||P) \geq 0$ (Gibbs' inequality) with equality only if $Q{=}P$. In general $D_{\mathrm{KL}}(Q||P) \neq D_{\mathrm{KL}}(P||Q)$.

Probability distributions in statistical physics

In statistical physics one often encounters probability distributions of the form

$$P(\mathbf{x}|\beta, \mathbf{J}) = \frac{1}{Z(\beta, \mathbf{J})} \exp\left[-\beta E(\mathbf{x}; \mathbf{J})\right], \tag{26.2}$$

where for example the state vector is $\mathbf{x} \in \{-1, +1\}^N$, and $E(\mathbf{x}; \mathbf{J})$ is some energy function such as

$$E(\mathbf{x}; \mathbf{J}) = -\frac{1}{2} \sum_{m,n} J_{mn} x_m x_n - \sum_n h_n x_n. \tag{26.3}$$

The partition function is

$$Z(\beta, \mathbf{J}) \equiv \sum_{\mathbf{x}} \exp\left[-\beta E(\mathbf{x}; \mathbf{J})\right] \tag{26.4}$$

The probability distribution of equation (26.2) is complex. Not unbearably complex — we can, after all, evaluate $E(\mathbf{x}; \mathbf{J})$ for any $\mathbf{x}$ in a time polynomial

in the number of spins. But evaluating the normalizing constant $Z(\beta, \mathbf{J})$ is difficult, as we saw in chapter 23, and describing the properties of the probability distribution is also hard. Knowing the value of $E(\mathbf{x}; \mathbf{J})$ at a few arbitrary points $\mathbf{x}$, for example, gives no useful information about what the average properties of the system are.

An evaluation of $Z(\beta, \mathbf{J})$ would be particularly desirable because from the partition function we can derive all the thermodynamic properties of the system.

Variational free energy minimization is a method for *approximating* the complex distribution $P(\mathbf{x})$ by a simpler ensemble $Q(\mathbf{x}; \theta)$ that is parameterized by adjustable parameters θ. We adjust these parameters so as to get Q to best approximate P, in some sense. A by-product of this approximation is a lower bound on $Z(\beta, \mathbf{J})$.

The variational free energy

The objective function chosen to measure the quality of the approximation is the *variational free energy*

$$\beta\tilde{F}(\theta) = \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \log \frac{Q(\mathbf{x}; \theta)}{\exp\left[-\beta E(\mathbf{x}; \mathbf{J})\right]}. \tag{26.5}$$

This expression can be manipulated into a couple of interesting forms: first,

$$\begin{aligned} \beta\tilde{F}(\theta) &= \beta \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) E(\mathbf{x}; \mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \log \frac{1}{Q(\mathbf{x}; \theta)} & (26.6) \\ &\equiv \beta \left\langle E(\mathbf{x}; \mathbf{J}) \right\rangle_Q - S_Q, & (26.7) \end{aligned}$$

where $\left\langle E(\mathbf{x}; \mathbf{J}) \right\rangle_Q$ is the average of the energy function under the distribution $Q(\mathbf{x}; \theta)$, and S_Q is the entropy of the distribution $Q(\mathbf{x}; \theta)$ (we set k_{B} to one in the definition of S so that it is identical to the definition of the entropy H in part 1).

Second, we can use the definition of $P(\mathbf{x}|\beta, \mathbf{J})$ to write:

$$\begin{aligned} \beta\tilde{F}(\theta) &= \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \log \frac{Q(\mathbf{x}; \theta)}{P(\mathbf{x}|\beta, \mathbf{J})} - \log Z(\beta, \mathbf{J}) & (26.8) \\ &= D_{\mathrm{KL}}(Q||P) + \beta F, & (26.9) \end{aligned}$$

where F is the true free energy, defined by

$$\beta F \equiv -\log Z(\beta, \mathbf{J}), \tag{26.10}$$

and $D_{\mathrm{KL}}(Q||P)$ is the relative entropy between the approximating distribution $Q(\mathbf{x}; \theta)$ and the true distribution $P(\mathbf{x}|\beta, \mathbf{J})$. Thus by Gibbs' inequality, the variational free energy $\tilde{F}(\theta)$ is bounded below by F and only attains this value for $Q(\mathbf{x}; \theta) = P(\mathbf{x}|\beta, \mathbf{J})$.

Our strategy is thus to vary θ in such a way that $\beta\tilde{F}(\theta)$ is minimized. The approximating distribution then gives a simplified approximation to the true distribution that may be useful, and the value of $\beta\tilde{F}(\theta)$ will be an upper bound for βF. Equivalently, $\tilde{Z} \equiv e^{-\beta\tilde{F}(\theta)}$ is a lower bound for Z.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Can $\beta\tilde{F}$ be evaluated?

We have already agreed that the evaluation of various interesting sums over $\mathbf{x}$ is intractable. For example, the partition function

$$Z = \sum_{\mathbf{x}} \exp\left(-\beta E(\mathbf{x};\mathbf{J})\right), \tag{26.11}$$

the energy

$$\langle E \rangle_P = \frac{1}{Z} \sum_{\mathbf{x}} E(\mathbf{x};\mathbf{J}) \exp\left(-\beta E(\mathbf{x};\mathbf{J})\right), \tag{26.12}$$

and the entropy

$$S \equiv \sum_{\mathbf{x}} P(\mathbf{x}|\beta,\mathbf{J}) \log \frac{1}{P(\mathbf{x}|\beta,\mathbf{J})} \tag{26.13}$$

are all presumed to be impossible to evaluate. So why should we suppose that this objective function $\beta\tilde{F}(\theta)$, which is also defined in terms of a sum over all $\mathbf{x}$ (26.5), should be a convenient quantity to deal with? Well, for a range of interesting energy functions, and for sufficiently simple approximating distributions, the variational free energy *can* be efficiently evaluated.

26.2 Variational free energy minimization for Ising models

An example of a tractable variational free energy is given by the spin system whose energy function was given in equation (26.3), which we can approximate with a *separable* approximating distribution,

$$Q(\mathbf{x};\mathbf{a}) = \frac{1}{Z_Q} \exp\left(\sum_n a_n x_n\right). \tag{26.14}$$

The variational parameters θ of the variational free energy (26.5) are the components of the vector $\mathbf{a}$. To evaluate the variational free energy we need the entropy of this distribution,

$$S_Q = \sum_{\mathbf{x}} Q(\mathbf{x};\theta) \log \frac{1}{Q(\mathbf{x};\theta)} \tag{26.15}$$

and the mean of the energy,

$$\langle E(\mathbf{x};\mathbf{J}) \rangle_Q = \sum_{\mathbf{x}} Q(\mathbf{x};\theta) E(\mathbf{x};\mathbf{J}). \tag{26.16}$$

The entropy of the separable approximating distribution is simply the sum of the entropies of the individual spins (exercise 8.1, p. 156),

$$S_Q = \sum_n H_2^{(e)}(q_n), \tag{26.17}$$

where q_n is the probability that spin n is $+1$,

$$q_n = \frac{e^{a_n}}{e^{a_n} + e^{-a_n}} = \frac{1}{1 + \exp(-2a_n)}, \tag{26.18}$$

and

$$H_2^{(e)}(q) = q \log \frac{1}{q} + (1-q) \log \frac{1}{(1-q)}, \tag{26.19}$$

all logs being natural logarithms. The mean energy is easy to obtain because $\sum_{m,n} J_{mn} x_m x_n$ is a sum of terms each involving the product of two independent random variables. (There are no self-couplings, so $J_{mn} = 0$ when $m = n$.) If we define the mean value of x_n to be $\bar{x}_n$, which is given by

$$\bar{x}_n = \frac{e^{a_n} - e^{-a_n}}{e^{a_n} + e^{-a_n}} = \tanh(a_n) = 2q_n - 1, \tag{26.20}$$

we obtain

$$\begin{aligned} \langle E(\mathbf{x}; \mathbf{J}) \rangle_Q &= \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \left[-\frac{1}{2} \sum_{m,n} J_{mn} x_m x_n - \sum_n h_n x_n \right] & (26.21) \\ &= -\frac{1}{2} \sum_{m,n} J_{mn} \bar{x}_m \bar{x}_n - \sum_n h_n \bar{x}_n. & (26.22) \end{aligned}$$

So the variational free energy is given by

$$\beta \tilde{F}(\mathbf{a}) = \beta \langle E(\mathbf{x}; \mathbf{J}) \rangle_Q - S_Q = \beta \left(-\frac{1}{2} \sum_{m,n} J_{mn} \bar{x}_m \bar{x}_n - \sum_n h_n \bar{x}_n \right) - \sum_n H_2^{(e)}(q_n). \tag{26.23}$$

We now consider minimizing this function with respect to the variational parameters $\mathbf{a}$. If $q = 1/(1 + e^{-2a})$, the derivative of the entropy is

$$\frac{\partial}{\partial q} H_2^e(q) = \log \frac{1-q}{q} = -2a. \tag{26.24}$$

So we obtain

$$\begin{aligned} \frac{\partial}{\partial a_m} \beta \tilde{F}(\mathbf{a}) &= \beta \left[-\sum_n J_{mn} \bar{x}_n - h_m \right] \left(2 \frac{\partial q_m}{\partial a_m} \right) - \log \left(\frac{1-q_m}{q_m} \right) \left(\frac{\partial q_m}{\partial a_m} \right) \\ &= 2 \left(\frac{\partial q_m}{\partial a_m} \right) \left[-\beta \left(\sum_n J_{mn} \bar{x}_n + h_m \right) + a_m \right]. & (26.25) \end{aligned}$$

This derivative is equal to zero when

$$a_m = \beta \left(\sum_n J_{mn} \bar{x}_n + h_m \right). \tag{26.26}$$

So $\tilde{F}(\mathbf{a})$ is extremized at any point that satisfies equation (26.26) and

$$\bar{x}_n = \tanh(a_n) \tag{26.27}$$

The variational free energy $\tilde{F}(\mathbf{a})$ may be a multimodal function in which case each stationary point (maximum, minimum or saddle) will satisfy equations (26.26) and (26.27). One way of using these equations, in the case of a system with an arbitrary coupling matrix $\mathbf{J}$, is to update each parameter a_m and the corresponding value of $\bar{x}_m$ using equation (26.26), one at a time. This *asynchronous updating of the parameters* is guaranteed to decrease $\beta \tilde{F}(\mathbf{a})$.

Equations (26.26) and (26.27) may be recognized as the mean field equations for a spin system. The variational parameter a_n may be thought of as the strength of a fictitious field applied to an isolated spin n. Equation (26.27) describes the mean response of spin n, and equation (26.26) describes how the field a_m is set in response to the mean state of all the other spins.

The variational free energy derivation is a helpful viewpoint for mean field theory for two reasons.

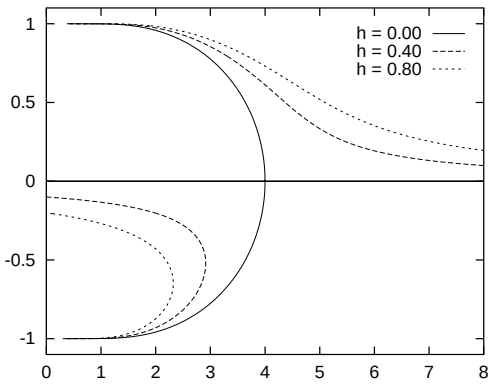


Figure 26.1. Solutions of the variational free energy extremization problem for the Ising model. Horizontal axis: temperature $T = 1/\beta$. Vertical axis: magnetization $\bar{x}$. The critical temperature found by mean field theory is $T_c^{\rm mft} = 4$.

1. This approach associates an objective function $\beta\tilde{F}$ with the mean field equations; such an objective function is useful because it can help identify alternative dynamical systems that minimize the same function;

2. The theory is readily generalized to other approximating distributions. We can imagine introducing a more complex approximation $Q(\mathbf{x};\theta)$ that might, for example, capture correlations among the spins instead of modelling them as independent. One could then evaluate the variational free energy and optimize the parameters θ of this more complex approximation. The more degrees of freedom the approximating distribution has, the tighter the bound on the free energy becomes. However, if the complexity of an approximation is increased, the evaluation of either the mean energy or the entropy typically becomes more challenging.

26.3 Example: mean field theory for the ferromagnetic Ising model

In the simple Ising model studied in chapter 24, every coupling J_{mn} is equal to J if m and n are neighbours and zero otherwise. There is an applied field $h_n = h$ that is the same for all spins. A very simple approximating distribution is one with just a single variational parameter a, which defines a separable distribution

$$Q(\mathbf{x};a) = \frac{1}{Z_Q}\exp\left(\sum_n a x_n\right) \tag{26.28}$$

in which all spins are independent and have the same probability

$$q_n = \frac{1}{1+\exp(-2a)} \tag{26.29}$$

of being up. The mean magnetization is

$$\bar{x} = \tanh(a) \tag{26.30}$$

and the equation (26.26) which defines the minimum of the variational free energy becomes

$$a = \beta\left(CJ\bar{x} + h\right), \tag{26.31}$$

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

where C is the number of couplings that a spin is involved in, $C = 4$ in the case of a rectangular two-dimensional Ising model. We can solve equations (26.30) and (26.31) for $\bar{x}$ numerically — in fact, it is easiest to vary $\bar{x}$ and solve for β — and obtain graphs of the free energy minima and maxima as a function of temperature as shown in figure 26.1. This figure shows $\bar{x}$ versus $T = 1/\beta$ for the case $C = 4, J = 1$.

When $h = 0$, there is a pitchfork bifurcation at a critical temperature $T_c^{\rm mft}$. Above this temperature, there is only one minimum in the variational free energy, at $a = 0$ and $\bar{x} = 0$; this minimum corresponds to an approximating distribution that is uniform over all states. Below the critical temperature, there are two minima corresponding to approximating distributions that are symmetry-broken, with all spins more likely to be up, or all spins more likely to be down. The state $\bar{x} = 0$ persists as a stationary point of the variational free energy, but now it is a local *maximum* of the variational free energy.

When $h > 0$, there is a global variational free energy minimum at any temperature for a positive value of $\bar{x}$, shown by the uppermost curves in figure 26.1. As long as h is not too large, that is, $h < JC$, there is also a second local minimum in the free energy, if the temperature is sufficiently small. This second minimum corresponds to a self-preserving state of magnetization in the opposite direction to the applied field. The temperature at which the second minimum appears is smaller than $T_c^{\rm mft}$, and when it appears, it is accompanied by a saddle point located between the two minima. A name given to this type of bifurcation is a saddle-node bifurcation.

The variational free energy per spin is given by

$$\beta\tilde{F} = \beta\left(-\frac{C}{2}J\bar{x}^2 - h\bar{x}\right) - H_2^{(e)}\left(\frac{\bar{x}+1}{2}\right). \tag{26.32}$$

Exercise 26.1:[A2] Sketch the variational free energy as a function of its one parameter $\bar{x}$ for a variety of values of the temperature T and the applied field h.

Figure 26.1 reproduces the key properties of the real Ising system — that, for $h = 0$, there is a critical temperature below which the system has long-range order, and that it can adopt one of two macroscopic states. However, by probing a little more we can reveal some inadequacies of the variational approximation. To start with, the critical temperature $T_c^{\rm mft}$ is 4, which is nearly a factor of 2 greater than the true critical temperature $T_c = 2.27$.

For the case $h = 0$ we can follow the trajectory of the global minimum as a function of β and find the entropy, heat capacity and fluctuations of the approximating distribution and compare them with those of a real 8×8 fragment using the matrix method of chapter 24. As shown in figure 26.2, one of the biggest differences is in the fluctuations in energy. The real system has large fluctuations near the critical temperature, whereas the approximating distribution has no correlations among its spins and thus has an energy-variance which scales simply linearly with the number of spins.

26.4 Variational methods in inference and data modelling

In statistical data modelling we are interested in the posterior probability distribution of a parameter vector $\mathbf{w}$ given data D and model assumptions $\mathcal{H}$,

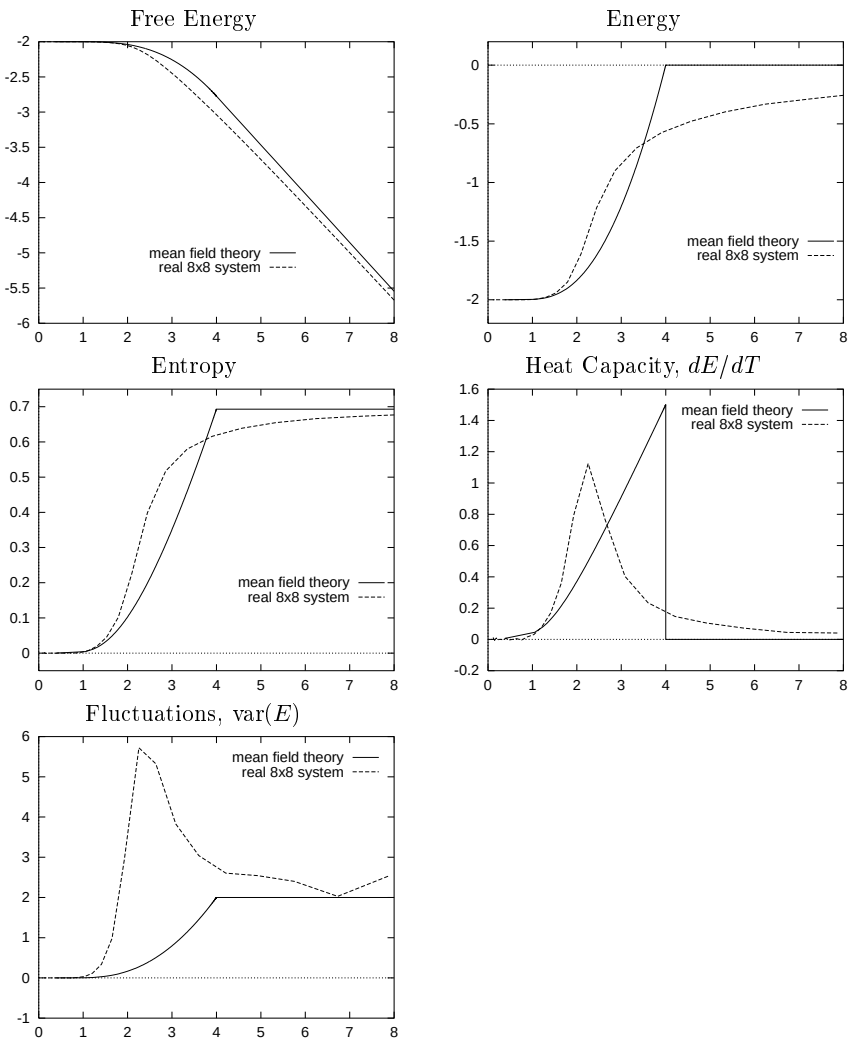


Figure 26.2. Comparison of approximating distribution's properties with those of a real 8×8 fragment. Notice that the variational free energy of the approximating distribution is indeed an upper bound on the free energy of the real system. All quantities are shown 'per spin'.

$P(\mathbf{w}|D,\mathcal{H})$.

$$P(\mathbf{w}|D,\mathcal{H}) = \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})}{P(D|\mathcal{H})}. \tag{26.33}$$

In traditional approaches to model fitting, a single parameter vector $\mathbf{w}$ is optimized to find the mode of this distribution. What is really of interest is the whole distribution. We may also be interested in its normalizing constant $P(D|\mathcal{H})$ if we wish to do model comparison. The probability distribution $P(\mathbf{w}|D,\mathcal{H})$ is often a complex distribution, so we may be interested in working in terms of an approximating *ensemble* $Q(\mathbf{w};\theta)$, which is a probability distribution over the parameters, and optimize the ensemble (by varying its own parameters θ) so that it approximates the posterior distribution of the parameters $P(\mathbf{w}|D,\mathcal{H})$ well.

One objective function we may choose to measure the quality of the approximation is the variational free energy

$$\tilde{F}(\theta) = \int d^k\mathbf{w}\, Q(\mathbf{w};\theta) \log \frac{Q(\mathbf{w};\theta)}{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})}. \tag{26.34}$$

The denominator $P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})$ is, within a multiplicative constant, equal to the posterior probability $P(\mathbf{w}|D,\mathcal{H}) = P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})/P(D|\mathcal{H})$. So the variational free energy $\tilde{F}(\theta)$ can be viewed as the sum of $-\log P(D|\mathcal{H})$ and the relative entropy between $Q(\mathbf{w};\theta)$ and $P(\mathbf{w}|D,\mathcal{H})$. $\tilde{F}(\theta)$ is bounded below by $-\log P(D|\mathcal{H})$ and only attains this value for $Q(\mathbf{w};\theta) = P(\mathbf{w}|D,\mathcal{H})$. For certain models and certain approximating distributions, this free energy, and its derivatives with respect to the ensemble's parameters, can be evaluated.

The approximation of posterior probability distributions using variational free energy minimization provides a useful approach to approximating Bayesian inference in a number of fields ranging from neural networks to the decoding of error-correcting codes (Hinton and van Camp, 1993; Hinton and Zemel, 1994; Dayan *et al.*, 1995; Neal and Hinton, 1993; MacKay, 1995c). We have given this idea the name of 'ensemble learning' in contrast with traditional learning processes in which a single parameter vector is optimized. Let us examine how ensemble learning works in the simple case of a Gaussian distribution.

26.5 The case of an unknown Gaussian: approximating the posterior distribution of μ and σ

We will fit an approximating ensemble $Q(\mu,\sigma)$ to the posterior distribution that we studied in chapter 3,

$$P(\mu,\sigma|\{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N|\mu,\sigma)P(\mu,\sigma)}{P(\{x_n\}_{n=1}^N)} \tag{26.35}$$

$$= \frac{\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N(\mu-\bar{x})^2+S}{2\sigma^2}\right) \frac{1}{\sigma_\mu}\frac{1}{\sigma}.}{P(\{x_n\}_{n=1}^N)} \tag{26.36}$$

Let us make the single assumption that the approximating ensemble is separable in the form $Q(\mu,\sigma) = Q_\mu(\mu)Q_\sigma(\sigma)$. No restrictions on the functional form of $Q_\mu(\mu)$ and $Q_\sigma(\sigma)$ are made.

We write down a variational free energy,

$$\tilde{F}(Q) = \int d\mu \, d\sigma \, Q_\mu(\mu) Q_\sigma(\sigma) \log \frac{Q_\mu(\mu) Q_\sigma(\sigma)}{P(D|\mu,\sigma) P(\mu,\sigma)}. \tag{26.37}$$

We can find the optimal separable distribution Q by considering separately the optimization of $\tilde{F}$ over $Q_\mu(\mu)$ for fixed $Q_\sigma(\sigma)$, and then the optimization of $Q_\sigma(\sigma)$ for fixed $Q_\mu(\mu)$.

Optimization of $Q_\mu(\mu)$

As a functional of $Q_\mu(\mu)$, $\tilde{F}$ is:

$$\tilde{F} = -\int d\mu \, Q_\mu(\mu) \left[\int d\sigma \, Q_\sigma(\sigma) \log P(D|\mu,\sigma) + \log[P(\mu)/Q(\mu)] \right] + (26.38)$$

$$= \int d\mu \, Q_\mu(\mu) \left[\int d\sigma \, Q_\sigma(\sigma) N\beta \frac{1}{2} (\mu - \bar{x})^2 + \log Q(\mu) \right] + \kappa', \tag{26.39}$$

where κ denotes constants that do not depend on $Q_\mu(\mu)$. The dependence on Q_σ thus collapses down to a simple dependence on the mean

$$\bar{\beta} \equiv \int d\sigma \, Q_\sigma(\sigma) 1/\sigma^2. \tag{26.40}$$

Now we can recognize the function $-N\bar{\beta}\frac{1}{2}(\mu - \bar{x})^2$ as the log of a Gaussian identical to the posterior distribution for a particular value of $\beta = \bar{\beta}$. Since a divergence $\int Q \log(Q/P)$ is minimized by setting $Q = P$, we can immediately write down the distribution $Q_\mu^{\rm opt}(\mu)$ that minimizes $\tilde{F}$ for fixed Q_σ:

$$Q_\mu^{\rm opt}(\mu) = P(\mu|D, \bar{\beta}, \mathcal{H}) = \text{Normal}(\mu; \bar{x}, \sigma^2_{\mu|D}). \tag{26.41}$$

where $\sigma^2_{\mu|D} = 1/(N\bar{\beta})$.

Optimization of $Q_\sigma(\sigma)$

As a functional of $Q_\sigma(\sigma)$, $\tilde{F}$ is (neglecting additive constants):

$$\tilde{F} = -\int d\sigma \, Q_\sigma(\sigma) \left[\int d\mu \, Q_\mu(\mu) \log P(D|\mu,\sigma) + \log[P(\sigma)/Q_\sigma(\sigma)] \right] \tag{26.42}$$

$$= \int d\sigma \, Q_\sigma(\sigma) \left[(N\sigma^2_{\mu|D} + S)\beta/2 - \left(\tfrac{N}{2} - 1\right) \log\beta + \log Q_\sigma(\sigma) \right] \tag{26.43}$$

where the integral over μ is performed assuming $Q_\mu(\mu) = Q_\mu^{\rm opt}(\mu)$. Here, the β-dependent expression in the brackets can be recognized as the log of a gamma distribution over β — see equation (21.2) — giving as the distribution that minimizes $\tilde{F}$ for fixed Q_μ:

$$Q_\sigma^{\rm opt}(\beta) = \Gamma(\beta; b', c'), \tag{26.44}$$

with

$$\frac{1}{b'} = \frac{1}{2}(N\sigma^2_{\mu|D} + S) \quad \text{and} \quad c' = \frac{N}{2}. \tag{26.45}$$

In figure 26.3, these two update rules (26.41,26.44) are applied alternately, starting from an arbitrary initial condition. The algorithm converges to the optimal approximating ensemble in a few iterations.

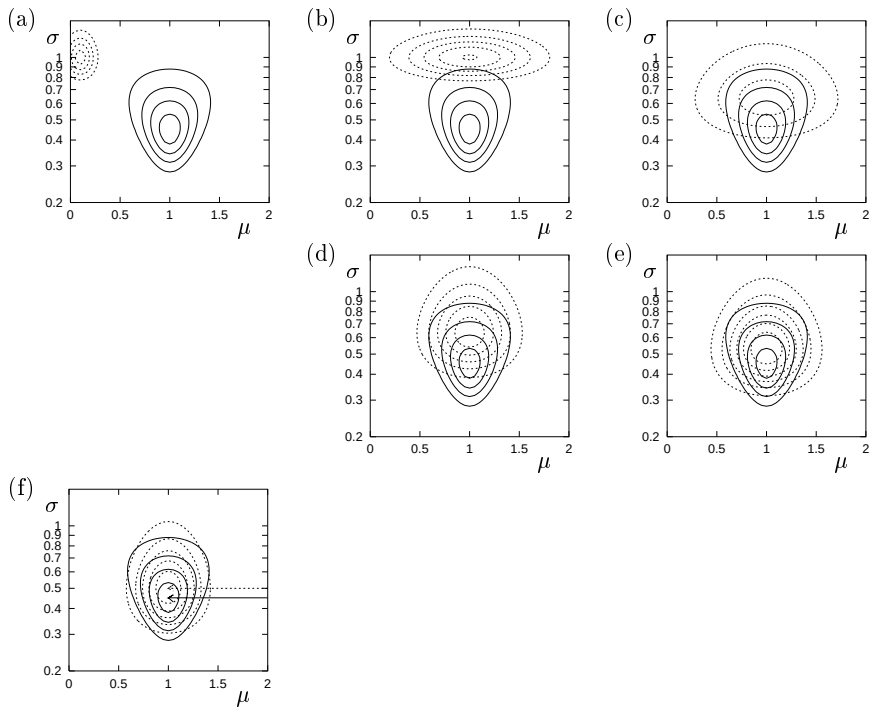


Figure 26.3. Optimization of an approximating distribution. The posterior distribution $P(\mu, \sigma|\{x_n\})$, which is the same as that in figure 21.1, is shown by solid contours. (a) Initial condition. The approximating distribution $Q(\mu, \sigma)$ (dotted contours) is an arbitrary separable distribution. (b) Q_μ has been updated, using equation (26.41). (c) Q_σ has been updated, using equation (26.44). (d) Q_μ updated again. (e) Q_σ updated again. (f) Converged approximation (after 15 iterations). The arrows point to the peaks of the two distribution, which are at $\sigma_N = 0.45$ (for P) and $\sigma_{N-1} = 0.5$ (for Q).

Direct solution for the joint optimum $Q_\mu(\mu)Q_\sigma(\sigma)$

In this problem, we do not need to resort to computation to find the optimal approximating ensemble. Equations (26.41) and (26.44) define the optimum implicitly. We must simultaneously have $\sigma^2_{\mu|D} = 1/(N\bar{\beta})$, and $\bar{\beta} = b'c'$. The solution is:

$$1/\bar{\beta} = S/(N-1). \tag{26.46}$$

This is similar to the true posterior distribution of σ, which is a gamma distribution with $c' = \frac{N-1}{2}$ and $1/b' = 1/b + S/2$. This true posterior also has a mean value of β satisfying $1/\bar{\beta} = S/(N-1)$; the only difference is that the approximating distribution's parameter c' is too large by $1/2$. The approximations given by variational free energy minimization always tend to be more compact than the true distribution.

In conclusion, ensemble learning gives an approximation to the posterior that agrees nicely with the conventional estimators. The approximate posterior distribution over β is a gamma distribution with mean $\bar{\beta}$ corresponding to a variance of $\sigma^2 = S/(N-1) = \sigma^2_{N-1}$. And the approximate posterior distribution over μ is a Gaussian with mean $\bar{x}$ and standard deviation $\sigma_{N-1}/\sqrt{N}$.

26.6 Exercises

Exercise 26.2:[A2] This exercise explores the assertion, made above, that the approximations given by variational free energy minimization always tend to be more compact than the true distribution. Consider a two dimensional Gaussian distribution $P(\mathbf{x})$ with axes aligned with the directions $\mathbf{e}^{(1)} =$

$(1,1)$ and $\mathbf{e}^{(2)} = (1,-1)$. Let the variances in these two directions be σ_1^2 and σ_2^2. What is the optimal variance if this distribution is approximated by a *spherical* Gaussian with variance σ_Q^2, optimized by variational free energy minimization? If we instead optimized the objective function

$$G = \int d\mathbf{x}\, P(\mathbf{x}) \log_2 \frac{P(\mathbf{x})}{Q(\mathbf{x};\sigma^2)}, \tag{26.47}$$

what would be the optimal value of σ^2? Sketch a contour of the true distribution $P(\mathbf{x})$ and the two approximating distributions in the case $\sigma_1/\sigma_2 = 10$.

[Note that in general it is not possible to evaluate the objective function G, because integrals under the true distribution $P(\mathbf{x})$ are usually intractable.]

Exercise 26.3:^{A2} What do you think of the idea of using a variational method to optimize an approximating distribution Q which we then use as a proposal density for importance sampling?

Solutions to Chapter 26's exercises

Solution to exercise 26.2 (p. 350): This problem rests on knowing the relative entropy between two one–dimensional Gaussian distributions:

$$\begin{aligned}
&\int dx\, \mathrm{Normal}(x;0,\sigma_Q) \log \frac{\mathrm{Normal}(x;0,\sigma_Q)}{\mathrm{Normal}(x;0,\sigma_P)} \\
&= \int dx\, \mathrm{Normal}(x;0,\sigma_Q) \left[\log \frac{\sigma_P}{\sigma_Q} - \frac{1}{2} x^2 \left(\frac{1}{\sigma_Q^2} - \frac{1}{\sigma_P^2} \right) \right] \qquad (26.48) \\
&= \frac{1}{2} \left(\log \frac{\sigma_P^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_P^2} \right). \qquad (26.49)
\end{aligned}$$

So, if we approximate P, whose variances are σ_1^2 and σ_2^2, by Q, whose variances are both σ_Q^2, we find

$$F(\sigma_Q^2) = \frac{1}{2} \left(\log \frac{\sigma_1^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_1^2} + \log \frac{\sigma_2^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_2^2} \right); \qquad (26.50)$$

differentiating,

$$\frac{d}{d \log(\sigma_Q^2)} F = \frac{1}{2} \left(-2 + \frac{\sigma_Q^2}{\sigma_1^2} + \frac{\sigma_Q^2}{\sigma_2^2} \right), \qquad (26.51)$$

which is zero when

$$\frac{1}{\sigma_Q^2} = \frac{1}{2} \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right). \qquad (26.52)$$

Thus we set the approximating distribution's inverse variance to the mean inverse variance of the target distribution P.

In the case $\sigma_1 = 10$ and $\sigma_2 = 1$, we obtain $\sigma_Q \simeq \sqrt{2}$, which is just a factor of $\sqrt{2}$ larger than σ_2, pretty much *independent* of the value of the larger standard deviation σ_1. *Variational free energy minimization typically leads to approximating distributions whose length scales match the shortest length scale of the target distribution.* The approximating distribution might be viewed as *too compact*.

In contrast, if we use the objective function G then we find:

$$G(\sigma_Q^2) = \frac{1}{2} \left(\log \sigma_Q^2 + \frac{\sigma_1^2}{\sigma_Q^2} + \log \sigma_Q^2 + \frac{\sigma_2^2}{\sigma_Q^2} \right) + \text{constant}, \qquad (26.53)$$

where the constant depends on σ_1 and σ_2 only. Differentiating,

$$\frac{d}{d \log \sigma_Q^2} G = \frac{1}{2} \left(2 - \frac{\sigma_1^2}{\sigma_Q^2} + \frac{\sigma_2^2}{\sigma_Q^2} \right), \qquad (26.54)$$

which is zero when

$$\sigma_Q^2 = \frac{1}{2} \left(\sigma_1^2 + \sigma_2^2 \right). \qquad (26.55)$$

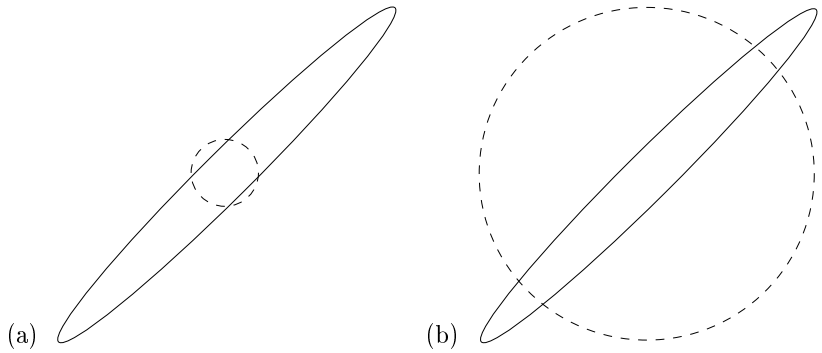


Figure 26.4. Two separable Gaussian approximations (dotted lines) to a bivariate Gaussian distribution (solid line). (a) The approximation that minimizes the variational free energy. (b) The approximation that minimizes the objective function G. In each figure, the lines show the contours at which $\mathbf{x}\mathbf{A}\mathbf{x} = 1$, where $\mathbf{A}$ is the inverse covariance matrix of the Gaussian.

Thus we set the approximating distribution's variance to the mean variance of the target distribution P.

In the case $\sigma_1 = 10$ and $\sigma_2 = 1$, we obtain $\sigma_Q \simeq 10/\sqrt{2}$, which is just a factor of $\sqrt{2}$ smaller than σ_1, independent of the value of σ_2.

The two approximations are shown to scale in figure 26.4.

Solution to exercise 26.3 (p. 351): The best possible variational approximation is of course the target distribution P. Assuming that this is not possible, a good variational approximation is more compact than the true distribution. In contrast, a good sampler is more heavy tailed than the true distribution. An over–compact distribution would make for a lousy sampler with a large variance.

27

Bayesian Inference and Sampling Theory

There are two schools of statistics. Sampling theorists concentrate on having methods guaranteed to work most of the time, given minimal assumptions. Bayesians try to make inferences that take into account all available information and answer the question that is of interest given the particular data set.

Sampling theory is the widely used approach to statistics, and most papers in most journals report their experiments using quantities like confidence intervals and p–values. A p–value is the probability, given a null hypothesis for the probability distribution of the data, that the outcome would be as extreme as, or more extreme than, the observed outcome. The untrained reader — and perhaps, more worryingly, the author — usually interprets such a p–value as if it is a Bayesian probability (for example, the posterior probability of the null hypothesis), an interpretation that both sampling theorists and Bayesians would agree is incorrect.

In this chapter we study a couple of simple inference problems in order to compare these two approaches to statistics.

While in some cases, the answers from a Bayesian approach and from sampling theory are very similar, we can also find cases where there are significant differences.

27.1 A medical example

> We are trying to reduce the incidence of an unpleasant disease called *microsoftus*. Two vaccinations, A and B, are tested on a group of volunteers. Vaccination B is a control treatment, that is, a placebo treatment with no active ingredients. Of the 40 subjects, 30 are randomly assigned to have treatment A and the other 10 are given the control treatment B. We observe the subjects for one year after their vaccinations. Of the 30 in group A, one contracts *microsoftus*. Of the 10 in group B, three contract *microsoftus*.
>
> Is treatment A better than treatment B?
>
> Is it a good idea to treat everyone in Tanzania with vaccination A if the cost of the vaccination is 1 unit, and the cost of treating someone after they get the disease is 100 units? (The 'unit' here could be a unit measuring human suffering or a unit measuring financial cost, or a combination of both.)

28

Further exercises on inference

I've been asked to include some exercises without worked solutions. Here's a start.

Exercise 28.1:[A2] Define the *relative entropy* or *Kullback–Leibler divergence* between two probability distributions P and Q, and state Gibbs's inequality.

Consider the problem of approximating a joint distribution $P(x, y)$ by a separable distribution $Q(x, y) = Q_X(x)Q_Y(y)$. Show that if the objective function for this approximation is

$$G(Q_X, Q_Y) = \sum_{x,y} P(x, y) \log_2 \frac{P(x, y)}{Q_X(x)Q_Y(y)}$$

that the minimal value of G is achieved when Q_X and Q_Y are equal to the marginal distributions over x and y.

Now consider the alternative objective function

$$F(Q_X, Q_Y) = \sum_{x,y} Q_X(x)Q_Y(y) \log_2 \frac{Q_X(x)Q_Y(y)}{P(x, y)};$$

the probability distribution $P(x, y)$ defined by

$P(x,y)$		x			
		1	2	3	4
y	1	1/8	1/8	0	0
	2	1/8	1/8	0	0
	3	0	0	1/4	0
	4	0	0	0	1/4

is to be approximated by a separable distribution $Q(x, y) = Q_X(x)Q_Y(y)$. State the value of $F(Q_X, Q_Y)$ if Q_X and Q_Y are set to the marginal distributions over x and y.

Show that $F(Q_X, Q_Y)$ has three distinct minima, identify those minima, and evaluate F at each of them.

Part V

Neural networks

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

29

Introduction to neural networks

The field of neural networks is broad and interdisciplinary. Three motivations underlie work on neural networks.

Biology. The task of understanding how the brain works is one of the outstanding unsolved problems in science. Some neural network models are intended to shed light on the way in which 'neural computation' is performed by brains.

Engineering. Many researchers would like to create machines that can 'learn', perform 'pattern recognition' or 'discover patterns in data'.

Complex systems. A third motivation for being interested in neural networks is that they are complex adaptive systems whose properties are interesting in their own right.

I should emphasize several points at the outset.

- This course is only going to give a taste of this field. There are many interesting neural network models which we will not have time to touch on.
- The models that we discuss are not intended to be faithful models of biological systems. If they are at all relevant to biology, their relevance is on an abstract level.
- I will describe some neural network methods that are widely used in nonlinear data modelling, but I will not be able to give a full description of the state of the art in neural network modelling. If you wish to solve real problems with neural networks, please read the relevant papers.

29.1 Memories

In the next few chapters we will meet several neural network models which come with simple 'learning algorithms' which make them function as memories. Perhaps we should dwell for a moment on the conventional idea of memory in digital computation. A memory (a string of 5000 bits describing the name of a person and an image of their face, say) is stored in a digital computer at an address. To retrieve the memory you have to know the address. The address has nothing to do with the memory itself. Notice the properties that this scheme does *not* have:

1. Address–based memory is *not* associative. Imagine you know half of a memory, say someone's face, and you would like to recall their name. If your memory is address–based then you can't get at that memory without knowing the address. [Computer scientists have devoted effort to content–addressable memories, but content–addressability does not come naturally. It has to be added on.]

2. Address–based memory is *not* robust or fault–tolerant. If a one–bit mistake is made in specifying the *address* then a completely different memory will be retrieved. If one bit of a *memory* is flipped then whenever that memory is retrieved the error will be present. Of course, in all modern computers, error correcting codes are used in the memory, so that small numbers of errors can be detected and corrected. But this error–tolerance is not an intrinsic property of the memory system. If minor damage occurs to certain hardware that implements memory retrieval, it is likely that all functionality will be catastrophically lost.

   It is interesting to ask whether there are models of computation and memory that have in–built error correcting capabilities.

3. Address–based memory is non–distributed. In a serial computer that is accessing a particular memory, only a tiny fraction of the devices participate in the memory recall: the CPU and the circuits that are storing the required byte. All the other millions of devices in the machine are sitting idle.

   Is is interesting to ask whether there are models of truly parallel computation, in which multiple devices participate in all computations. [Present–day parallel computers scarcely differ from serial computers from this point of view. Memory retrieval works in just the same way, and control of the computation process resides in CPUs. There are just a few more CPUs.]

Biological memory systems are completely different.

1. Biological memory is associative. Memory recall is *content addressable*. Memories are apparently recalled spontaneously, not just at the request of some CPU.

2. Biological memory recall is error–tolerant and robust.

   - Errors in the cues for memory recall can be corrected. An old example asks you to recall 'an American actor who is a politician and very intelligent'. Most people think of ex–president Reagan — even though one of the cues contains an error.
   - Hardware faults can also be tolerated. Our brains are noisy lumps of meat that are alive and are in a continual state of change, with cells being damaged or dying by natural processes and through alcohol consumption. While the cells in our brains and the proteins in our cells are continually changing, many of our memories persist unaffected.

3. Biological memory is parallel and distributed — not distributed throughout the whole brain: there does appear to be functional specialization — but in the parts of the brain where memories are stored, it seems many neurons participate in the storage of multiple memories.

These properties of biological memory systems motivate the study of 'artificial neural networks' — parallel distributed computational systems consisting of many interacting simple elements. The hope is that these model systems might give some hints as to how neural computation is achieved in real biological neural networks.

29.2 Terminology

Each time we describe a neural network algorithm we will typically specify three things.

Architecture. The architecture specifies what variables are involved in the model and their topological relationships — for example, the variables involved in a neural net may be the *weights* of the connections between the neurons, along with the *activities* of the neurons.

Activity rule. Most neural network models have short time–scale dynamics that are defined by local rules for the way that the **activities** of the neurons change in response to each other. Typically the activity rule depends on the **weights** (the parameters) in the network.

Learning rule. The learning rule specifies the way in which the neural network's **weights** change with time. This learning is usually viewed as taking place on a longer time scale than the time scale of the dynamics under the activity rule. Usually the learning rule will depend on the **activities** of the neurons. It may also depend on the values of **target** values supplied by a **teacher** and on the current value of the weights.

Where do these rules come from? Often, activity rules and learning rules are invented by imaginative researchers. Alternatively, activity rules and learning rules may be **derived** from carefully chosen **objective functions**.

Neural network algorithms can be roughly divided into two classes.

Supervised neural networks are given data in the form of *inputs* and *targets*, the targets being a specification by a *teacher* of what the neural network's response to the input should be.

Unsupervised neural networks are given data in an undivided form — simply a set of examples $\{\mathbf{x}\}$. The learning algorithms of some unsupervised neural networks are intended simply to memorize these data in such a way that the examples can be recalled in the future. Other algorithms are intended to 'generalize', to discover 'patterns' in the data or extract the underlying 'features' from them.

Some unsupervised algorithms are able to make predictions — for example, some algorithms can 'fill in' missing variables in an example $\mathbf{x}$ — and so can also be viewed as supervised networks.

30

The single neuron as a classifier

30.1 The single neuron

We will study a single neuron for two reasons. First, other neural network models are built out of single neurons, so it is good to understand them in detail. And second, a single neuron is itself capable of 'learning' — indeed, various standard statistical methods can be viewed in terms of single neurons — so this model will serve as a first example of a *supervised neural network*.

Definition of a single neuron

We will start by defining the architecture and the activity rule of a single neuron, and we will then derive a learning rule.

Architecture. A single neuron has a number I of *inputs* x_i and one *output* which we will here call y. (See figure 30.1.) Associated with each input is a weight w_i ($i = 1, \ldots, I$). There may be an additional parameter w_0 of the neuron called a *bias* which we may view as being the weight associated with an input x_0 which is permanently set to 1. The single neuron is a *feedforward* device — the connections are directed from the inputs to the output of the neuron.

Activity rule. The activity rule has two steps.

1. First, in response to the imposed inputs $\mathbf{x}$, we compute the *activation* of the neuron,

$$a = \sum_i w_i x_i, \tag{30.1}$$

where the sum is over $i = 0, \ldots, I$ if there is a bias and $i = 1, \ldots, I$ otherwise.

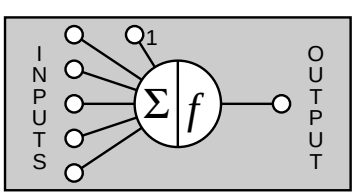


Figure 30.1. A single neuron

2. Second, the *output* y is set as a function $f(a)$ of the activation. The output is also called the *activity* of the neuron, not to be confused with the activation a. There are several possible *activation functions*.

   (a) Deterministic activation functions:

      i. Linear.

$$y(a) = a \tag{30.2}$$

      ii. Sigmoid (logistic function).

$$y(a) = \frac{1}{1+e^{-a}} \qquad (y \in (0,1)) \tag{30.3}$$

      iii. Sigmoid (tanh).

$$y(a) = \tanh(a) \qquad (y \in (-1,1)) \tag{30.4}$$

      iv. Threshold function.

$$y(a) = \Theta(a) \equiv \begin{cases} 1 & a > 0 \\ -1 & a \leq 0 \end{cases} \tag{30.5}$$

   (b) Stochastic activation functions: y is stochastically selected from ± 1.

      i. Heat bath.

$$y(a) = \begin{cases} 1 & \text{with probability } \frac{1}{1+e^{-a}} \\ -1 & \text{otherwise} \end{cases} \tag{30.6}$$

      ii. The Metropolis rule produces the output in a way that depends on the previous output state y: we compute $\Delta = ay$.

$$\begin{array}{l} \text{If } \Delta \leq 0, \text{ flip } y \text{ to the other state} \\ \text{Else flip } y \text{ to the other state with probability } e^{-\Delta}. \end{array} \tag{30.7}$$

30.2 Basic neural network concepts

A neural network implements a function $y(\mathbf{x}; \mathbf{w})$; the 'output' of the network, y, is a nonlinear function of the 'inputs' $\mathbf{x}$; this function is parameterized by 'weights' $\mathbf{w}$.

We will study a single neuron which produces an output between 0 and 1 as the following function of $\mathbf{x}$:

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}} \tag{30.8}$$

Exercise 30.1:[A1] In what contexts have we encountered the function $y(\mathbf{x}; \mathbf{w}) = 1/(1+e^{-\mathbf{w}\cdot\mathbf{x}})$ already?

Motivations for the linear logistic function

In section 11.2 we studied 'the best detection of pulses', assuming that one of two signals $\mathbf{x}_0$ and $\mathbf{x}_1$ had been transmitted over a Gaussian channel with variance–covariance matrix $\mathbf{A}^{-1}$. We found that the probability that the source signal was $s = 1$ rather than $s = 0$, given the received signal $\mathbf{y}$, was

$$P(s = 1|\mathbf{y}) = \frac{1}{1 + \exp(-a(\mathbf{y}))}, \tag{30.9}$$

where $a(\mathbf{y})$ was a linear function of the received vector,

$$a(\mathbf{y}) = \mathbf{w}^{\mathsf{T}}\mathbf{y} + \theta, \tag{30.10}$$

with $\mathbf{w} \equiv \mathbf{A}(\mathbf{x}_1 - \mathbf{x}_0)$.

The linear logistic function can be motivated in several other ways — see the exercises.

Input space and weight space

For convenience let us study the case where the input vector $\mathbf{x}$ and the parameter vector $\mathbf{w}$ are both two-dimensional: $\mathbf{x} = (x_1, x_2)$, $\mathbf{w} = (w_1, w_2)$. Then we can spell out the function performed by the neuron thus:

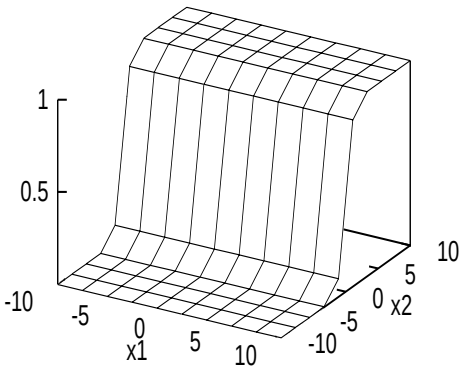
$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}. \tag{30.11}$$

Figure 30.2 shows the output of the neuron as a function of the input vector, for $\mathbf{w} = (0, 2)$. The two horizontal axes of this figure are the inputs x_1 and x_2, with the output y on the vertical axis. Notice that on any line perpendicular to $\mathbf{w}$, the output is constant; and along a line in the direction of $\mathbf{w}$, the output is a 'sigmoid' function.

We now introduce the idea of *weight space*, that is, the parameter space of the network. In this case, there are two parameters w_1 and w_2, so the weight space is two dimensional. This weight space is shown in figure 30.3. For a selection of different values of the parameter vector $\mathbf{w}$, smaller inset figures show the function of $\mathbf{x}$ performed by the network when $\mathbf{w}$ is set to those given values. Each of these smaller figures is equivalent to figure 30.2. Thus each *point* in $\mathbf{w}$ space corresponds to a *function* of $\mathbf{x}$. Notice that the gain of the sigmoid function (the gradient of the ramp) increases as the magnitude of $\mathbf{w}$ increases.

Now, the central idea of supervised neural networks is this. Given *examples* of a relationship between an input vector $\mathbf{x}$, and a target t, we hope to make the neural network 'learn' a model of the relationship between $\mathbf{x}$ and t. A successfully trained network will, for any given $\mathbf{x}$, give an output y that is close (in some sense) to the target value t. '*Training*' the network involves searching in the weight space of the network for a value of $\mathbf{w}$ that produces a function that fits the provided training data well.

Typically an '*objective function*' or '*error function*' is defined as a function of $\mathbf{w}$ to measure how well the network, with its weights set to $\mathbf{w}$, solves the task. The objective function is a sum of terms, one for each input/target pair $\{\mathbf{x}, t\}$, measuring how close the output $y(\mathbf{x}; \mathbf{w})$ is to the target t. The training process is an exercise in *function minimization* — *i.e.*, adjusting $\mathbf{w}$ in such



$\mathbf{w} = (0, 2)$

Figure 30.2. Output of a simple neural network as a function of its input

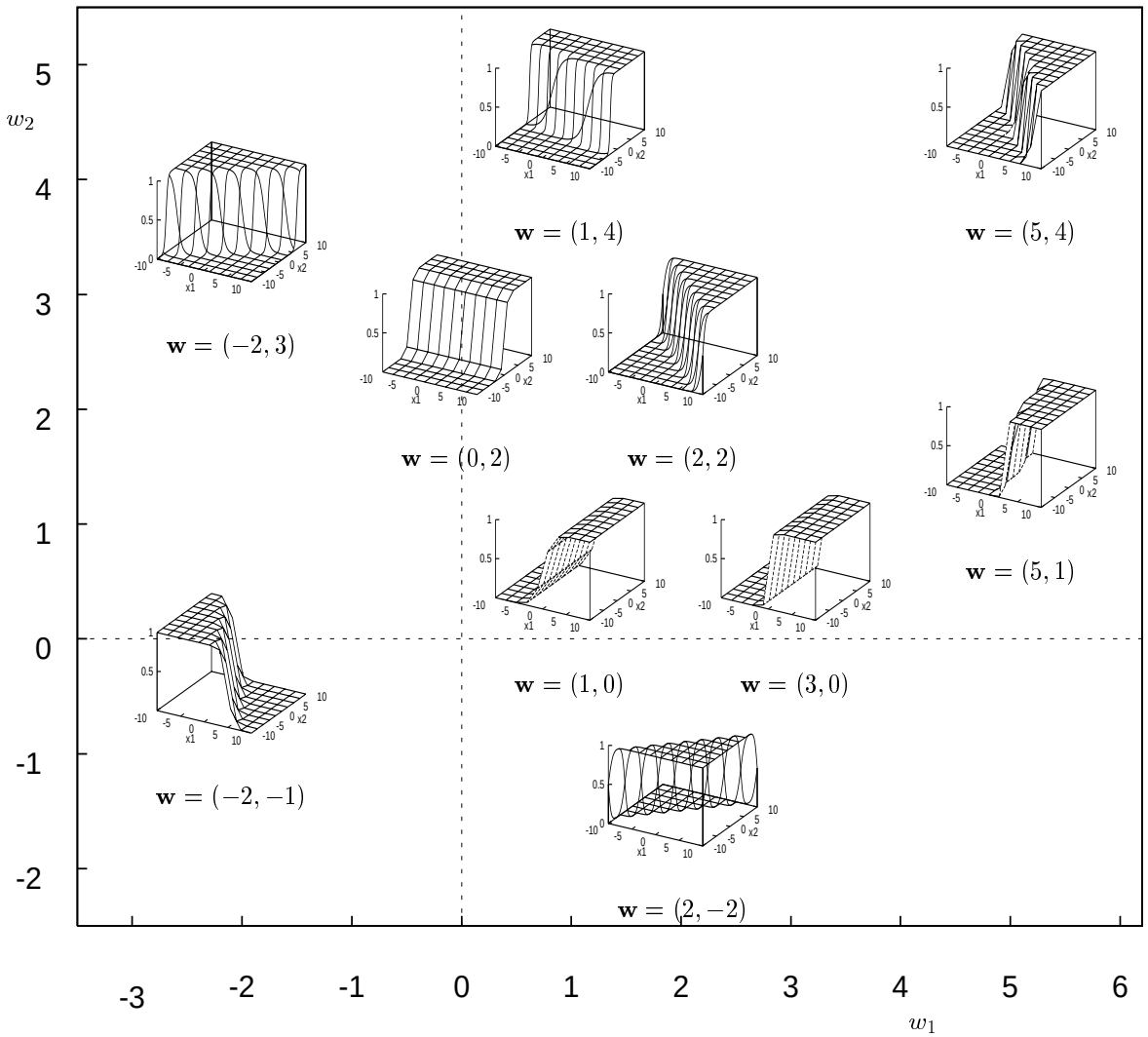


Figure 30.3. Weight space.

a way as to find a $\mathbf{w}$ that minimizes the objective function. Many function minimization algorithms make use not only of the objective function, but also its *gradient* with respect to the parameters $\mathbf{w}$. For general feedforward neural networks the *backpropagation* algorithm efficiently evaluates the gradient of the output y with respect to the parameters $\mathbf{w}$, and thence the gradient of the objective function with respect to $\mathbf{w}$.

30.3 Training the single neuron as a binary classifier

We assume we have a data set of inputs $\{\mathbf{x}^{(n)}\}_{n=1}^N$ with binary labels $\{t^{(n)}\}_{n=1}^N$, and a neuron whose output $y(\mathbf{x};\mathbf{w})$ is bounded between 0 and 1. We can then write down the following *error function*:

$$G(\mathbf{w}) = -\sum_n \left[t^{(n)} \log y(\mathbf{x}^{(n)};\mathbf{w}) + (1-t^{(n)}) \log(1-y(\mathbf{x}^{(n)};\mathbf{w})) \right]. \quad (30.12)$$

Each term in this objective function may be recognized as a relative entropy between the empirical probability distribution $(t^{(n)}, 1-t^{(n)})$ and the probability distribution implied by the output of the neuron $(y, 1-y)$. The objective function is bounded below by zero and only attains this value if $y(\mathbf{x}^{(n)};\mathbf{w}) = t^{(n)}$ for all n.

We now differentiate this objective function with respect to $\mathbf{w}$.

Exercise 30.2:^{A2} Show that the derivative $\mathbf{g} = \partial G/\partial \mathbf{w}$ is given by:

$$g_j = \frac{\partial G}{\partial w_j} = \sum_{n=1}^N -(t^{(n)} - y^{(n)}) x_j^{(n)}. \quad (30.13)$$

Notice that the quantity $e^{(n)} \equiv t^{(n)} - y^{(n)}$ is the *error* on example n — the difference between the target and the output. The simplest thing to do with a gradient of an error function is to *descend* it (even though this is often dimensionally incorrect), so equation (30.13) motivates the following learning algorithm. Since the derivative $\partial G/\partial \mathbf{w}$ is a sum of terms $\mathbf{g}^{(n)}$ defined by

$$g_j^{(n)} \equiv -(t^{(n)} - y^{(n)}) x_j^{(n)} \quad (30.14)$$

for $n = 1, \ldots, N$, we can obtain a simple on–line algorithm by putting each input through the network one at a time, and adjusting $\mathbf{w}$ a little in a direction opposite to $\mathbf{g}^{(n)}$.

The on–line gradient–descent learning algorithm

Architecture. A single neuron has a number I of *inputs* x_i and one *output* y. Associated with each input is a weight w_i $(i = 1, \ldots, I)$.

Activity rule. 1. First, in response to the received inputs $\mathbf{x}$ (which may be arbitrary real numbers), we compute the *activation* of the neuron,

$$a = \sum_i w_i x_i, \quad (30.15)$$

where the sum is over $i = 0, \ldots, I$ if there is a bias and $i = 1, \ldots, I$ otherwise.

```
global x ;        # x is an N * K matrix containing all the input vectors
global t ;        # t is a vector of length N containing all the targets

for l = 1:L       # loop L times

   a = x * w  ;                          # compute all activations
   y = sigmoid(a) ;                      # compute outputs
   e = t - y   ;                         # compute errors
   g = - x' * e ;                        # compute the gradient vector
   w = w - eta * ( g + alpha * w )  ;    # make step, using learning rate eta
                                         #    and weight decay alpha
endfor

function f = sigmoid ( v )
   f = 1.0 ./ ( 1.0 .+ exp ( - v ) ) ;
endfunction
```

Figure 30.4. `Octave` source code for a gradient descent optimizer of a single neuron, with optional weight decay. The weight decay rate is controlled by the scalar `alpha`. `Octave` is a convenient language for matrix operations. Notation: the single instruction `a = x * w` causes the $(N \times K)$ *matrix* `x` consisting of all the input vectors to be multiplied by the weight vector `w`, giving the *vector* `a` listing the activations for all N input vectors; `x'` means `x`–transpose; the single command `y = sigmoid(a)` computes the sigmoid function of all elements of the vector `a`.

2. Second, the *output* y is set as a sigmoid function of the activation.

$$y(a) = \frac{1}{1 + e^{-a}} \tag{30.16}$$

This output might be viewed as stating the probability, according to the neuron, that the given input is in class 1 rather than class 0.

Learning rule. The *teacher* supplies a *target* value $t \in \{0, 1\}$ which says what the correct answer is for the given input. We compute the *error signal*

$$e = t - y \tag{30.17}$$

then adjust the weights $\mathbf{w}$ in a direction that would reduce the magnitude of this error:

$$\Delta w_i = \eta e x_i, \tag{30.18}$$

where η is the 'learning rate'. Commonly η is set by trial and error to a constant value or to a decreasing function of simulation time τ such as η_0/τ.

The above rules are repeated for each input / target pair $(\mathbf{x}, t)$ that is presented. If there is a fixed data set of size N, we can cycle through the data multiple times.

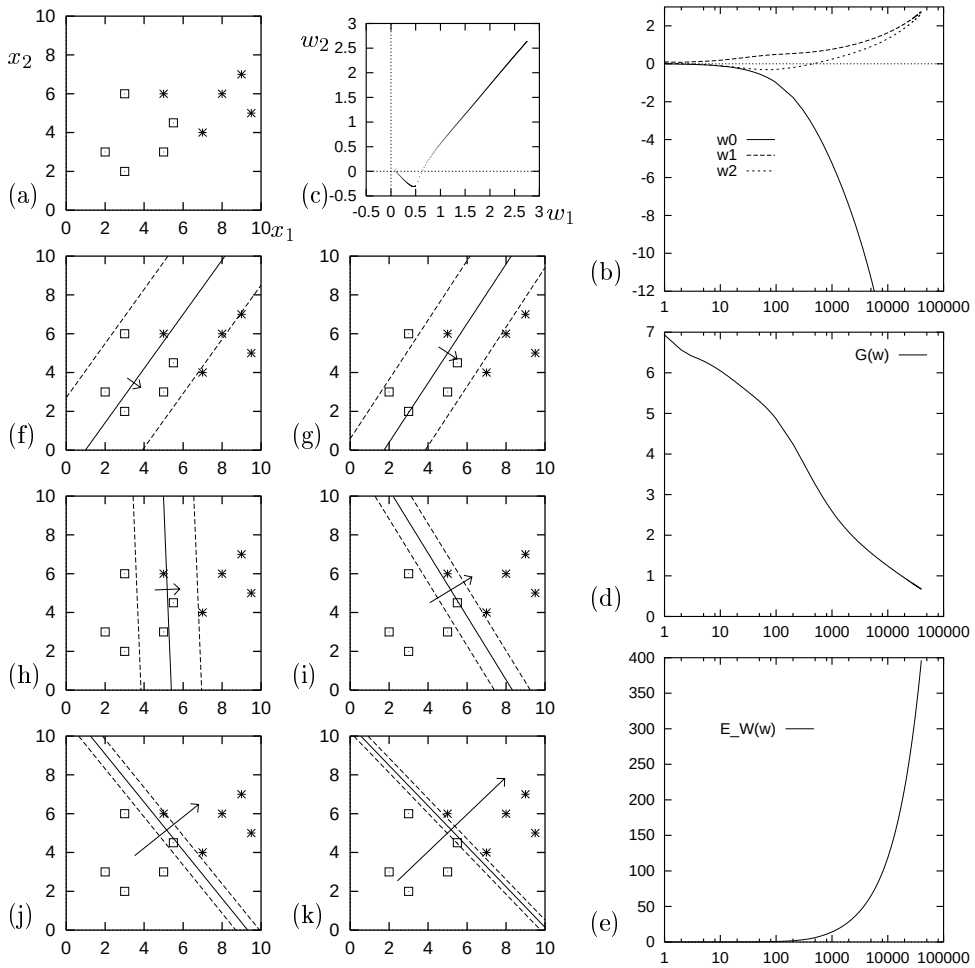


Figure 30.5. A single neuron learning to classify by gradient descent. The neuron has two weights w_1 and w_2 and a bias w_0. The learning rate was set to $\eta = 0.01$ and batch–mode gradient descent was performed using the code displayed in figure 30.4. (a) The training data. (b) Evolution of weights w_0, w_1 and w_2 as a function of number of iterations (on log scale). (c) Evolution of weights w_1 and w_2 in weight space. (d) The objective function $G(\mathbf{w})$ as a function of number of iterations. (e) The magnitude of the weights $E_W(\mathbf{w})$ as a function of time. (f–k) The function performed by the neuron (shown by three of its contours) after 30, 80, 500, 3000, 10000 and 40000 iterations. The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. Also shown is a vector proportional to (w_1, w_2). The larger the weights are, the bigger this vector becomes, and the closer together are the contours.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Batch learning versus on–line learning

Here we have described the *on-line* learning algorithm, in which a change in the weights is made after every example is presented. An alternative paradigm is to go through a *batch* of examples, computing the outputs and errors and accumulating the changes specified in equation (30.18) which are then made at the end of the batch.

Batch learning for the single neuron classifier

For each input/target pair $(\mathbf{x}^{(n)}, t^{(n)})$ $(n = 1, \ldots, N)$, compute $y^{(n)} = y(\mathbf{x}^{(n)}; \mathbf{w})$, where

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp\left(-\sum_i w_i x_i\right)}, \tag{30.19}$$

define $e^{(n)} = t^{(n)} - y^{(n)}$, and compute for each weight w_i

$$g_i^{(n)} = -e^{(n)} x_i^{(n)}. \tag{30.20}$$

Then let

$$\Delta w_i = -\eta \sum_n g_i^{(n)}. \tag{30.21}$$

Source code implementing batch learning gradient descent is given in figure 30.4. This algorithm is demonstrated in figure 30.5 for a neuron with two inputs with weights w_1 and w_2 and a bias w_0, performing the function

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}. \tag{30.22}$$

The bias w_0 is included, in contrast to the figure 30.3, where it was omitted. The neuron is trained on a data set of ten labelled examples.

30.4 Beyond descent on the error function: regularization

If the parameter η is set to an appropriate value, this algorithm works: the algorithm finds a setting of $\mathbf{w}$ which correctly classifies as many of the examples as possible.

If the examples are in fact *linearly separable* then the neuron finds this linear separation and its weights diverge to ever-larger values as the simulation continues. This can be seen happening in figure 30.5(f–k). This is an example of *overfitting*, where a model fits the data so well that its generalization performance is likely to be adversely affected.

This behaviour may be viewed as undesirable. How can it be rectified?

An ad hoc solution to overfitting is to use *early stopping*, that is, use an algorithm originally intended to minimize the error function $G(\mathbf{w})$, then prevent it from doing so by halting the algorithm at some point.

A principled solution to overfitting makes use of *regularization*. Regularization involves modifying the objective function in such a way as to incorporate a bias against the sorts of solution $\mathbf{w}$ which we dislike. In the above example, what we dislike is the development of a very sharp decision boundary; this

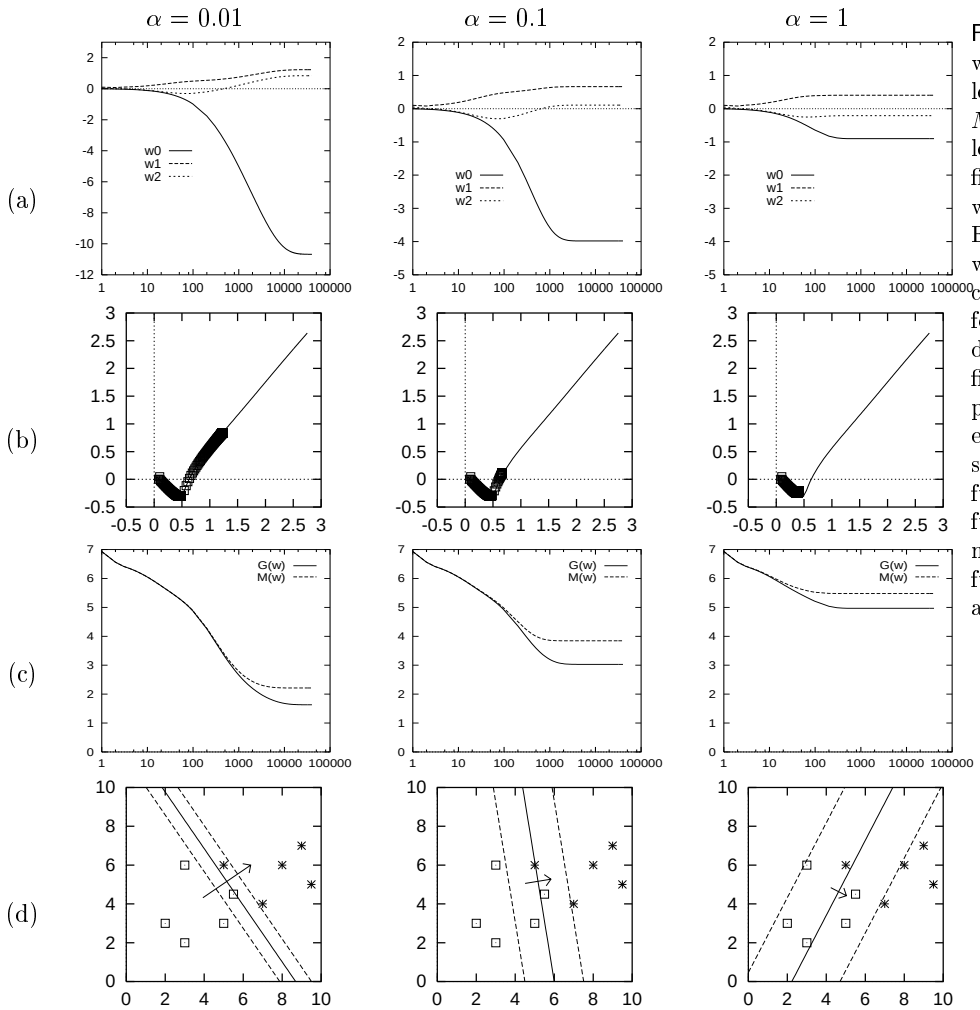


Figure 30.6. The influence of weight decay on a single neuron's learning. The objective function is $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$. The learning method was as in figure 30.5. (a) Evolution of weights w_0, w_1 and w_2. (b) Evolution of weights w_1 and w_2 in weight space shown by points, contrasted with the trajectory followed in the case of zero weight decay, shown by a thin line (from figure 30.5). Notice that for this problem weight decay has an effect very similar to 'early stopping'. (c) The objective function $M(\mathbf{w})$ and the error function $G(\mathbf{w})$ as a function of number of iterations. (d) The function performed by the neuron after 40000 iterations.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

sharp boundary is associated with large weight values, so we use a regularizer that penalizes large weight values. We modify the objective function to:

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{30.23}$$

where the simplest choice of regularizer is the '*weight decay*' regularizer

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_i w_i^2. \tag{30.24}$$

The regularization constant α is called the 'weight decay rate'. This additional term favours small values of $\mathbf{w}$ and decreases the tendency of a model to overfit fine details of the training data. The quantity α is known as a *hyperparameter*. Hyperparameters play a role in the learning algorithm but play no role in the activity rule of the network.

Exercise 30.3:^{A2} Compute the derivative of $M(\mathbf{w})$ with respect to w_i. Why is the above regularizer known as the 'weight decay' regularizer?

The gradient descent source code of figure 30.4 implements weight decay. This gradient descent algorithm is demonstrated in figure 30.6 using weight decay rates $\alpha = 0.01, 0.1$ and 1. It may be noticed that as the weight decay rate is increased the solution becomes biased towards broader sigmoid functions with decision boundaries that are closer to the origin.

Note

Gradient descent with a step size η is in general *not* the most efficient way to minimize a function. A modification of gradient descent known as *momentum* is also not recommended. [An aside to those who are familiar with momentum: please do not confuse momentum, which is sometimes given the symbol α, with weight decay.] Most neural network experts use more advanced optimizers such as conjugate gradient algorithms.

30.5 Exercises

Motivations for the linear neuron

Exercise 30.4:^{B2} Consider the task of recognizing which of two Gaussian distributions a vector $\mathbf{z}$ comes from. Unlike the case studied in section 11.2, where the distributions had different means but a common variance–covariance matrix, we will assume that the two distributions have exactly the same mean but different variances. Let the probability of $\mathbf{z}$ given s ($s \in \{0, 1\}$) be

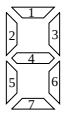
$$P(\mathbf{z}|s) = \prod_{i=1}^{I} \text{Normal}(z_i; 0, \sigma_{si}^2), \tag{30.25}$$

where σ_{si}^2 is the variance of z_i when the source symbol is s. Show that $P(s = 1|\mathbf{z})$ can be written in the form

$$P(s = 1|\mathbf{z}) = \frac{1}{1 + \exp(-\mathbf{w}^{\mathsf{T}}\mathbf{x} + \theta)}, \tag{30.26}$$

where x_i is an appropriate function of z_i, $x_i = g(z_i)$.

Exercise 30.5:^{A2} **The noisy LED**.



$\mathbf{c}(2) =$ $\mathbf{c}(3) =$ $\mathbf{c}(8) =$

Consider an LED display with 7 elements numbered as shown above. The state of the display is a vector $\mathbf{x}$. When the controller wants the display to show character number s, *e.g.*, $s = 2$, each element x_j $(j = 1, \ldots, 7)$ either adopts its intended state $c_j(s)$, with probability $1 - f$, or is flipped, with probability f. Let's call the two states of x '+1' and '−1'.

(a) Assuming that the intended character s is actually a 2 or a 3, what is the probability of s, given the state $\mathbf{x}$? Show that $P(s = 2|\mathbf{x})$ can be written in the form

$$P(s = 2|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^{\mathsf{T}}\mathbf{x} + \theta)}, \tag{30.27}$$

and compute the values of the weights $\mathbf{w}$ in the case $f = 0.1$.

(b) Assuming that s is one of $\{0, 1, 2, \ldots, 9\}$, with prior probabilities p_s, what is the probability of s, given the state $\mathbf{x}$? Put your answer in the form

$$P(s|\mathbf{x}) = \frac{e^{a_s}}{\sum_{s'} e^{a_{s'}}}, \tag{30.28}$$

where $\{a_s\}$ are functions of $\{c_j(s)\}$ and $\mathbf{x}$.

Could you make a better alphabet of 10 characters for a noisy LED, *i.e.*, an alphabet less susceptible to confusion?

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 30's exercises

Solution to exercise 30.1 (p. 361): One answer, given in the text on page 361, is that the single neuron function was encountered under 'the best detection of pulses'. The same function has also appeared in the chapter on variational methods when we derived mean field theory for a spin system. Several of the solutions to the inference problems in chapter 1 were also written in terms of this function.

Solution to exercise 30.2 (p. 364): ...

Solution to exercise 30.3 (p. 369): ...

Solution to exercise 30.4 (p. 369): ...

Solution to exercise 30.5 (p. 370): Useful stuff: let $\mathbf{x}$ and $\mathbf{s}$ be binary $\in \{\pm 1\}^7$. The likelihood is $(1-f)^N f^M$, where $N = (\mathbf{s}^{\mathsf{T}}\mathbf{x} + 7)/2$ and $M = (7 - \mathbf{s}^{\mathsf{T}}\mathbf{x})/2$.

The LED displays a binary code of length 7 with 10 codewords. Some codewords are very confusable — 8 and 9 differ by just one bit, for example. A superior binary code of length 7 is the (7,4) Hamming code. This code has 15 non–zero codewords, all separated by a distance of at least 3 bits.

Here are those 15 codewords, along with a suggested mapping to the integers 0–14.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

Problems to look at before Chapter 31

Exercise 30.6: What is $\sum_{K=0}^{N} \binom{N}{K}$? [The symbol $\binom{N}{K}$ means the combination $\frac{N!}{K!(N-K)!}$.]

Exercise 30.7: If the top row of Pascal's triangle (which contains the single number '1') is denoted row zero, what is the sum of all the numbers in the triangle above row N?

Exercise 30.8: 3 points are selected at random on the surface of a sphere. What is the probability that all of them lie on a single hemisphere?

General position

Definition 30.1 *A set of points $\{\mathbf{x}_n\}$ in K–dimensional space are in* general position *if any subset of size $\leq K$ is linearly independent.*

In $K = 3$ dimensions, for example, a set of points are in general position if no three points are colinear and no four points are coplanar.

Exercise 30.9: Can a finite set of $2N$ distinct points in a two–dimensional space be split in half by a straight line

- if the points are in general position?
- if the points are not in general position?

Can $2N$ points in a K dimensional space be split in half by a $K-1$ dimensional hyperplane?

31

Capacity of a single neuron

31.1 Neural network learning as communication

Many neural network models involve the adaptation of a set of weights $\mathbf{w}$ in response to a set of data points, for example a set of N target values $D_N = \{t_n\}_{n=1}^N$ at given locations $\{\mathbf{x}_n\}_{n=1}^N$. The adapted weights are then used to process subsequent input data. This process can be viewed as a communication process, in which the sender examines the data D_N and creates a message $\mathbf{w}$ that depends on those data. The receiver then uses $\mathbf{w}$; for example, the receiver might use the weights to try to reconstruct what the data D_N was. [In neural network parlance, this is using the neuron for 'memory' rather than for 'generalization'; 'generalizing', means extrapolating from the observed data to the value of t_{N+1} at some new location $\mathbf{x}_{N+1}$.] Just as a disc drive or a piece of

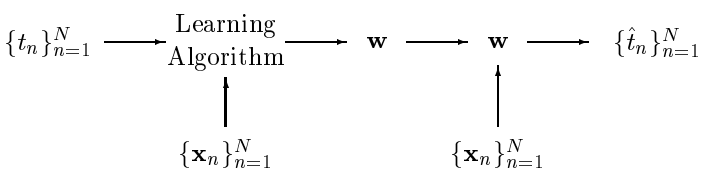


Figure 31.1. Neural network learning viewed as communication.

RAM is a communication channel, the adapted network weights $\mathbf{w}$ therefore play the role of a communication channel, conveying information to a future user about the training data. The question we now address is, "what is the capacity of this channel?" — that is, "how much information can be stored by training a neural network?"

If we had a learning algorithm that either produces a network whose response to all inputs is +1 or a network whose response to all inputs is 0, depending on the training data, then the weights allow us to distinguish between just two sorts of data set. The maximum information such a learning algorithm could convey about the data is therefore 1 bit, this information content being achieved if the two sorts of data set are equiprobable. How much more information can be conveyed if we make full use of a neural network's ability to represent other functions?

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

31.2 The capacity of a single neuron

We will look at the simplest case, that of a single binary threshold neuron. We will find that the capacity of such a neuron is *two bits per connection.* A neuron with K inputs can store $2K$ bits of information.

To obtain this result we will need to lay down some rules to exclude less interesting answers, such as: "the capacity of a neuron is infinite, because each of its weights is a real number and so can convey an infinite number of bits". We exclude this answer by saying that the receiver is not able to examine the weights directly, nor is the receiver allowed to probe the weights by observing the output of the neuron for arbitrarily chosen inputs. We will constrain the receiver to observe the output of the neuron at the same fixed set of N points $\{\mathbf{x}_n\}$ that were in the training set. What matters now is how many different distinguishable functions our neuron can produce, given that we can only observe the function at these N points. How many different binary labellings of N points can a linear threshold function produce? And how does this number compare with the maximum possible number of binary labellings, 2^N? If nearly all of the 2^N labellings can be realised by our neuron, then it is a communication channel that can convey all N bits (the target values $\{t_n\}$) with small probability of error. We will identify the capacity of the neuron as the maximum value that N can have such that the probability of error is very small.

We thus examine the following scenario. The sender is given a neuron with K inputs and a data set D_N which is a labelling of N points in general position. The sender uses an adaptive algorithm to try to find a $\mathbf{w}$ that can reproduce this labelling exactly. For this analysis, we will assume that they have a perfect algorithm, that finds such a $\mathbf{w}$ if it exists. Otherwise, $\mathbf{w}$ is set to some other value. The receiver then evaluates the threshold function on the N input values. What is the probability that *all* N bits are correctly reproduced? How large can N become, for a given K, without this probability becoming substantially less than one?

General position

One technical detail needs to be pinned down: what set of inputs $\{\mathbf{x}_n\}$ are we considering? Our answer might depend on this choice. We will assume that the points are in *general position*, which means in $K = 3$ dimensions, for example, that no three points are colinear and no four points are coplanar.

Definition 31.1 *A set of points $\{\mathbf{x}_n\}$ in K–dimensional space are in* general position *if any subset of size $\leq K$ is linearly independent.*

The linear threshold function

The neuron we will consider performs the function

$$y = f\left(\sum_{k=1}^{K} w_k x_k\right). \tag{31.1}$$

where

$$f(a) = \begin{cases} 1 & a > 0 \\ 0 & a \leq 0 \end{cases} \tag{31.2}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

We will not have a bias w_0; the capacity for a neuron with a bias can be obtained by replacing K by $K+1$ in the final result below, *i.e.*, considering one of the inputs to be fixed to 1. (This is not incompatible with the assumption that the points are in general position.)

31.3 Counting threshold functions

Let us denote by $T(N, K)$ the number of distinct threshold functions on N points in general position in K dimensions. We will derive[1] a formula for $T(N, K)$.

To start with, let us work out a few cases by hand.

In $K = 1$ dimension, for any N

The N points lie on a line. By changing the sign of the one weight w_1 we can label all points on the right side of the origin 1 and the others 0, or vice versa. Thus there are two distinct threshold functions. $T(N, 1) = 2$.

With $N = 1$ point, for any K

If there is just one point $\mathbf{x}^{(1)}$ then we can realise both possible labellings by setting $\mathbf{w} = \pm\mathbf{x}^{(1)}$. Thus $T(1, K) = 2$.

In $K = 2$ dimensions

In two dimensions with N points, we are free to spin the separating line around the origin. Each time the line passes over a point we obtain a new function. Once we have spun the line through 2π we reproduce the function we started from. Because the points are in general position, the separating plane crosses only one point at a time. In one revolution, every point is passed over twice. There are therefore $2N$ distinct threshold functions. $T(N, 2) = 2N$.

Comparing with the total number of binary functions, 2^N, we may note that for $N \geq 3$, not all binary functions can be realised by a linear threshold function. One famous example of an unrealisable function with $N = 4$ and $K = 2$ is the exclusive–or function on the points $\mathbf{x} = (\pm 1, \pm 1)$.[2]

In $K = 2$ dimensions, from the point of view of weight space

There is another way of visualizing this problem. Instead of visualizing a plane separating points in the two–dimensional input space, we can consider the two–dimensional weight space and count the number of threshold functions by counting how many distinguishable regions there are in it. We can colour regions in weight space different colours if they label the given points differently. Consider weight space and consider the set of weight vectors that classify a particular example $\mathbf{x}^{(n)}$ as a 1. For example, figure 31.2(a) shows a single point in our two–dimensional $\mathbf{x}$–space, and figure 31.2(b) shows the two

[1] This derivation is originally due to Cover (1965) and the exposition that follows is due to Yaser Abu–Mostafa. Another explanation of this material can be found in Hertz *et al.* (1991), page 111.

[2] These points are not in general position, but you may confirm that the function remains unrealisable if the points are perturbed into general position.

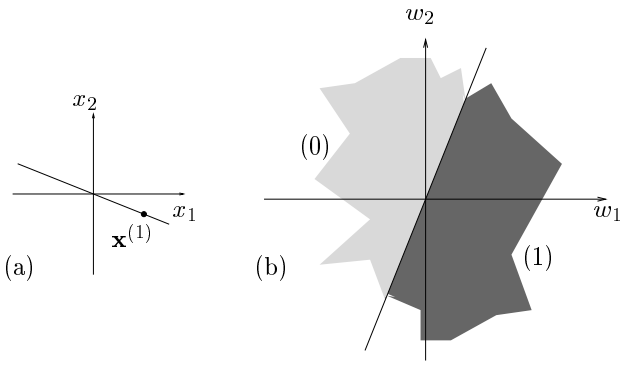


Figure 31.2. One data point in a two–dimensional input space, and the two regions of weight space that give the two alternative labellings of that point.

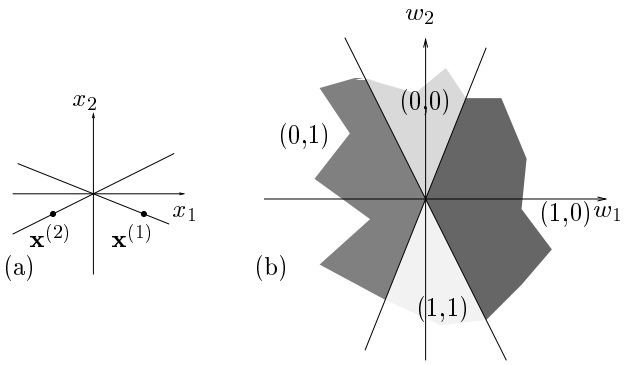


Figure 31.3. Two data points in a two–dimensional input space, and the four regions of weight space that give the four alternative labellings.

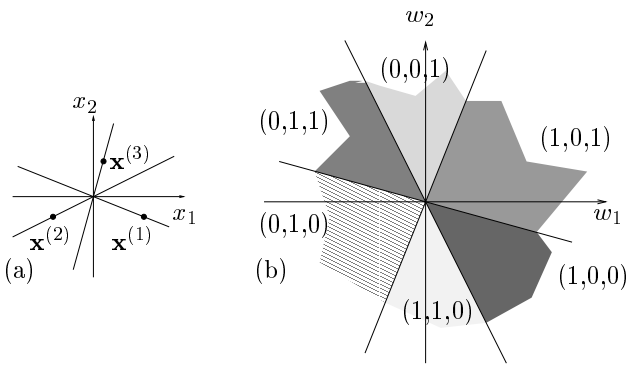


Figure 31.4. Three data points in a two–dimensional input space, and the six regions of weight space that give alternative labellings of those points. In this case, the labellings (0,0,0) and (1,1,1) cannot be realised. For any three points in general position there are always two labellings that cannot be realised.

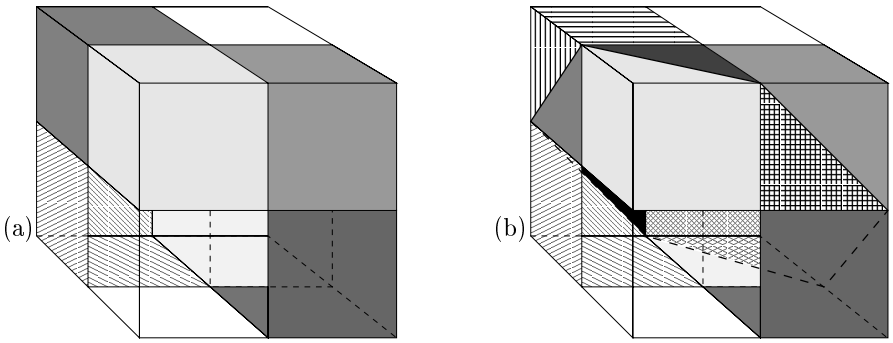


Figure 31.5. Weight space illustrations for $T(3,3)$ and $T(4,3)$. (a) $T(3,3) = 8$. Three hyperplanes (corresponding to three points in general position) divide 3–space into 8 regions, shown here by colouring the relevant part of the surface of a hollow, semi–transparent cube centred on the origin. (b) $T(4,3) = 14$. Four hyperplanes divide 3–space into 14 regions, of which this figure shows 13 (the 14th region is out of view on the right hand face. Compare with figure 31.5a: all of the regions that are not coloured white have been cut into two.

	K							
N	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	4	4					
3	2	6	8					
4	2	8	14					
5	2	10						
6	2	12						

Table 31.1. Values of $T(N,K)$ deduced by hand

corresponding sets of points in $\mathbf{w}$–space. One set of weight vectors occupy the half space

$$\mathbf{x}^{(n)} \cdot \mathbf{w} > 0, \tag{31.3}$$

and the others occupy $\mathbf{x}^{(n)} \cdot \mathbf{w} < 0$. In figure 31.3(a) we have added a second point in the input space. There are now 4 possible labellings: (1,1), (1,0), (0,1) and (0,0). Figure 31.3(b) shows the two hyperplanes $\mathbf{x}^{(1)} \cdot \mathbf{w} = 0$ and $\mathbf{x}^{(2)} \cdot \mathbf{w} = 0$ which separate the sets of weight vectors that produce each of these labellings. When $N = 3$ (figure 31.4), weight space is divided by three hyperplanes into six regions. Not all of the eight possible labellings can be realised. Thus $T(3,2) = 6$.

In $K = 3$ dimensions

We now use this weight space visualization to study the three dimensional case.

Let us imagine adding one point at a time and count the number of threshold functions as we do so. When $n = 2$, weight space is divided by two hyperplanes $\mathbf{x}^{(1)} \cdot \mathbf{w} = 0$ and $\mathbf{x}^{(2)} \cdot \mathbf{w} = 0$ into four regions; in any one region all vectors $\mathbf{w}$ produce the same function on the 2 input vectors. Thus $T(2,3) = 4$.

Adding a third point in general position produces a third plane in $\mathbf{w}$ space, so that there are 8 distinguishable regions. $T(3,3) = 8$. The three bisecting planes are shown in figure 31.5a.

At this point matters become slightly more tricky. The fourth plane in the three–dimensional $\mathbf{w}$ space cannot transect all eight of the sets created by the first three planes. A sketch confirms that for points in general position six of the exisiting regions are cut into two and the remaining two are unaffected

	K							
N	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		

Table 31.2. Pascal's triangle

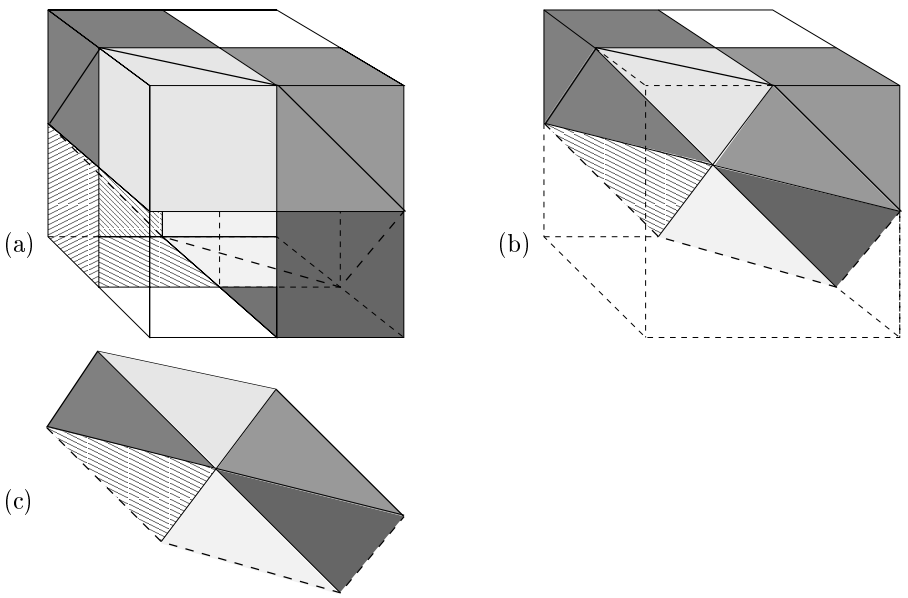


Figure 31.6. Illustration of the cutting process going from $T(3,3)$ to $T(4,3)$. (a) The eight regions of figure 31.5a with one added hyperplane. All of the regions that are not coloured white have been cut into two. (b) Here, the hollow cube has been made solid, so we can see which regions are cut by the fourth plane. The front half of the cube has been cut away. (c) This figure shows the new two dimensional hyperplane, which is divided into six regions by the three one–dimensional hyperplanes (lines) which cross it. Each of these regions corresponds to one of the three–dimensional regions in figure 31.6a which is cut into two by this new hyperplane. This shows that $T(4,3) - T(3,3) = 6$. Figure 31.6c should be compared with figure 31.4b.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

(figure 31.5b). So $T(4,3) = 14$. Thus two of the binary functions on 4 points in 3 dimensions cannot be realised by a linear threshold function.

We have now filled in the values of $T(N,K)$ shown in table 31.1. Can we obtain any insights into our derivation of $T(4,3)$ in order to fill in the rest of the table for $T(N,K)$? Why was $T(4,3)$ greater than $T(3,3)$ by six?

Six is the number of regions that the new hyperplane bisected in $\mathbf{w}$–space (figure 31.6a). Equivalently, if we look in the $K-1$ dimensional subspace that is the Nth hyperplane, that subspace is divided up into six regions by the $N-1$ previous hyperplanes (figure 31.6c). Now this is a concept we have met before. Compare figure 31.6c with figure 31.4b. How many regions are created by $N-1$ hyperplanes in a $K-1$ dimensional space? Why, $T(N-1,K-1)$, of course! In the present case $N = 4, K = 3$, we can look up $T(3,2) = 6$ in the previous section. So

$$T(4,3) = T(3,3) + T(3,2). \tag{31.4}$$

Recurrence relation for any N, K

Generalizing this picture, we see that when we add an Nth hyperplane in K dimensions, it will bisect $T(N-1,K-1)$ of the $T(N-1,K)$ regions that were created by the previous $N-1$ hyperplanes. This means we can write down $T(N,K)$ in terms of $T(N-1,K-1)$ and $T(N-1,K)$:

$$T(N,K) = T(N-1,K) + T(N-1,K-1). \tag{31.5}$$

Now all that remains is to solve this recurrence relation given the boundary conditions $T(N,1) = 2$ and $T(1,K) = 2$.

Does the recurrence relation (31.5) look familiar? Maybe you remember building Pascal's triangle by adding together two adjacent numbers in one row to get the number below. The N, K element of Pascal's triangle is equal to

$$C(N,K) \equiv \binom{N}{K} \equiv \frac{N!}{(N-K)!K!}. \tag{31.6}$$

So combinations $\binom{N}{K}$ satisfy the equation

$$C(N,K) = C(N-1,K-1) + C(N-1,K), \quad \text{for all } N > 0. \tag{31.7}$$

[Here we are adopting the convention that $\binom{N}{K} \equiv 0$ if $K > N$ or $K < 0$.] So $\binom{N}{K}$ satisfies the required recurrence relation (31.5). This doesn't mean $T(N,K) = \binom{N}{K}$, since there can be many functions that satisfy one recurrence relation. But perhaps we can express $T(N,K)$ as a linear superposition of combination functions of the form $C_{\alpha,\beta}(N,K) \equiv \binom{N+\alpha}{K+\beta}$. By comparing tables 31.2 and 31.1 we can see how to satisfy the boundary conditions: we simply need to translate Pascal's triangle to the right by 1, 2, 3, ...; superpose; add; multiply by two, and drop the whole table by one line. Thus:

$$T(N,K) = 2 \sum_{k=0}^{K-1} \binom{N-1}{k} \tag{31.8}$$

Using the fact that the Nth row of Pascal's triangle sums to 2^N, that is, $\sum_{k=0}^{N-1} \binom{N-1}{k} = 2^{N-1}$, we can simplify the cases where $K-1 \geq N-1$.

$$T(N,K) = \begin{cases} 2^N & K \geq N \\ 2 \sum_{k=0}^{K-1} \binom{N-1}{k} & K < N \end{cases} \tag{31.9}$$

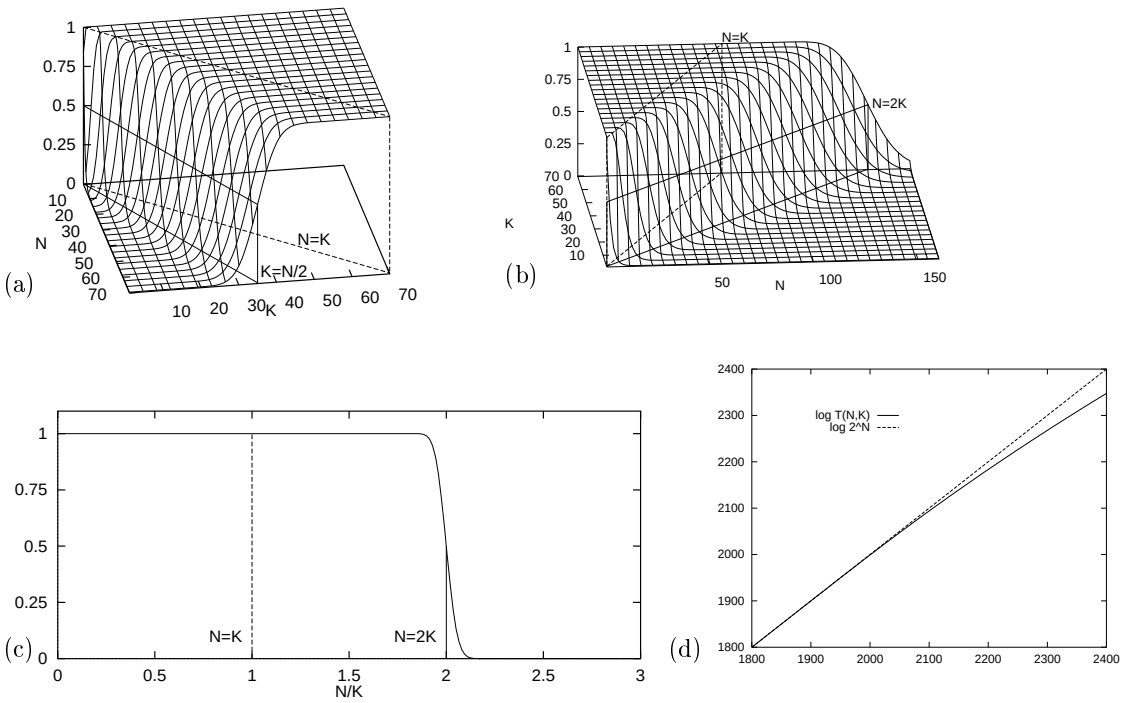


Figure 31.7. The fraction of functions on N points in K dimensions that are linear threshold functions, $T(N,K)/2^N$, shown from various viewpoints. In (a) we see the dependence on K, which is approximately an error function passing through 0.5 at $K = N/2$; the fraction reaches 1 at $K = N$. In (b) we see the dependence on N, which is 1 up to $N = K$ and drops sharply at $N = 2K$. Figure (c) shows the dependence on N/K for $K = 1000$. There is a sudden drop in the fraction of realizable labellings when $N = 2K$. Figure (d) shows the values of $\log_2 T(N,K)$ and $\log_2 2^N$ as a function of N for $K = 1000$. These figures were plotted using the approximation of $T/2^N$ by the error function.

Interpretation

It is natural to compare $T(N,K)$ with the total number of binary functions on N points, 2^N. The ratio $T(N,K)/2^N$ tells us the probability that an arbitrary labelling $\{t_n\}_{n=1}^N$ can be memorized by our neuron. The two functions are equal for all $N \leq K$. The line $N = K$ is thus a special line, namely, it defines the maximum number of points on which *any* arbitrary labelling can be realised. This number of points is referred to as the Vapnik–Chervonenkis dimension (VC dimension) of the class of functions. The VC dimension of a binary threshold function on K dimensions is thus K.

What is interesting is (for large K) the number of points N such that *almost* any labelling can be realised. The ratio $T(N,K)/2^N$ is, for $N < 2K$, still greater than $1/2$, and for large K the ratio is less than 1 by an exponentially small quantity.

For our purposes the familiar sum in equation (31.9) is well approximated by the error function,

$$\sum_0^K \binom{N}{k} \simeq 2^N \, \Phi\left(\frac{K - (N/2)}{\sqrt{N}/2}\right), \tag{31.10}$$

where $\Phi(z) \equiv \int_{-\infty}^{z} \exp(-z^2/2)/\sqrt{2\pi}$. Figure 31.7 shows the fraction $T(N,K)/2^N$ as a function of N and K. The take–home message is shown in figure 31.7(c): although the fraction $T(N,K)/2^N$ is less than 1 for $N > K$, it is only negligibly less than 1 up to $N = 2K$; there, there is a catastrophic drop to zero, so that for $N > 2K$, only a tiny fraction of the binary labellings can be realised by the threshold function.

Conclusion

The capacity of a linear threshold neuron, for large K, is 2 bits per connection.

A single neuron can almost certainly memorize up to $N = 2K$ random binary labels, but will almost certainly fail to memorize more.

31.4 Problems

Exercise 31.1:^{A2} 4 points are selected at random on the surface of a sphere. What is the probability that all of them lie on a single hemisphere? How does this question relate to $T(N,K)$?

Exercise 31.2:^{A2} Consider the binary threshold neuron in $K = 3$ dimensions, and the set of points $\{\mathbf{x}\} = \{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}$. Find a parameter vector $\mathbf{w}$ such that the neuron memorizes the labels: (a) $\{t\} = \{1,1,1,1\}$; (b) $\{t\} = \{1,1,0,0\}$.

Find an unrealizable labelling $\{t\}$.

Exercise 31.3:^{B2} Estimate in bits the total sensory experience that you have had in your life — visual information, auditory information, etc. Estimate how much information you have memorized. Estimate the information content of the works of Shakespeare. Compare these with the capacity of your brain assuming you have 10^{11} neurons each making 1000 synaptic connections, and that the capacity result for one neuron (two bits per connection) applies. Is your brain full yet?

Solutions to Chapter 31's exercises

Solution to exercise 31.1 (p. 381): The probability that all 4 points lie on a single hemisphere is

$$T(4,3)/2^4 = 14/16 = 7/8. \tag{31.11}$$

Solution to exercise 31.2 (p. 381):

(a) $\mathbf{w} = (1,1,1)$.

(b) $\mathbf{w} = (1/4, 1/4, -1)$.

The two unrealizable labellings are $\{0,0,0,1\}$ and $\{1,1,1,0\}$.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

32

Learning as Inference

32.1 Neural network learning as inference

In chapter 30 we trained a simple neural network as a classifier by minimizing an objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{32.1}$$

made up of an error function

$$G(\mathbf{w}) = -\sum_n \left[t^{(n)} \log y(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log(1 - y(\mathbf{x}^{(n)}; \mathbf{w})) \right] \tag{32.2}$$

and a regularizer

$$E_W(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2. \tag{32.3}$$

This neural network learning process can be given the following probabilistic interpretation.

We interpret the output $y(\mathbf{x}; \mathbf{w})$ of the neuron literally as defining (when its parameters $\mathbf{w}$ are specified) the probability that an input $\mathbf{x}$ belongs to class $t = 1$, rather than the alternative $t = 0$. Thus $y(\mathbf{x}; \mathbf{w}) \equiv P(t=1|\mathbf{x}, \mathbf{w})$. Then each value of $\mathbf{w}$ defines a different hypothesis about the probability of class 1 relative to class 0 as a function of $\mathbf{x}$.

We define the observed data D to be the *targets* $\{t\}$ — the inputs $\{\mathbf{x}\}$ are assumed to be given, and not to be modelled. To infer $\mathbf{w}$ given the data, we require a likelihood function and a prior probability over $\mathbf{w}$. The likelihood function measures how well the parameters $\mathbf{w}$ predict the observed data; it is the probability assigned to the observed t values by the model with parameters set to $\mathbf{w}$. Now the two equations

$$\begin{array}{rcl} P(t = 1|\mathbf{w}, \mathbf{x}) & = & y \\ P(t = 0|\mathbf{w}, \mathbf{x}) & = & 1 - y \end{array} \tag{32.4}$$

can be rewritten as the single equation

$$P(t|\mathbf{w}, \mathbf{x}) = y^t (1 - y)^{1-t} = \exp\left[t \log y + (1 - t) \log(1 - y) \right]. \tag{32.5}$$

So the error function G can be interpreted as minus the log likelihood:

$$P(D|\mathbf{w}) = \exp[-G(\mathbf{w})]. \tag{32.6}$$

Similarly the regularizer can be interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{32.7}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$, and $1/Z_W(\alpha)$ is equal to $(\alpha/2\pi)^{K/2}$, where K is the number of parameters in the vector $\mathbf{w}$.

The objective function $M(\mathbf{w})$ then corresponds to the **inference** of the parameters $\mathbf{w}$, given the data:

$$P(\mathbf{w}|D,\alpha) = \frac{P(D|\mathbf{w})P(\mathbf{w}|\alpha)}{P(D|\alpha)} \tag{32.8}$$

$$= \frac{e^{-G(\mathbf{w})}\, e^{-\alpha E_W(\mathbf{w})}/Z_W(\alpha)}{P(D|\alpha)} \tag{32.9}$$

$$= \frac{1}{Z_M} \exp(-M(\mathbf{w})). \tag{32.10}$$

So the $\mathbf{w}$ *found by (locally) minimizing* $M(\mathbf{w})$ *can be interpreted as the (locally) most probable parameter vector,* $\mathbf{w}_{\rm MP}$.

Why is it natural to interpret the error functions as *log* probabilities? Error functions are usually additive. For example, G is a *sum* of relative entropies, and E_W is a *sum* of squared weights. Probabilities, on the other hand, are multiplicative: for independent events X and Y, the joint probability is $P(x, y) = P(x)P(y)$. The logarithmic mapping maintains this correspondence.

The interpretation of $M(\mathbf{w})$ as a log probability has numerous benefits, some of which we will discuss in a moment.

32.2 Illustration for a neuron with two weights

In the case of a neuron with just two inputs and no bias,

$$y(\mathbf{x};\mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}, \tag{32.11}$$

we can plot the posterior probabillity of $\mathbf{w}$, $P(\mathbf{w}|D,\alpha) \propto \exp(-M(\mathbf{w}))$. Imagine that we receive some data as shown in the left column of figure 32.1. Each data point consists of a two dimensional input vector $\mathbf{x}$ and a t value indicated by $\times$ ($t = 1$) or $\Box$ ($t = 0$). The likelihood function $\exp(-G(\mathbf{w}))$ is shown as a function of $\mathbf{w}$ in the second column. It is a product of functions of the form (32.11).

The product of traditional learning is a point in $\mathbf{w}$–space, the estimator $\mathbf{w}^*$, which maximizes the posterior probability density. In contrast, in the Bayesian view, the product of learning is an *ensemble* of plausible parameter values (bottom right of figure 32.1). We do not choose one particular hypothesis $\mathbf{w}$; rather we evaluate their posterior probabilities. The posterior distribution is obtained by multiplying the likelihood by a prior distribution over $\mathbf{w}$ space (shown as a broad Gaussian at the upper right of figure 32.1). The posterior ensemble (within a multiplicative constant) is shown in the third column of figure 32.1, and as a contour plot in the fourth column. As the amount of data increases (from top to bottom), the posterior ensemble becomes increasingly concentrated around the most probable value $\mathbf{w}^*$. This illustrates the Bayesian interpretation and generalization of traditional neural network learning.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

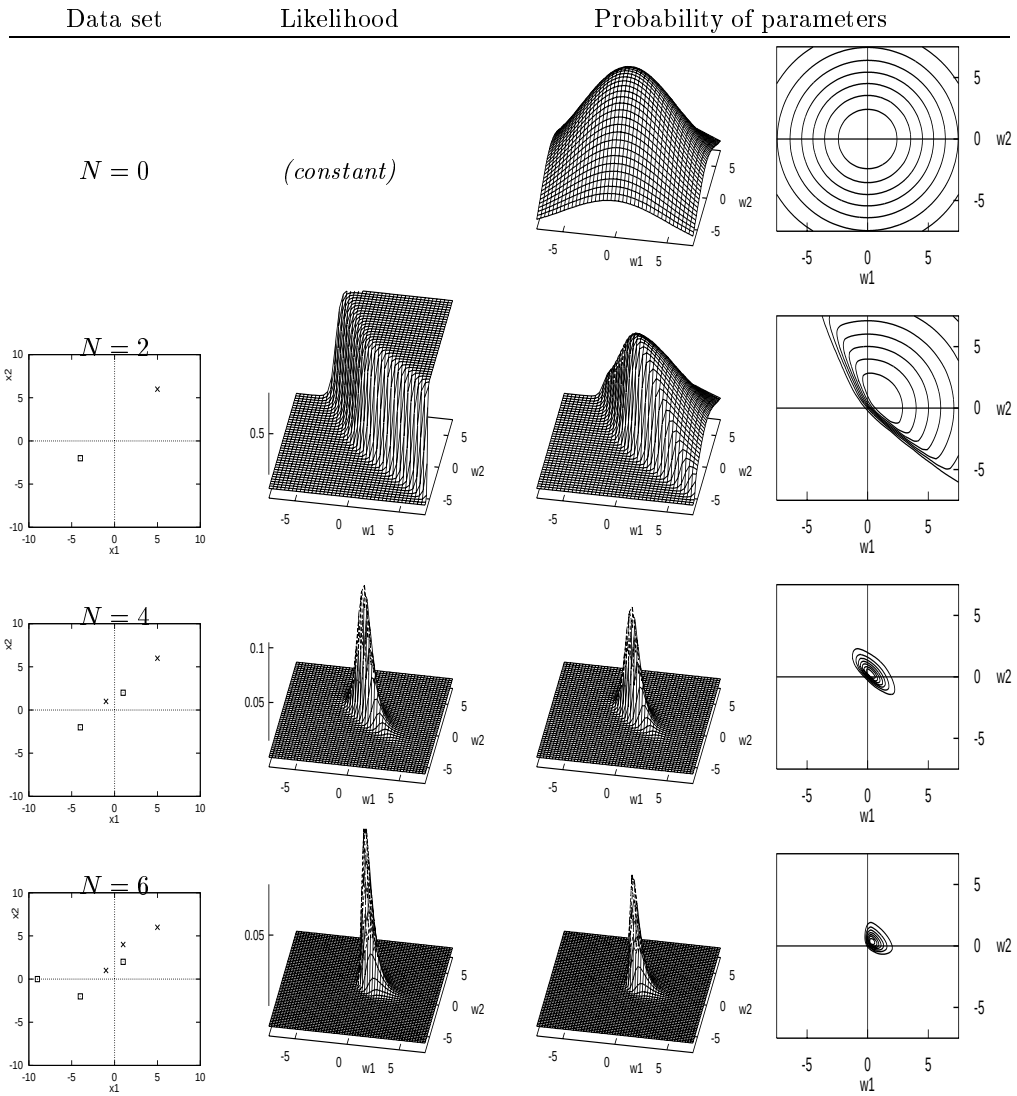


Figure 32.1. Evolution of the probability distribution over parameters as data arrives.

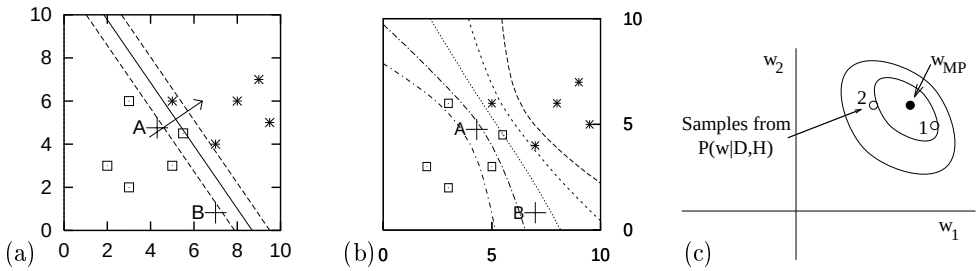


Figure 32.2. Making predictions. (a) The function performed by an optimized neuron (shown by three of its contours) trained with weight decay, $\alpha = 0.01$ (from figure 30.6). The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. (b) Are these predictions more reasonable? (Contours shown are for $y = 0.5, 0.27$, 0.73, 0.12 and 0.88.) (c) The posterior probability of $\mathbf{w}$ (schematic); the Bayesian predictions shown in (b) were obtained by averaging together the predictions made by each possible value of the weights $\mathbf{w}$, with each value of $\mathbf{w}$ receiving a vote proportional to its probability under the posterior ensemble. The method used to create (b) is described in section 32.4.

32.3 Beyond optimization: making predictions

Let us consider the task of making predictions with the neuron which we trained as a classifier in section 30.3. This was a neuron with two inputs and a bias.

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}} \tag{32.12}$$

When we last played with it, we trained it by minimizing the objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}). \tag{32.13}$$

The resulting optimized function for the case $\alpha = 0.01$ is reproduced in figure 32.2(a).

We now consider the task of predicting the class $t^{(N+1)}$ corresponding to a new input $\mathbf{x}^{(N+1)}$. It is common practice when making predictions simply to use a neural network with its weights fixed to their optimized value $\mathbf{w}_{\rm MP}$, but this is not optimal, as can be seen intuitively by considering the predictions shown in figure 32.2a. Are these reasonable predictions? Consider new data arriving at points A and B. The best fit model assigns both of these examples probability 0.2 of being in class 1, because they have the same value of $\mathbf{w}_{\rm MP} \cdot \mathbf{x}$. If we really knew that $\mathbf{w}$ was equal to $\mathbf{w}_{\rm MP}$, then these predictions would be correct. But we do not know $\mathbf{w}$. The parameters are *uncertain*. Intuitively we might be inclined to assign a less confident probability (closer to 0.5) at B than at A, as shown in figure 32.2(b), since point B is far from the training data. *The best fit parameters* $\mathbf{w}_{\rm MP}$ *often give over–confident predictions.* A non–Bayesian approach to this problem is to down–weight all predictions uniformly, by an empirically determined factor (Copas, 1983). This is not ideal, since intuition suggests the strength of the predictions at B should be downweighted more than those at A. A Bayesian viewpoint helps us to understand the cause of the problem, and provides a straightforward solution that is demonstrably superior to this ad hoc procedure. In a nutshell, we obtain Bayesian predictions by taking into account the whole posterior ensemble, shown schematically in figure 32.2(c).

The Bayesian prediction of a new datum $\mathbf{t}^{(N+1)}$ involves *marginalizing* over the parameters (and over anything else about which we are uncertain). For simplicity, let us assume that the weights $\mathbf{w}$ are the only uncertain quantities — the weight decay rate α and the model $\mathcal{H}$ itself are assumed to be fixed. Then by the sum rule, the predictive probability of a new target $\mathbf{t}^{(N+1)}$ at a location $\mathbf{x}^{(N+1)}$ is:

$$P(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, D, \alpha) = \int d^k\mathbf{w}\, P(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) P(\mathbf{w}|D, \alpha). \tag{32.14}$$

Thus the predictions are obtained by weighting together the prediction for a given $\mathbf{w}$,

$$\begin{array}{rcl} P(\mathbf{t}^{(N+1)}{=}1|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & y(\mathbf{x}^{(N+1)}; \mathbf{w}) \\ P(\mathbf{t}^{(N+1)}{=}0|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & 1 - y(\mathbf{x}^{(N+1)}; \mathbf{w}) \end{array} , \tag{32.15}$$

with a weight given by the posterior probability of $\mathbf{w}$, $P(\mathbf{w}|D, \alpha)$, which we most recently wrote down in equation (32.10). This posterior probability is

$$P(\mathbf{w}|D, \alpha) = \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{32.16}$$

where

$$Z_M = \int d^k\mathbf{w} \exp(-M(\mathbf{w})). \tag{32.17}$$

In summary, we can get the Bayesian predictions if we can find a way of computing the integral

$$P(\mathbf{t}^{(N+1)}{=}1|\mathbf{x}^{(N+1)}, D, \alpha) = \int d^k\mathbf{w}\, y(\mathbf{x}^{(N+1)}; \mathbf{w})\, \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{32.18}$$

which is the average of the output of the neuron at $\mathbf{x}^{(N+1)}$ under the posterior distribution of $\mathbf{w}$.

Implementation

How shall we compute the integral (32.18)? For our toy problem, the weight space is three dimensional; for a realistic neural network the dimensionality might be in the thousands.

Bayesian inference for general data modelling problems may be implemented by exact methods (chapter 20), by Monte Carlo sampling (chapter 23), or by deterministic approximate methods, for example, methods that make Gaussian approximations to $P(\mathbf{w}|D, \alpha)$ using Taylor expansions (chapter 22) or variational methods (chapter 26). For neural networks there are few analytic methods. The two main approaches to implementing Bayesian inference for neural networks are the Monte Carlo methods developed by Neal (1996) and the Gaussian approximation methods developed by MacKay (1991).

32.4 Monte Carlo implementation of a single neuron

First we will use a Monte Carlo approach in which the task of evaluating the integral (32.18) is solved by treating $y(\mathbf{x}^{(N+1)}; \mathbf{w})$ as a function 'f' of $\mathbf{w}$ whose mean we compute using

$$\langle f(\mathbf{w}) \rangle \simeq \frac{1}{R} \sum_r f(\mathbf{w}^{(r)}). \tag{32.19}$$

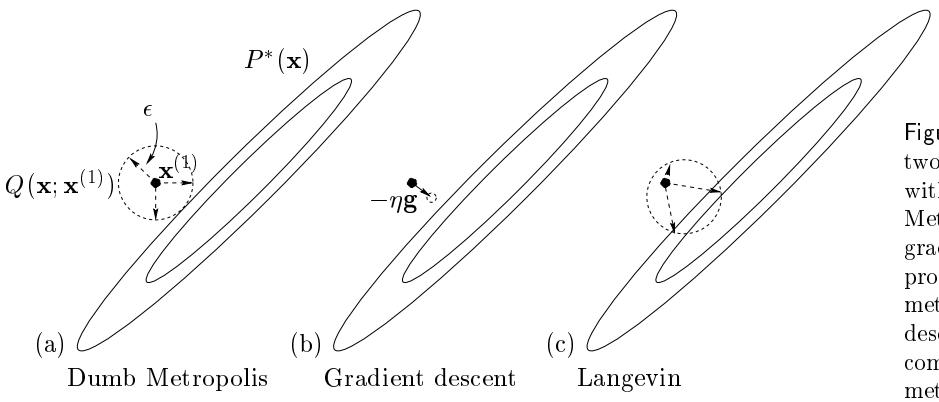


Figure 32.3. Langevin method in two dimensions (c), contrasted with a traditional 'dumb' Metropolis method (a) and with gradient descent (b). The proposal density of the Langevin method is given by 'gradient descent with noise'. For a complete description of the method see the text.

where $\{\mathbf{w}^{(r)}\}$ are random samples from $\frac{1}{Z_M}\exp(-M(\mathbf{w}))$ (c.f. equation (23.4)). We will obtain the samples using a Metropolis method (section 23.4). As an aside, a possible disadvantage of this Monte Carlo approach is that it is a poor way of estimating the probability of an improbable event, *i.e.*, a $P(t|D,\mathcal{H})$ that is very close to zero, if the improbable event is most likely to occur in conjunction with improbable parameter values.

Radford Neal's *hybrid Monte Carlo* method for neural networks is a sophisticated Metropolis method making use of gradient information. The Metropolis method we now demonstrate is a simple version of Hybrid Monte Carlo called the *Langevin Monte Carlo method*.

The Langevin Monte Carlo method

The algorithm is given in figure 32.4. It may be summarised as 'gradient descent with added noise', as shown pictorially in figure 32.3. A noise vector $\mathbf{p}$ is generated from a Gaussian with unit variance. The gradient $\mathbf{g}$ is computed, and a step in $\mathbf{w}$ is made, given by

$$\Delta\mathbf{w} = -\tfrac{1}{2}\epsilon^2\mathbf{g} + \epsilon\mathbf{p}. \tag{32.20}$$

Notice that if the $\epsilon\mathbf{p}$ term were omitted this would simply be gradient descent with learning rate $\eta = \frac{1}{2}\epsilon^2$. This step in $\mathbf{w}$ is accepted or rejected depending on the change in the value of the objective function $M(\mathbf{w})$ and on the change in gradient, with a probability of acceptance such that detailed balance holds.

Demonstration of Langevin method

The Langevin method is demonstrated in figures 32.6, 32.7 and 32.8. Here, the objective function is $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$, with $\alpha = 0.01$. These figures include, for comparison, the results of the previous optimization method using gradient descent on the same objective function (figure 30.6). It can be seen that the mean evolution of $\mathbf{w}$ is similar to the evolution of the parameters under gradient descent. The Monte Carlo method appears to have converged to the posterior distribution after about 10000 iterations.

The average acceptance rate during this simulation was 93%; only 7% of the proposed moves were rejected. Probably faster progress around the state space would have been made if a larger step size ϵ had been used, but the value was chosen so that $\eta = \frac{1}{2}\epsilon^2$ matched the step size of the earlier simulations.

```
g = gradM ( w ) ;                       # set gradient using initial w
M = findM ( w ) ;                       # set objective function too

for l = 1:L                             # loop L times
   p = randn ( size(w) ) ;              # initial momentum is Normal(0,1)
   H = p' * p / 2 + M ;                 # evaluate H(w,p)

*  p = p - epsilon * g / 2 ;            # make half-step in p
*  wnew = w + epsilon * p ;             # make step in w
*  gnew = gradM ( wnew ) ;              # find new gradient
*  p = p - epsilon * gnew / 2 ;         # make half-step in p

   Mnew = findM ( wnew ) ;              # find new objective function
   Hnew = p' * p / 2 + Mnew ;           # evaluate new value of H
   dH = Hnew - H ;                      # Decide whether to accept
   if ( dH < 0 )                accept = 1 ;
   elseif ( rand() < exp(-dH) ) accept = 1 ;
   else                         accept = 0 ;
   endif
   if ( accept )      g = gnew ;   w = wnew ;    M = Mnew ;    endif
endfor

function gM = gradM ( w )               # gradient of objective function
   a = x * w  ;                         # compute activations
   y = sigmoid(a) ;                     # compute outputs
   e = t - y   ;                        # compute errors
   g = - x' * e ;                       # compute the gradient of G(w)
   gM = alpha * w + g ;
endfunction

function M = findM ( w )                # objective function
   G = - (t' * log(y) + (1-t') * log( 1-y )) ;
   EW = w' * w / 2 ;
   M = G + alpha * EW ;
endfunction
```

Figure 32.4. `Octave` source code for the Langevin Monte Carlo method. To obtain a 'hybrid Monte Carlo' method, we repeat the four lines marked * multiple times (figure 32.5).

```
wnew = w ;
gwnew = gw ;
for tau = 1:Tau

   p = p - epsilon * gwnew / 2 ;     # make half-step in p
   wnew = wnew + epsilon * p ;       # make step in w

   gwnew = gradM ( wnew ) ;          # find new gradient
   p = p - epsilon * gwnew / 2 ;     # make half-step in p

endfor
```

Figure 32.5. `Octave` source code for the hybrid Monte Carlo method. The code is identical to that for the Langevin method in figure 32.4, except for the replacement of the four lines marked * in that figure by the above fragment.

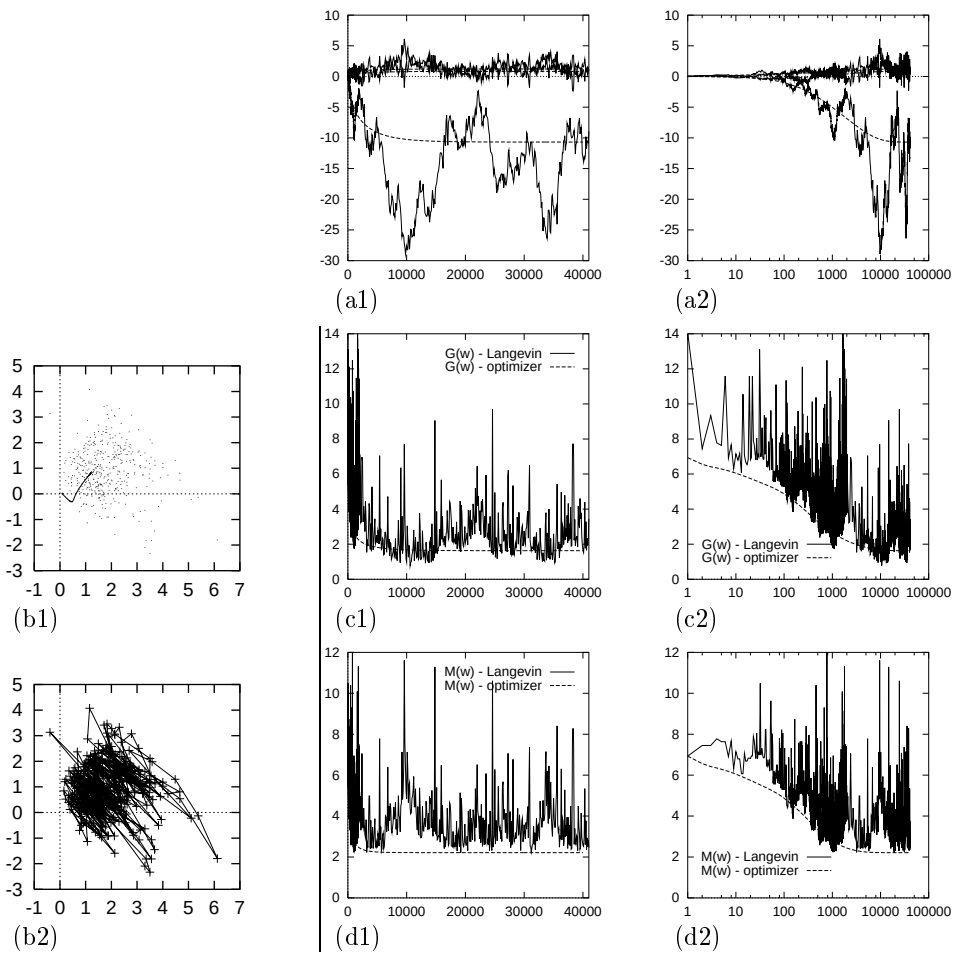


Figure 32.6. A single neuron learning under the Langevin Monte Carlo method. (a) Evolution of weights w_0, w_1 and w_2 as a function of number of iterations (a1: linear scale; a2: log scale). (b) Evolution of weights w_1 and w_2 in weight space. Also shown by a line is the evolution of the weights using the optimizer of figure 30.6. b2: every 100th state, connected by lines to show the simulation sequence. (c) The error function $G(\mathbf{w})$ as a function of number of iterations. Also shown is the error function during the optimization of figure 30.6. (d) The objective function $M(\mathbf{x})$ as a function of number of iterations. See also figures 32.7 and 32.8.

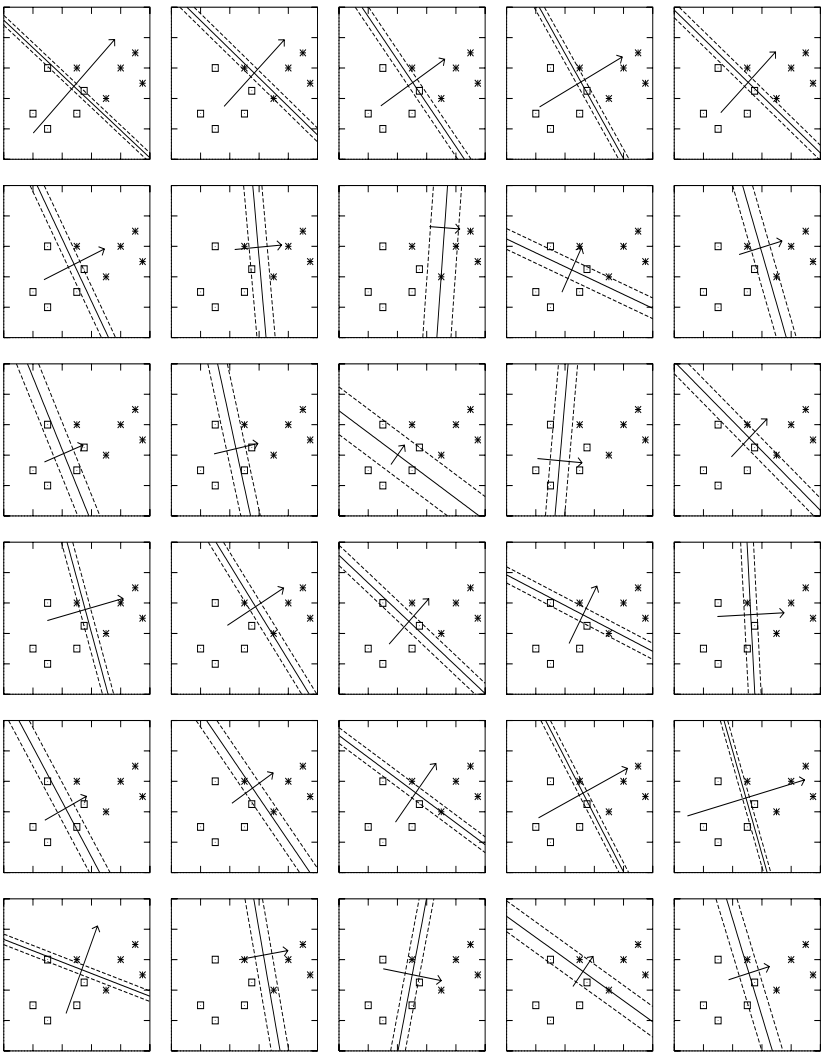


Figure 32.7. Samples obtained by the Langevin Monte Carlo method. The learning rate was set to $\eta = 0.01$ and the weight decay rate to $\alpha = 0.01$. The step size is given by $\epsilon = \sqrt{2\eta}$. The function performed by the neuron is shown by three of its contours every 1000 iterations from iteration 10000 to 40000. The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. Also shown is a vector proportional to (w_1, w_2).

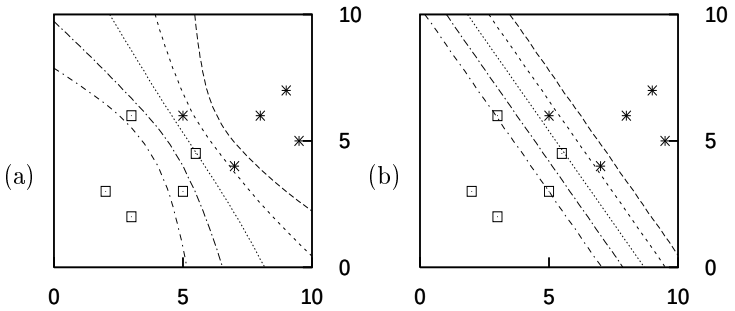


Figure 32.8. Bayesian predictions found by the Langevin Monte Carlo method compared with the predictions using the optimized parameters. (a) The predictive function obtained by averaging the predictions for 30 samples uniformly spaced between iterations 10000 and 40000, shown in figure 32.7. The contours shown are those corresponding to $a = 0, \pm 1, \pm 2$, namely $y = 0.5, 0.27$, 0.73, 0.12 and 0.88. (b) For contrast, the predictions given by the 'most probable' setting of the neuron's parameters, as given by optimization of $M(\mathbf{w})$.

Making Bayesian predictions

From the 10000th iteration to the 40000th, the weights were sampled every 1000 iterations and the corresponding functions of $\mathbf{x}$ are plotted in figure 32.7. There is a considerable variety of plausible functions. We obtain a Monte Carlo approximation to the Bayesian predictions by averaging these thirty functions of $\mathbf{x}$ together. The result is shown in figure 32.8 and contrasted with the predictions given by the optimized parameters. The Bayesian predictions become satisfyingly 'moderate' as we move away from the region of highest data density.

The Bayesian classifier is better able to identify the points where the classification is uncertain. This pleasing behaviour results simply from a mechanical application of the rules of probability.

Optimization and typicality

A final observation concerns the behaviour of the functions $G(\mathbf{w})$ and $M(\mathbf{w})$ during the Monte Carlo sampling process, compared with the values of G and M at the optimum $\mathbf{w}_{\rm MP}$ (figure 32.6). The function $G(\mathbf{w})$ fluctuates around the value of $G(\mathbf{w}_{\rm MP})$, though not in a symmetrical way. The function $M(\mathbf{w})$ also fluctuates, but it does not fluctuate *around* $M(\mathbf{w}_{\rm MP})$ — obviously it cannot, because M is minimized at $\mathbf{w}_{\rm MP}$, so M could not go any smaller — furthermore, M only very rarely decreases close to $M(\mathbf{w}_{\rm MP})$. In the language of information theory, *the typical set of* $\mathbf{w}$ *has different properties from the most probable state* $\mathbf{w}_{\rm MP}$.

A general message therefore emerges — applicable to all data models, not just neural networks: one should be cautious about making use of *optimized* parameters, as the properties of optimized parameters may be unrepresentative of the properties of typical, plausible parameters; and the predictions obtained using optimized parameters alone will often be unreasonably over–confident.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

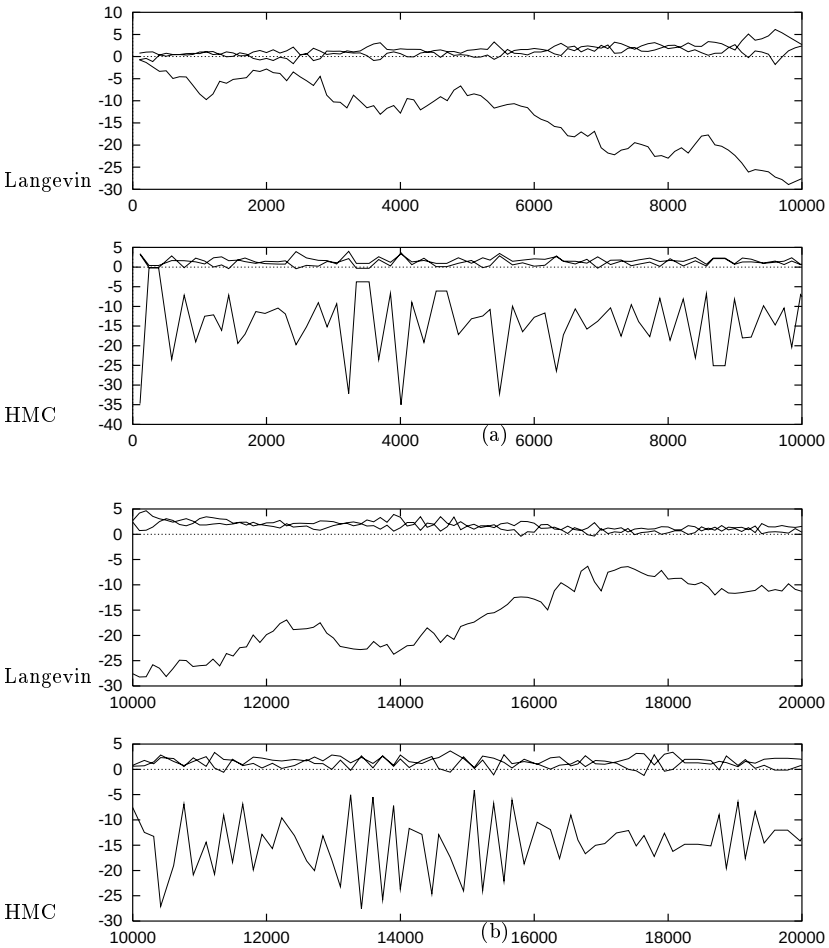


Figure 32.9. Comparison of sampling properties of the Langevin Monte Carlo method and the hybrid Monte Carlo (HMC) method. The horizontal axis is the number of gradient evalutations made. (a) The weights during the first 10000 iterations. (b) The weights during iterations 10000–20000.

Reducing random walk behaviour using hybrid Monte Carlo

As a final study of Monte Carlo methods, we now compare the Langevin Monte Carlo method with its big brother, the hybrid Monte Carlo method. The change to hybrid Monte Carlo is simple to implement, as shown in figure 32.5. Each single proposal makes use of multiple gradient evaluations along a dynamical trajectory in $\mathbf{w}, \mathbf{p}$ space, where $\mathbf{p}$ are the extra 'momentum' variables of the Langevin and hybrid Monte Carlo methods. The number of steps '`Tau`' was set at random to a number between 100 and 200 for each trajectory. The step size ϵ was kept fixed so as to retain comparability with the simulations that have gone before; Neal recommends that one randomize the step size in practical applications, however.

A comparison of sampling properties of the Langevin Monte Carlo method and the hybrid Monte Carlo method is made in figure 32.9. It can be seen that the autocorrelation of the state of the hybrid Monte Carlo simulation falls much more rapidly with simulation time than that of the Langevin method.

The rejection rate during this hybrid Monte Carlo simulation was 8%.

32.5 Implementing inference with Gaussian approximations

Physicists love to take non–linearities and locally linearize them, and they love to approximate probability distributions by Gaussians. Such approximations offer an alternative strategy for dealing with the integral

$$P(\mathbf{t}^{(N+1)} = 1 | \mathbf{x}^{(N+1)}, D, \alpha) = \int d^k \mathbf{w}\; y(\mathbf{x}^{(N+1)}; \mathbf{w}) \frac{1}{Z_M} \exp(-M(\mathbf{w})), \qquad (32.21)$$

which we just evaluated using Monte Carlo methods.

We start by making a Gaussian approximation to the posterior probability. We go to the minimum of $M(\mathbf{w})$ (using a gradient–based optimizer) and Taylor–expand M there:

$$M(\mathbf{w}) \simeq M(\mathbf{w}_{\rm MP}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_{\rm MP})^{\mathsf{T}} \mathbf{A} (\mathbf{w} - \mathbf{w}_{\rm MP}) + \ldots, \qquad (32.22)$$

where $\mathbf{A}$ is the matrix of second derivatives, also known as the **Hessian**, defined by

$$A_{ij} \equiv \left. \frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) \right|_{\mathbf{w}=\mathbf{w}_{\rm MP}} . \qquad (32.23)$$

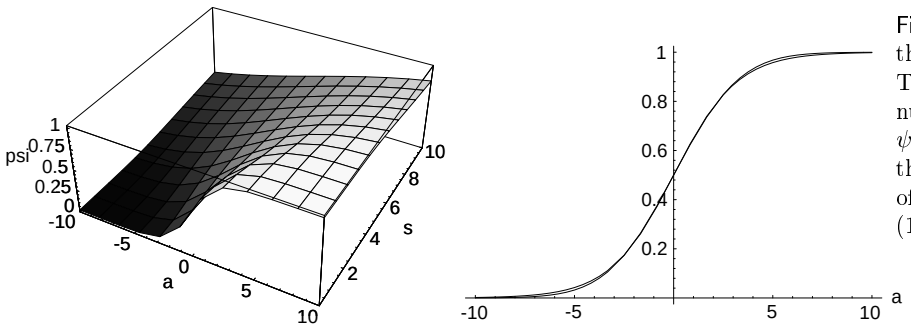
We thus define our Gaussian approximation:

$$Q(\mathbf{w}; \mathbf{w}_{\rm MP}, \mathbf{A}) = [\det(\mathbf{A}/2\pi)]^{1/2} \exp\left[-\frac{1}{2}(\mathbf{w} - \mathbf{w}_{\rm MP})^{\mathsf{T}} \mathbf{A} (\mathbf{w} - \mathbf{w}_{\rm MP}) \right]. \qquad (32.24)$$

We can think of the matrix $\mathbf{A}$ as defining **error bars** on $\mathbf{w}$. To be precise, Q is a normal distribution whose variance–covariance matrix is $\mathbf{A}^{-1}$.

Exercise 32.1:[A2] Show that the second derivative of $M(\mathbf{w})$ with respect to $\mathbf{w}$ is given by

$$\frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) = \sum_{n=1}^{N} f'(a^{(n)}) x_i^{(n)} x_j^{(n)} + \alpha \delta_{ij}, \qquad (32.25)$$



(a) (b)

Figure 32.10. Approximation to the marginalized probability. (a) The function $\psi(a, s^2)$, evaluated numerically. In (b) the functions $\psi(a, s^2)$ and $\phi(a, s^2)$ defined in the text are shown as a function of a for $s^2 = 4$. From MacKay (1992b).

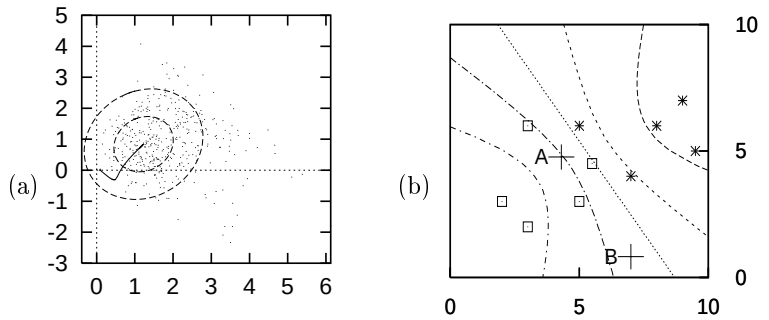


Figure 32.11. The Gaussian approximation and its approximate Bayesian predictions. (a) A projection of the Gaussian approximation onto the (w_1, w_2) plane of weight space. The one– and two–standard–deviation contours are shown. Also shown are the trajectory of the optimizer, and the Monte Carlo method's samples. The Monte Carlo samples are *not* involved in the creation of the Gaussian approximation. (b) The predictive function obtained from the Gaussian approximation and equation (32.31). The contours shown are those corresponding to $a = 0, \pm 1, \pm 2$, namely $y = 0.5, 0.27, 0.73, 0.12$ and 0.88.

where $f'(a)$ is the first derivative of

$$f(a) \equiv 1/(1 + e^{-a}) \tag{32.26}$$

which is given by

$$f'(a) = \frac{d}{da} f(a) = f(a)(1 - f(a)), \tag{32.27}$$

and

$$a^{(n)} = \sum_j w_j x_j^{(n)}. \tag{32.28}$$

Having computed the Hessian, our task is then to perform the integral (32.21) using our Gaussian approximation.

Calculating the marginalized probability

The output $y(\mathbf{x}; \mathbf{w})$ only depends on $\mathbf{w}$ through the scalar $a(\mathbf{x}; \mathbf{w})$, so let us reduce the dimensionality of the integral by finding the probability density of a. We are assuming a locally Gaussian posterior probability distribution over $\mathbf{w} = \mathbf{w}_{\rm MP} + \Delta\mathbf{w}$, $P(\mathbf{w}|D, \alpha) \simeq (1/Z_Q) \exp(-\frac{1}{2}\Delta\mathbf{w}^{\rm T}\mathbf{A}\Delta\mathbf{w})$. For our single neuron, the activation $a(\mathbf{x}; \mathbf{w})$ is a linear function of $\mathbf{w}$ with $\partial a/\partial \mathbf{w} = \mathbf{x}$, so for any $\mathbf{x}$, the activation a is Gaussian distributed.

Exercise 32.2:^{B2} Assuming $\mathbf{w}$ is Gaussian with mean $\mathbf{w}_{\rm MP}$ and variance–covariance matrix $\mathbf{A}^{-1}$, show that the probability distribution of $a(\mathbf{x})$ is

$$P(a|\mathbf{x}, D, \alpha) = \text{Normal}(a_{\rm MP}, s^2) = \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{(a - a_{\rm MP})^2}{2s^2}\right), \quad (32.29)$$

where $a_{\rm MP} = a(\mathbf{x}; \mathbf{w}_{\rm MP})$ and $s^2 = \mathbf{x}^{\rm T}\mathbf{A}^{-1}\mathbf{x}$.

This means that the marginalized output is:

$$P(t=1|\mathbf{x}, D, \alpha) = \psi(a_{\rm MP}, s^2) \equiv \int da\, f(a)\, \text{Normal}(a_{\rm MP}, s^2). \quad (32.30)$$

This is to be contrasted with the most probable network's output, $y(\mathbf{x}; \mathbf{w}_{\rm MP}) = f(a_{\rm MP})$. The integral of a sigmoid times a Gaussian cannot be solved analytically, but there are easy ways to handle such one–dimensional integrals; a simple numerical approximation to this integral (MacKay, 1992b) is:

$$\psi(a_{\rm MP}, s^2) \simeq \phi(a_{\rm MP}, s^2) \equiv f(\kappa(s) a_{\rm MP}) \quad (32.31)$$

with $\kappa = 1/\sqrt{1 + \pi s^2/8}$. This approximation is not globally accurate over $(a_{\rm MP}, s^2)$, (for large $s^2 > a$ the function should tend to an error function, not a logisitic) but it breaks down gracefully. The value of κ was chosen so that the approximation has the correct gain at $a_{\rm MP} = 0$, as $s^2 \to \infty$. A representative of this approximation is given in figure 32.10. The errors in the approximation are most significant when $a \gg s$.

Demonstration

Figure 32.11 shows the result of fitting a Gaussian approximation at the optimum $\mathbf{w}_{\rm MP}$, and the results of using that Gaussian approximation and equation (32.31) to make predictions. Comparing these predictions with those of the Langevin Monte Carlo method (figure 32.8) we observe that, whilst qualitatively the same, the two are clearly numerically different. So at least one of the two methods is not completely accurate.

Exercise 32.3:^{B2} Is the Gaussian approximation to $P(\mathbf{w}|D, \alpha)$ too heavy–tailed or too light–tailed, or both? It may help to consider $P(\mathbf{w}|D, \alpha)$ as a function of one parameter w_i and to think of the two distributions on a logarithmic scale.

Discuss the conditions under which the Gaussian approximation is most accurate.

Why marginalize?

If the output is immediately used to make a (0/1) decision and the costs associated with error are symmetrical, then the use of marginalized outputs under this Gaussian approximation will make no difference to the performance of the classifier, compared with using the outputs given by the most probable parameters, since both functions pass through 0.5 at $a_{\rm MP} = 0$. But these Bayesian outputs will make a difference if, for example, there is an option of saying 'I don't know', in addition to saying '0' and '1'. And even if there are just the two choices '0' and '1', if the costs associated with error are unequal, the the decision boundary will be some contour other than the 0.5 contour, and the boundary will be affected by marginalization.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Solutions to Chapter 32's exercises

Solution to exercise 32.2 (p. 396): The task is to show that when $\mathbf{w} \sim \text{Normal}(\mathbf{w}_{\text{MP}}, \mathbf{A}^{-1})$, the scalar $a = a(\mathbf{x}; \mathbf{w}_{\text{MP}}) + (\mathbf{w} - \mathbf{w}_{\text{MP}}) \cdot \mathbf{x}$ is Gaussian distributed with mean $a(\mathbf{x}; \mathbf{w}_{\text{MP}})$ and variance $s^2 = \mathbf{x}^{\mathsf{T}} \mathbf{A}^{-1} \mathbf{x}$.

This is easily shown by simply computing the mean and variance of a, then arguing that a's distribution must be Gaussian. The mean is

$$\langle a \rangle = \langle a(\mathbf{x}; \mathbf{w}_{\text{MP}}) + (\mathbf{w} - \mathbf{w}_{\text{MP}}) \cdot \mathbf{x} \rangle = a(\mathbf{x}; \mathbf{w}_{\text{MP}}) + \langle (\mathbf{w} - \mathbf{w}_{\text{MP}}) \rangle \cdot \mathbf{x} = a(\mathbf{x}; \mathbf{w}_{\text{MP}}).$$

The variance is

$$\begin{aligned} \left\langle (a - a(\mathbf{x}; \mathbf{w}_{\text{MP}}))^2 \right\rangle &= \langle \mathbf{x} \cdot (\mathbf{w} - \mathbf{w}_{\text{MP}})(\mathbf{w} - \mathbf{w}_{\text{MP}}) \cdot \mathbf{x} \rangle \qquad (32.32) \\ &= \mathbf{x}^{\mathsf{T}} \left\langle (\mathbf{w} - \mathbf{w}_{\text{MP}})(\mathbf{w} - \mathbf{w}_{\text{MP}})^{\mathsf{T}} \right\rangle \mathbf{x} \;=\; \mathbf{x}^{\mathsf{T}} \mathbf{A}^{-1} \mathbf{x}. \end{aligned}$$

Finally, we know a must be Gaussian because the marginals of a multivariate Gaussian are Gaussian.

Solution to exercise 32.3 (p. 396): In the case of a single data point, the likelihood function, as a function of one parameter w_i, is a sigmoid function; an example of a sigmoid function is shown on a logarithmic scale in figure 32.12a. The same figure shows a Gaussian distribution on a log scale. The prior distribution in this problem is assumed to be Gaussian; and the approximation Q is also a Gaussian, fitted at the maximum of the sum of the log likelihood and the log prior.

We can make some observations. The log likelihood and log prior are both concave functions, so the curvature of $\log Q$ must necessarily be greater than the curvature of the log prior. But asymptotically the log likelihood function is linear, so the curvature of the log posterior for large $|a|$ decreases to the curvature of the log prior. Thus for sufficiently large values of w_i, the approximating distribution is *lighter–tailed* than the true posterior.

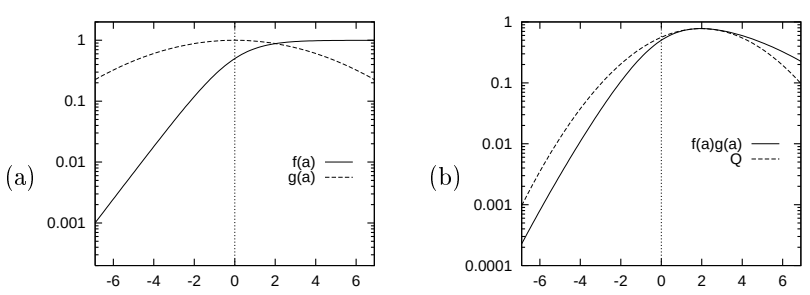


Figure 32.12. (a) The log of the sigmoid function $f(a) = 1/(1 + e^{-a})$ and the log of a Gaussian $g(a) \propto \text{Normal}(0, 4^2)$. (b) The product $P = f(a)g(a)$ and a Gaussian approximation to it, fitted at its mode. Notice that for a range of negative values of a, the Gaussian approximation Q is bigger than P, while for values of a beyond the mode, Q is smaller than P.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

This conclusion may be a little misleading however. If we multiply the likelihood and the prior and find the maximum and fit a Gaussian there, we might obtain a picture like figure 32.12b. Here issues of normalization have been ignored. The important point to note is that since the Gaussian is fitted at a point where the log likelihood's curvature is not very great, the approximating Gaussian's curvature is *too small* for a between $a_{\rm MP}$ and $-a_{\rm MP}$, with the consequence that the approximation Q is substantially *larger* than P for a wide range of negative values of a. On the other hand, for values of a greater than $a_{\rm MP}$, the approximation Q is smaller in value than P.

Thus whether Q is for practical purposes a heavy–tailed or light–tailed approximation to P depends which direction one looks in, and how far one looks.

The Gaussian approximation becomes most accurate when the amount of data increases, because the log of the posterior is a sum of more and more bent functions all of which contribute curvature to the log posterior, making it more and more Gaussian (c.f. figure 32.1). The greatest curvature is contributed by data points that are close (in terms of a) to the decision boundary, so the Gaussian approximation becomes good fastest if the optimized parameters are such that all the points are close to the decision boundary, that is, if the data are noisy.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

33

The Hopfield network

We have now spent three chapters studying the single neuron. The time has come to connect multiple neurons together, making the output of one neuron be the input to another, so as to make neural networks.

Neural networks can be divided into two classes on the basis of their connectivity.

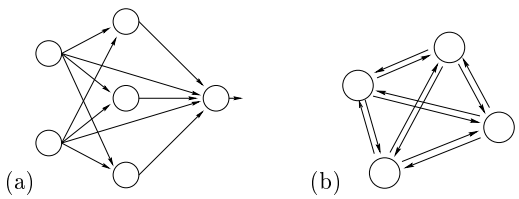


Figure 33.1. (a) A feedforward network. (b) A feedback network.

Feedforward networks. In a feedforward network, all the connections are directed such that the network forms a directed acyclic graph.

Feedback networks. Any network that is not a feedforward network will be called a feedback network.

In this chapter we will discuss a fully connected feedback network called the Hopfield network. The weights in the Hopfield network are constrained to be *symmetric*, *i.e.*, the weight from neuron i to neuron j is equal to the weight from neuron j to neuron i.

Hopfield networks have two applications. First, they can act as *associative memories*. Second, they can be used to solve *optimization problems*. We will discuss the idea of associative memory first.

33.1 Hebbian learning

In chapter 29, we discussed the contrast between traditional digital memories and biological memories. Perhaps the most striking difference is the *associative* nature of biological memory.

A simple model due to Donald Hebb captures the idea of associative memory. Imagine that all neurons are connected to each other, and imagine that the weights are adjusted so that the weights between neurons whose activities

are *positively correlated* are *increased*:

$$\frac{dw_{ij}}{dt} \sim \text{Correlation}(x_i, x_j). \tag{33.1}$$

Now imagine that neuron m is associated with stimulus m (for example, the smell of a banana), such that when stimulus m is present, the activity of neuron m increases; and neuron n is associated with another stimulus, n (for example, the sight of a yellow object). If these two stimuli co–occur in the environment, then the Hebbian learning rule (33.1) will increase the values of the weights w_{nm} and w_{mn}. This means that when, on a later occasion, stimulus n occurs in isolation, making the activity x_n large, the positive weight from n to m will cause neuron m also to be activated. Thus the response to the sight of a yellow object is an automatic association with the smell of a banana. We could call this 'pattern completion'. It should be emphasised that no outside intelligence or teacher is required for this associative memory to work. No signal is needed to indicate that a correlation has been detected or that an association should be made. The unsupervised, local learning algorithm and the unsupervised, local activity rule spontaneously produce associative memory.

This idea seems so simple and so effective that it must be relevant to how memories work in the brain.

33.2 Definition of the binary Hopfield network

Convention for weights. Our convention in general will be that w_{ij} denotes the connection *from* neuron j *to* neuron i.

Architecture. A Hopfield network consists of I neurons. They are fully connected through **symmetric, bidirectional** connections with weights $w_{ij} = w_{ji}$. There are no self-connections, so $w_{ii} = 0$ for all i. Biases w_{i0} may be included (these may be viewed as weights from a neuron '0' whose activity is permanently $x_0 = 1$). We will denote the activity of neuron i (its output) by x_i.

Activity rule. Roughly, a Hopfield network's activity rule is for each neuron to update its state as if it were a single neuron with the threshold activation function

$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases} \tag{33.2}$$

Since there is **feedback** in a Hopfield network (every neuron's output is an input to all the other neurons) we will have to specify an order for the updates to occur. The updates may be synchronous or asynchronous.

synchronous updates — all neurons compute their activations

$$a_i = \sum_j w_{ij} x_j \tag{33.3}$$

then update their states simultaneously to

$$x_i = \Theta(a_i); \tag{33.4}$$

asynchronous updates — one neuron at a time computes its activation and updates its state. The sequence of selected neurons may be a fixed sequence or a random sequence.

The properties of a Hopfield network may be sensitive to the above choices.

Learning rule. The learning rule sets the weights with the intention that a set of desired **memories** $\{\mathbf{x}^{(n)}\}$ will be **stable states** of the Hopfield network's activity rule. Each memory is a binary pattern, with $x_i \in \{-1, 1\}$.

The weights are set using the **sum of outer products** or **Hebb rule**,

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)}, \tag{33.5}$$

where η is an unimportant constant. To prevent the largest possible weight from growing with N we might choose to set $\eta = 1/N$.

Exercise 33.1:[A1] Explain why the value of η is not important for the Hopfield network defined above.

33.3 Definition of the continuous Hopfield network

Using the identical architecture we can define a Hopfield network whose activities are real numbers between -1 and 1.

Activity rule. A Hopfield network's activity rule is for each neuron to update its state as if it were a single neuron with the sigmoid (tanh) activation function.

The updates may be synchronous or asynchronous, and involve the equations

$$a_i = \sum_j w_{ij} x_j \tag{33.6}$$

and

$$x_i = \tanh(a_i). \tag{33.7}$$

The learning rule is the same as in the binary Hopfield network, but the value of η becomes relevant. Alternatively, we may fix η and introduce a **gain** $\beta \in (0, \infty)$ into the activation function:

$$x_i = \tanh(\beta a_i). \tag{33.8}$$

Exercise 33.2:[A1] Where have we encountered equations (33.6, 33.7 & 33.8) before?

33.4 Convergence of the Hopfield network

The hope is that the Hopfield networks we have defined will perform associative memory recall, as shown schematically in figure 33.2. We hope that the activity rule of a Hopfield network will take a partial memory or a corrupted memory, and perform pattern completion or error correction to restore the original memory.

(a)

moscow_____russia
lima_____peru
london_____england
tokyo_____japan
edinburgh_scotland
ottawa_____canada
oslo_____norway
stockholm___sweden
paris_____france

(b)
```
moscow___::::::::::  ⟹  moscow_____russia
::::::::::__canada   ⟹  ottawa_____canada
```

(c)
```
otowa_____canada     ⟹  ottawa_____canada
egindurrh_sxotland   ⟹  edinburgh_scotland
```

Figure 33.2. Associative memory (schematic). (a) A list of desired memories. (b) The first function of an associative memory is pattern completion, given a partial pattern. (c) The second function of a memory is error correction.

But why should we expect *any* pattern to be stable under the activity rule, let alone the desired memories?

We address the continuous Hopfield network, since the binary network is a special case of it. We have already encountered the activity rule (33.6, 33.8) when we discussed variational methods: when we approximated the spin system whose energy function was

$$E(\mathbf{x};\mathbf{J}) = -\frac{1}{2}\sum_{m,n} J_{mn}x_m x_n - \sum_n h_n x_n \tag{33.9}$$

with a separable distribution

$$Q(\mathbf{x};\mathbf{a}) = \frac{1}{Z_Q}\exp\left(\sum_n a_n x_n\right) \tag{33.10}$$

and optimized the latter so as to minimize the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})E(\mathbf{x};\mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})\log\frac{1}{Q(\mathbf{x};\mathbf{a})}, \tag{33.11}$$

we found that the pair of iterative equations

$$a_m = \beta\left(\sum_n J_{mn}\bar{x}_n + h_m\right) \tag{33.12}$$

and

$$\bar{x}_n = \tanh(a_n) \tag{33.13}$$

were guaranteed to decrease the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\left(-\frac{1}{2}\sum_{m,n} J_{mn}\bar{x}_m\bar{x}_n - \sum_n h_n\bar{x}_n\right) - \sum_n H_2^{(e)}(q_n). \tag{33.14}$$

This objective function is lower–bounded by the true free energy βF.

If we simply replace J by w, $\bar{x}$ by x, and h_n by w_{i0}, we see that the equations of the Hopfield network are identical to a set of mean field equations that minimize

$$\beta\tilde{F}(\mathbf{x}) = -\beta\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x} - \sum_i H_2^{(e)}[(1+x_i)/2]. \tag{33.15}$$

There is a general name for a function that decreases under the dynamical evolution of a system and that is bounded below: such a function is a *Lyapunov function* for the system. It is useful to be able to prove the existence of Lyapunov functions: if a system has a Lyapunov function then its dynamics are bound to settle down to a *fixed point*, which is a local minimum of the Lyapunov function, or a *limit cycle*, along which the Lyapunov function is a constant. Chaotic behaviour is not possible for a system with a Lyapunov function. If a system has a Lyapunov function then its state space can be divided into *basins of attraction*, one basin associated with each attractor.

So, the continuous Hopfield network's activity rules (if implemented asynchronously) have a Lyapunov function. This Lyapunov function is a convex function of each parameter a_i so a Hopfield network's dynamics will always converge to a stable fixed point.

This convergence proof depends crucially on the fact that the Hopfield network's connections are *symmetric*. It also depends on the updates being made asynchronously.

Exercise 33.3:[A2] Show by constructing an example that if a feedback network does not have symmetric connections then its dynamics may fail to converge to a fixed point.

Exercise 33.4:[A2] Show by constructing an example that if a Hopfield network is updated synchronously that, from some initial conditions, it may fail to converge to a fixed point.

33.5 The associative memory in action

Figure 33.3 shows the dynamics of a 25-unit binary Hopfield network that has learnt four patterns by Hebbian learning. The four patterns are displayed as five by five binary images in figure 33.3(a). Parts (b-m) show the state of the network, iteration by iteration, all 25 units being updated asynchronously in each iteration. For an initial condition randomly perturbed from a memory, it often only takes one iteration for all the errors to be corrected. The network has more stable states in addition to the four desired memories. First, the inverse of any stable state is also a stable state. There are also several stable states that can be interpreted as mixtures of the memories.

Brain damage

The network can be severely damaged and still work fine as an associative memory. If we take the 300 weights of the network shown in figure 33.3 and randomly set 50 or 100 of them to zero, we still find that the desired memories are attracting stable states. Imagine a digital computer which still works fine even when 20% of its components are destroyed!

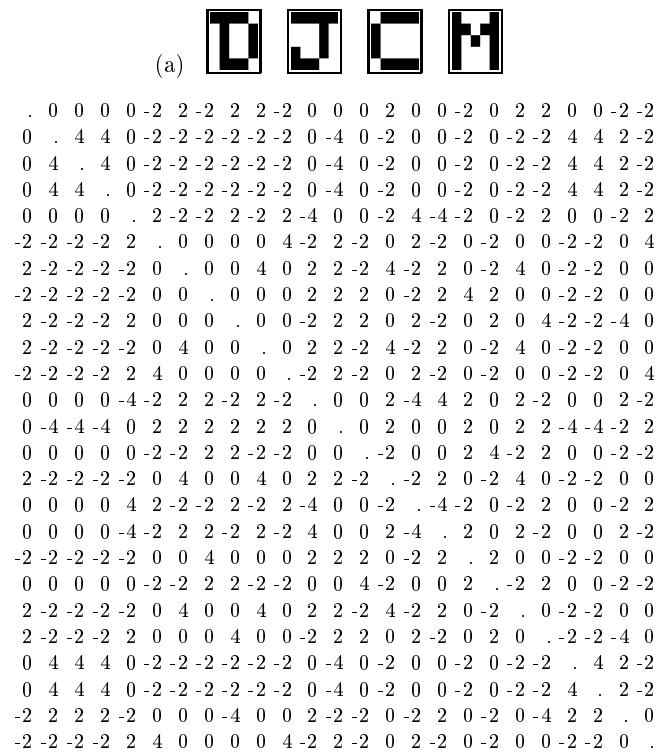


Figure 33.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to other stable states; in (i), the stable state looks like a mixture of two memories, 'D' and 'J'; stable state (j) is like a mixture of 'J' and 'C'; in (k), we find a corrupted version of the 'M' memory (two bits distant); in (l) a corrupted version of 'J' (four bits distant) and in (m), a state which looks spurious until we recognize that it is the inverse of the stable state (l).

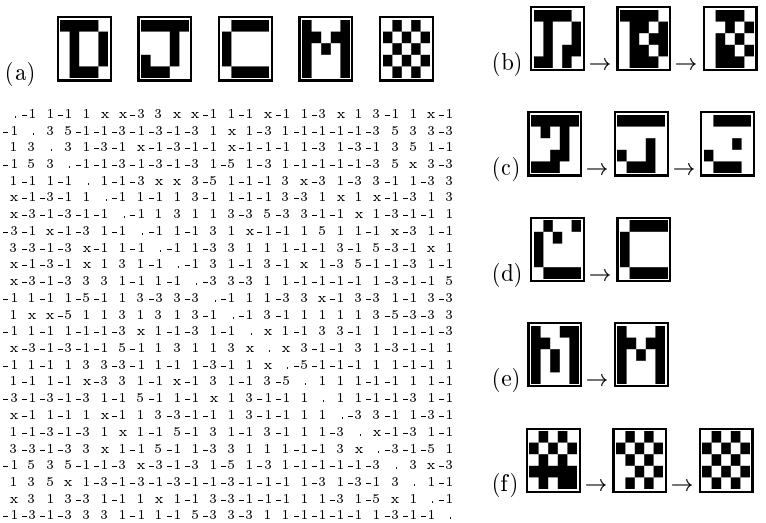


Figure 33.4. Hopfield network storing five memories, and suffering deletion of 26 of its 300 weights. (a) The five memories, and the weights of the network, with deleted weights shown by 'x'. (b–f) Initial states that differ by three random bits from a memory: some are restored, but some converge to other states.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

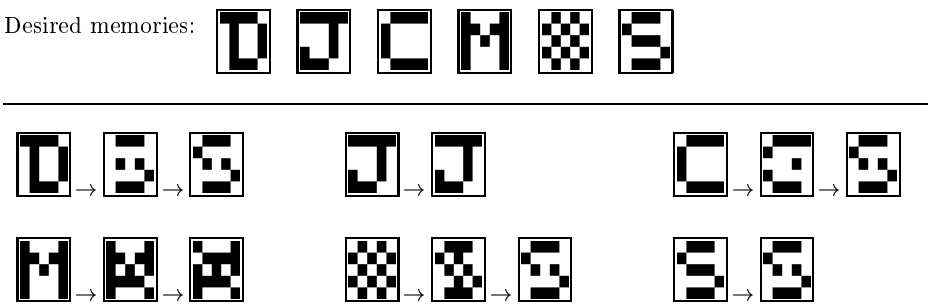


Figure 33.5. An overloaded Hopfield network trained on six memories, most of which are not stable.

Desired memories
moscow_____russia
lima_____peru
london_____england
tokyo_____japan
edinburgh_scotland
ottawa_____canada
oslo_____norway
stockholm___sweden
paris_____france

$\rightarrow$ **W** $\rightarrow$

Attracting stable states	
moscow_____russia	
lima_____peru	
londog_____englard	(1)
tonco_____japan	(1)
edinburgh_scotland	(2)
oslo_____norway	
stockholm___sweden	
paris_____france	
wzkmhewn__xqwqwpoq	(3)
paris_____sweden	(4)
ecnarf_____sirap	(4)

Figure 33.6. Failure modes of a Hopfield network (highly schematic). A list of desired memories, and the resulting list of attracting stable states of a Hopfield network. This list shows (1) some memories that are retained with a small number of errors; (2) desired memories that are completely lost (there is no attracting stable state at the desired memory or near it); (3) spurious stable states unrelated to the original list; (4) spurious stable states that are confabulations of desired memories.

More memories

We can squash more memories into the network too. Figure 33.4(a) shows a set of five memories. When we train the network with Hebbian learning, all five memories are stable states, even when 26 of the weights are randomly deleted (as shown by the 'x's in the figure). However, the basins of attraction are smaller when we have more memories: figure 33.4(b–f) shows the dynamics resulting from randomly chosen starting states close to each of the memories (3 bits flipped). Only three of the memories are recovered correctly.

If we try to store too many patterns, the associative memory fails catastrophically. If we add a sixth pattern, as shown in figure 33.5, only one of the patterns is stable; the others all flow into one of two spurious stable states.

33.6 The continuous–time continuous Hopfield network

The fact that the Hopfield network's properties are not robust to the minor change from asynchronous to synchronous updates might be a cause for concern; can this model be a useful model of biological networks? It turns out that once we move to a continuous–time version of the Hopfield networks, this issue melts away.

We assume that each neuron's activity x_i is a continuous function of time $x_i(t)$ and that the activations $a_i(t)$ are computed instantaneously in accordance with

$$a_i(t) = \sum_j w_{ij} x_j(t). \tag{33.16}$$

The neuron's response to its activation is assumed to be mediated by the differential equation:

$$\frac{d}{dt} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)), \tag{33.17}$$

where $f(a)$ is the activation function, for example $f(a) = \tanh(a)$. For a steady activation a_i, the activity $x_i(t)$ relaxes exponentially to $f(a_i)$ with time–constant τ.

Now, here is the nice result: as long as the weight matrix is symmetric, this system has the variational free energy (33.15) as its Lyapunov function.

Exercise 33.5:[B1] Prove that the variational free energy $\tilde{F}(\mathbf{x})$ is a Lyapunov function for the continuous–time Hopfield network by computing $\frac{d}{dt}\tilde{F}$.

The simplest Lyapunov functions are those on which a system's dynamics perform steepest descents, that is $\frac{d}{dt}x_i(t) \propto \frac{\partial}{\partial x_i}L$. In this case, it is not quite so simple, but every component of $\frac{d}{dt}x_i(t)$ does have the same sign as $\frac{\partial}{\partial x_i}\tilde{F}$, which means that with an appropriately defined metric, the Hopfield network dynamics do perform steepest descents on $\tilde{F}(\mathbf{x})$.

33.7 The capacity of the Hopfield network

One way in which we viewed learning in the single neuron was as communication — communication of the labels of the training data set from one point in time to a later point in time. We found that the capacity of a linear threshold neuron was 2 bits per weight.

Similarly, we might view the Hopfield associative memory as a communication channel (figure 33.6). A list of desired memories is encoded into a set of weights $\mathbf{W}$ using the Hebb rule of equation (33.5), or perhaps some other learning rule. The receiver, receiving the weights $\mathbf{W}$ only, finds the stable states of the Hopfield network, which he interprets to be the original memories. This communication system can fail in various ways, as illustrated in the figure.

1. Individual bits in some memories might be corrupted, that is, a stable state of the Hopfield network is displaced a little from the desired memory.

2. Entire memories might be absent from the list of attractors of the network; or a stable state might be present but have such a small basin of attraction that it is of no use for pattern completion and error correction.

3. Spurious additional memories unrelated to the desired memories might be present.

4. Spurious additional memories derived from the desired memories by operations such as mixing and inversion may also be present.

Of these failure modes, modes 1 and 2 are clearly undesirable, mode 2 especially so. Mode 3 might not matter so much as long as each of the desired memories has a large basin of attraction. The fourth failure mode might in some contexts actually be viewed as beneficial. For example, if a network is required to memorize examples of valid sentences such as 'John loves Mary' and 'John gets cake', we might be happy to find that 'John loves cake' was also a stable state of the network. We might call this behaviour 'generalization'.

The capacity of a Hopfield network with I neurons might be defined to be the number of random patterns N that can be stored without failure–mode 2 having substantial probability. If we also require failure–mode 1 to have tiny probability then the resulting capacity is much smaller. We will study the capacity of the Hopfield network in the next section.

The capacity of the Hopfield network — stringent definition

We will first explore the information storage capabilities of a binary Hopfield network that learns using the Hebb rule by considering the stability of just one bit of one of the desired patterns, assuming that the state of the network is set to that desired pattern $\mathbf{x}^{(n)}$. We will assume that the patterns to be stored are randomly selected binary patterns.

The activation of a particular neuron i is

$$a_i = \sum_j w_{ij} x_j^{(n)}, \tag{33.18}$$

where the weights are, for $i \neq j$,

$$w_{ij} = x_i^{(n)} x_j^{(n)} + \sum_{m \neq n} x_i^{(m)} x_j^{(m)}. \tag{33.19}$$

Here we have split $\mathbf{W}$ into two terms, the first of which will contribute 'signal', reinforcing the desired memory, and the second 'noise'. Substituting for w_{ij}, the activation is

$$a_i = \sum_{j \neq i} x_i^{(n)} x_j^{(n)} x_j^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)} \tag{33.20}$$

$$= (I-1) x_i^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)}. \tag{33.21}$$

The first term is $(I-1)$ times the desired state $x_i^{(n)}$. If this were the only term, it would keep the neuron firmly clamped in the desired state. The second term is a sum of $(I-1)(N-1)$ random quantities $x_i^{(m)} x_j^{(m)} x_j^{(n)}$. A moment's reflection confirms that these quantities are independent random binary variables with mean 0 and variance 1.

Thus, considering the statistics of a_i under the ensemble of random patterns, we conclude that a_i has mean $(I-1)x_i^{(n)}$ and variance $(I-1)(N-1)$.

For brevity, we will now assume I and N are large enough that we can neglect the distinction between I and $I-1$, and between N and $N-1$. Then we can restate our conclusion: a_i is Gaussian distributed with mean $Ix_i^{(n)}$ and variance IN.

What then is the probability that the selected bit is stable, if we put the network into the state $\mathbf{x}^{(n)}$? The probability that bit i will flip on the first iteration of the Hopfield network's dynamics is

$$P(i \text{ unstable}) = \Phi\left(-\frac{I}{\sqrt{IN}}\right) = \Phi\left(-\frac{1}{\sqrt{N/I}}\right), \tag{33.22}$$

where

$$\Phi(z) \equiv \int_{-\infty}^{z} dz \, \tfrac{1}{\sqrt{2\pi}} e^{-z^2/2}. \tag{33.23}$$

The important quantity N/I is the ratio of the number of patterns stored to the number of neurons. If, for example, we try to store $N \simeq 0.18I$ patterns in the Hopfield network then there is a chance of 1% that a specified bit in a specified pattern will be unstable on the first iteration.

We are now in a position to derive our first capacity result, for the case where no corruption of the desired memories is permitted.

Exercise 33.6:^{B2} Assume that we wish all the desired patterns to be completely stable — we don't want any of the bits to flip when the network is put into any desired pattern state — and the total probability of any error at all is required to be less than a small number ϵ. Using the approximation to the error function for large z,

$$\Phi(-z) \simeq \frac{1}{\sqrt{2\pi}} \frac{e^{-z^2/2}}{z}, \tag{33.24}$$

show that the maximum number of patterns that can be stored, $N_{\max}$, is

$$N_{\max} \simeq \frac{I}{4 \log I + 2 \log(1/\epsilon)}. \tag{33.25}$$

If, however, we allow a small amount of corruption of memories to occur, the number of patterns that can be stored increases.

The statistical physicists' capacity

The analysis that led to equation (33.22) tells us that if we try to store $N \simeq 0.18I$ patterns in the Hopfield network then, starting from a desired memory, about 1% of the bits will be unstable on the first iteration. Our analysis does not shed light on what is expected to happen on subsequent iterations. The flipping of these bits might make some of the other bits unstable too, causing an increasing number of bits to be flipped. This process might lead to an avalanche in which the network's state ends up a long way from the desired memory.

In fact, when N/I is large, such avalanches do happen. When N/I is small, they tend not to — there is a stable state near to the desired memory. For the limit of large I, Amit *et al.* (1985) have used methods from statistical physics

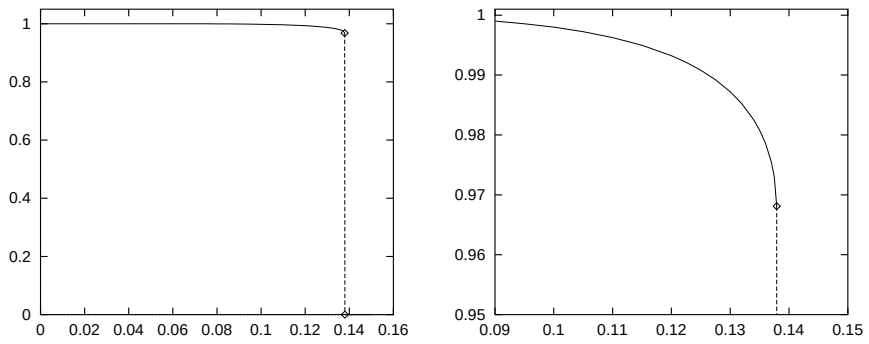


Figure 33.7. Overlap between a desired memory and the stable state nearest to it as a function of the loading fraction N/I. The overlap is defined to be the scaled inner product $\sum_i x_i x_i^{(n)}/I$, which is 1 when recall is perfect and zero when the stable state has 50% of the bits flipped. There is an abrupt transition at $N/I = 0.138$, where the overlap drops from 0.97 to zero.

to find numerically the transition between these two behaviours. There is a sharp discontinuity at

$$N_{\rm crit} = 0.138I. \tag{33.26}$$

Below this critical value, there is likely to be a stable state near every desired memory, in which a small fraction of the bits are flipped. When N/I exceeds 0.138, the system has only spurious stable states, known as *spin glass states*, none of which is correlated with any of the desired memories. Just below the critical value, the fraction of bits that are flipped when a desired memory has evolved to its associated stable state is 1.6%. Figure 33.7 shows the overlap between the desired memory and the nearest stable state as a function of N/I.

Some other transitions in properties of the model occur at some additional values of N/I, as summarised below.

For all N/I, stable spin glass states exist.

For $N/I > 0.138$, these spin glass states are the only stable states.

For $N/I \in (0, 0.138)$, there are stable states close to the desired memories, and spurious spin glass states.

For $N/I \in (0, 0.05)$, the stable states associated with the desired memories have lower energy than the spin glass states.

For $N/I \in (0.05, 0.138)$, the spin glass states dominate — there are spin glass states that have lower energy than the stable states associated with the desired memories.

For $N/I \in (0, 0.03)$, there are additional *mixture* states, which are combinations of several desired memories. These stable states do not have as low energy as the stable states associated with the desired memories.

In conclusion, the capacity of the Hopfield network with I neurons, if we define the capacity in terms of the abrupt discontinuity discussed above, is $0.138I$ random binary patterns, each of length I, each of which is received with 1.6% of its bits flipped. In bits, this capacity is

$$0.138I^2 \times (1 - H_2(0.016)) = 0.122I^2 \text{ bits.} \tag{33.27}$$

Since there are $I^2/2$ weights in the network, we can also express the capacity as *0.24 bits per weight.*

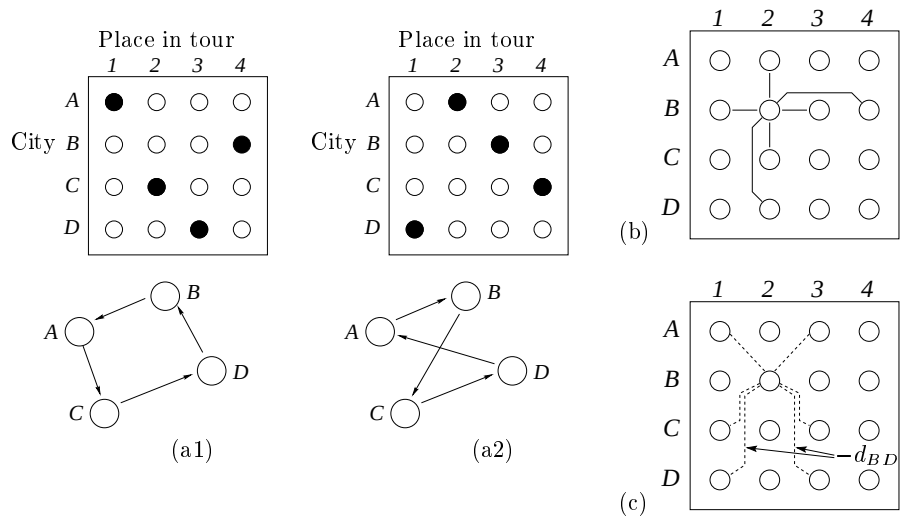


Figure 33.8. Hopfield network for solving a travelling salesman problem with $K = 4$ cities. (a1,2) Two solution states of the 16-neuron network, with activites represented by black = 1, white = 0; and the tours corresponding to these network states. (b) The negative weights between node $B2$ and other nodes; these weights enforce validity of a tour. (c) The negative weights that embody the distance objective function.

33.8 Hopfield networks for optimization problems

Since a Hopfield network's dynamics minimize an energy function, it is natural to ask whether we can map interesting optimization problems onto Hopfield networks. Biological data processing problems often involve an element of *constraint satisfaction* — in scene interpretation, for example, one might wish to infer the spatial location, orientation, brightness and texture of each visible element, and which visible elements are connected together in objects. These inferences are constrained by the given data and by prior knowledge about continuity of objects.

Hopfield and Tank suggested that one might take an interesting constraint satisfaction problem and design the weights of a binary or continuous Hopfield network such that the settling process of the network would minimize the objective function of the problem.

The travelling salesman problem

The classic constraint satisfaction problem to which Hopfield networks have been applied is the travelling salesman problem.

A set of K cities is given, and a matrix of the $K(K-1)/2$ distances between those cities. The task is to find a closed tour of the cities, visiting each city once, that has the smallest total distance. The travelling salesman problem is equivalent in difficulty to an NP-complete problem.

The method suggested by Hopfield and Tank is to represent a tentative solution to the problem by the state of a network with $I = K^2$ neurons arranged in a square, with each neuron representing the hypothesis that a particular city comes at a particular point in the tour. It will be convenient to consider the states of the neurons as being between 0 and 1 rather than -1 and 1. Two solution states for a four-city travelling salesman problem are shown in figure 33.8(a).

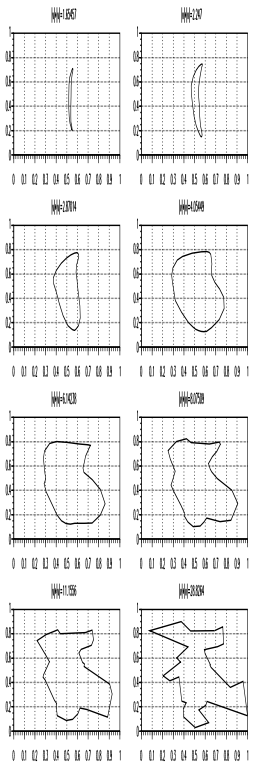
The weights in the Hopfield network play two roles. First, they must define

an energy function which is only minimized when the state of the network represents a **valid** tour. A valid state is one which looks like a permutation matrix, having exactly one '1' in every row and one '1' in every column. This rule can be enforced by putting large negative weights between any pair of neurons that are in the same row or the same column, and setting a positive bias for all neurons to ensure that K neurons do turn on. Figure 33.8(b) shows the negative weights that are connected to one neuron, '$B2$', which represents the statement 'city B comes second in the tour'.

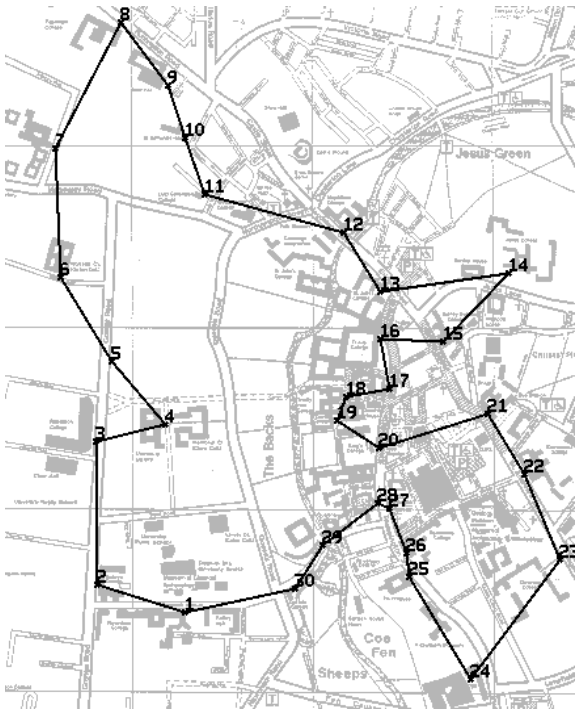
Second, the weights must encode the objective function — the total distance. This can be done by putting positive weights proportional to the appropriate distances between the nodes in adjacent columns. For example, between the B and D nodes in adjacent columns, the weight would be $-d_{BD}$. The negative weights that are connected to neuron $B2$ are shown in figure 33.8(c). The result is that when the network is in a valid state, its total energy will be the total distance of the corresponding tour, plus a constant given by the energy associated with the biases.

Now, since a Hopfield network minimizes its energy, it is hoped that the binary or continuous Hopfield network's dynamics will take the state to a minimum that is a valid tour and which might be an optimal tour. This hope is not fulfilled for large travelling salesman problems, however, without some careful modifications. We have not specified the size of the weights that enforce the tour's validity, relative to the size of the distance weights, and setting this scale factor poses difficulties. If 'large' validity-enforcing weights are used, the network's dynamics will rattle into a valid state with little regard for the distances. If 'small' validity-enforcing weights are used, it is possible that the distance weights will cause the network to adopt an *invalid* state that has lower energy than any valid state. Our original formulation of the energy function puts the objective function and the solution's validity in potential conflict with each other. This difficulty has been resolved by the work of Sree Aiyer (1991), who showed how to modify the distance weights so that they would not interfere with the solution's validity, and how to define a continuous Hopfield network whose dynamics are at all times confined to a 'valid subspace'. Aiyer used a 'graduated non-convexity' or 'deterministic annealing' approach to find good solutions using these Hopfield networks. The deterministic annealing approach involves gradually increasing the gain β of the neurons in the network from 0 to ∞, at which point the state of the network corresponds to a valid tour. A sequence of trajectories generated by applying this method to a thirty-city travelling salesman problem is shown in figure 33.9(a).

A solution to the 'travelling scholar problem' found by Aiyer using a continuous Hopfield network is shown in figure 33.9(b).



(a)



(b)

Figure 33.9. (a) Evolution of the state of a continous Hopfield network solving a travelling salesman problem using Aiyer's (1991) graduated non-convexity method; the state of the network is projected into the two-dimensional space in which the cities are located by finding the centre of mass for each point in the tour, using the neuron activities as the mass function. (b) The travelling scholar problem. The shortest tour linking the 27 Cambridge Colleges, the Engineering Department, the University Library, and Sree Aiyer's house. From Aiyer (1991).

Solutions to Chapter 33's exercises

Solution to exercise 33.3 (p. 403): Take a binary feedback network with 2 neurons and let $w_{12} = 1$ and $w_{21} = -1$. Then whenever neuron 1 is updated, it will match neuron 2, and whenever neuron 2 is updated, it will flip to the opposite state from neuron 1. There is no stable state.

Solution to exercise 33.4 (p. 403): Take a binary Hopfield network with 2 neurons and let $w_{12} = w_{21} = 1$, and let the initial condition be $x_1 = 1$, $x_2 = -1$. Then if the dynamics are synchronous, on every iteration both neurons will flip their state. The dynamics do not converge to a fixed point.

34

From Hopfield networks to Boltzmann machines

34.1 The Boltzmann machine

We have noticed that the binary Hopfield network minimizes an energy function

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x} \tag{34.1}$$

and that the continuous Hopfield network with activation function $x_n = \tanh(a_n)$ can be viewed as *approximating* the probability distribution associated with that energy function,

$$P(\mathbf{x}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})}\exp[-E(\mathbf{x})] = \frac{1}{Z(\mathbf{W})}\exp\left[\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x}\right]. \tag{34.2}$$

These observations motivate the idea of working with a neural network model that actually *implements* the above probability distribution.

The *stochastic Hopfield network* or *Boltzmann machine* has the following activity rule:

Activity rule of Boltzmann machine: after computing the activation a_n,

$$\begin{aligned} &\text{set } x_n = +1 \text{ with probability } \frac{1}{1+e^{-2a}} \\ &\text{else set } x_n = -1. \end{aligned} \tag{34.3}$$

This rule implements Gibbs sampling for the probability distribution (34.2).

Boltzmann machine learning

Given a set of examples $\{\mathbf{x}^{(n)}\}_1^N$ from the real world, we might be interested in adjusting the weights $\mathbf{W}$ such that the generative model

$$P(\mathbf{x}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})}\exp\left[\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x}\right] \tag{34.4}$$

is well matched to those examples. We can derive a learning algorithm by writing down Bayes' theorem to obtain the posterior probability of the weights given the data:

$$P(\mathbf{W}|\{\mathbf{x}^{(n)}\}_1^N\}) = \frac{\left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)}|\mathbf{W})\right] P(\mathbf{W})}{P(\{\mathbf{x}^{(n)}\}_1^N\})}. \tag{34.5}$$

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

We concentrate on the first term in the numerator, the likelihood, and derive a maximum likelihood algorithm (though there might be advantages in pursuing a full Bayesian approach as we did in the case of the single neuron; indeed Hinton and Sejnowski (1986) used weight decay in their work on Boltzmann machines). It is convenient to differentiate the logarithm of the likelihood.

$$\log \left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)} | \mathbf{W}) \right] = \sum_{n=1}^{N} \left[\frac{1}{2} \mathbf{x}^{\mathsf{T}} \mathbf{W} \mathbf{x} - \log Z(\mathbf{W}) \right] \tag{34.6}$$

We differentiate with respect to w_{ij}, bearing in mind that $\mathbf{W}$ is defined to be symmetric with $w_{ji} = w_{ij}$.

Exercise 34.1:[A2] Show that the derivative of $\log Z(\mathbf{W})$ with respect to w_{ij} is

$$\frac{\partial}{\partial w_{ij}} \log Z(\mathbf{W}) = \sum_{\mathbf{x}} x_i x_j P(\mathbf{x}|\mathbf{W}) = \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} . \tag{34.7}$$

The derivative of the log likelihood is therefore:

$$\frac{\partial}{\partial w_{ij}} \log P(\{\mathbf{x}^{(n)}\}_1^N\}|\mathbf{W}) = \sum_{n=1}^{N} \left[x_i^{(n)} x_j^{(n)} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \right] \tag{34.8}$$

$$= N \left[\langle x_i x_j \rangle_{\text{Data}} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \right] . \tag{34.9}$$

This gradient is thus proportional to a difference of two terms. The first term is the *empirical* correlation between x_i and x_j,

$$\langle x_i x_j \rangle_{\text{Data}} \equiv \frac{1}{N} \sum_{n=1}^{N} \left[x_i^{(n)} x_j^{(n)} \right] , \tag{34.10}$$

and the second term is the correlation between x_i and x_j under the given model,

$$\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \equiv \sum_{\mathbf{x}} x_i x_j P(\mathbf{x}|\mathbf{W}). \tag{34.11}$$

The first correlation $\langle x_i x_j \rangle_{\text{Data}}$ is readily evaluated — it is just the empirical correlation between the activities in the real world. The second correlation, $\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})}$, is not so easy to evaluate, but it can be estimated by Monte Carlo methods, that is, by observing the average value of $x_i x_j$ while the activity rule of the Boltzmann machine, equation (34.3), is iterated.

In the special case $\mathbf{W} = 0$, we can evaluate the gradient exactly because, by symmetry, the correlation $\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})}$ must be zero. If the weights are adjusted by gradient descent with learning rate η, then, after one iteration, the weights will be

$$w_{ij} = \eta \sum_{n=1}^{N} \left[x_i^{(n)} x_j^{(n)} \right] , \tag{34.12}$$

precisely the value of the weights given by the Hebb rule, equation (16.5), with which we trained the Hopfield network.

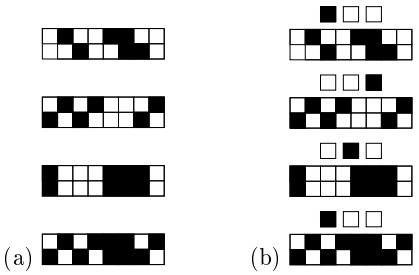


Figure 34.1. The 'shifter' ensembles. (a) Four samples from the plain shifter ensemble. (b) Four corresponding samples from the labelled shifter ensemble.

Interpretation of Boltzmann machine learning

One way of viewing the two terms in the gradient (34.9) is as 'waking' and 'sleeping' rules. While the network is 'awake', it measures the correlation between x_i and x_j in the real world, and weights are increased in proportion. While the network is 'asleep', it 'dreams' about the world using the generative model (34.4), and measures the correlations between x_i and x_j in the model world; these correlations control a proportional decrease in the weights. If the second order correlations in the dream world match the correlations in the real world, then the two terms balance and the weights do not change.

Criticism of Hopfield networks and simple Boltzmann machines

Up to this point we have discussed Hopfield networks and Boltzmann machines in which all of the neurons correspond to *visible* variables x_i. The result is a probabilistic model, which, when optimized, can capture the second order statistics of the environment. The real world, however, often has higher order correlations that must be included if our description of it is to be effective. Often the second order correlations in themselves may carry little or no useful information.

Consider, for example, the ensemble of binary images of chairs. We can imagine images of chairs with various designs — four legged chairs, comfy chairs, chairs with five legs and wheels, wooden chairs, cushioned chairs, chairs with rockers instead of legs. A child can easily learn to distinguish these images from images of carrots and parrots, beetles and beets. But the second order statistics of the raw data are useless for describing the ensemble. Second order statistics only capture whether two pixels are likely to be in the same state as each other. Higher order concepts are needed in order to make a good generative model of images of chairs.

A simpler ensemble of images in which high order statistics are important is the 'shifter ensemble', which comes in two flavours. Figure 34.1(a) shows a few samples from the 'plain shifter ensemble'. In each image, the bottom eight pixels are a copy of the top eight pixels, either shifted one pixel to the left, or unshifted, or shifted one pixel to the right. (The top eight pixels are set at random.) This ensemble is a simple model of the visual signals from the two eyes arriving at early levels of the brain. These signals are similar to each other but may differ by small translations because of the varying depth of the visual world. This ensemble is simple to describe, but its second order statistics convey no useful information at all. The correlation between one pixel and one of the three pixels above it and to either side is 1/3. The

correlation between any other two pixels is zero. *(It would be nice here if I had an example with exactly zero correlation between all pixels.)*

Figure 34.1(b) shows a few samples from the 'labelled shifter ensemble'. Here, the problem has been made easier by including an extra three neurons that label the visual image as being an instance of either the 'shift left', 'no shift', or 'shift right' sub–ensemble. But with this extra information, the ensemble is still not learnable using second order statistics alone. The second order correlation between any two neurons is still zero. We need models that can capture higher order statistics of an environment.

So, how can we develop such models? One instinct might be to create models that directly capture higher order correlations, such as:

$$P'(\mathbf{x}|\mathbf{W},\mathbf{V},\ldots) = \frac{1}{Z'}\exp\left(\frac{1}{2}\sum_{ij}W_{ij}x_ix_j + \frac{1}{6}\sum_{ij}V_{ijk}x_ix_jx_k + \ldots\right). \quad (34.13)$$

Such 'higher order Boltzmann machines' (Sejnowski, 1986) are equally easy to simulate using stochastic updates, and the learning rule for the higher order parameters v_{ijk} is equivalent to the learning rule for w_{ij}.

Exercise 34.2:^{B2} Derive the gradient of the log likelihood with respect to v_{ijk}.

It is conceivable that the 'spines' found on biological neurons are responsible for detecting correlations between multiple incoming signals. However, to capture statistics of high enough order to describe the ensemble of images of chairs well would require an unimaginable number of terms. To capture merely the fourth order statistics in a 128×128 pixel image, we need more than 10^7 parameters.

So measuring moments of images is *not* a good way to describe their underlying structure. Perhaps what we need instead or in addition are *hidden variables*, also known to statisticians as *latent variables*. This is the innovation introduced by Hinton and Sejnowski (1986). The idea is that the high–order correlations among the visible variables are described by including extra hidden variables and sticking to a model that has only second order interactions between its variables; the hidden variables induce higher order correlations between the visible variables.

34.2 Boltzmann machine with hidden units

The simplest way to turn our stochastic model into a latent variable model is to add *hidden neurons* to it. These are neurons that do not correspond to observed variables; they are free to play any role in the probabilistic model defined by equation (34.4). They might actually take on interpretable roles, performing 'feature extraction', or discovering the underlying structure of the given ensemble.

Learning in Boltzmann machines with hidden units

The activity rule of a Boltzmann machine with hidden units is identical to that of the original Boltzmann machine. The learning rule can again be derived by maximum likelihood, but now we need to take into account the fact that the states of the hidden units are unknown. We will denote the states of the

visible units by $\mathbf{x}$, the states of the hidden units by $\mathbf{h}$, and the generic state of a neuron (either visible or hidden) by y_i, with $\mathbf{y} \equiv (\mathbf{x}, \mathbf{h})$. We will denote the state of the network when the visible neurons are clamped in state $\mathbf{x}^{(n)}$ by $\mathbf{y}^{(n)} \equiv (\mathbf{x}^{(n)}, \mathbf{h})$. The likelihood of $\mathbf{W}$ given a single data example $\mathbf{x}^{(n)}$ is

$$P(\mathbf{x}^{(n)}|\mathbf{W}) = \sum_{\mathbf{h}} P(\mathbf{x}^{(n)}, \mathbf{h}|\mathbf{W}) = \sum_{\mathbf{h}} \frac{1}{Z(\mathbf{W})} \exp\left[\frac{1}{2}[\mathbf{y}^{(n)}]^{\mathsf{T}} \mathbf{W} \mathbf{y}^{(n)}\right], \quad (34.14)$$

where

$$Z(\mathbf{W}) = \sum_{\mathbf{x},\mathbf{h}} \exp\left[\frac{1}{2}\mathbf{y}^{\mathsf{T}} \mathbf{W} \mathbf{y}\right]. \quad (34.15)$$

Equation (34.14) may also be written

$$P(\mathbf{x}^{(n)}|\mathbf{W}) = \frac{Z_{\mathbf{x}^{(n)}}(\mathbf{W})}{Z(\mathbf{W})} \quad (34.16)$$

where

$$Z_{\mathbf{x}^{(n)}}(\mathbf{W}) = \sum_{\mathbf{h}} \exp\left[\frac{1}{2}[\mathbf{y}^{(n)}]^{\mathsf{T}} \mathbf{W} \mathbf{y}^{(n)}\right]. \quad (34.17)$$

Differentiating the likelihood as before, we find that the derivative with respect to any weight w_{ij} is again the difference between a 'waking' term and a 'sleeping' term,

$$\frac{\partial}{\partial w_{ij}} \log P(\{\mathbf{x}^{(n)}\}_1^N|\mathbf{W}) = \langle y_i y_j \rangle_{P(\mathbf{h}|\mathbf{x}^{(n)},\mathbf{W})} - \langle y_i y_j \rangle_{P(\mathbf{x},\mathbf{h}|\mathbf{W})}. \quad (34.18)$$

The first term $\langle y_i y_j \rangle_{P(\mathbf{h}|\mathbf{x}^{(n)},\mathbf{W})}$ is the correlation between y_i and y_j if the Boltzmann machine is simulated with the visible variables clamped to $\mathbf{x}^{(n)}$ and the hidden variables freely sampling from their conditional distribution.

The second term $\langle y_i y_j \rangle_{P(\mathbf{x},\mathbf{h}|\mathbf{W})}$ is the correlation between y_i and y_j when the Boltzmann machine generates samples from its model distribution.

Hinton and Sejnowski demonstrated that non–trivial ensembles such as the labelled shifter ensemble can be learned using a Boltzmann machine with hidden units. The hidden units take on the role of feature detectors that spot patterns likely to be associated with one of the three shifts.

The Boltzmann machine is time–consuming to simulate because the computation of the gradient of the log likelihood depends on taking the difference of two gradients, both found by Monte Carlo methods. So Boltzmann machines are not in widespread use. It is an area of active research to create models that embody the same capabilities in a more efficient computational manner (Hinton *et al.*, 1995; Dayan *et al.*, 1995; Hinton and Ghahramani, 1997).

In my opinion, Hinton and Sejnowski's conception of the Boltzmann machine is an outstandingly important step towards understanding how the brain works.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

35

Supervised learning in multilayer networks

35.1 Multilayer perceptrons

No course on neural networks could be complete without a discussion of supervised multilayer networks, also known as backpropagation networks.

The *multilayer perceptron* is a feedforward network. It has input neurons, hidden neurons and output neurons. The hidden neurons may be arranged in a sequence of layers. The most common multilayer perceptrons have a single hidden layer, and are known as 'two–layer' networks, the number 'two' counting the number of layers of neurons not including the inputs.

Such a feedforward network defines a non-linear parameterized mapping from an input $\mathbf{x}$ to an output $\mathbf{y} = \mathbf{y}(\mathbf{x}; \mathbf{w}, \mathcal{A})$. The output is a continuous function of the input and of the parameters $\mathbf{w}$; the architecture of the net, *i.e.*, the functional form of the mapping, is denoted by $\mathcal{A}$. Feedforward networks can be 'trained' to perform regression and classification tasks.

Regression networks

In the case of a regression problem, the mapping for a network with one hidden layer may have the form:

$$\text{Hidden layer:} \quad a_j^{(1)} = \sum_l w_{jl}^{(1)} x_l + \theta_j^{(1)}; \quad h_j = f^{(1)}(a_j^{(1)}) \tag{35.1}$$

$$\text{Output layer:} \quad a_i^{(2)} = \sum_j w_{ij}^{(2)} h_j + \theta_i^{(2)}; \quad y_i = f^{(2)}(a_i^{(2)}) \tag{35.2}$$

where, for example, $f^{(1)}(a) = \tanh(a)$, and $f^{(2)}(a) = a$. Here l runs over the inputs $x_1, \ldots, x_L$, j runs over the hidden units, and i runs over the outputs. The 'weights' w and 'biases' θ together make up the parameter vector $\mathbf{w}$. The non-linear 'sigmoid' function $f^{(1)}$ at the hidden layer gives the neural network greater computational flexibility than a standard linear regression model. Graphically, we can represent the neural network as a set of layers of connected neurons. One neuron is shown in figure 35.1(a). The entire network is shown in figure 35.1(b).

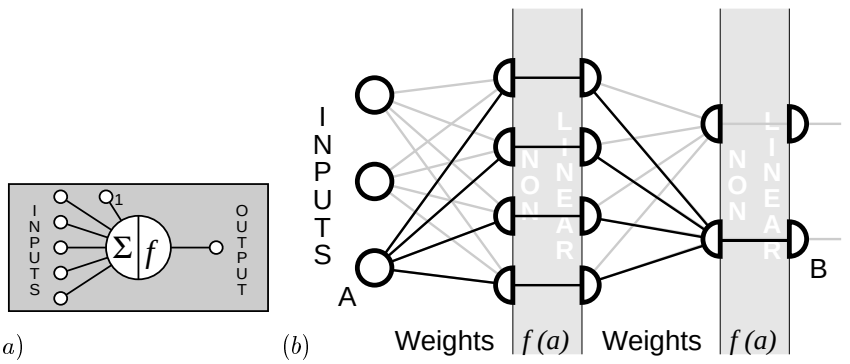


Figure 35.1. Graphical representation of a neural network. (a) A single neuron. (b) A two layer network with 3 inputs, four hidden units and 2 outputs. The weights from input A to the hidden layer have been highlighted, and the weights from the hidden layer to output B.

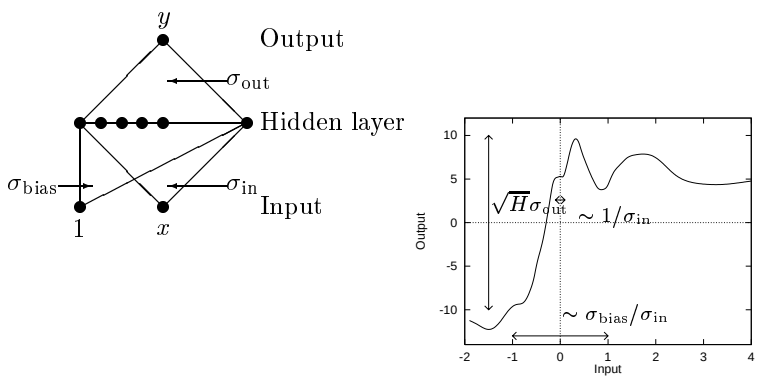


Figure 35.2. Properties of a function produced by a random network. The vertical scale of a typical function produced by the network with random weights is of order $\sqrt{H}\sigma_{\rm out}$; the horizontal range in which the function varies significantly is of order $\sigma_{\rm bias}/\sigma_{\rm in}$; and the shortest horizontal length scale is of order $1/\sigma_{\rm in}$. This network had $H = 400$, and Gaussian weights were generated with $\sigma_{\rm bias} = 4$, $\sigma_{\rm in} = 8$, and $\sigma_{\rm out} = 0.5$.

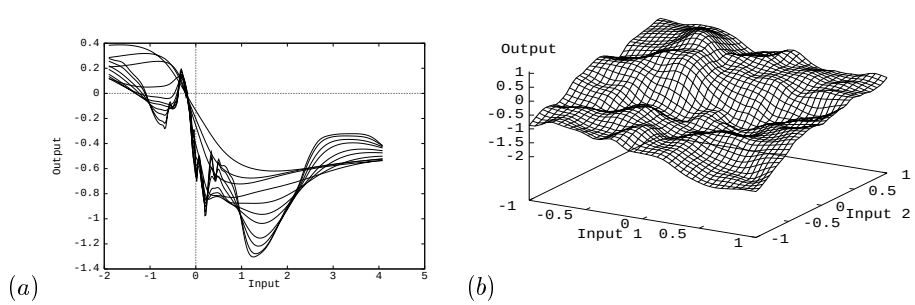


Figure 35.3. Samples from the prior of (a) a one input network; and (b) a two input network. Figure 35.3a shows one function for each of a sequence of values of $\sigma_{\rm bias}$ and $\sigma_{\rm in}$. with $H = 400$, $\sigma^w_{\rm out} = 0.05$. For each graph the parameter $\sigma^w_{\rm bias}$ takes a different value in the sequence: 8, 6, 4, 3, 2, 1.6, 1.2, 0.8, 0.4, 0.3, 0.2. The parameter $\sigma^w_{\rm in}$ was also varied such that $\sigma^w_{\rm in}/\sigma^w_{\rm bias} = 5.0$. The larger values of $\sigma^w_{\rm in}$ and $\sigma^w_{\rm bias}$ produce the more complex functions with more fluctuations. (b) A typical function produced by a two input network with $\{H, \sigma^w_{\rm in}, \sigma^w_{\rm bias}, \sigma^w_{\rm out}\} = \{400, 8.0, 8.0, 0.05\}$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

What sorts of functions can these networks implement?

Just as we explored the weight space of the single neuron in chapter 30, examining the functions it could produce, let us make a small exploration of the weight space of a regression network. In figure 35.2 I take a network with one input and one output and a large number H of hidden units, set the biases and weights $\theta_j^{(1)}$, $w_{jl}^{(1)}$, $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values, and plot the resulting function $y(x)$. I set the hidden unit biases $\theta_j^{(1)}$ to random values from a Gaussian with zero mean and standard deviation $\sigma_{\rm bias}$; the input to hidden weights $w_{jl}^{(1)}$ to random values with standard deviation $\sigma_{\rm in}$; and the bias and output weights $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values with standard deviation $\sigma_{\rm out}$.

The sort of functions that we obtain depend on the values of $\sigma_{\rm bias}$, $\sigma_{\rm in}$ and $\sigma_{\rm out}$ (Neal, 1996) (see captions of figures 35.2 & 35.3). As the weights and biases are made bigger we obtain more complex functions with more features and a greater sensitivity to the input variable. Neal (1996) has also shown that in the limit as $H \to \infty$ the statistical properties of the functions generated by randomizing the weights are independent of the number of hidden units; so, interestingly, the complexity of the functions is independent of the number of parameters in the model. What determines the complexity of the typical functions is the characteristic magnitude of the weights. Thus we anticipate that when we fit these models to real data, an important way of controlling the complexity of the fitted function will be by controlling the characteristic magnitude of the weights.

Figure 35.3b shows one typical function produced by a network with two inputs and one output. This should be contrasted with the function produced by a traditional linear regression model, which is a flat plane. Clearly neural networks can create functions with much more structure than a linear regression.

35.2 How a regression network is traditionally trained

This network is trained using a data set $D = \{\mathbf{x}^{(n)}, \mathbf{t}^{(n)}\}$ by adjusting $\mathbf{w}$ so as to minimize an error function, *e.g.*,

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_m \sum_i \left(t_i^{(n)} - y_i(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 . \tag{35.3}$$

This objective function is a sum of terms, one for each input/target pair $\{\mathbf{x}, t\}$, measuring how close the output $\mathbf{y}(\mathbf{x}; \mathbf{w})$ is to the target t.

This minimization is based on repeated evaluation of the gradient of E_D. This gradient can be efficiently computed using the *backpropagation* algorithm (Rumelhart *et al.*, 1986), which can be viewed as embodying the chain rule for the relevant derivatives.

Often, regularization (also known as 'weight decay') is included, modifying the objective function to:

$$M(\mathbf{w}) = \beta E_D + \alpha E_W \tag{35.4}$$

where, for example, $E_W = \frac{1}{2} \sum_i w_i^2$. This additional term favours small values of $\mathbf{w}$ and decreases the tendency of a model to 'overfit' noise in the training data.

Rumelhart *et al.* (1986) showed that multilayer perceptrons can be trained, by gradient descent on $M(\mathbf{w})$, to discover solutions to non–trivial problems such as deciding whether an image is symmetric or not. These networks have been successfully applied to real–world tasks as varied as pronouncing English text (Sejnowski and Rosenberg, 1987) and focussing multiple–mirror telescopes (Angel *et al.*, 1990).

35.3 Neural network learning as inference

The neural network learning process above can be given the following probabilistic interpretation. [Here we duplicate and generalize the discussion of chapter 32.]

The error function is interpreted as minus the log likelihood for a noise model:

$$P(D|\mathbf{w},\beta,\mathcal{H}) = \frac{1}{Z_D(\beta)}\exp(-\beta E_D). \tag{35.5}$$

Thus, the use of the sum-squared error E_D (35.3) corresponds to an assumption of Gaussian noise on the target variables, and the parameter β defines a noise level $\sigma_\nu^2 = 1/\beta$.

Similarly the regularizer is interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w}|\alpha,\mathcal{H}) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E_W). \tag{35.6}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$. The probabilistic model $\mathcal{H}$ specifies the functional form $\mathcal{A}$ of the network, the likelihood (35.5), and the prior (35.6).

The objective function $M(\mathbf{w})$ then corresponds to the *inference* of the parameters $\mathbf{w}$, given the data:

$$P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}) = \frac{P(D|\mathbf{w},\beta,\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})}{P(D|\alpha,\beta,\mathcal{H})} \tag{35.7}$$

$$= \frac{1}{Z_M}\exp(-M(\mathbf{w})). \tag{35.8}$$

The $\mathbf{w}$ found by (locally) minimizing $M(\mathbf{w})$ is then interpreted as the (locally) most probable parameter vector, $\mathbf{w}_{\rm MP}$.

The interpretation of $M(\mathbf{w})$ as a log probability adds little new at this stage. But new tools will emerge when we proceed to other inferences. First, though, let us establish the probabilistic interpretation of classification networks, to which the same tools apply.

Binary classification networks

If the targets t in a data set are binary classification labels (0,1), it is natural to use a neural network whose output $y(\mathbf{x};\mathbf{w},\mathcal{A})$ is bounded between 0 and 1, and is interpreted as a probability $P(t{=}1|\mathbf{x},\mathbf{w},\mathcal{A})$. For example, a network with one hidden layer could be described by equations (35.1) and (35.2), with $f^{(2)}(a) = 1/(1+e^{-a})$. The error function βE_D is replaced by the negative log likelihood:

$$G(\mathbf{w}) = -\left[\sum_m t^{(n)}\log y(\mathbf{x}^{(n)};\mathbf{w}) + (1-t^{(n)})\log(1-y(\mathbf{x}^{(n)};\mathbf{w}))\right]. \tag{35.9}$$

The total objective function is then $M = G + \alpha E_W$. Note that this includes no parameter β.

Multi-class classification networks

For a multi-class classification problem, we can represent the targets by a vector, $\mathbf{t}$, in which a single element is set to 1, indicating the correct class, and all other elements are set to 0. In this case it is appropriate to use a 'softmax' network (Bridle, 1989) having coupled outputs which sum to one and are interpreted as class probabilities $y_i = P(t_i{=}1|\mathbf{x}, \mathbf{w}, \mathcal{A})$. The last part of equation (35.2) is replaced by:

$$y_i = \frac{e^{a_i}}{\sum_{i'} e^{a_{i'}}}. \tag{35.10}$$

The negative log likelihood in this case is

$$G = -\sum_m \sum_i t_i \log y_i(\mathbf{x}^{(n)}; \mathbf{w}). \tag{35.11}$$

As in the case of the regression network, the minimization of the objective function $M(\mathbf{w}) = G + \alpha E_W$ corresponds to an inference of the form (35.8). A variety of useful results can be built on this interpretation.

35.4 Implementation of Bayesian inference

Bayesian inference for data modelling problems may be implemented by analytical methods, by Monte Carlo sampling, or by deterministic methods employing Gaussian approximations. For neural networks there are few analytic methods. Sophisticated Monte Carlo methods which make use of gradient information have been developed by Neal (1996). Methods based on Gaussian approximations to the posterior distribution have been developed by MacKay (1992c).

35.5 Benefits of the Bayesian approach to supervised feedforward neural networks

The statistical data modelling perspective essentially views supervised neural networks as nothing but nonlinear curve–fitting devices. Curve fitting is not a trivial task however. The effective complexity of an interpolating model is of crucial importance, as illustrated in figure 35.4. Consider a control parameter which influences the complexity of a model, for example a regularization constant α (weight decay parameter). As the control parameter is varied to increase the complexity of the model (descending from figure 35.4*a-c* and going from left to right across figure 35.4*d*), the best fit to the *training* data that the model can achieve becomes increasingly good. However, the empirical performance of the model, the *test error*, has a minimum as a function of the control parameters. *An over-complex model overfits the data and generalizes poorly.* This problem may also complicate the choice of architecture in a multilayer perceptron, the radius of the basis functions in a radial basis function network, and the choice of the input variables themselves in any regression

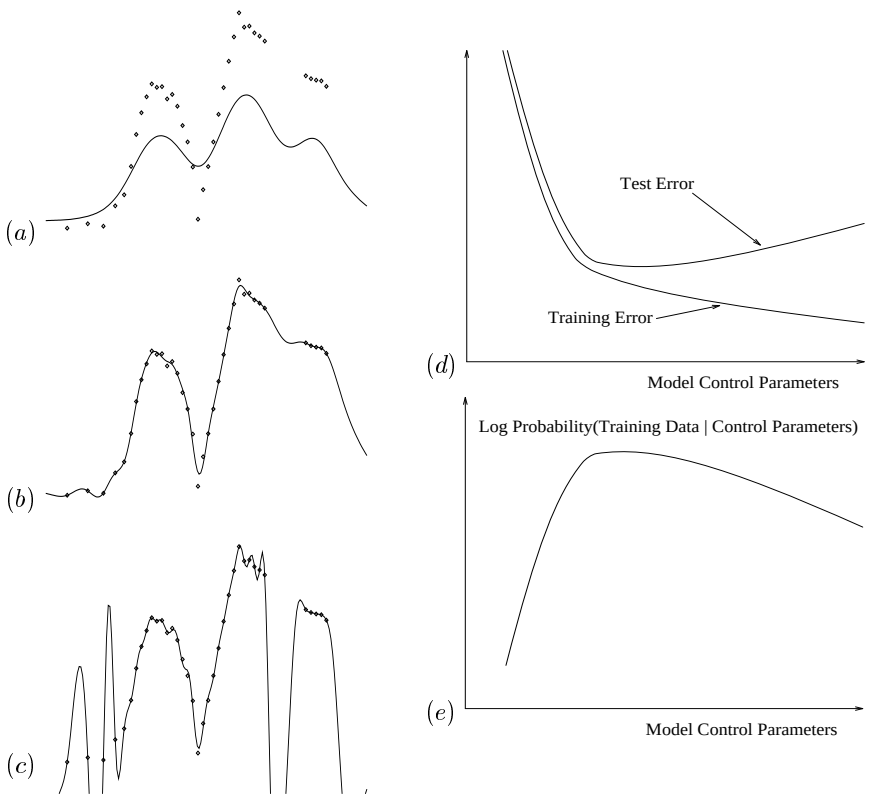


Figure 35.4. Optimization of model complexity. Figures a-c show a radial basis function model interpolating a simple data set with one input variable and one output variable. As the regularization constant is varied to increase the complexity of the model (from a to c), the interpolant is able to fit the training data increasingly well, but beyond a certain point the generalization ability (test error) of the model deteriorates. Probability theory allows us to optimize the control parameters without needing a test set.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

problem. Finding values for model control parameters that are appropriate for the data is therefore an important and non-trivial problem.

An important message is illustrated in figure 35.4*e*. If we give a probabilistic interpretation to the model, then we can evaluate the 'evidence' for alternative values of the control parameters. Over-complex models turn out to be less probable, and the quantity $P(\text{Data}|\text{Control Parameters})$ can be used as an objective function for optimization of model control parameters. The setting of α that maximizes this quantity is displayed in figure 35.4*b*.

Bayesian optimization of model control parameters has four important advantages. (1) No 'test set' or 'validation set' is involved, so all available training data can be devoted to both model fitting and model comparison. (2) Regularization constants can be optimized on-line, *i.e.*, simultaneously with the optimization of ordinary model parameters. (3) The Bayesian objective function is not noisy, in contrast to a cross-validation measure. (4) The gradient of the evidence with respect to the control parameters can be evaluated, making it possible to simultaneously optimize a large number of control parameters.

The *overfitting problem* can thus be solved by using Bayesian methods to control model complexity.

Probabilistic modelling also handles *uncertainty* in a natural manner. There is a unique prescription, *marginalization*, for incorporating uncertainty about parameters into predictions; this procedure yields better predictions, as we saw in chapter 30.

Bayesian probability theory provides a unifying framework for data modelling in which it is easy to define more sophisticated probabilistic models with new capabilities. For example, if we believe that some input variables in a problem may be irrelevant to the predicted quantity, but we don't know which, it is easy to define a new model with multiple hyperparameters that captures the idea of uncertain input variable relevance (MacKay, 1994; Neal, 1996); these models then infer automatically from the data which are the relevant input variables for a problem.

36

Complexity control and model comparison

This chapter pulls together various threads: model comparison can be quantified in terms of 'Occam factors' that penalize over–complex models. The logarithms of these Occam factors are relative entropies that measure how much information has been learned from the data about the model in question.

Complexity control is important from a practical point of view for neural networks.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

37

Approximate Methods for Handling Hyperparameters

Summary

I examine two approximate methods for computational implementation of Bayesian hierarchical models, that is, models which include unknown hyperparameters such as regularization constants and noise levels. In the 'evidence framework' the model parameters are *integrated* over, and the resulting evidence is *maximized* over the hyperparameters. The optimized hyperparameters are used to define a Gaussian approximation to the posterior distribution. In the alternative 'MAP' method, the true posterior probability is found by *integrating* over the hyperparameters. The true posterior is then *maximized* over the model parameters, and a Gaussian approximation is made. The similarities of the two approaches, and their relative merits, are discussed, and comparisons are made with the ideal hierarchical Bayesian solution.

In moderately ill–posed problems, integration over hyperparameters yields a probability distribution with a skew peak which causes significant biases to arise in the MAP method. In contrast, the evidence framework is shown to introduce negligible predictive error, under straightforward conditions. General lessons are drawn concerning inference in many dimensions.

37.1 The overfitting problem and hyperparameters in neural networks

Feedforward neural networks are often trained to solve regression and classification problems using algorithms that minimize an *error function*, a measure of goodness of fit to the training data (Rumelhart *et al.*, 1986). If nothing is done to control the complexity of the resulting neural network, an inevitable consequence of error–minimization will be *overfitting* — the neural network will learn a function which fits spurious details and noise in the data.

There are several approaches to the overfitting problem in neural networks. A crude technique known as 'early stopping' attempts to track a measure of generalization performance during optimization and halt the learning algorithm at the point where this generalization error appears to start to increase. However, most generalization measures are themselves noisy, so the turning point is not easy to identify. Furthermore, the outcome of early stopping will depend on the details of the optimizer chosen to perform the minimization and the initial conditions. And early stopping is unable to independently control

multiple dimensions of complexity; if, as seems reasonable in the case of large models, there is more than one degree of freedom in the model's 'complexity', early stopping would seem too crude a method for complexity control, since it controls complexity using only one degree of freedom — the simulation time.

A more principled approach to overfitting, and one that is less implementation–dependent, is to change the objective function by adding one or more *regularizers* that penalize complex functions. There are various regularizers, the simplest and most popular being 'weight decay' (Hinton and Sejnowski, 1986) (also known as 'ridge regression'). The regularizer in this case is αE_W where E_W is half the sum of the squares of the weights $\{w_i\}$ in the neural network,

$$E_W = \frac{1}{2} \sum_i w_i^2. \tag{37.1}$$

The motivation for this regularizer is that functions with a complex dependence on the inputs of a network require larger weights than simple functions, so this regularizer penalizes the more complex functions and favours smooth ones. This is known as a weight decay regularizer because its derivative with respect to w_i is $\partial(\alpha E_W)/\partial w_i = \alpha w_i$, a term which under gradient descent causes the weights to decay exponentially to zero with a 'weight decay rate' of α. When such a regularizer is used the overfitting problem reappears as the task of setting this complexity control 'hyperparameter' α. Too large a value of α will cause the interpolant to be too smooth so that genuine structure is neglected. Too small a value of α will also give poor generalization because of overfitting. Other regularization schemes have been suggested (Weigend *et al.*, 1991), and the same problem of controlling the hyperparameters applies to those models too.

One way of describing the overfitting problem is to view the neural network as an approximation or estimation tool and describe the control of complexity as a trade–off between *bias* and *variance* (see Bishop (1995) for a review). This might be termed the sampling theory approach to the problem.

This chapter is concerned with an alternative Bayesian viewpoint of neural network learning (MacKay, 1991; Buntine and Weigend, 1991; MacKay, 1992c; Neal, 1993a; Ripley, 1996; Neal, 1996) in which the data error is interpreted as defining a likelihood function, and the regularizer corresponds to a prior probability distribution over the weights. From this viewpoint the question of what value α should take can be thought of as a model comparison question, where the models being compared differ by assigning different priors to the parameters. In (MacKay, 1991; MacKay, 1992c) it was shown that it made theoretical sense, and could be practically beneficial, to use multiple hyperparameters $\{\alpha_c\}$, each one controlling a different aspect of the prior probability distribution. Methods for controlling these multiple hyperparameters were developed by MacKay (1991) using Gaussian approximations and by Neal (1993a) using Markov chain Monte Carlo methods. The approach to implementing Bayesian neural networks suggested by Buntine and Weigend (1991) was subtly different in its treatment of the hyperparameters. As in MacKay's (1991) approach, the use of Gaussian approximations was suggested, but the hyperparameters were 'integrated out' of the problem analytically *before* the Gaussian approximation.

In this chapter I compare the approximate strategies of MacKay (1991) and Buntine and Weigend (1991) for handling hyperparameters, assuming

a Bayesian approach to neural networks. This comparison is also relevant to other ill–posed problems such as image reconstruction (Gull, 1989). For simplicity I will concentrate on the case of a single hyperparameter α, and I will assume that the prior is Gaussian over $\mathbf{w}$, and that the likelihood function is also a Gaussian function of $\mathbf{w}$. I believe that the insights obtained concerning the differences between the approximate methods also apply to models that have more complex likelihood functions and that have priors with multiple hyperparameters.

37.2 The model studied

In inference problems, a Bayesian model $\mathcal{H}$ commonly takes the form:

$$P(D, \mathbf{w}, \alpha, \beta|\mathcal{H}) = P(D|\mathbf{w}, \beta, \mathcal{H})P(\mathbf{w}|\alpha, \mathcal{H})P(\alpha, \beta|\mathcal{H}), \tag{37.2}$$

where D is the data, $\mathbf{w}$ is the parameter vector, β defines a noise variance $\sigma_\nu^2 = 1/\beta$, and α is a regularization constant. In a regression problem, for example, D might be a set of data points, $\{\mathbf{t}\}$, at given locations $\{\mathbf{x}\}$, and the vector $\mathbf{w}$ might parameterize a function $f(\mathbf{x}; \mathbf{w})$. The model $\mathcal{H}$ states that for some $\mathbf{w}$, the dependent variables $\{\mathbf{t}\}$ arise from the addition of noise to $\{f(\mathbf{x}; \mathbf{w})\}$; the likelihood function $P(D|\mathbf{w}, \beta, \mathcal{H})$ describes the assumed noise process, parameterized by a noise level $1/\beta$; the prior probability of the parameters $P(\mathbf{w}|\alpha, \mathcal{H})$ embodies assumptions about the spatial correlations and smoothness that the true function is expected to have, parameterized by a regularization constant α. The variables α and β are known as hyperparameters. Problems for which models can be written in the form (37.2) include linear interpolation with a fixed basis set (Gull, 1988; MacKay, 1992a), nonlinear regression with a neural network (MacKay, 1992c), nonlinear classification (MacKay, 1992b), and image deconvolution (Gull, 1989).

In the simplest case (linear models, Gaussian noise), the first factor in (37.2), the likelihood, can be written in terms of a quadratic function of $\mathbf{w}$, $E_D(\mathbf{w})$:

$$P(D|\mathbf{w}, \beta, \mathcal{H}) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D(\mathbf{w})), \tag{37.3}$$

where $Z_D(\beta)$ is a normalization constant with no $\mathbf{w}$–dependence. In the case of 'ill–posed' problems, the hessian $\nabla\nabla E_D$ is ill–conditioned — some of its eigenvalues are very small, so that the maximum likelihood parameters depend undesirably on the noise in the data. The model is 'regularized' by the second factor in (37.2), the prior, which in the simplest case is a spherical Gaussian:

$$P(\mathbf{w}|\alpha, \mathcal{H}) = \frac{1}{Z_W(\alpha)} \exp(-\alpha \tfrac{1}{2} \mathbf{w}^{\mathsf{T}} \mathbf{w}). \tag{37.4}$$

where $Z_W(\alpha) = \int d^k \mathbf{w} \, \exp(-\alpha \mathbf{w}^{\mathsf{T}} \mathbf{w}/2)$, with k denoting the dimensionality of the parameter vector $\mathbf{w}$. The regularization constant α defines the variance $\sigma_W^2 = 1/\alpha$ of the components w_i of $\mathbf{w}$ under the prior. This simple linear model will be studied in this chapter because it provides a convenient test–bed for comparing approximate inference methods. If a method behaves pathologically in this simple case, how can we expect it to behave well when applied to more complex nonlinear models?

Much interest has centred on the question, for models like the one defined in equations (37.3–37.4), of how the constants α and β — or the ratio α/β — should be set, and Gull (1989) has derived an appealing Bayesian prescription for these constants (see also MacKay (1992a) for a review). This 'evidence framework' *integrates* over the *parameters* $\mathbf{w}$ to give the 'evidence' $P(D|\alpha, \beta, \mathcal{H})$. The evidence is then *maximized* over the *regularization constant* α *and noise level* β. A Gaussian approximation is then made with the hyperparameters fixed to their optimized values. This relates closely to the 'generalized maximum likelihood' or 'MLII' method in statistics (Wahba, 1975). This method can be applied to nonlinear models by making appropriate local linearizations (so that the integral over the parameters is made approximately rather than exactly) and has been used successfully in image reconstruction (Gull, 1989; Weir, 1991) and in neural networks (MacKay, 1992c; Thodberg, 1996; MacKay, 1996a).

An alternative procedure for computing inferences under the same Bayesian model has been suggested by Buntine and Weigend (1991), Strauss *et al.* (1993) and Wolpert (1993). In this approach, one *integrates* over the *regularization constant* α first to obtain the 'true prior', and over the *noise level* β to obtain the 'true likelihood'; then *maximizes* the 'true posterior' (which is proportional to the product of the true prior and the true likelihood) over the *parameters* $\mathbf{w}$. A Gaussian approximation is then made around this true probability density maximum. I will call this the 'MAP' method (for *maximum a posteriori*) although this use of the term 'MAP' may not coincide precisely with its general usage. In the MAP method, the integrations over α can typically be performed exactly, and the posterior probability density maximum is found without any approximations being made. The MAP method is an approximation in that the Gaussian fitted at the posterior maximum is an approximation to the true posterior distribution.

The purpose of this chapter is to examine the choice between these two Gaussian approximations, both of which might be used to approximate predictive inference for high–dimensional problems. Of course the *ideal* Bayesian approach would be to obtain predictions by integrating out *all* the parameters and hyperparameters, and this would certainly be preferred. The assumption here is that this is a challenging integral to perform, and that we are only able to analytically integrate over *either* the parameters (for fixed hyperparameters), as in the evidence framework, *or* over the hyperparameters (for fixed parameters) as in the MAP method.

It is assumed that predictive *distributions* are of interest, rather than point *estimates*. Estimation will only appear as a computational stepping stone in the process of approximating a predictive distribution. I concentrate on the simplest case of the linear model with Gaussian noise, but the insights obtained are expected to apply to more general nonlinear models and to models with multiple hyperparameters. When a nonlinear model has multiple local optima, one can approximate the posterior by a sum of Gaussians, one fitted at each optimum. There is then an analogous choice between either (a) optimizing α separately at each local optimum in $\mathbf{w}$ and using a Gaussian approximation conditioned on α (MacKay, 1992c); or (b) fitting multiple Gaussians to local maxima of the true posterior with the hyperparameter α integrated out. The results of this chapter shed light on this choice.

We will assume for simplicity that the noise level β is known precisely, so

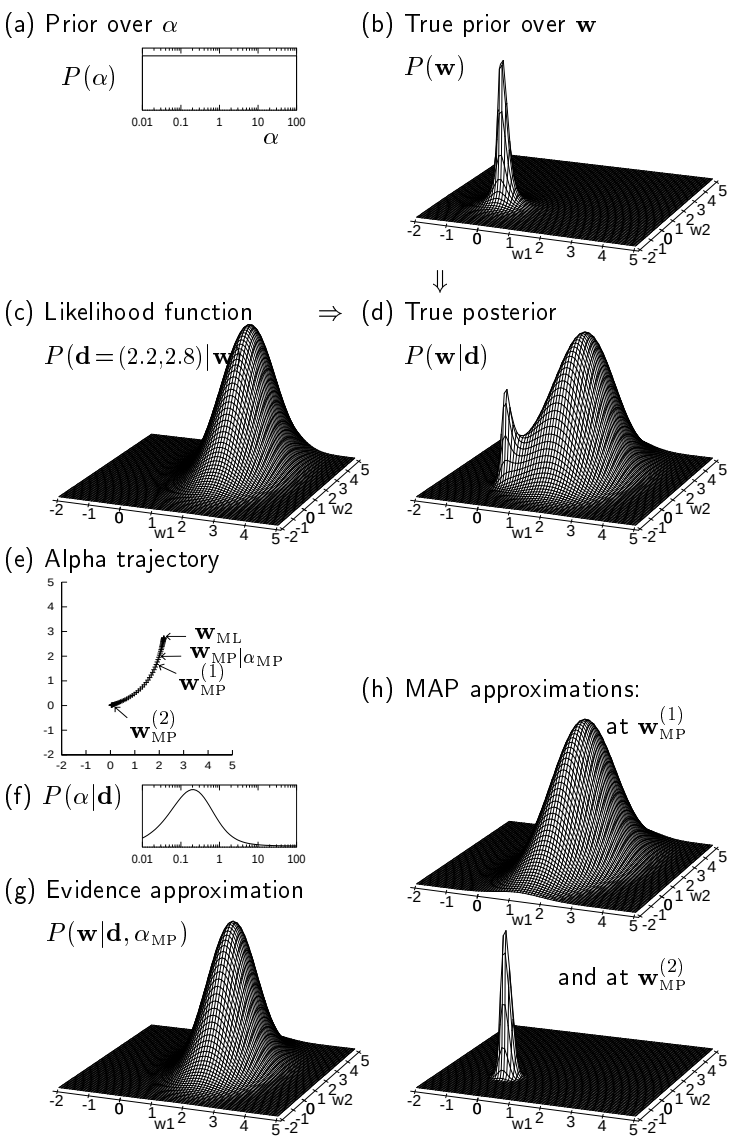


Figure 37.1. Comparison of the evidence approximation and the MAP approximation for a two–dimensional problem with data $\mathbf{d} = (2.2, 2.8)$.

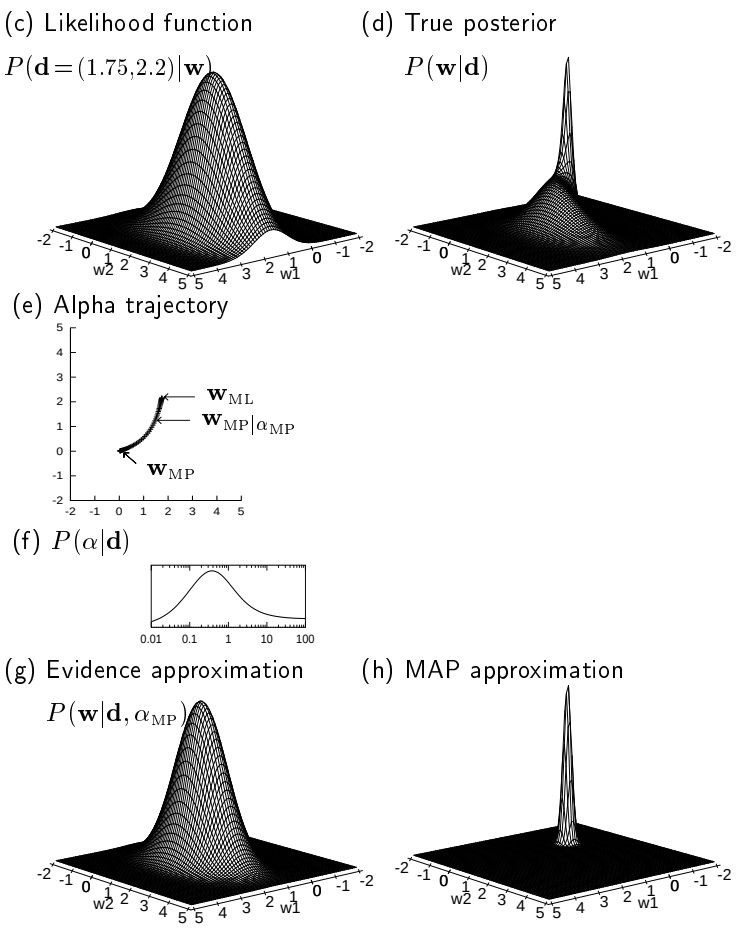


Figure 37.2. Comparison of the evidence approximation and the MAP approximation when $\mathbf{d} = (1.75, 2.2)$.

that only the regularization constant α is respectively optimized or integrated over. Comments about α can apply equally well to β.

37.3 Pictorial comparison of the two methods

The two approximations are illustrated graphically for a simple two–parameter problem in figures 37.1 and 37.2. There are two unknown parameters w_1, w_2, with a prior distribution that is Gaussian with mean zero and variance $1/\alpha$,

$$P(w_1, w_2|\alpha) = \frac{\alpha}{2\pi} \exp\left(-\frac{\alpha}{2}(w_1^2 + w_2^2)\right), \tag{37.5}$$

where α is an unknown hyperparameter whose prior distribution (figure 37.1(a)) is uniform over $\log \alpha$ from $\alpha = 0.01$ to $\alpha = 100$. This prior expresses a belief that w_1 and w_2 are likely to be similar in magnitude, and that their magnitudes might be about 0.1, 1.0, or 10. There are two data points d_1 and d_2 which differ from w_1 and w_2 by additive Gaussian noise of known variance $\sigma_1^2 = 0.5$ and $\sigma_2^2 = 2$ respectively.

$$P(d_1, d_2|w_1, w_2) = \frac{1}{2\pi\sigma_1\sigma_2} \exp\left[-\frac{(d_1 - w_1)^2}{2\sigma_1^2} + \frac{(d_2 - w_2)^2}{2\sigma_2^2}\right]. \tag{37.6}$$

(Or equivalently, there could be more than two data points, all having Gaussian distributions with equal variance, for example, if w_1 is measured independently sixteen times, and w_2 is measured once, with the measurements having variance $\sigma^2 = 2$.)

The 'true prior',

$$P(w_1, w_2) = \int d\alpha \, P(w_1, w_2|\alpha) P(\alpha), \tag{37.7}$$

is shown in figure 37.1(b). It is obtained by integrating the prior conditional on α (equation (37.5)) with respect to the prior on α,

$$P(\log \alpha) = \begin{cases} 1/\log \frac{100}{0.01} & \alpha \in (0.01, 100) \\ 0 & \text{otherwise.} \end{cases} \tag{37.8}$$

We are interested in the posterior distribution of w_1 and w_2 conditional on $\{d_1, d_2\}$; the 'true posterior' (as distinct from the posterior distribution conditional on some value of α) is:

$$P(w_1, w_2|d_1, d_2) \propto P(d_1, d_2|w_1, w_2) P(w_1, w_2). \tag{37.9}$$

Let the data be $\{d_1, d_2\} = \{2.2, 2.8\}$

The likelihood function for the case $\{d_1, d_2\} = \{2.2, 2.8\}$ is shown in figure 37.1(c). The 'true posterior' (which is proportional to the product of the likelihood and the prior) is shown in figure 37.1(d). At this point we notice that the true posterior has two maxima — one associated with a large peak that encompasses the maximum likelihood parameters, and one close to the origin which is associated with a very narrow peak. The 'alpha trajectory' is shown in figure 37.1(e). This is the path that is followed by the maximum of the posterior conditional on α, $P(\mathbf{w}|\mathbf{d}, \alpha)$, as α is varied from a large value

(which puts the posterior maximum near the origin) to a small value (which puts it close to the maximum likelihood value, $\mathbf{w} = \mathbf{w}_{\rm ML}$). We will see in section 37.4 that the maxima and saddle points of the true posterior happen to lie exactly on the alpha trajectory. The posterior probability of α, which is maximized in the evidence framework, is shown in figure 37.1(f). The evidence approximation, $P(\mathbf{w}|\mathbf{d}, \alpha_{\rm MP})$, is shown in figure 37.1(g). The Gaussian approximations found by the MAP method (there are two, because the true posterior has two maxima) are shown in figure 37.1(h).

In this first example, it is not clear if one approximation is superior to the other. We note that whereas the true posterior (figure 37.1(d)) is multimodal, the posterior probability of α is unimodal in this case, and the posterior probability of $\mathbf{w}$ given $\alpha_{\rm MP}$ is also unimodal.

Let us now study the situation for a slightly different data set.

Let the data be $\{d_1, d_2\} = \{1.75, 2.2\}$

The likelihood function for the case $\{d_1, d_2\} = \{1.75, 2.2\}$ is shown in figure 37.2(c). The true posterior is shown in figure 37.2(d). In this case, unlike figure 37.1(d), the true posterior has only one maximum. Both the maximum formerly associated with the large peak and the saddle point between the maxima have vanished. The sole maximum of the true posterior is a sharp peak close to the origin. The posterior probability of α is shown in figure 37.2(f). The evidence approximation $P(\mathbf{w}|\mathbf{d}, \alpha_{\rm MP})$, is shown in figure 37.2(g). The Gaussian approximation found by the MAP method is shown in figure 37.2(h).

In this case, it seems that the MAP method is being led astray by the tall but narrow and skew peak of the probability density. Although the *density* is maximized at this peak, most of the posterior probability *mass* is elsewhere. The Gaussian fitted by the method suggested by Buntine and Weigend (1991), Strauss *et al.* (1993) and Wolpert (1993) appears to be a poor representation of the true posterior. The evidence approximation is not a perfect approximation either — it fails to capture the narrow peak where the true posterior is maximized; but it appears to robustly capture most of the posterior probability mass.

Of course, we cannot judge between two approximate methods on the basis of a toy problem alone. The rest of this chapter aims to fill out the picture, with an emphasis on what is expected to happen in high–dimensional problems in which there are ill–determined as well as well–determined parameters. What we will see is that figure 37.2 gives a good intuition for what happens in high dimensions. We will show that the true posterior distribution usually has a skew peak if there are ill–determined parameters, and the true posterior density's maximum is usually unrepresentative of the true posterior density.

37.4 The alternative methods in detail

Given the Bayesian model defined in (37.2), we might be interested in the following inferences.

Problem A: Infer the parameters, *i.e.*, obtain a compact representation of $P(\mathbf{w}|D, \mathcal{H})$ and the marginal distributions $P(w_i|D, \mathcal{H})$.

Problem B: Infer the relative model plausibility, which requires the 'evidence' $P(D|\mathcal{H})$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Problem C: Make predictions, *i.e.*, obtain some representation of $P(D_2|D,\mathcal{H})$, where D_2, in the simplest case, is a single new datum.

The ideal approach

Ideally, if we were able to do all the necessary integrals, we would just generate the probability distributions $P(\mathbf{w}|D,\mathcal{H})$, $P(D|\mathcal{H})$, and $P(D_2|D,\mathcal{H})$ by direct integration over everything that we are not concerned with. The pioneering work of Box and Tiao (1973) used this approach to develop Bayesian robust statistics.

For real problems of interest, however, such exact integration methods are seldom available. A partial solution can still be obtained by using Monte Carlo methods to simulate the full probability distribution (see Neal (1993b) for an excellent review of Monte Carlo methods and Neal (1996) for the application of these methods to hierarchical models). Thus one can obtain (problem A) a set of samples $\{\mathbf{w}\}$ which represent the posterior $P(\mathbf{w}|D,\mathcal{H})$, and (problem C) a set of samples $\{D_2\}$ which represent the predictive distribution $P(D_2|D,\mathcal{H})$. Unfortunately, the evaluation of the evidence $P(D|\mathcal{H})$ with Monte Carlo methods (problem B) is a difficult undertaking. Recent developments (Neal, 1993a; Skilling, 1993) now make it possible to use gradient and curvature information so as to sample high–dimensional spaces more effectively, even for highly non–Gaussian distributions. Let us come down from these clouds however, and turn attention to the two deterministic approximations under study.

The evidence framework

The evidence framework divides our inferences into distinct 'levels of inference':

Level 1: Infer the parameters $\mathbf{w}$ for a given value of α:

$$P(\mathbf{w}|D,\alpha,\mathcal{H}) = \frac{P(D|\mathbf{w},\alpha,\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})}{P(D|\alpha,\mathcal{H})}. \tag{37.10}$$

Level 2: Infer α:

$$P(\alpha|D,\mathcal{H}) = \frac{P(D|\alpha,\mathcal{H})P(\alpha|\mathcal{H})}{P(D|\mathcal{H})}. \tag{37.11}$$

Level 3: Compare models:

$$P(\mathcal{H}|D) \propto P(D|\mathcal{H})P(\mathcal{H}). \tag{37.12}$$

There is a pattern in these three applications of Bayes' rule: at each of the higher levels 2 and 3, the data–dependent factor (*e.g.* in level 2, $P(D|\alpha,\mathcal{H})$) is the normalizing constant (the 'evidence') from the preceding level of inference.

The inference problems listed at the beginning of this section are solved approximately using the following procedure.

- The level 1 inference is approximated by making a quadratic expansion of $\log P(D|\mathbf{w},\alpha,\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})$ around a maximum of $P(\mathbf{w}|D,\alpha,\mathcal{H})$; this expansion defines a Gaussian approximation to the posterior. The evidence $P(D|\alpha,\mathcal{H})$ is estimated by evaluating the appropriate determinant. For linear models the Gaussian approximation is exact.

- By maximizing the evidence $P(D|\alpha, \mathcal{H})$ at level 2, we find the most probable value of the regularization constant, $\alpha_{\rm MP}$, and by Taylor–expanding $\log P(D|\alpha, \mathcal{H})$ with respect to $\log \alpha$, we obtain error bars on $\log \alpha$, $\sigma_{\log \alpha|D}$. (Because α is a positive scale variable, it is natural to represent its uncertainty on a log scale.)

- The value of $\alpha_{\rm MP}$ is substituted at level 1. This defines a probability distribution $P(\mathbf{w}|D, \alpha_{\rm MP}, \mathcal{H})$ which is intended to be a good approximation (in a sense we will clarify later) to the posterior $P(\mathbf{w}|D, \mathcal{H})$. The solution offered for problem A is a Gaussian distribution around the maximum of this distribution, $\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}}$, with covariance matrix $\mathbf{\Sigma}$ defined by

$$\mathbf{\Sigma}^{-1} = -\nabla\nabla \log P(\mathbf{w}|D, \alpha_{\rm MP}, \mathcal{H}). \tag{37.13}$$

  Marginals for the components of $\mathbf{w}$ are easily obtained from this distribution.

- The evidence for model $\mathcal{H}$ (problem B) is estimated using Laplace's approximation:

$$P(D|\mathcal{H}) \simeq P(D|\alpha_{\rm MP}, H) P(\log \alpha_{\rm MP}|\mathcal{H}) \sqrt{2\pi}\, \sigma_{\log \alpha|D}. \tag{37.14}$$

- Problem C: The predictive distribution $P(D_2|D, \mathcal{H})$ is approximated by using the posterior distribution with $\alpha = \alpha_{\rm MP}$:

$$P(D_2|D, \alpha_{\rm MP}, \mathcal{H}) = \int d^k\mathbf{w}\ \ P(D_2|\mathbf{w}, \mathcal{H}) P(\mathbf{w}|D, \alpha_{\rm MP}, \mathcal{H}), \tag{37.15}$$

  where k is the dimensionality of the parameter vector $\mathbf{w}$. For a locally linear model with Gaussian noise, both the distributions inside the integral are Gaussian, and this integral is straightforward to perform.

As reviewed in MacKay (1992a), the most probable value of α satisfies a simple implicit equation,

$$\frac{1}{\alpha_{\rm MP}} = \frac{\sum_1^k w_i^2}{\gamma} \tag{37.16}$$

where w_i are the components of the vector $\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}}$ and γ is the *number of well–determined parameters*, which can be expressed in terms of the eigenvalues λ_a of the matrix $\beta\nabla\nabla E_D(\mathbf{w})$:

$$\gamma = k - \alpha {\rm Trace} \mathbf{\Sigma} = \sum_1^k \frac{\lambda_a}{\lambda_a + \alpha}. \tag{37.17}$$

This quantity is a number between 0 and k. Recalling that α can be interpreted as the variance σ_W^2 of the distribution from which the parameters w_i come, we see that equation (37.16) corresponds to an intuitive prescription for a variance estimator. The idea is that we are estimating the variance of the distribution of w_i from only γ well–determined parameters, the other $(k-\gamma)$ having been set roughly to zero by the regularizer and therefore not contributing to the sum in the numerator.

In principle, there may be multiple optima in α, but this is not the typical case for a model well matched to the data. Under general conditions, the error bars on $\log \alpha$ are $\sigma_{\log \alpha|D} \simeq \sqrt{2/\gamma}$ (MacKay, 1992a) (see section 37.8). Thus $\log \alpha$ is well determined by the data if $\gamma \gg 1$.

The central computation can be summarised thus:

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Evidence approximation: find a self–consistent solution $\{\mathbf{w}_{\mathrm{MP}|\alpha_{\mathrm{MP}}}, \alpha_{\mathrm{MP}}\}$ such that $\mathbf{w}_{\mathrm{MP}|\alpha_{\mathrm{MP}}}$ maximizes $P(\mathbf{w}|D, \alpha_{\mathrm{MP}}, \mathcal{H})$ and α_{MP} satisfies equation (37.16).

If one is concerned that there may be multiple optima in α, then one may explicitly evaluate the evidence as a function of α.

Justification for the evidence approximation

The central approximation in this scheme can be stated as follows: when we integrate out a parameter α, the effect for most purposes is to estimate the parameter from the data, and then constrain the parameter to that value (Box and Tiao, 1973; Bretthorst, 1988). When we predict an observable D_2, the predictive distribution is dominated by the value $\alpha = \alpha_{\mathrm{MP}}$. In symbols,

$$P(D_2|D, \mathcal{H}) = \int P(D_2|D, \alpha, \mathcal{H}) P(\log \alpha|D, \mathcal{H}) \, d \log \alpha \simeq P(D_2|D, \alpha_{\mathrm{MP}}, \mathcal{H}). \tag{37.18}$$

This approximation is accurate (in a sense that will be made more precise in section 37.8) as long as $P(D_2|D, \alpha, \mathcal{H})$ is insensitive to changes in $\log \alpha$ on a scale of $\sigma_{\log \alpha|D}$, so that the distribution $P(\log \alpha|D, \mathcal{H})$ is effectively a delta function.

This is a well–established idea. A similar equivalence of two probability distributions arises in statistical thermodynamics. The 'canonical ensemble' over all states r of a system,

$$P(r|\beta) = \exp(-\beta E_r)/Z, \tag{37.19}$$

describes equilibrium with a heat bath at temperature $1/\beta$. Although the energy of the system is not fixed, the probability distribution of the energy is usually sharply peaked about the mean energy $\bar{E}$. The corresponding 'microcanonical ensemble' describes the system when it is isolated and has fixed energy:

$$P(r|E{=}\bar{E}) = \begin{cases} 1/\Omega & E_r \in [\bar{E} \pm \delta E/2] \\ 0 & \text{otherwise} \end{cases} . \tag{37.20}$$

Under these two distributions, a particular microstate r may have numerical probabilities that are completely different. For example, the most probable microstate under the canonical ensemble is always the ground state, for any temperature $1/\beta \geq 0$; whereas its probability under the microcanonical ensemble is zero. But if the system has a large number of degrees of freedom, it is well known (Reif, 1965) that for most macroscopic purposes, the two distributions are indistinguishable, because most of the probability *mass* of the canonical ensemble is concentrated in the states in a small interval around $\bar{E}$.

The same reasoning justifies the evidence approximation for ill–posed problems, with particular values of $\mathbf{w}$ corresponding to microstates. If the number of well–determined parameters is large, then α, like the energy above, is well determined. This does not imply that the two densities $P(\mathbf{w}|D, \mathcal{H})$ and $P(\mathbf{w}|D, \alpha_{\mathrm{MP}}, \mathcal{H})$ are numerically close in value, but we have no interest in the probability of the high–dimensional vector $\mathbf{w}$. For practical purposes, we only care about distributions of low–dimensional quantities (*e.g.*, an individual parameter w_i or a new datum); what matters, and what is asserted

here, is that when we project the distributions down in order to predict low–dimensional quantities, the approximating distribution $P(\mathbf{w}|D, \alpha_{\rm MP}, \mathcal{H})$ puts most of its probability mass in the right place. A more precise discussion of this approximation is given in section 37.8.

The MAP method

The alternative procedure studied in this chapter is first to integrate out α to obtain the true prior:

$$P(\mathbf{w}|\mathcal{H}) = \int d\alpha \; P(\mathbf{w}|\alpha, \mathcal{H}) P(\alpha|\mathcal{H}). \tag{37.21}$$

We can then write down the true posterior directly (except for its normalizing constant):

$$P(\mathbf{w}|D, \mathcal{H}) \propto P(D|\mathbf{w}, \mathcal{H}) P(\mathbf{w}|\mathcal{H}). \tag{37.22}$$

This posterior can be maximized to find the MAP parameters, $\mathbf{w}_{\rm MP}$. How does this relate to the desired inferences listed at the head of this section? Not all authors describe how they intend the true posterior to be used in practical problems (*e.g.*, Wolpert (1993)); here I describe a method based on the suggestions of Buntine and Weigend (1991).

Problem A: The posterior distribution $P(\mathbf{w}|D, \mathcal{H})$ is approximated by a Gaussian distribution, fitted around the most probable parameters, $\mathbf{w}_{\rm MP}$; to find the Hessian of the log posterior, one needs the Hessian of the log prior, derived below. [A simple evaluation of the factors on the right hand side of (37.22) is not a satisfactory solution of problem A, since (a) the normalizing constant is missing; (b) even if the right hand side of (37.22) were normalized, the ability to evaluate the local value of this density would be of little use as a summary of the distribution in the high–dimensional space; for example, the marginal distribution over one parameter w_i can only be obtained from (37.22) by somehow performing the marginalization integral over the other parameters.]

Problem B: An estimate of the evidence is obtained from the determinant of the covariance matrix of this Gaussian distribution.

Problem C: The parameters $\mathbf{w}_{\rm MP}$ with error bars are used to generate predictions as in (37.15).

A simple example will illustrate that this approach actually gives results qualitatively similar to the evidence framework. Let us consider the weight decay prior. If we apply the improper prior over α, $P_{\rm Imp}(\log \alpha) = 1$, and evaluate the true prior over the parameters $\mathbf{w}$, we obtain a particularly simple result:[1]

$$P_{\rm Imp}(\mathbf{w}|\mathcal{H}) = \int_{\alpha=0}^{\infty} \frac{e^{-\alpha \sum_{i=1}^{k} w_i^2/2}}{Z_W(\alpha)} \, d\log\alpha \; \propto \frac{1}{(\sum_i w_i^2)^{k/2}}. \tag{37.23}$$

The derivative of the true log prior with respect to $\mathbf{w}$ is $-(k/\sum_i w_i^2)\mathbf{w}$. This 'weight decay' term can be directly viewed in terms of an 'effective α',

$$\frac{1}{\alpha_{\rm eff}(\mathbf{w})} = \frac{\sum_i w_i^2}{k}. \tag{37.24}$$

[1] If a uniform prior over α from 0 to ∞ is used (instead of a uniform prior over $\log\alpha$) then the exponent in equation (37.23) changes from $k/2$ to $(k/2+1)$.

Any maximum of the true posterior $P(\mathbf{w}|D,\mathcal{H})$ is therefore also a maximum of the conditional posterior $P(\mathbf{w}|D,\alpha,\mathcal{H})$, with α set to $\alpha_{\rm eff}$. The similarity of equation (37.24) to equation (37.16) of the evidence framework is clear. We can therefore describe the MAP method thus:

> **MAP method** (improper prior over α): find a self–consistent solution $\{\mathbf{w}_{\rm MP}, \alpha_{\rm eff}\}$ such that $\mathbf{w}_{\rm MP}$ maximizes $P(\mathbf{w}|D,\alpha_{\rm eff},\mathcal{H})$ and $\alpha_{\rm eff}$ satisfies equation (37.24).

This procedure is suggested in (MacKay, 1992c) as a 'quick and dirty' approximation to the evidence framework. What the above result shows is that it is also an exact method for locating the weights that maximize the true posterior probability density.

The effective α and the curvature resulting from a general prior over α

We have just established that, when the improper prior over α (37.23) is used, the MAP solution lies exactly on the 'alpha trajectory' — the graph of $\mathbf{w}_{{\rm MP}|\alpha}$ — for a particular value of $\alpha = \alpha_{\rm eff}$. This result still holds when a proper prior over α is used to define the true prior over $\mathbf{w}$ (37.21). The derivative of $\log P(\mathbf{w}|\mathcal{H})$ with respect to $\mathbf{w}$ is

$$\frac{\partial}{\partial \mathbf{w}} \log P(\mathbf{w}|\mathcal{H}) = \frac{\int d\alpha\, (-\alpha \mathbf{w}) \exp(-\alpha \mathbf{w}^2/2)/Z_W(\alpha) P(\alpha|\mathcal{H})}{P(\mathbf{w}|\mathcal{H})} = -\alpha_{\rm eff}(\mathbf{w})\mathbf{w} \tag{37.25}$$

where the effective $\alpha(\mathbf{w})$ is:

$$\alpha_{\rm eff}(\mathbf{w}) = \int d\alpha\, \alpha\, P(\alpha|\mathbf{w},\mathcal{H}), \tag{37.26}$$

and

$$P(\alpha|\mathbf{w},\mathcal{H}) = \frac{P(\mathbf{w}|\alpha,\mathcal{H})P(\alpha|\mathcal{H})}{P(\mathbf{w}|\mathcal{H})}. \tag{37.27}$$

So at any stationary point of the true posterior, it must be the case that

$$-\beta \frac{\partial}{\partial \mathbf{w}} E_D(\mathbf{w}) - \alpha_{\rm eff}(\mathbf{w})\mathbf{w} = 0, \tag{37.28}$$

which shows that all maxima, minima and saddle points of the true posterior lie on the alpha trajectory. In summary, optima $\mathbf{w}_{\rm MP}$ found by the MAP method can be described thus:

> **MAP method** (proper prior over α): find the self–consistent solution $\{\mathbf{w}_{\rm MP}, \alpha_{\rm eff}\}$ such that $\mathbf{w}_{\rm MP}$ maximizes $P(\mathbf{w}|D,\alpha_{\rm eff},\mathcal{H})$ and $\alpha_{\rm eff}$ satisfies equation (37.26).

The curvature of the true prior over $\mathbf{w}$ is needed for evaluation of the error bars on $\mathbf{w}$ in the MAP method. The true posterior probability maximum $\mathbf{w}_{\rm MP}$ coincides with the maximum of the distribution $P(\mathbf{w}|D,\alpha_{\rm eff},\mathcal{H})$, but the curvature of the true log posterior is not equal to the curvature of $\log P(\mathbf{w}|D,\alpha_{\rm eff},\mathcal{H})$. By direct differentiation of the true log prior (37.21), we find:

$$-\nabla\nabla \log P(\mathbf{w}|\mathcal{H}) = \alpha_{\rm eff}\mathbf{I} - \sigma_\alpha^2(\mathbf{w})\mathbf{w}\mathbf{w}^{\rm T}, \tag{37.29}$$

where $\alpha_{\rm eff}(\mathbf{w})$ is defined in (37.26), and the effective variance of α is:

$$\sigma_\alpha^2(\mathbf{w}) \equiv \overline{\alpha^2}(\mathbf{w}) - \alpha_{\rm eff}(\mathbf{w})^2 \equiv \int d\alpha\, \alpha^2\, P(\alpha|\mathbf{w},\mathcal{H}) - \left(\int d\alpha\, \alpha\, P(\alpha|\mathbf{w},\mathcal{H})\right)^2. \tag{37.30}$$

This is an intuitive result: if α were fixed to $\alpha_{\rm eff}$, then the curvature would just be the first term in (37.29), $\alpha_{\rm eff}\mathbf{I}$. The fact that α is uncertain depletes the curvature in the radial direction $\hat{\mathbf{w}} = \mathbf{w}/|\mathbf{w}|$. To obtain the Hessian for the MAP method's Gaussian approximation, the curvature of the log prior in equation (37.29) would be added to the curvature of the log likelihood $\log P(D|\mathbf{w},\mathcal{H})$.

Condition satisfied by typical samples

The conditions (37.16) and (37.24), satisfied by the optima $(\alpha_{\rm MP}, \mathbf{w}_{{\rm MP}|\alpha_{\rm MP}})$ and $(\alpha_{\rm eff}, \mathbf{w}_{\rm MP})$ respectively, are complemented by an additional result concerning typical samples from posterior distributions conditioned on α. The maximum $\mathbf{w}_{{\rm MP}|\alpha}$ of a Gaussian distribution is not typical of that distribution: the maximum has an atypically small value of $\mathbf{w}^{\rm T}\mathbf{w}$, because, as discussed in section 37.6, nearly all of the mass of a Gaussian is in a shell at some distance surrounding the maximum.

Consider samples $\{\mathbf{w}\}$ from the Gaussian posterior distribution with α fixed to $\alpha_{\rm MP}$, $P(\mathbf{w}|D,\alpha_{\rm MP},\mathcal{H})$. The average value of $\mathbf{w}^{\rm T}\mathbf{w} = \sum_i w_i^2$ for these samples satisfies:

$$\alpha_{\rm MP} = \frac{k}{\langle \sum_i w_i^2 \rangle_{|D,\alpha_{\rm MP}}}. \tag{37.31}$$

Proof: The deviation $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{{\rm MP}|\alpha_{\rm MP}}$ is Gaussian distributed with $\Delta\mathbf{w}\Delta\mathbf{w}^{\rm T} = \mathbf{\Sigma}$. So $\alpha_{\rm MP}\langle \sum_i w_i^2 \rangle_{|D,\alpha_{\rm MP}} = \alpha_{\rm MP}(\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}} + \Delta\mathbf{w})^{\rm T}(\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}} + \Delta\mathbf{w}) = \alpha_{\rm MP}\mathbf{w}^2_{{\rm MP}|\alpha_{\rm MP}} + \alpha_{\rm MP}{\rm Trace}\mathbf{\Sigma} = k$, using equations (37.16) and (37.17).

Thus a typical sample from the evidence approximation prefers just the same value of α as does the evidence $P(D|\alpha,\mathcal{H})$, in the sense that if one were to draw samples $\{\mathbf{w}\}$ from $P(\mathbf{w}|D,\alpha_{\rm MP},\mathcal{H})$ and then estimate α so as to maximize the probability of those samples, α would be set to $\alpha_{\rm MP}$.

37.5 Pros and cons

The algorithms for finding the evidence framework's $\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}}$ and the MAP method's $\mathbf{w}_{\rm MP}$ have been seen to be very similar. Is there any significant distinction to be drawn between these two aproaches?

The MAP method has the advantage that it involves no approximations until after we have found the MAP parameters $\mathbf{w}_{\rm MP}$; in contrast, the evidence framework approximates an integral over α.

In the MAP method the integrals over α and β need only be performed once and can then be used repeatedly for different data sets; in the evidence framework, each new data set has to receive individual attention, with a sequence of (Gaussian) integrations being performed each time α and β are optimized.

So why not always integrate out hyperparameters whenever possible? Let us answer this question by magnifying the systematic differences between the two approaches. With sufficient magnification it will become evident to the

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

intuition that the approximation of the evidence framework is superior to the MAP approximation.

The distinction between $\mathbf{w}_{\rm MP}$ and $\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}}$ is similar to that between the two estimators of standard deviation on a calculator, σ_N and σ_{N-1}, the former being the biased maximum likelihood estimator, whereas the latter is unbiased. The true posterior distribution has a skew peak, so that the MAP parameters are not representative of the whole posterior distribution. This is best illustrated by an example.

The widget example

A collection of widgets $i = 1..k$ have a property called 'wodge', w_i, which we measure, widget by widget, in noisy experiments with a known noise level $\sigma_\nu = 1.0$. Our model for these quantities is that they come from a Gaussian prior $P(w_i|\alpha, \mathcal{H})$, where $\alpha = 1/\sigma_W^2$ is not known. Our prior for this variance is flat over $\log \sigma_W$ from $\sigma_W = 0.1$ to $\sigma_W = 10$.

Scenario 1. Suppose four widgets have been measured and give the following data: $\{d_1, d_2, d_3, d_4\} = \{2.2, -2.2, 2.8, -2.8\}$. The task (problem A) is to infer the wodges of these four widgets, *i.e.*, to produce a representative $\mathbf{w}$ with error bars.

Evidence framework: using equation (37.16) iteratively we find $\alpha_{\rm MP} = 0.19$, $\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}} = \{1.9, -1.9, 2.4, -2.4\}$, each with error bars ± 0.9.

MAP method: We can identify maxima of the true posterior by finding attracting fixed points of equation (37.26) using a computer algebra system. For scenario 1, there are two attracting fixed points, corresponding to two maxima like those in figure 37.1(f): the fixed point with the smaller value of $\alpha_{\rm eff}$ has $\alpha_{\rm eff} = 0.25$, $\mathbf{w}_{\rm MP} = \{1.8, -1.8, 2.2, -2.2\}$, each with error bars ± 0.9. The other maximum is located at $\mathbf{w}_{\rm MP} = \{0.03, -0.03, 0.04, -0.04\}$ and is associated with $\alpha_{\rm eff} = 65$; here, each parameter has error bars ± 0.1.

Concentrating our attention on the sensible maximum, we might note that $\mathbf{w}_{{\rm MP}|\alpha_{\rm MP}}$ is slightly less regularized than $\mathbf{w}_{\rm MP}$, but there is not much disagreement between the two methods when all the parameters are well–determined.

Scenario 2. Suppose in addition to the four measurements above we are now informed that there are an additional four widgets that have been measured with a much less accurate instrument, having $\sigma'_\nu = 100.0$. Thus we now have both well–determined and ill–determined parameters, as in a typical ill–posed problem. The data from these measurements were a string of uninformative values, $\{d_5, d_6, d_7, d_8\} = \{100, -100, 100, -100\}$.

We are again asked to infer the wodges of the widgets. Intuitively, we would like our inferences about the well–measured widgets to be negligibly affected by this vacuous information about the poorly–measured widgets, just as the true Bayesian predictive distributions are unaffected. But clearly with $k = 8$, the difference between k and γ in equations (37.16) and (37.24) is going to become significant. The value of $\alpha_{\rm eff}$ will be substantially greater than that of $\alpha_{\rm MP}$.

In the evidence framework the value of γ is almost exactly the same, since each of the ill–determined parameters has $\lambda_i \simeq 0$ and adds nothing to the number of well–determined parameters (37.17). So the value of $\alpha_{\rm MP}$ and the predictive distributions are unchanged.

In contrast, the MAP solution changes drastically. The maximum associated with $\alpha_{\rm eff} = 0.25$ vanishes, and the only maximum of the true posterior probability is the spike $\mathbf{w}_{\rm MP}$ which is squashed close to zero. Solving equation (37.26) in a computer algebra system, we find: $\alpha_{\rm eff} = 79.5$, $\mathbf{w}_{\rm MP} = \{0.03, -0.03, 0.03, -0.03, 0.0001, -0.0001, 0.0001, -0.0001\}$, with marginal error bars on all eight parameters $\sigma_{w|D} = 0.11$.

Thus the MAP Gaussian approximation is terribly biased towards zero. The final disaster of this approach is that the error bars on the parameters are also very small.

This is not a contrived example. It contains the basic feature of ill–posed problems: that there are both well–determined and poorly–determined parameters. To aid comprehension, the two sets of parameters are separated. This example can be transformed into a typical ill–posed problem simply by rotating the basis to mix the parameters together. In neural networks, a pair of scenarios identical to those discussed above can arise if there are a large number of poorly determined parameters which have been set to zero by the regularizer, and we consider two scenarios. In scenario 1, the network is 'pruned', removing the ill–determined parameters. In scenario 2, the parameters are retained, and take on their most probable value, zero. In each case, what is the optimal setting of the weight decay rate α (assuming the traditional regularizer $\mathbf{w}^{\rm T}\mathbf{w}/2$)? We would expect the answer to be unchanged. Yet the MAP method effectively sets α to a much larger value in the second scenario.

The MAP method may locate the true posterior maximum, but it fails to capture most of the true probability mass. Figure 37.2 conveys in two dimensions this difference between the MAP Gaussian approximation and the Gaussian approximation given by evidence maximization. The larger the number of dimensions we are in, the higher the density in the skew peak becomes, and the more it dominates the maximization of the density. But the mass associated with the peak is not increasing.

If we maximize a probability density which is equal to a superposition of Gaussians, the location of the maximum will be chiefly determined by the locations of the Gaussians with *smallest standard deviation*, rather than the locations of the Gaussians with greatest probability mass.

37.6 Inference in many dimensions

In many dimensions, therefore, new intuitions are needed.

Nearly all of the volume of a k–dimensional hypersphere is in a thin shell near its surface. For example, in 1000 dimensions, 90% of a hypersphere of radius 1.0 is within a depth of 0.0023 of its surface. A central core of the hypersphere, with radius 0.5, contains less than $1/10^{300}$ of the volume.

This has an important effect on high–dimensional probability distributions. Consider a Gaussian distribution $P(\mathbf{w}) = (1/\sqrt{2\pi}\,\sigma_W)^k \exp(-\sum_1^k w_i^2/2\sigma_W^2)$. Nearly all of the probability mass of a Gaussian is in a thin shell of radius $r = \sqrt{k}\sigma_W$ and of thickness $\propto r/\sqrt{k}$. For example, in 1000 dimensions, 90% of the mass of a Gaussian with $\sigma_W = 1$ is in a shell of radius 31.6 and thickness 2.8. However, the probability *density* at the origin is $e^{k/2} \simeq 10^{217}$ times bigger than the density at this shell where most of the probability mass is.

Now consider two Gaussian densities in 1000 dimensions which differ in

radius σ_W by just 1%, and which contain equal total probability mass. The maximum probability density is greater at the centre of the Gaussian with smaller σ_W by a factor of $\sim \exp(0.01k) \simeq 20{,}000$.

A typical true posterior distribution for an ill–posed problem is a weighted superposition of Gaussians with varying means and standard deviations, so the true posterior has a skew peak, with the maximum of the probability density located near the mean of the Gaussian distribution that has the smallest standard deviation, not the Gaussian with the greatest weight. Thus a Gaussian fitted at the MAP parameters is a bad approximation to the distribution: it is in the wrong place, and its error bars are far too small. In contrast, the evidence approximation is given by selecting from the superposition of Gaussians the Gaussian component which has the biggest weight, and which thus captures most of the probability mass of the true posterior.

In summary, probability density maxima often have very little associated probability mass — even though the value of the probability density there may be immense — because they have so little associated volume. If a distribution is composed of a mixture of Gaussians with different σ_W, the probability density maxima are strongly dominated by smaller values of σ_W. This is why the MAP method finds a silly solution in the widget example. Recall that in the case of a thermodynamic system in its canonical ensemble (section 37.4), the state of the system that has maximum probability density is the ground state, regardless of the temperature of the system.

Thus the locations of probability density maxima in many dimensions are generally misleading and irrelevant. Probability densities should only be maximized if there is good reason to believe that the location of the maximum conveys useful information about the whole distribution, *e.g.*, if the distribution is approximately Gaussian.

37.7 Relationship between evidence maximization and 'ensemble learning'

A novel approach to the approximation of Bayesian inference has recently been introduced by Hinton and van Camp (1993). I will first review the concept of *Ensemble Learning by Free Energy Minimization* for a simplified model with the hyperparameter α omitted.

In traditional approaches to neural networks, a *single* parameter vector $\mathbf{w}$ is optimized by maximum likelihood or penalized maximum likelihood. In the Bayesian interpretation, these optimized parameters are viewed as defining the mode of a posterior probability distribution $P(\mathbf{w}|D, \mathcal{H})$ (given data D and model assumptions $\mathcal{H}$), which can be approximated, with a Gaussian distribution for example, in order to obtain predictive distributions and optimize model control parameters.

The new concept introduced by Hinton and van Camp (1993) is to work in terms of an approximating *ensemble* $Q(\mathbf{w}; \theta)$, that is, a probability distribution over the parameters, and optimize the ensemble (by varying its own parameters θ) so that it approximates the posterior distribution of the parameters $P(\mathbf{w}|D, \mathcal{H})$ as well as possible. The objective function chosen to measure the quality of the approximation is a *variational free energy* (Feynman, 1972),

$$F(\theta) = - \int d^k \mathbf{w}\, Q(\mathbf{w}; \theta) \log \frac{P(D|\mathbf{w}, \mathcal{H}) P(\mathbf{w}|\mathcal{H})}{Q(\mathbf{w}; \theta)}. \tag{37.32}$$

The free energy $F(\theta)$ is bounded below by $-\log P(D|\mathcal{H})$ and only attains this value for $Q(\mathbf{w};\theta) = P(\mathbf{w}|D,\mathcal{H})$. $F(\theta)$ can be viewed as the sum of $-\log P(D|\mathcal{H})$ and the Kullback–Leibler divergence between $Q(\mathbf{w};\theta)$ and $P(\mathbf{w}|D,\mathcal{H})$. For certain models and certain approximating distributions, this free energy, and its derivatives with respect to the ensemble's parameters, can be evaluated. [This is the main reason for choosing the objective function $F(\theta)$ rather than some other measure of distance between $Q(\mathbf{w};\theta)$ and $P(\mathbf{w}|D,\mathcal{H})$.] A longer review of Ensemble Learning including references to applications may be found in (MacKay, 1995b).

In this section I demonstrate that a free energy approximation for the model studied in this chapter reproduces the method of the evidence framework precisely. This result is not viewed as a justification for the evidence framework, but rather as giving insight into the nature of the approximations made by this framework.

Free energy approximation for a model with a hyperparameter

Let us assume, in addition to the likelihood function and prior over $\mathbf{w}$ of equations (37.3) and (37.4), that the prior over α is a gamma distribution, $P(\alpha|\mathcal{H}) = \Gamma(\alpha; b_\alpha, c_\alpha)$, where this notation means:

$$\Gamma(\alpha; b_\alpha, c_\alpha) = \frac{1}{\Gamma(c_\alpha)} \frac{\alpha^{c_\alpha - 1}}{b_\alpha^{c_\alpha}} \exp\left(-\frac{\alpha}{b_\alpha}\right), 0 \leq \alpha < \infty. \tag{37.33}$$

This distribution has mean $b_\alpha c_\alpha$ and variance $b_\alpha^2 c_\alpha$.

Let us consider approximating the joint distribution of $\mathbf{w}$ and α given the data,

$$P(\mathbf{w}, \alpha|D,\mathcal{H}) = \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})P(\alpha|\mathcal{H})}{P(D|\mathcal{H})}, \tag{37.34}$$

by a distribution $Q(\mathbf{w},\alpha)$. I make one assumption only, namely we will use an approximating distribution that is constrained to have the separable form $Q(\mathbf{w},\alpha) = Q_{\mathbf{w}}(\mathbf{w})Q_\alpha(\alpha)$. *No functional form for these distributions is assumed.* [The reason for choosing this separable form is that this is the most complex approximating distribution for which the computations are tractable — we don't necessarily believe the posterior density is approximately separable.] We write down a variational free energy,

$$F(Q) = -\int d\mathbf{w}\, d\alpha\, Q_{\mathbf{w}}(\mathbf{w})Q_\alpha(\alpha) \log \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})P(\alpha|\mathcal{H})}{Q_{\mathbf{w}}(\mathbf{w})Q_\alpha(\alpha)}. \tag{37.35}$$

This functional is bounded below by the evidence for the model thus: $F \geq -\log P(D|\mathcal{H})$, with equality if and only if $Q(\mathbf{w},\alpha) = P(\mathbf{w},\alpha|D,\mathcal{H})$. We can find the optimal separable distribution Q by considering separately the optimization of F over $Q_{\mathbf{w}}(\mathbf{w})$ for fixed $Q_\alpha(\alpha)$, and then the optimization of $Q_\alpha(\alpha)$ for fixed $Q_{\mathbf{w}}(\mathbf{w})$.

Optimization of $Q_{\mathbf{w}}(\mathbf{w})$

As a functional of $Q_{\mathbf{w}}(\mathbf{w})$, F is:

$$F = -\int d\mathbf{w}\, Q_{\mathbf{w}}(\mathbf{w}) \left[\int d\alpha\, Q_\alpha(\alpha) \log P(\mathbf{w}|\alpha) + \log P(D|\mathbf{w},\mathcal{H}) - \log Q(\mathbf{w})\right] + \text{const.} \tag{37.36}$$

$$= \int d\mathbf{w}\, Q_{\mathbf{w}}(\mathbf{w}) \left[\int d\alpha\, Q_\alpha(\alpha) \alpha \frac{1}{2}\mathbf{w}\mathbf{w}^{\mathsf{T}} + \beta E_D(\mathbf{w}) + \log Q(\mathbf{w})\right] + \text{const.}' \tag{37.37}$$

The dependence on Q_α thus collapses down to a dependence simply on the mean value of α,

$$\bar{\alpha} \equiv \int d\alpha \, Q_\alpha(\alpha) \alpha. \tag{37.38}$$

$$F = \int d\mathbf{w} \, Q_{\mathbf{w}}(\mathbf{w}) \left[\bar{\alpha} \frac{1}{2} \mathbf{w}\mathbf{w}^{\mathsf{T}} + \beta E_D(\mathbf{w}) + \log Q(\mathbf{w}) \right] + \text{const.}' \tag{37.39}$$

Noting that the $\mathbf{w}$–dependent terms $-\bar{\alpha}\frac{1}{2}\mathbf{w}\mathbf{w}^{\mathsf{T}} - \beta E_D(\mathbf{w})$ are the log of a posterior distribution, and using the theorem that a divergence $\int Q \log(Q/P)$ is minimized by setting $Q = P$, we can immediately write down the distribution $Q_{\mathbf{w}}(\mathbf{w})$ that minimizes this expression. For given data D and Q_α, the optimizing distribution $Q_{\mathbf{w}}^{\text{opt}}(\mathbf{w})$ is a Gaussian identical to the posterior distribution for a particular value of $\alpha = \bar{\alpha}$.

$$Q_{\mathbf{w}}^{\text{opt}}(\mathbf{w}) = P(\mathbf{w}|D, \bar{\alpha}, \mathcal{H}) = \text{Normal}(\mathbf{w}_{\text{MP}|\bar{\alpha}}, \mathbf{\Sigma}). \tag{37.40}$$

Optimization of $Q_\alpha(\alpha)$

As a functional of $Q_\alpha(\alpha)$, F is:

$$F = -\int d\alpha \, Q_\alpha(\alpha) \left[\int d\mathbf{w} \, Q_{\mathbf{w}}(\mathbf{w}) \log P(\mathbf{w}|\alpha, \mathcal{H}) + \log P(\alpha|\mathcal{H}) - \log Q_\alpha(\alpha) \right] + \text{const.} \tag{37.41}$$

$$= \int d\alpha \, Q_\alpha(\alpha) \left[\tfrac{\alpha}{2} \textstyle\int d\mathbf{w} \, Q_{\mathbf{w}}(\mathbf{w}) \mathbf{w}^{\mathsf{T}}\mathbf{w} - \tfrac{k}{2} \log \alpha - (c_\alpha - 1) \log \alpha + \tfrac{\alpha}{b_\alpha} + \log Q_\alpha(\alpha) \right] \tag{37.42}$$

$$= \int d\alpha \, Q_\alpha(\alpha) \left[\left(\frac{1}{2} \mathbf{w}_{\text{MP}|\bar{\alpha}}{}^{\mathsf{T}} \mathbf{w}_{\text{MP}|\bar{\alpha}} + \frac{1}{2} \text{Trace}\, \mathbf{\Sigma} + \frac{1}{b_\alpha} \right) \alpha \right.$$
$$\left. - \left(\frac{k}{2} + c_\alpha - 1 \right) \log \alpha + \log Q_\alpha(\alpha) \right] + \text{const.}' \tag{37.43}$$

where c_α, b_α are the parameters of the gamma prior on α. Here, the α–dependent expression in the brackets can be recognized as the log of a gamma distribution, giving as the optimal distribution that minimizes F for fixed $Q_{\mathbf{w}}$:

$$Q_\alpha^{\text{opt}}(\alpha) = \Gamma(\alpha; b', c') \tag{37.44}$$

where

$$\begin{aligned} 1/b' &= 1/b_\alpha + \tfrac{1}{2} \mathbf{w}_{\text{MP}|\bar{\alpha}}{}^{\mathsf{T}} \mathbf{w}_{\text{MP}|\bar{\alpha}} + \tfrac{1}{2} \text{Trace}\, \mathbf{\Sigma} \\ c' &= k/2 + c_\alpha \end{aligned} . \tag{37.45}$$

This completes our derivation of the free energy optimization. The optimal approximating distribution is given by finding the gamma distribution for α and the normal distribution for $\mathbf{w}$ that satisfy the simultaneous equations (37.38), (37.40) and (37.45).

Comparison with the evidence framework

To understand this result we complete the loop by evaluating the mean $\bar{\alpha}'$ for this optimized gamma distribution, which is:

$$\bar{\alpha}' = b'c' = \frac{\frac{k}{2} + c_\alpha}{\frac{1}{b_\alpha} + \frac{1}{2} \mathbf{w}_{\text{MP}|\bar{\alpha}}{}^{\mathsf{T}} \mathbf{w}_{\text{MP}|\bar{\alpha}} + \frac{1}{2} \text{Trace}\, \mathbf{\Sigma}}. \tag{37.46}$$

In the special case of an uninformative prior on α ($c_\alpha \to 0$ and $\frac{1}{b_\alpha} \to 0$) we obtain:

$$\bar{\alpha}' = \frac{k}{\mathbf{w}_{\mathrm{MP}|\bar{\alpha}}{}^{\mathsf{T}} \mathbf{w}_{\mathrm{MP}|\bar{\alpha}} + \mathrm{Trace}\, \mathbf{\Sigma}}. \tag{37.47}$$

Is this the same optimal α as that found by evidence maximization?[2] The answer is yes. Substituting (equation 37.16) $\mathbf{w}^{\mathsf{T}}_{\mathrm{MP}|\alpha_{\mathrm{MP}}} \mathbf{w}_{\mathrm{MP}|\alpha_{\mathrm{MP}}} = \gamma/\alpha_{\mathrm{MP}}$, and using $\gamma = k - \alpha \mathrm{Trace} \mathbf{\Sigma}$, we find that if we set $\alpha = \bar{\alpha} = \alpha_{\mathrm{MP}}$ on the right hand side we obtain

$$\bar{\alpha}' = \frac{k}{\gamma/\bar{\alpha} + (k-\gamma)/\bar{\alpha}} = \bar{\alpha}. \tag{37.48}$$

Thus any optimum of the evidence approximation also corresponds to a minimum of the free energy. This relationship is only exact in the case of the linear regression model studied in this chapter. If the likelihood is non–Gaussian then $P(\mathbf{w}|D, \bar{\alpha}, \mathcal{H})$ is no longer a Gaussian, so the step at equation (37.40) does not follow.

Intuition for the relationship between evidence maximization and ensemble learning

These two approaches give complementary views of the task of inferring α given the data.

In the evidence framework we examine the optimized value of $\mathbf{w}$, $\mathbf{w}_{\mathrm{MP}|\alpha}$, and think of $(\mathbf{w}_{\mathrm{MP}|\alpha})^2$ as giving information about the variance σ_W^2 of the prior distribution of $\mathbf{w}$. The maximum likelihood estimator of σ_W^2 would be $\sigma^2_{W(\mathrm{ML})} = (\mathbf{w}_{\mathrm{MP}|\alpha})^2/k$, but the evidence framework modifies this estimator to take into account the fact that some of the k parameters have not been determined by the data, and have effectively been set to zero by the prior. Thus the evidence–maximizing estimate replaces k by the effective number of well determined parameters γ: $\sigma^2_{W(\mathrm{MP})} = (\mathbf{w}_{\mathrm{MP}|\alpha})^2/\gamma$.

The free energy minimization approach is like an EM algorithm (Dempster *et al.*, 1977), in which we wish to find the most probable α and do this by introducing an E–step in which a distribution over $\mathbf{w}$ is obtained (Neal and Hinton, 1998). This distribution takes into account the $k - \gamma$ ill–determined parameters by assigning each of them a variance of σ_W^2 in the matrix $\mathbf{\Sigma}$. Then when the M–step occurs, finding the optimal α, the maximum likelihood equation $\sigma^2_{W(\mathrm{ML})} = (\mathbf{w}_{\mathrm{MP}|\alpha})^2/k$ is modified by adding these variance terms to the numerator: $\sigma^2_{W(\mathrm{FE})} = \left[(\mathbf{w}_{\mathrm{MP}|\alpha})^2 + \mathrm{Trace}\, \mathbf{\Sigma}\right]/k$.

Thus evidence maximization decrements the denominator of the equation $\sigma^2_{W(\mathrm{ML})} = (\mathbf{w}_{\mathrm{MP}|\alpha})^2/k$ to take into account the smallness of the ill–determined parameters, whereas free energy minimization increments the numerator to take into account their variability. As we have seen, the two formulae converge on the identical result.

Further comments

There are two small differences between evidence maximization and free energy minimization.

[2] Or 'are *these* the same as *those* found by evidence maximization?' if there are multiple optima.

1. The variance of the optimized gamma distribution for α is, in the limit of the uninformative prior,

$$\mathrm{var}(\alpha) = b'^2 c' = 2k/(k/\bar{\alpha})^2 = \bar{\alpha}^2/k \tag{37.49}$$

so that $\log \alpha$ has standard error $\sqrt{2/k}$. This contrasts with the result $\sqrt{2/\gamma}$ from the evidence framework.

2. This free energy approximation for $Q_{\mathbf{w}}(\mathbf{w})$ fails to produce the small order correction terms to be identified in section 37.8, which arise because of the uncertainty in α. This failure is caused by the separability assumption in the ensemble approximation.

37.8 Conditions for the evidence approximation

We have observed in section 37.5 that the MAP method can lead to absurdly biased answers if there are many ill–determined parameters. In contrast, I now discuss conditions under which the evidence approximation works. I discuss again the case of linear models with Gaussian probability distributions.

What do we care about when we approximate a complex probability distribution by a simple one? My definition of a good approximation is a practical one, concerned with (A) estimating parameters; (B) estimating the evidence accurately; and (C) getting the predictive mass in the right place. Estimation of individual parameters (A) is a special case of prediction (C), so in the following I will address only problems (C) and (B).

For convenience let us work in the eigenvector basis where the prior over $\mathbf{w}$ (given α) and the likelihood are both diagonal Gaussian functions. The curvature of the log likelihood is represented by eigenvalues $\{\lambda_a\}$. For a typical ill–posed problem these eigenvalues vary in value by several orders of magnitude. Without loss of generality let us assume k data measurements $\{d_a\}$, such that $d_a = \sqrt{\lambda_a} w_a + \nu$, where the noise standard deviation is $\sigma_\nu = 1$. We define the probability distribution of everything by the product of the distributions:

$$P(\log\alpha|\mathcal{H}) = \frac{1}{\log(\alpha_{\max}/\alpha_{\min})}, P(\mathbf{w}|\alpha,\mathcal{H}) = \left(\frac{\alpha}{2\pi}\right)^{k/2} \exp\left(-\frac{1}{2}\alpha \sum_1^k w_a^2\right), \text{ and} \tag{37.50}$$

$$P(D|\mathbf{w},\mathcal{H}) = (2\pi)^{-k/2} \exp\left\{-\frac{1}{2}\sum_1^k \left(\sqrt{\lambda_a} w_a - d_a\right)^2\right\}. \tag{37.51}$$

The discussion proceeds in two steps. First, the posterior distribution over α must have a single sharp peak at α_{MP}. No general guarantee can be given for this to be the case, but various pointers are given. Second, given a sharp Gaussian posterior over $\log\alpha$, it is proved that the evidence approximation introduces negligible error.

Concentration of $P(\log\alpha|D,\mathcal{H})$ in a single maximum

Condition 1 *In the posterior distribution over $\log\alpha$, all the probability mass should be contained in a single sharp maximum.*

For this to hold, several sub–conditions are needed. If there is any doubt whether these conditions are sufficient, it is straightforward (at least in the

case of a single hyperparameter) to iterate all the way down the α trajectory, explicitly evaluating $P(\log\alpha|D,\mathcal{H})$.

The prior over α must be such that the posterior has negligible mass at $\log\alpha \to \pm\infty$. In cases where the signal to noise ratio of the data is very low, there may be a significant tail in the evidence for large α. There may even be no maximum in the evidence, in which case the evidence framework gives singular behaviour, with α going to infinity. But often the tails of the evidence are small, and contain negligible mass if our prior over $\log\alpha$ has cutoffs at some $\alpha_{\min}$ and $\alpha_{\max}$ surrounding $\alpha_{\rm MP}$. For each data analysis problem, one may evaluate the critical $\alpha_{\max}$ above which the posterior would be measurably affected by the large α tail of the evidence (Gull, 1989). Often, as Gull points out, this critical value of $\alpha_{\max}$ has bizarrely large magnitude.

Even if a flat prior between appropriate $\alpha_{\min}$ and $\alpha_{\max}$ is used, it is possible in principle for the posterior $P(\log\alpha|D,\mathcal{H})$ to be multi–modal. However this is not expected when the model space is well matched to the data. Examples of multi–modality only arise if the data are grossly at variance with the model. For example, if some large eigenvalue measurements give small $d_{a(l)}$, and some measurements with small eigenvalue give large $d_{a(s)}$, then the posterior over α can have two peaks, one at large α which nicely explains $d_{a(l)}$, but must attribute $d_{a(s)}$ to unusually large amounts of noise, and one at small α which nicely explains $d_{a(s)}$, but must attribute $d_{a(l)}$ to $w_{a(l)}$ being unexpectedly close to zero. This concept may be formalized into a quantitative test as follows.

If we accept the model, then we believe that there is a true value of $\alpha = \alpha_{\rm T}$, and that given $\alpha_{\rm T}$, the data measurements d_a are the sum of two independent Gaussian variables $\sqrt{\lambda_a}w_a$ and ν_a, so that $P(d_a|\alpha_{\rm T},\mathcal{H}) = \mathrm{Normal}(0,\sigma^2_{a|\alpha_T})$, where $\sigma^2_{a|\alpha_T} = \frac{\lambda_a}{\alpha_T} + 1$. The expectation of d_a^2 is $\langle d_a^2\rangle = \frac{\lambda_a}{\alpha_T} + 1$. We therefore expect that there is an $\alpha_{\rm T}$ such that the quantities $\{d_a^2/\sigma^2_{a|\alpha_T}\}$ are independently distributed like χ^2 with one degree of freedom.

Definition 1 *A data set $\{d_a\}$ is* grossly at variance *with the model for a given value of α at significance level τ, if any of the quantities $j_a = d_a^2/(\frac{\lambda_a}{\alpha}+1)$ is not in the interval $[e^{-\tau}, 1+\tau]$.*

It is conjectured that if we find a value of $\alpha = \alpha_{\rm MP}$ which locally maximizes the evidence, and with which the data are not grossly at variance, then there are no other maxima over α.

Conversely, if the data are grossly at variance with a local maximum $\alpha_{\rm MP}$, then there may be multiple maxima in α, and the evidence approximation may be inaccurate. In these circumstances one might also suspect that the entire model is inadequate in some way.

Assuming that $P(\log\alpha|D,\mathcal{H})$ has a single maximum over $\log\alpha$, how sharp is it expected to be? I now establish conditions under which the $P(\log\alpha|D,\mathcal{H})$ is locally Gaussian and sharp.

Definition 2 *The symbol n_e is defined by:*

$$n_e \equiv \sum_a \frac{4\lambda_a\alpha_{\rm MP}}{(\lambda_a+\alpha_{\rm MP})^2}. \tag{37.52}$$

This is a measure of the number of eigenvalues λ_a within approximately e–fold of $\alpha_{\rm MP}$.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

In the following, I will assume that $n_e \ll \gamma$, but this condition is not essential for the evidence approximation to be valid. If $n_e \ll \gamma$, and the data are not grossly at variance with $\alpha_{\rm MP}$, then the Taylor expansion of $\log P(\alpha|D,\mathcal{H})$ about $\alpha = \alpha_{\rm MP}$ is:

$$\left.\frac{\partial \log P(D|\alpha,\mathcal{H})}{\partial \log \alpha}\right|_{\alpha_{\rm MP}} = \frac{1}{2}\left(\gamma - \alpha \mathbf{w}^2_{{\rm MP}|\alpha_{\rm MP}}\right) = 0 \tag{37.53}$$

$$\left.\frac{\partial^2 \log P(D|\alpha,\mathcal{H})}{\partial (\log \alpha)^2}\right|_{\alpha_{\rm MP}} \simeq -\alpha \mathbf{w}^2_{{\rm MP}|\alpha_{\rm MP}} = -\frac{\gamma}{2} \tag{37.54}$$

$$\left.\frac{\partial^3 \log P(D|\alpha,\mathcal{H})}{\partial (\log \alpha)^3}\right|_{\alpha_{\rm MP}} \simeq -\alpha \mathbf{w}^2_{{\rm MP}|\alpha_{\rm MP}} = -\frac{\gamma}{2}. \tag{37.55}$$

The first derivative is exact, assuming that the eigenvalues λ_a are independent of α, which is true in the case of a Gaussian prior on $\mathbf{w}$ (Bryan, 1990). The second and third derivatives are approximate, with terms proportional to n_e being omitted. Now, if $\gamma \gg 1$, then the second derivative is relatively large, and the third derivative is relatively small (even though they are numerically equal), since in the expansion $P(l) = \exp(-\frac{c}{2}l^2 + \frac{d}{6}l^3 + \ldots)$, the second term gives a negligible perturbation for $l \sim c^{-1/2}$ if $d \ll c^{3/2}$. In this case, since $d \simeq c \simeq \gamma \gg 1$, the perturbation introduced by the higher order terms is $O(\gamma^{-1/2})$. Thus the posterior distribution over $\log \alpha$ has a maximum that is both locally Gaussian and sharp if $\gamma \gg 1$ and $n_e \ll \gamma$. The expression for the evidence (37.14) follows.

Error of low–dimensional predictive distributions

I will now assume that the posterior distribution $P(\log \alpha|D,\mathcal{H})$ is Gaussian with standard deviation $\sigma_{\log\alpha|D} = 1/\sqrt{\kappa\gamma}$, with $\kappa\gamma \gg 1$, and $\kappa = O(1)$.

Theorem 1 *Consider a scalar which depends linearly on* $\mathbf{w}$, $y = \mathbf{g}\cdot\mathbf{w}$. *The evidence approximation's predictive distribution for* y *is close to the exact predictive distribution, for nearly all projections* $\mathbf{g}$. *In the case* $\mathbf{g} = \mathbf{w}$, *the error (measured by a cross–entropy) is of order* $\sqrt{n_e/\kappa\gamma}$. *For all* $\mathbf{g}$ *perpendicular to this direction, the error is of order* $\sqrt{1/\kappa\gamma}$.

A similar result is expected still to hold when the dimensionality of y is greater than one, provided that it is much less than $\sqrt{\gamma}$.

Proof: At 'level 1', we infer $\mathbf{w}$ for a fixed value of α:

$$P(\mathbf{w}|D,\alpha,\mathcal{H}) \propto \exp\left\{-\frac{1}{2}\sum_a (\lambda_a + \alpha)\left(w_a - \frac{\sqrt{\lambda_a}d_a}{\lambda_a+\alpha}\right)^2\right\}. \tag{37.56}$$

The most probable $\mathbf{w}$ given this value of α is: $w_a^{{\rm MP}|\alpha} = \sqrt{\lambda_a}d_a/(\lambda_a+\alpha)$. The posterior distribution is Gaussian about this most probable $\mathbf{w}$. We introduce a *typical* $\mathbf{w}$, that is, a sample from the posterior for a particular value of α:

$$w_a^{{\rm TYP}|\alpha} = \frac{\sqrt{\lambda_a}d_a}{\lambda_a+\alpha} + \frac{r_a}{\sqrt{\lambda_a+\alpha}}, \tag{37.57}$$

where r_a is a sample from Normal(0,1).

Now, assuming that $\log \alpha$ has a Gaussian posterior distribution with standard deviation $1/\sqrt{\kappa\gamma}$, a typical α, *i.e.*, a sample from this posterior, is given to leading order by

$$\alpha^{\rm TYP} = \alpha_{\rm MP}\left(1 + \frac{s}{\sqrt{\kappa\gamma}}\right), \tag{37.58}$$

where s is a sample from Normal(0,1). We now substitute this $\alpha^{\rm TYP}$ into (37.57) and obtain a typical $\mathbf{w}$ from the true posterior distribution, which depends on k+1 random variables $\{r_a\}, s$. We expand each component of this vector $\mathbf{w}^{\rm TYP}$ in powers of $1/\gamma$:

$$\begin{aligned} w_a^{\rm TYP} &= \frac{\sqrt{\lambda_a} d_a}{\lambda_a + \alpha_{\rm MP}}\left(1 - \frac{s}{\sqrt{\kappa\gamma}}\frac{\alpha_{\rm MP}}{\lambda_a+\alpha_{\rm MP}} + \frac{s^2}{\kappa\gamma}\frac{\alpha_{\rm MP}^2}{(\lambda_a+\alpha_{\rm MP})^2} + \ldots\right) + \\ & \frac{r_a}{\sqrt{\lambda_a+\alpha_{\rm MP}}}\left(1 - \frac{1}{2}\frac{s}{\sqrt{\kappa\gamma}}\frac{\alpha_{\rm MP}}{\lambda_a+\alpha_{\rm MP}} + \frac{3}{8}\frac{s^2}{\kappa\gamma}\frac{\alpha_{\rm MP}^2}{(\lambda_a+\alpha_{\rm MP})^2}\ldots\right) \end{aligned} \tag{37.59}$$

We now examine the mean and variance of $y^{\rm TYP} = \sum_a g_a w_a^{\rm TYP}$. Setting $\langle r_a^2\rangle = \langle s^2\rangle = 1$ and dropping terms of higher order than $1/\gamma$, we find that whereas the evidence approximation gives a Gaussian predictive distribution for y which has mean and variance:

$$\mu_0 = \sum_a g_a w_a^{{\rm MP}|\alpha_{\rm MP}}, \quad \sigma_0^2 = \sum_a \frac{g_a^2}{\lambda_a+\alpha_{\rm MP}}, \tag{37.60}$$

the true predictive distribution is, to order $1/\gamma$, Gaussian with mean and variance:

$$\mu_1 = \mu_0 + \frac{1}{\kappa\gamma}\sum_a g_a w_a^{{\rm MP}|\alpha_{\rm MP}}\frac{\alpha_{\rm MP}^2}{(\lambda_a+\alpha_{\rm MP})^2}, \tag{37.61}$$

$$\sigma_1^2 = \sigma_0^2 + \frac{1}{\kappa\gamma}\left\{\left(\sum_a g_a w_a^{{\rm MP}|\alpha_{\rm MP}}\frac{\alpha_{\rm MP}}{(\lambda_a+\alpha_{\rm MP})}\right)^2 + \sum_a \frac{g_a^2}{\lambda_a+\alpha_{\rm MP}}\frac{\alpha_{\rm MP}^2}{(\lambda_a+\alpha_{\rm MP})^2}\right\}. \tag{37.62}$$

How wrong can the evidence approximation be? Since both distributions are Gaussian, it is simple to evaluate the Kullback–Leibler distance between them. The cross entropy between p_0 =Normal(μ_0,σ_0^2) and p_1 =Normal(μ_1,σ_1^2) is

$$H(p_0,p_1) \equiv \int p_1 \log\frac{p_1}{p_0} = \frac{1}{2}\frac{(\mu_1-\mu_0)^2}{\sigma_0^2} + \frac{1}{4}\left(\frac{\sigma_1^2-\sigma_0^2}{\sigma_0^2}\right)^2 + O\left\{\left(\frac{\sigma_1^2-\sigma_0^2}{\sigma_0^2}\right)^3\right\}. \tag{37.63}$$

We consider the two dominant terms separately. The difference in means gives the term

$$\frac{(\mu_1-\mu_0)^2}{\sigma_0^2} = \frac{1}{\kappa^2\gamma^2}\left(\sum_a h_a w_a^{{\rm MP}|\alpha_{\rm MP}}\frac{\alpha_{\rm MP}^2}{(\lambda_a+\alpha_{\rm MP})^{3/2}}\right)^2 \Big/ \sum h_a^2 \tag{37.64}$$

where $h_a = g_a/\sqrt{\lambda_a+\alpha_{\rm MP}}$. The worst case is given by the direction $\mathbf{g}$ such that $h_a = w_a^{{\rm MP}|\alpha_{\rm MP}}\frac{\alpha_{\rm MP}^2}{(\lambda_a+\alpha_{\rm MP})^{3/2}}$. This worst case gives an upper bound to the contribution to the cross entropy:

$$\begin{aligned} \frac{(\mu_1-\mu_0)^2}{\sigma_0^2} &\leq \frac{1}{\kappa^2\gamma^2}\sum_a \frac{{w_a^{{\rm MP}|\alpha_{\rm MP}}}^2\alpha_{\rm MP}^4}{(\lambda_a+\alpha_{\rm MP})^3} & (37.65)\\ &\leq \frac{\alpha_{\rm MP}}{\kappa^2\gamma^2}\sum_a {w_a^{{\rm MP}|\alpha_{\rm MP}}}^2 = \frac{1}{\kappa^2\gamma} \ll 1. & (37.66) \end{aligned}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

So the change in μ *never* has a significant effect.

The variance term can be split into two terms:

$$\left(\frac{\sigma_1^2-\sigma_0^2}{\sigma_0^2}\right)^2 = \frac{1}{\kappa\gamma}\left\{\left(\sum_a \frac{h_a w_a^{\mathrm{MP}|\alpha_{\mathrm{MP}}}\alpha_{\mathrm{MP}}}{\sqrt{\lambda_a+\alpha_{\mathrm{MP}}}}\right)^2 + \sum_a h_a^2 \frac{\alpha_{\mathrm{MP}}^2}{(\lambda_a+\alpha_{\mathrm{MP}})^2}\right\} \Bigg/ \sum_a h_a^2, \tag{37.67}$$

where, as above, $h_a = g_a/\sqrt{\lambda_a+\alpha_{\mathrm{MP}}}$.

For the first term, the worst case is the direction $h_a = w_a^{\mathrm{MP}|\alpha_{\mathrm{MP}}} \frac{\alpha_{\mathrm{MP}}}{\sqrt{\lambda_a+\alpha_{\mathrm{MP}}}}$, *i.e.*, the radial direction $\mathbf{g} = \alpha_{\mathrm{MP}}\mathbf{w}_{\mathrm{MP}|\alpha_{\mathrm{MP}}}$. Substituting in this direction, we find:

$$\text{First term} \leq \frac{1}{\kappa\gamma}\sum_a {w_a^{\mathrm{MP}|\alpha_{\mathrm{MP}}}}^2 \frac{\alpha_{\mathrm{MP}}^2}{\lambda_a+\alpha_{\mathrm{MP}}} \tag{37.68}$$

$$\leq \frac{\alpha_{\mathrm{MP}}}{\kappa\gamma}\sum_a {w_a^{\mathrm{MP}|\alpha_{\mathrm{MP}}}}^2 = \frac{1}{\kappa} = O(1). \tag{37.69}$$

We can improve this bound by substituting for $w_a^{\mathrm{MP}|\alpha_{\mathrm{MP}}}$ in terms of d_a and making use of the definition of n_e. Only n_e of the terms in the sum in equation (37.68) are significant. Thus

$$\text{First term} \lesssim \frac{n_e}{\kappa\gamma}. \tag{37.70}$$

So this term can give a significant effect, but only in one direction; for any direction orthogonal (in $\mathbf{h}$) to this radial direction, this term is zero.

Finally, we examine the second term:

$$\frac{1}{\kappa\gamma}\sum_a h_a^2 \frac{\alpha_{\mathrm{MP}}^2}{(\lambda_a+\alpha_{\mathrm{MP}})^2} \Bigg/ \sum_a h_a^2 < \frac{1}{\kappa\gamma} \ll 1. \tag{37.71}$$

So this term never has a significant effect.

Conclusion

The evidence approximation affects the mean and variance of properties y of $\mathbf{w}$, but only to within $O(\gamma^{-1/2})$ of the property's standard deviation; this error is insignificant, for large γ. The sole exception is the direction $\mathbf{g} = \mathbf{w}_{\mathrm{MP}|\alpha_{\mathrm{MP}}}$, along which the variance is erroneously small, with a cross–entropy error of order $O(n_e/\gamma)$.

A correction term

This result motivates a straightforward term which could be added to the inverse Hessian of the evidence approximation, to correct the predictive variance in this direction. The predictive variance for a general $y = \mathbf{g}^{\mathsf{T}}\mathbf{w}$ could be estimated by

$$\sigma_y^2 = \mathbf{g}^{\mathsf{T}}\left(\mathbf{\Sigma} + \sigma^2_{\log\alpha|D}\mathbf{w}'_{\mathrm{MP}|\alpha}{\mathbf{w}'_{\mathrm{MP}|\alpha}}^{\mathsf{T}}\right)\mathbf{g}, \tag{37.72}$$

where $\mathbf{w}'_{\mathrm{MP}|\alpha} \equiv \partial\mathbf{w}_{\mathrm{MP}|\alpha}/\partial(\log\alpha) = \alpha\mathbf{\Sigma}\mathbf{w}_{\mathrm{MP}|\alpha}$, and $\sigma^2_{\log\alpha|D} = \frac{2}{\gamma}$. With this correction, the predictive distribution for any direction would be in error only by order $O(1/\gamma)$. If the noise variance $\sigma_\nu^2 = \beta^{-1}$ is also uncertain, then the factor $\sigma^2_{\log\alpha|D}$ is incremented by $\sigma^2_{\log\beta|D} = \frac{2}{N-\gamma}$.

37.9 Discussion

The MAP method, though it can give exact values for the relative probability densities of two weight vectors, is capable of giving a Gaussian approximation which is highly unrepresentative of the true posterior. In high–dimensional spaces, maxima of densities are misleading. MAP estimates play no fundamental role in Bayesian inference, and they can change arbitrarily with arbitrary re–parameterizations. The problem with MAP estimates is that they maximize the probability *density*, without taking account of the complementary *volume* information. What matters is where the probability *mass* is, and mass is equal to density times volume. When there are many ill–determined parameters, the MAP method's integration over α yields a $\mathbf{w}_{\rm MP}$ which is severely over–regularized. Integration over the noise level $1/\beta$ to give the true likelihood leads to a bias in the other direction. [These two biases may cancel: the evidence framework's $\mathbf{w}_{\rm MP|\alpha_{MP},\beta_{MP}}$ coincides with $\mathbf{w}_{\rm MP}$ if the number of well–determined parameters happens to obey the condition $\gamma/k = N/(N+k)$, where N is the number of data points.]

There are two general take–home messages.

(1) When one has a choice of which variables to integrate over and which to maximize over, one should integrate over as many variables as possible, in order to capture the relevant volume information. There are typically far fewer regularization constants and other hyperparameters than there are 'level 1' parameters.

(2) If practical Bayesian methods involve approximations such as fitting a Gaussian to a posterior distribution, then one should think twice before integrating out hyperparameters (Gull, 1988). The probability density which results from such an integration typically has a skew peak; a Gaussian fitted at the peak may not approximate the distribution well. In contrast, optimization of the hyperparameters can give a Gaussian approximation which, for predictive purposes, puts most of the probability mass in the right place.

The evidence approximation, which sets hyperparameters so as to maximize the evidence, is not intended to produce an accurate approximation to the numerical value of the true posterior density over $\mathbf{w}$; and it does not. But what matters is whether low–dimensional properties of $\mathbf{w}$ (*i.e.*, predictions) are seriously mis–calculated as a result of the evidence approximation. The main conditions for the evidence approximation are that the data should not be grossly at variance with the model, and that the number of well–determined parameters γ should be large. How large depends on the problem, but often a value as small as $\gamma \simeq 3$ is sufficient, because this means that α is determined to within a factor of e (recall $\sigma_{\log \alpha|D} \simeq \sqrt{2/\gamma}$); predictive distributions are often insensitive to changes of α of this magnitude. Thus the approximation is usually good if we have enough data to determine a few parameters.

If satisfactory conditions do not hold for the evidence approximation (*e.g.*, if γ is too small), then it should be emphasized that this would not motivate integrating out α first. The MAP approximation is systematically inferior to the evidence approximation. Practical alternative methods for dealing with hyperparameters include the deterministic method of Bryan (1990), who finds it most convenient numerically to retain α as an explicit variable, and integrate it out *last*, and the Markov chain Monte Carlo implementation of Neal (1996) which samples the hyperparameters and parameters from the joint distribution

$P(\mathbf{w}, \alpha|D, \mathcal{H})$.

The relationship between evidence maximization and ensemble learning derived in section 37.7 gives a convergence proof (at least for linear models) for a re–estimation formula for α (equation 37.46) which previous work on the evidence framework had not provided. The steps of re–estimating $\bar{\alpha}$ and computing the new distribution $Q_{\mathbf{w}}(\mathbf{w})$ both decrease F, and F is bounded below, so the iterative procedure must converge.

A final point in favour of the evidence framework is that it can be naturally extended (at least approximately) to more elaborate priors such as mixture models; it would be difficult to integrate over the mixture hyperparameters in order to evaluate the true prior in these cases.

38

Gaussian processes

A review of work by Neal, Williams, Rasmussen and Gibbs on Gaussian processes as a replacement for supervised neural networks.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

39

Deconvolution

39.1 Traditional image reconstruction methods

Optimal linear filters

In many imaging problems, the data measurements $\{d_m\}$ are linearly related to the underlying image $\mathbf{f}$:

$$d_m = \sum_j R_{mj} f_j + \nu_m. \tag{39.1}$$

The vector ν denotes the inevitable noise which corrupts real data. In the case of a camera which produces a blurred picture, the vector $\mathbf{f}$ denotes the true image, $\mathbf{d}$ denotes the blurred and noisy picture, and the linear operator $\mathbf{R}$ is a convolution defined by the point spread function of the camera. In this special case, the true image and the data vector reside in the same space; but it is important to maintain a distinction between them. We will use the subscript $m = 1, \ldots, N$ to run over data measurements, and the subscripts $i, j = 1, \ldots, k$ to run over image pixels.

One might speculate that since the blur was created by a linear operation, then perhaps it might be deblurred by another linear operation. We derive the *optimal linear filter* in two ways.

Bayesian derivation

We assume that the linear operator $\mathbf{R}$ is known, and that the noise ν is Gaussian and independent, with a known standard deviation σ_ν.

$$P(\mathbf{d}|\mathbf{f}, \sigma_\nu, \mathcal{H}) = \frac{1}{(2\pi\sigma_\nu^2)^{N/2}} \exp\left(-\sum_m \left(d_m - \textstyle\sum_j R_{mj} f_j \right)^2 \Big/ \left(2\sigma_\nu^2\right) \right) \tag{39.2}$$

We assume that the prior probability of the image is also Gaussian, with a standard deviation σ_f.

$$P(\mathbf{f}|\sigma_f, \mathcal{H}) = \frac{\det^{-\frac{1}{2}} \mathbf{C}}{(2\pi\sigma_f^2)^{k/2}} \exp\left(-\sum_{i,j} f_i C_{ij} f_j / \left(2\sigma_f^2\right) \right) \tag{39.3}$$

If we assume no correlations among the pixels then the symmetric, full rank matrix $\mathbf{C}$ is equal to the identity matrix $\mathbf{I}$. The more sophisticated 'intrinsic correlation function' model uses $\mathbf{C} = [\mathbf{G}\mathbf{G}^{\mathrm{T}}]^{-1}$, where $\mathbf{G}$ is a convolution that

takes us from an imaginary 'hidden' image, which is uncorrelated, to the real correlated image. The intrinsic correlation function should not be confused with the point spread function $\mathbf{R}$ which defines the image to data mapping. A zero-mean Gaussian prior is clearly a poor assumption if it is known that all elements of the image $\mathbf{f}$ are positive but let us proceed. We are now able to infer the posterior probability of an image $\mathbf{f}$ given the data $\mathbf{d}$.

$$P(\mathbf{f}|\mathbf{d},\sigma_\nu,\sigma_f,\mathcal{H}) = \frac{P(\mathbf{d}|\mathbf{f},\sigma_\nu,\mathcal{H})P(\mathbf{f}|\sigma_f,\mathcal{H}))}{P(\mathbf{d}|\sigma_\nu,\sigma_f,\mathcal{H})} \tag{39.4}$$

In words,

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}. \tag{39.5}$$

The 'evidence' $P(\mathbf{d}|\sigma_\nu,\sigma_f,\mathcal{H})$ is the normalizing constant for this posterior distribution. Here it is unimportant, but it is used in a more sophisticated analysis to compare, for example, different values of σ_ν and σ_f, or different point spread functions $\mathbf{R}$.

Since the posterior distribution is the product of two Gaussian functions of $\mathbf{f}$, it is also a Gaussian, and can therefore be summarized by its mean, which is also the *most probable image*, $\mathbf{f}_{\rm MP}$, and its covariance matrix:

$$\Sigma_{\mathbf{f}|\mathbf{d}} \equiv [-\nabla\nabla \log P(\mathbf{f}|\mathbf{d},\sigma_\nu,\sigma_f,\mathcal{H})]^{-1}\,, \tag{39.6}$$

which defines the joint error bars on $\mathbf{f}$. In this equation, the symbol ∇ denotes differentiation with respect to the parameters $\mathbf{f}$. We can find $\mathbf{f}_{\rm MP}$ by differentiating the log of the posterior, and solving for the derivative being zero. We obtain:

$$\mathbf{f}_{\rm MP} = \left[\mathbf{R}^{\mathsf{T}}\mathbf{R} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}\right]^{-1}\mathbf{R}^{\mathsf{T}}\mathbf{d}. \tag{39.7}$$

The operator $\left[\mathbf{R}^{\mathsf{T}}\mathbf{R} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}\right]^{-1}\mathbf{R}^{\mathsf{T}}$ is called the optimal linear filter. When the term $\frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}$ can be neglected, the optimal linear filter is the pseudoinverse "$\mathbf{R}^{-1}$" $= [\mathbf{R}^{\mathsf{T}}\mathbf{R}]^{-1}\mathbf{R}^{\mathsf{T}}$. The term $\frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}$ 'regularizes' this ill-conditioned inverse.

The optimal linear filter can also be manipulated into the form:

$$\text{Optimal linear filter} = \mathbf{C}^{-1}\mathbf{R}^{\mathsf{T}}\left[\mathbf{R}\mathbf{C}^{-1}\mathbf{R}^{\mathsf{T}} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{I}\right]^{-1}. \tag{39.8}$$

Minimum square error derivation

The orthodox derivation of the optimal linear filter starts by assuming that we will 'estimate' the true image $\mathbf{f}$ by a linear function of the data:

$$\hat{\mathbf{f}} = \mathbf{W}\mathbf{d}. \tag{39.9}$$

The linear operator $\mathbf{W}$ is then 'optimized' by minimizing the expected sum-squared error between $\hat{\mathbf{f}}$ and the unknown true image . (Interestingly, any quadratic metric using a symmetric positive definite matrix gives the same optimal linear filter.) In the following equations, summations over repeated indices i, j, m are implicit. The expectation $\langle\cdot\rangle$ is over both the statistics of

the random variables $\{\nu_m\}$, and the ensemble of images $\mathbf{f}$ which we expect to bump into. We assume that the noise is zero mean and uncorrelated to second order with itself and everything else, with $\langle \nu_m \nu_{m'} \rangle = \sigma_\nu^2 \delta_{mm'}$.

$$\langle E \rangle = \frac{1}{2} \left\langle (W_{im} d_m - f_i)^2 \right\rangle \tag{39.10}$$

$$= \frac{1}{2} \left\langle (W_{im} R_{mj} f_j - f_i)^2 \right\rangle + \frac{1}{2} W_{im} W_{im} \sigma_\nu^2. \tag{39.11}$$

Differentiating, and introducing $\mathbf{F} \equiv \langle f_{j'} f_j \rangle$ (*c.f.* $\sigma_f^2 \mathbf{C}^{-1}$ in the Bayesian derivation above), we find that the optimal linear filter is:

$$\mathbf{W}_{\rm opt} = \mathbf{F}\mathbf{R}^{\rm T} \left[\mathbf{R}\mathbf{F}\mathbf{R}^{\rm T} + \sigma_\nu^2 \mathbf{I} \right]^{-1}. \tag{39.12}$$

If we identify $\mathbf{F} = \sigma_f^2 \mathbf{C}^{-1}$, we obtain the optimal linear filter (39.8) of the Bayesian derivation. The ad hoc assumptions made in this derivation were the choice of a quadratic error measure, and the decision to use a linear estimator. It is interesting that without explicit assumptions of Gaussian distributions, this derivation has reproduced the same estimator as the Bayesian posterior mode, $\mathbf{f}_{\rm MP}$.

Other image models

The better matched our model of images $P(\mathbf{f}|\mathcal{H})$ is to the real world, the better our image reconstructions will be, and the less data we will need to answer any given question. The Gaussian models which lead to the optimal linear filter fail to specify that all images are positive. This leads to the most pronounced problems where the image under observation has high contrast. Optimal linear filters applied to radio astronomical data give reconstructions with negative areas in them, corresponding to patches of sky that suck energy out of radio telescopes. The 'Maximum Entropy' model for image deconvolution (Gull and Daniell, 1978) was a great success principally because this model forced the reconstructed image to be positive. The spurious negative areas and complementary spurious positive areas are eliminated, and the dynamic range of the reconstruction is greatly enhanced.

The 'Classic maximum entropy' model assigns an entropic prior $P(\mathbf{f}|\alpha, \mathbf{m}, \mathcal{H}_{\rm Classic}) = \exp(\alpha S(\mathbf{f}, \mathbf{m}))/Z$, where $S(\mathbf{f}, \mathbf{m}) = \sum_i (f_i \log(m_i/f_i) + f_i - m_i)$ (Skilling, 1989). This model enforces positivity; the parameter α defines a characteristic dynamic range by which the pixel values are expected to differ from the default image $\mathbf{m}$.

The 'ICF maximum entropy' model (Gull, 1989) introduces an expectation of spatial correlations into the prior on $\mathbf{f}$ by writing $\mathbf{f} = \mathbf{G}\mathbf{h}$, where $\mathbf{G}$ is a convolution with an intrinsic correlation function, and putting a classic maxent prior on $\mathbf{h}$.

The 'Fermi-Dirac' model generalizes the entropy function so as to enforce an upper bound on intensity as well as the lower bound of positivity. This model is appropriate where the underlying image is bounded between two grey levels, as in the case of printed text.

All these models are implemented in the MemSys package.

39.2 Supervised neural networks for image deconvolution

'Neural network' researchers often exploit the following strategy. Given a problem currently solved with a standard data modelling algorithm: interpret the computations performed by the algorithm as a parameterized mapping from an input to an output, and call this mapping a neural network; then adapt the parameters to examples of the desired mapping so as to produce another mapping that solves the task better. By construction, the neural network can reproduce the standard algorithm, so this data-driven adaptation can (one expects) only make the performance better.

There are several reasons why standard algorithms can be bettered in this way. (1) Algorithms are often not designed to minimize the real objective function. For example, in speech recognition, a hidden Markov model is designed to model the speech signal, whereas the real objective is to discriminate between different words. If an inadequate model is being used, the neural-net-style training of the model will focus the resources of the model on the aspects relevant to the discrimination task. Discriminative training of hidden Markov models for speech recognition does improve their performance. (2) The neural network can be more flexible than the standard model; some of the adaptive parameters might have been viewed as fixed features by the original designers. (3) The net can find properties in the data that were not included in the original model.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part VI

Unsupervised learning and latent variable models

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

This part is only in draft form. The file lv.tex should be consulted to bring this up to date.

Latent Variable Models

Four lectures on models that discover the structure underlying data.

1. Introduction to latent variable models.
   - Adaptive clustering algorithms.
   - K–means clustering.
   - Soft clustering algorithms.
2. Examples of latent variable models. The EM algorithm.
   - Explanation of the EM algorithm in terms of variational free energy minimization.
3. Factor analysis and independent component analysis. Blind signal separation. Principal component analysis.
4. Temporal latent variable models.
   - Hidden Markov models. (temporal version of clustering.)
   - Kalman filters. (temporal version of factor analysis; also identical to Gaussian processes.)
   - The forward–backward algorithm.
   - Modelling of astronomical observation errors.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

40

Hidden Markov Models

40.1 Hidden Markov models

We use similar notation to Rabiner and Juang (1986).

- $S = \{s_1, s_2, ..., s_T\}$: hidden state sequence. $(s \in 1 \ldots I)$
- $X = \{x_1, x_2, ..., x_T\}$: observed sequence. $(x \in 1 \ldots M)$
- $\mathbf{A} = \{a_{ij}\}$, $a_{ij} = P(s_{t+1} = j | s_t = i)$: state transition probability matrix
- $\mathbf{B} = \{b_{im}\}$, $b_{im} = P(x_t = m | s_t = i)$: emission probabilities.
- $\pi = \{\pi_i\}$, $\pi_i = P(s_1 = i)$: initial state distribution.
- $\theta = \{\mathbf{A}, \mathbf{B}, \pi\}$: model's parameters.
- $\mathbf{U} = \{\mathbf{u}^{(A)}, \mathbf{u}^{(B)}, \mathbf{u}^{(\pi)}\}$: hyperparameters which define the prior over θ.

For given parameters θ, the probability of the hidden state sequence and the observed data is

$$P(X, S|\theta) = \left[\prod_{t=1}^{T-1} a_{s_t s_{t+1}}\right] \left[\prod_{t=1}^{T} b_{s_t x_t}\right] \pi_{s_1}. \tag{40.1}$$

The posterior probability of the hidden variables S given X and θ is given by

$$P(S|X, \theta) = \frac{1}{P(X|\theta)} \left[\prod_{t=1}^{T-1} a_{s_t s_{t+1}}\right] \left[\prod_{t=1}^{T} b_{s_t x_t}\right] \pi_{s_1}, \tag{40.2}$$

where

$$P(X|\theta) = \sum_S P(X, S|\theta). \tag{40.3}$$

We assume a prior probability over the parameters θ that is a product of Dirichlet distributions:

$$\text{Dirichlet}\,(\mathbf{p}; \mathbf{u}) \equiv \frac{1}{Z(\mathbf{u})} \prod_{i=1}^{I} p_i^{u_i - 1}\, \delta\left(\textstyle\sum_i p_i - 1\right) \tag{40.4}$$

where δ is a delta function which ensures $\mathbf{p}$ is normalized, and

$$Z(\mathbf{u}) = \frac{\prod_{i=1}^{I} \Gamma(u_i)}{\Gamma(u)}. \tag{40.5}$$

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

Here we have defined $u \equiv \sum u_i$. All the hyperparameters u_i are positive, with larger values of $\mathbf{u}$ corresponding to stronger priors. We set the prior on $\mathbf{A}$ to

$$P(\mathbf{A}|\mathbf{u}^{(A)}) = \prod_i \text{Dirichlet}\left(\{a_{i1} \ldots a_{iI}\}; \mathbf{u}^{(A)}\right), \tag{40.6}$$

with similar priors on $\mathbf{B}$ and π.

The reason for choosing these priors is that they give a direct correspondence between the standard penalized maximum likelihood method in which 'initial counts' or 'offsets' u_i are placed in the bins of the Baum-Welch algorithm and a maximum a posteriori method, if the posterior density is maximized in the 'softmax' basis (MacKay, 1998) where each probability vector $\mathbf{p}$ is represented by parameters $\mathbf{a}$ such that

$$p_i(\mathbf{a}) = e^{a_i} / \textstyle\sum_{i'} e^{a_{i'}} \ . \tag{40.7}$$

Inferring the parameters of a hidden Markov model

The traditional training algorithm for hidden Markov models is an expectation-maximization (EM) algorithm (Dempster *et al.*, 1977) known as the Baum-Welch algorithm. It is a maximum likelihood method, or, with a simple modification, a penalized maximum likelihood method, which can be viewed as maximizing a posterior probability density over the model parameters.

The Baum-Welch algorithm (with penalty terms $\mathbf{U}$) is an iterative algorithm that increases $P(\theta|X)$ at each iteration until a maximum is reached. In each iteration, a forward-backward step computes the probabilities of state sequences S conditioned on the current parameters θ, then an M-step updates the parameters θ.

The forward and backward probabilities $\alpha^{(t)}$ and $\beta^{(t)}$ are given by

$$\begin{array}{rclcrcl} \alpha_i^{(1)} & = & \pi_i b_{ix_1} & & \beta_i^{(T)} & = & b_{ix_T} \\ \alpha_j^{(t+1)} & = & \left[\sum_{i=1}^N \alpha_i^{(t)} a_{ij}\right] b_{jx_{t+1}} & & \beta_i^{(t)} & = & b_{ix_t}\left[\sum_{j=1}^N a_{ij}\beta_j^{(t+1)}\right] \end{array} \tag{40.8}$$

The M-step is expressed in terms of $n_{ij}^{(t)}$, the posterior probability that there was a transition between state i and state j at timestep t given X and θ,

$$n_{ij}^{(t)} = \sum_S P(S|X,\theta)\, \delta(s_t = i, s_{t+1} = j) \tag{40.9}$$

$$= \frac{1}{Z_n} \alpha_i^{(t)} a_{ij} \beta_j^{(t+1)} \tag{40.10}$$

where Z_n is a normalizing constant such that $\sum_{i,j=1}^I n_{ij}^{(t)} = 1$. Then the M-step is

$$a'_{ij} = \frac{u_j^{(A)} + \left[\sum_{t=1}^{T-1} n_{ij}^{(t)}\right]}{\sum_{j'=1}^I \left\{u_{j'}^{(A)} + \left[\sum_{t=1}^{T-1} n_{ij'}^{(t)}\right]\right\}}. \tag{40.11}$$

There are similar expressions for the updates of $\mathbf{B}$ and π.

Exercise 40.1:^{C3} Boltzmann chains and hidden Markov models

Under a linear Boltzmann chain model with state-state transition energies $A_{ii'}$ (going from state i to state i') and symbol emission energies B_{ij}, the

probability of an entire state $\{i_l, j_l\}_1^L$ given the length of the sequence L, is:

$$P(\{i_l, j_l\}_1^L | \Pi, \mathbf{A}, \mathbf{B}, L, \mathcal{H}_{\rm BC}) = \frac{1}{Z(\Pi, \mathbf{A}, \mathbf{B}, L)} \exp\left(-\Pi_{i_1} - \sum_{l=1}^{L-1} A_{i_l i_{l+1}} - \sum_{l=1}^{L} B_{i_l j_l} \right) \tag{40.12}$$

where $Z(\Pi, \mathbf{A}, \mathbf{B}, L)$ is the obvious normalizing constant. Here the symbol i runs over n discrete 'hidden' states, and j runs over m visible states.

In contrast, a hidden Markov model (HMM) assigns a probability distribution of the form:

$$P(\{i_l, j_l\}_1^L | \pi, \mathbf{a}, \mathbf{b}, L, \mathcal{H}_{\rm HMM}) = \pi_{i_1} \prod_{l=1}^{L-1} a_{i_l i_{l+1}} \prod_{l=1}^{L} b_{i_l j_l} \tag{40.13}$$

where π_i is a prior probability vector for the initial state, $a_{ii'}$ is a transition probability matrix, and b_{ij} is a matrix of emission probabilities satisfying respectively:

$$\sum_i \pi_i = 1, \quad \sum_{i'} a_{ii'} = 1 \ \ \forall i, \ \text{and} \ \sum_j b_{ij} = 1 \ \ \forall i. \tag{40.14}$$

Here again, the symbol i runs over an alphabet of n hidden states, and j runs over m visible states.

Whilst any HMM can be written as a linear Boltzmann chain by setting $\exp(-A_{ii'}) = a_{ii'}$, $\exp(-B_{ij}) = b_{ij}$ and $\exp(-\Pi_i) = \pi_i$, not all linear Boltzmann chains can be represented as HMMs (Saul and Jordan, 1995). However, the difference between the two models is minimal. Prove that *if the final hidden state i_L of a linear Boltzmann chain is constrained to be a particular end state, then the distribution over sequences is identical to that of a hidden Markov model.*

Solution to exercise 40.1 (p. 462): Start from the distribution (40.12) and consider the quantity in the exponent. The probability distribution over states $\{i_l, j_l\}_1^L$ is unchanged if we subtract arbitrary constants μ, ν from this exponent. The distribution will also be unaffected if we add arbitrary terms β_{i_l} to every appearance of $B_{i_l j_l}$, provided we also subtract β_{i_l} from every term $A_{i_l i_{l+1}}$. And we may similarly add $\alpha_{i_{l+1}}$ to every term $A_{i_l i_{l+1}}$ if we also subtract $\alpha_{i_{l+1}}$ from the following term $A_{i_{l+1} i_{l+2}}$. The probability distribution may therefore be rewritten unchanged (except for the normalizing constant) as:

$$P(\{i_l, j_l\}_1^L | \Pi, \mathbf{A}, \mathbf{B}, L, \mathcal{H}_{\rm BC}) \propto \exp\left(-(\Pi_{i_1} + \alpha_{i_1} + \mu) - \sum_{l=1}^{L-1} (-\alpha_{i_l} - \beta_{i_l} + A_{i_l i_{l+1}} + \alpha_{i_{l+1}} + \nu) - \sum_{l=1}^{L} (B_{i_l j_l} + \beta_{i_l}) + \alpha_{i_L} + \beta_{i_L} \right) \tag{40.15}$$

where $\mu, \nu, \{\alpha_i, \beta_i\}$ are arbitrary quantities. This probability distribution has the form of an HMM (equation 40.13) if

(a) the quantities $\pi_i \equiv \exp(-(\Pi_i + \alpha_i + \mu))$, $a_{ii'} \equiv \exp(\alpha_i + \beta_i - A_{ii'} - \alpha_{i'} - \nu)$ and $b_{ij} \equiv \exp(-(B_{ij} + \beta_i))$ satisfy the normalization conditions (40.14).

(b) the trailing term $\alpha_{i_L} + \beta_{i_L}$ can be treated as a constant, which holds if we assume that i_L is fixed to a particular end state $i_L = n$, say (a commonly applied constraint in the HMM literature).

Does a solution over $\mu, \nu, \{\alpha_i, \beta_i\}$ of the normalization conditions (40.14) exist? Trivially, we find for β_i:

$$\sum_j b_{ij} = 1 \;\Rightarrow\; \beta_i = \log \left[\sum_j \exp(-B_{ij}) \right] .$$

The normalization condition that $\{\alpha_i\}$ and ν must satisfy is:

$$\sum_{i'} \exp(\alpha_i + \beta_i - A_{ii'} - \alpha_{i'} - \nu) = 1 \quad \forall i$$

Rearranging, we obtain:

$$\sum_{i'} [\exp(\beta_i - A_{ii'})] [\exp(-\alpha_{i'})] = \exp(\nu) [\exp(-\alpha_i)] \quad \forall i \tag{40.16}$$

which can be recognised as an eigenvector/eigenvalue equation for the matrix $M_{ii'} \equiv [\exp(\beta_i - A_{ii'})]$, with $\exp(\nu)$ being the eigenvalue and $[\exp(-\alpha_i)]$ being the eigenvector. This eigenproblem has a solution, by the Perron-Frobenius theorem (Seneta, 1973:p.1), which states that a positive matrix (*i.e.*, one in which every element $M_{ii'}$ is positive) has a positive eigenvector with positive eigenvalue. A solution for $\{\alpha_i\}$ and ν therefore exists. Finally μ is given by:

$$\mu = \log \left[\sum_i \exp(-(\Pi_i + \alpha_i)) \right] .$$

This completes the proof.

The linear Boltzmann chain therefore can only differ from an HMM in having a pseudo-prior over its final state as well as a pseudo-prior over its initial state.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

41

Clustering

This chapter is a false start.

41.1 K–means clustering

As far as I know, the K in K–means clustering simply refers to the number of clusters assumed. If Newton had followed the same naming policy, maybe we would learn at school about 'calculus for the variable x'. It's a silly name, but we are stuck with it.

Competitive learning.

Imagine data in a space $\mathbf{x}$ superscript n and subscript. We will assume that the space is a real space and that we have a metric that defines distances between points

for example

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \sum_i (x_i - y_i)^2 \tag{41.1}$$

The K–means algorithm

In the K–means algorithm, K parameter vectors called the means $\mathbf{m}$ are initialized in some way, for example to random values.

The algorithm is then an iterative two–step algorithm.

Assignment step. Each data point n is assigned to the nearest mean. We denote our guess for the cluster $k^{(n)}$ that the point $\mathbf{x}^{(n)}$ belongs to by $\hat{k}^{(n)}$.

$$\hat{k}^{(n)} = \mathrm{argmin}_k \{ d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)}) \}. \tag{41.2}$$

An alternative representation of this assignment of points to clusters is given by 'responsibilities', which are indicator variables r_{nk}. In the assignment step, we set $r_k^{(n)}$ to one if mean k is the closest mean to datapoint $\mathbf{x}^{(n)}$; otherwise $r_k^{(n)}$ is zero.

$$r_k^{(n)} = \begin{cases} 1 & \text{if} \quad \hat{k}^{(n)} = k \\ 0 & \text{if} \quad \hat{k}^{(n)} \neq k \end{cases} . \tag{41.3}$$

What about ties? — We don't expect two means to be exactly the same distance from a data point, but if a tie does happen, $\hat{k}^{(n)}$ can be set to the smallest of the winning $\{k\}$.

Adaptation step. The model parameters, the means, are adjusted to match the sample means of the data points that they are responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \tag{41.4}$$

where $R^{(k)}$ is the total responsibility of mean k,

$$R^{(k)} = \sum_n r_k^{(n)}. \tag{41.5}$$

What about means with no responsibilities? — If $R^{(k)} = 0$, then we will leave the mean $\mathbf{m}^{(k)}$ where it is.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

42

Latent Variable Modelling and the E-M algorithm

42.1 Examples of Latent Variable Models

I will use the following notation. K will in general characterize the size of the latent variable space. I will characterize the size of the observable space. N denotes the number of data examples. $\mathbf{G}$ denotes the parameters of the generative model.

In mixture models, the vector $\mathbf{r}^{(n)} \equiv (r_1^{(n)}, r_2^{(n)}, \ldots, r_k^{(n)}, \ldots, r_K^{(n)})$ denotes the 'responsibilities' of mixture components for data item n. $k^{(n)}$ is the class of item n. Or should I call it $s^{(n)}$? Maybe $s^{(n)}$ will be the unary vector. Representing the *state* it was in. Also mnemonic for *source* vector. Instead of c for the unknown class. The observation vector will often be called $\mathbf{x}$.

Clustering

Factor Analysis and Independent Component Analysis

Algorithms for blind separation (Jutten and Herault, 1991; Comon *et al.*, 1991; Bell and Sejnowski, 1995; Hendin *et al.*, 1994) attempt to recover source signals $\mathbf{s}$ from observations $\mathbf{x}$ which are assumed to be linear mixtures (with unknown coefficients $\mathbf{V}$) of the source signals

$$\mathbf{x} = \mathbf{G}\mathbf{s}. \tag{42.1}$$

The algorithms attempt to create the inverse of $\mathbf{G}$ (within a post-multiplicative factor) given only a set of examples $\{\mathbf{x}\}$.

Similarly, factor analysis assumes that the observations can be described in terms of independent latent variables $\{s_k\}$ and independent additive noise. Thus the observable $\mathbf{x}$ is given by

$$\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}, \tag{42.2}$$

where $\mathbf{n}$ is a noise vector whose components have a separable probability distribution. In factor analysis t is often assumed that the probability distributions of $\{s_k\}$ and $\{n_i\}$ are zero–mean Gaussians; the noise terms may have different variances σ_i^2.

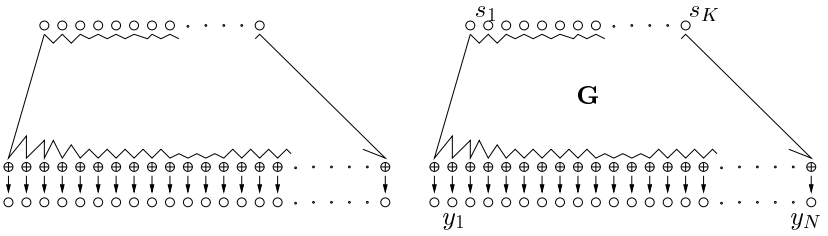


Figure 42.1. Error correcting codes as latent variable models.

Error Correcting Codes as Latent Variable Models

Imagine that a linear code with generator matrix $\mathbf{G}$ has been used to encode a random binary source vector $\mathbf{s}$ into a transmission $\mathbf{t}$, and the channel is a discrete memoryless channel with outputs $\mathbf{y}$. Then the probability distribution of the observable $\mathbf{y}$ is identical to that of a latent variable model. The K latent variables are the independent source bits $s_1, \ldots, s_K$; these give rise to the observables via the generator matrix $\mathbf{G}$ as shown in figure 42.1.

42.2 K–means Clustering

Motivations for clustering.

1. Vector quantization. This means lossy compression. Invent an assignment rule $\mathbf{x} \rightarrow k(\mathbf{x})$ for assigning datapoints $\mathbf{x}$ to one of K codes, and a reconstruction rule $k \rightarrow \mathbf{m}^{(k)}$, the aim being to minimize the expected distortion, which might be defined to be

$$D = \sum_{\mathbf{x}} P(\mathbf{x}) \frac{1}{2} \left[\mathbf{m}^{(k(\mathbf{x}))} - \mathbf{x} \right]^2. \tag{42.3}$$

   Vector quantization is used in some lossy image compression algorithms which represent small patches of image using a small alphabet of template images. The ideal objective function in this case would be to minimize the psychologically perceived distortion of the image. Since it is hard to quantify the distortion perceived by a human, vector quantization and lossy compression are not so crisply defined problems as data modelling and lossless compression.

   In vector quantization, we don't necessarily believe that the templates $\{\mathbf{m}^{(k)}\}$ have any natural meaning; they are simply tools to do a job.

2. Clustering as mixture density modelling. If we adopt the attitude of density modelling, then our aim is to find a good description of the observed data in terms of a mixture of probability densities. In contrast to vector quantization, we are likely to view the underlying clusters as having a natural meaning.

   For example, if we model handwritten characters with a mixture model, we might intend each cluster to correspond to a different character; if we model protein sequences with a mixture model, we might think of the clusters as representing protein families all of whose members descended by evolution from a common protein ancestor.

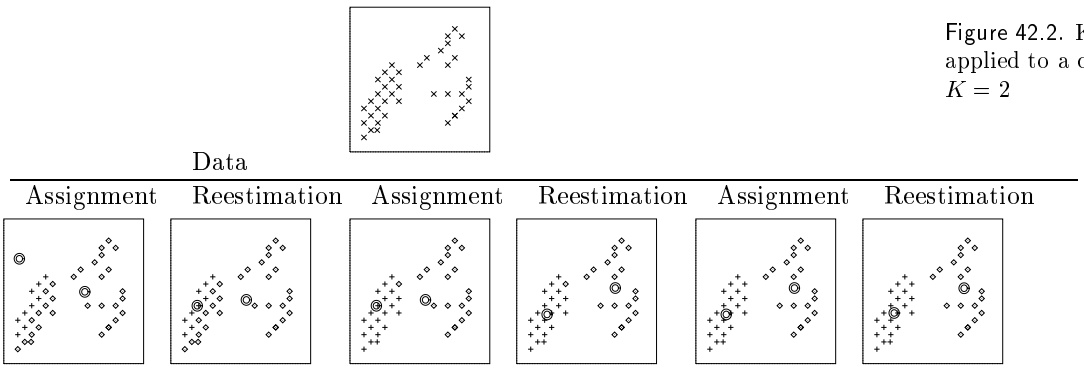


Figure 42.2. K means algorithm applied to a data set of 40 points. $K = 2$

The K–means algorithm

Complaints about this algorithm

It is a hard, rather than a soft algorithm. All points in a cluster are equally in that cluster. Borderline points have an equal vote in determining the location of the mean, if they are in, and no vote at all if they are out.

Where does the distance come from? In VQ it's chosen as a measure of perceived distortion, but in density modelling the choice of distance corresponds to a choice of density. And it certainly can have an effect on the resulting clusters.

Can get silly answers, see the big'n'small example. Algorithm implicitly assumes clusters have similar size and similar weight.

Why is the mean set to the mean?

How to do spaces other than real spaces? For example categorical spaces.

Choice of number of clusters

What about clusters with unequal width.

And clusters with unequal weight.

Cases where K means might be viewed as failing.

We can deliberately construct examples where K means gives inadequate answers, from a density modelling perspective.

Unequal weight and unequal width clusters

Figure 42.5(a) shows a set of 75 data points generated from a mixture of two Gaussians. The Gaussian centred at $(8, 5)$ differs from that centred at $(3, 5)$ in two ways: it has less weight (only one fifth of the data points); and it is a less broad cluster. The K means algorithm only takes account of the distance between the means and the data points; it has no representation of the weight or breadth of each cluster. Consequently, data points which actually belong to the broad cluster are incorrectly assigned to the narrow cluster. Once the algorithm converges, as shown in figure 42.5(b), the means have both become displaced to the left from their correct locations.

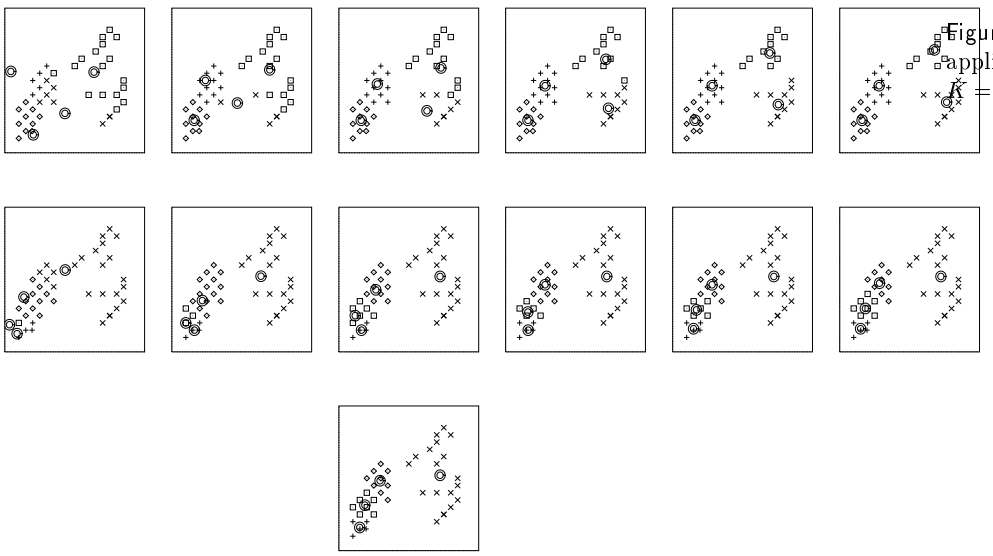


Figure 42.3. K means algorithm applied to a data set of 40 points. $K = 4$.

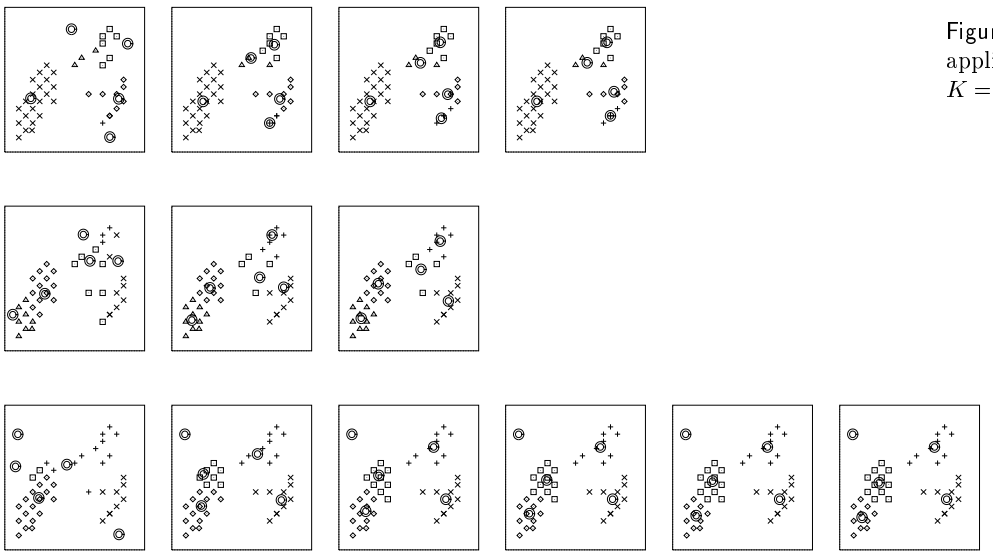


Figure 42.4. K means algorithm applied to a data set of 40 points. $K = 5$.

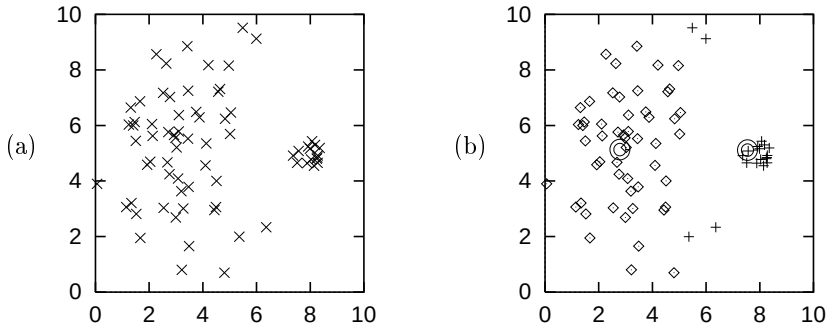


Figure 42.5. K means algorithm for a case with two dissimilar clusters. (a) Data. (b) Stable assignments and means.

Outliers

Similarly, one or two outlying data points can have a large effect on the stable state of the K means algorithm. The sample mean is a good estimator of the centre of a *Gaussian* distribution, but if a cluster is not Gaussian in shape, then the sample mean is not the most robust estimator of the centre of the cluster.

42.3 Soft K means clustering

Soft version. Write algorithm, showing how similar it is to hard K means. Give probabilistic interpretation. Spherical Gaussians version. Axis–aligned Gaussians version.

Then show that the update rules (EM) both increase a single variational objective function.

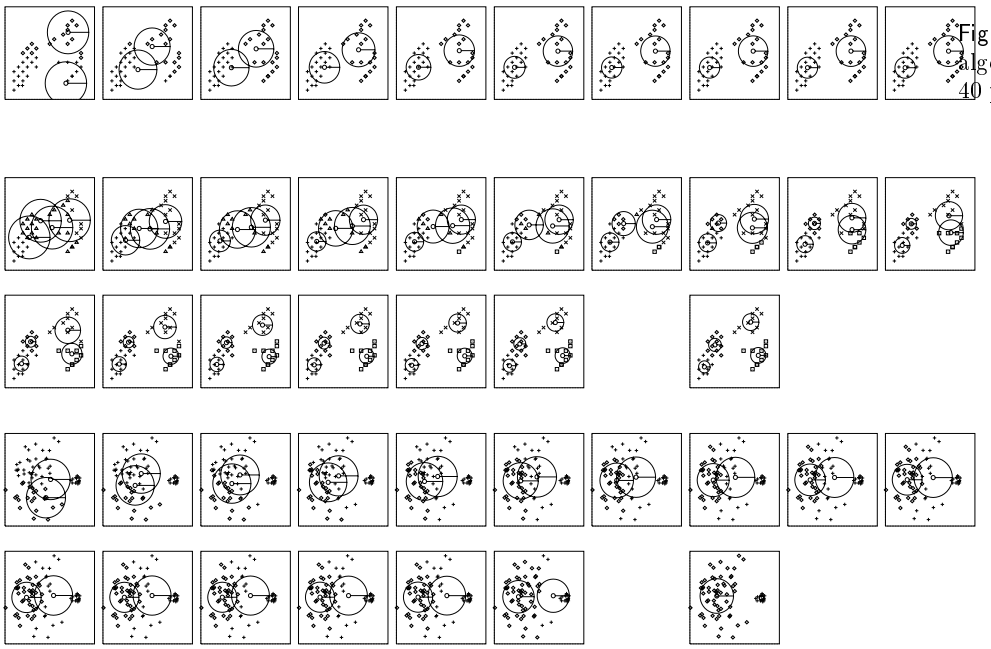


Figure 42.6. Soft K means algorithm applied to a data set of 40 points. $K = 2$.

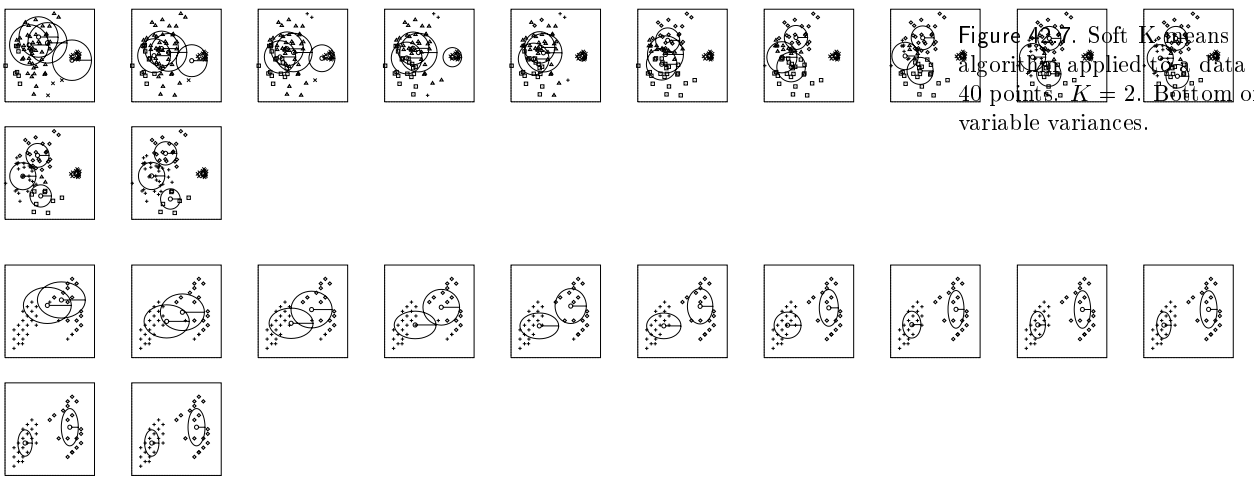
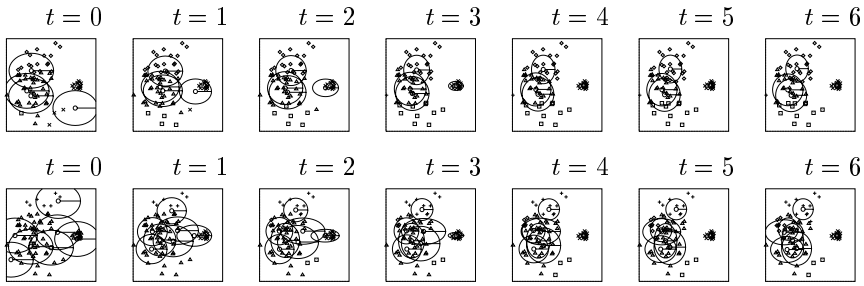


Figure 42.7. Soft K-means algorithm applied to a data set of 40 points. $K = 2$. Bottom one has variable variances.



© David J.C. MacKay. Draft 2.0.7. February 14, 2000

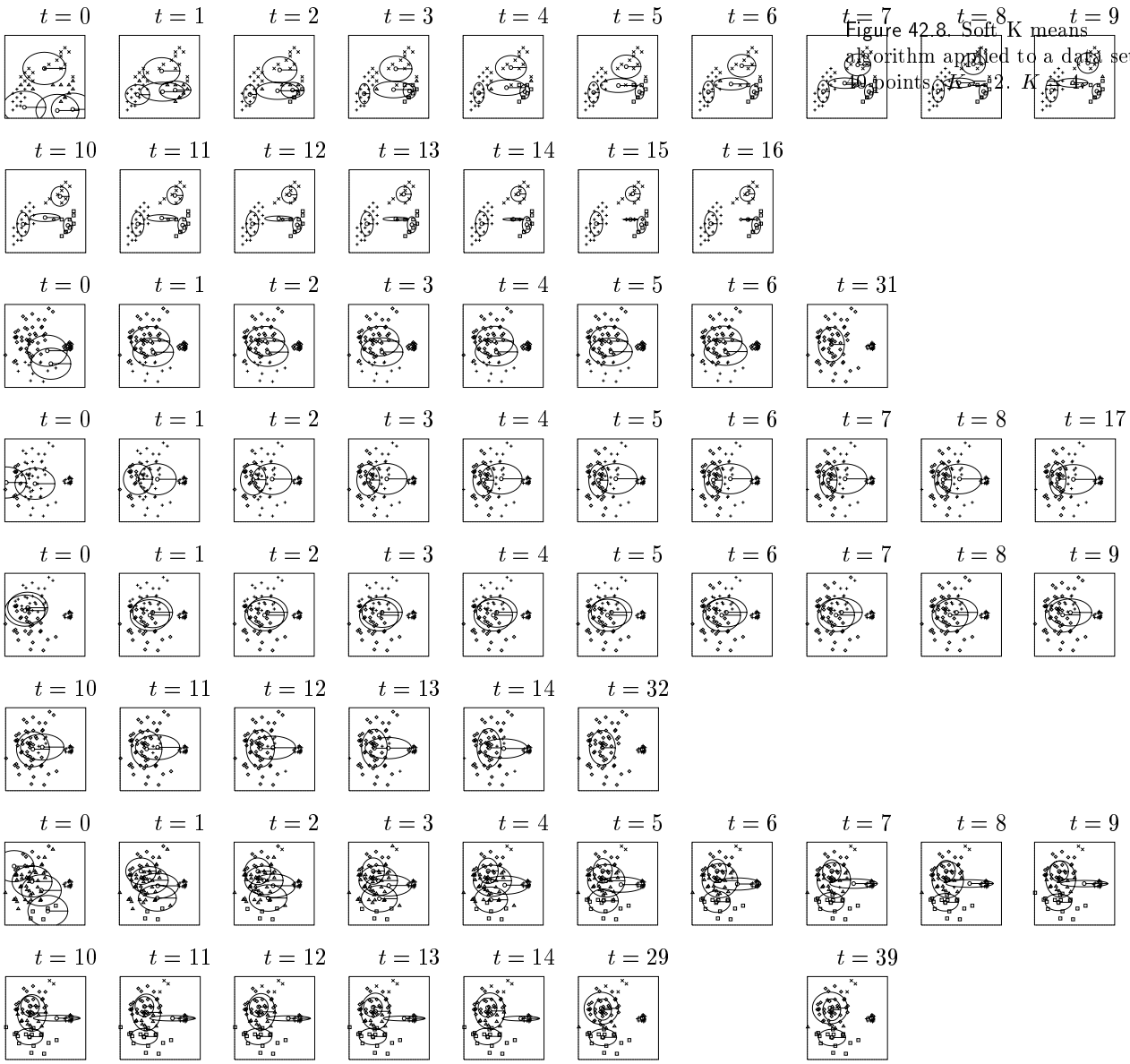


Figure 42.8. Soft K means algorithm applied to a data set of 40 points. $K = 2$. $K = 4$.

43

Continuous latent variable models

43.1 Algorithms for inferring latent variable models

Bell and Sejnowski (1995) have derived a blind separation algorithm from an information maximization viewpoint. The algorithm may be summarised as a linear mapping:

$$\mathbf{a} = \mathbf{W}\mathbf{x} \tag{43.1}$$

followed by a non-linear map:

$$z_i = \phi_i(a_i), \tag{43.2}$$

where, for example, $\phi = -\tanh(a_i)$, with a learning rule:

$$\Delta \mathbf{W} \propto [\mathbf{W}^{\mathsf{T}}]^{-1} + \mathbf{z}\mathbf{x}^{\mathsf{T}}. \tag{43.3}$$

Another non-linear function of a_i, $y_i = g(a_i)$, is also mentioned by Bell and Sejnowski, but it will not be needed here.

This paper has four parts. First it is shown that Bell and Sejnowski's (1995) algorithm may be derived as a maximum likelihood algorithm. This has been independently pointed out by Pearlmutter and Parra (1996) who also give an exciting generalization of the ICA algorithm.

Second, it is pointed out that the algorithm (43.3) is not *covariant*, and a covariant algorithm is described which is simpler, faster, and somewhat more biologically plausible. This covariant algorithm has been independently suggested by Amari *et al.* (1996) and is used by Pearlmutter and Parra (1996).

Third, this paper gives a partial proof of the 'folk-theorem' that any mixture of sources with high-kurtosis histograms is separable by the classic ICA algorithm.

Fourth, a collection of formulae are given that may be useful for the adaptation of the non-linearity in the ICA algorithm.

43.2 Maximum likelihood derivation of ICA

Latent variable models

Many statistical models are generative models that make use of latent variables to describe a probability distribution over observables (Everitt, 1984).

Examples of latent variable models include mixture models, which model the observables as coming from a superposed mixture of simple probability

distributions (Hanson *et al.*, 1991) (the latent variables are the unknown class labels of the examples); hidden Markov models (Rabiner and Juang, 1986); factor analysis; Helmholtz machines (Hinton *et al.*, 1995; Dayan *et al.*, 1995); and density networks (MacKay, 1995a; MacKay, 1996b).

Note that it is usual for the latent variables to have a simple distribution, often a separable distribution. Thus when we learn a latent variable model, we are finding a description of the data in terms of independent components. One thus might expect that an 'independent component analysis' algorithm should have a description in terms of a generative latent variable model. And this is indeed the case. Independent component analysis is latent variable modelling.

The generative model

Let us model the observable vector $\mathbf{x} = \{x_j\}_{j=1}^{J}$ as being generated from latent variables $\mathbf{s} = \{s_i\}_{i=1}^{I}$ via a linear mapping $\mathbf{G}$. The simplest derivation results if we assume $I = J$, *i.e.*, the number of sources is equal to the number of observations. The data we obtain are a set of N observations $D = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$. We assume that the latent variables are independently distributed, with marginal distributions $P(s_i|\mathcal{H}) \equiv p_i(s_i)$. Here $\mathcal{H}$ denotes the assumed form of this model and the assumed probability distributions p_i of the latent variables.

The probability of the observables and the hidden variables, given $\mathbf{G}$ and $\mathcal{H}$, is:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}, \{\mathbf{s}^{(n)}\}_{n=1}^{N}|\mathbf{G},\mathcal{H}) = \prod_{n=1}^{N}\left[P(\mathbf{x}^{(n)}|\mathbf{s}^{(n)},\mathbf{G},\mathcal{H})P(\mathbf{s}^{(n)}|\mathcal{H})\right] \quad (43.4)$$

$$= \prod_{n=1}^{N}\left[\left(\prod_j \delta\left(x_j^{(n)} - \textstyle\sum_i V_{ji}s_i^{(n)}\right)\right)\left(\prod_i p_i(s_i^{(n)})\right)\right] \quad (43.5)$$

Here it has been assumed that the vector $\mathbf{x}$ is generated without noise, because this is the assumption which leads to the Bell-Sejnowksi algorithm. It is straightforward to give another derivation in which the term $\delta\left(x_j^{(n)} - \sum_i V_{ji}s_i^{(n)}\right)$ is replaced by a probability distribution over $x_j^{(n)}$ with mean $\sum_i V_{ji}s_i^{(n)}$. If this noise distribution has sufficiently small standard deviation then the identical algorithm results.

The likelihood function

For learning about $\mathbf{G}$ from the data D, the relevant quantity is the likelihood function

$$P(D|\mathbf{G},\mathcal{H}) = \prod_n P(\mathbf{x}^{(n)}|\mathbf{G},\mathcal{H}) \quad (43.6)$$

which is a product of factors each of which is obtained by marginalizing over the latent variables. We adopt summation convention at this point, such that, for example, $V_{ji}s_i^{(n)} \equiv \sum_i V_{ji}s_i^{(n)}$. A single factor in the likelihood is given by[1]

$$P(\mathbf{x}^{(n)}|\mathbf{G},\mathcal{H}) = \int d^I\mathbf{s}^{(n)}\; P(\mathbf{x}^{(n)}|\mathbf{s}^{(n)},\mathbf{G})P(\mathbf{s}^{(n)}) \quad (43.7)$$

$$= \int d^I\mathbf{s}^{(n)}\; \prod_j \delta\left(x_j^{(n)} - V_{ji}s_i^{(n)}\right)\prod_i p_i(s_i^{(n)}) \quad (43.8)$$

[1] Recall that for scalars, $\int ds\, \delta(x - vs)f(s) = \frac{1}{v}f(x/v)$.

$$= \frac{1}{\det \mathbf{G}} \prod_i p_i(V_{ij}^{-1} x_j) \tag{43.9}$$

$$\Rightarrow \log P(\mathbf{x}^{(n)}|\mathbf{G}, \mathcal{H}) = -\log\det \mathbf{G} + \sum_i \log p_i(V_{ij}^{-1} x_j). \tag{43.10}$$

To obtain a maximum likelihood algorithm we find the gradient of the log likelihood. If we introduce $\mathbf{W} \equiv \mathbf{G}^{-1}$, the log likelihood for a single example may be written:

$$\log P(\mathbf{x}^{(n)}|\mathbf{G}, \mathcal{H}) = \log\det \mathbf{W} + \sum_i \log p_i(W_{ij} x_j). \tag{43.11}$$

We will need the following identities:

$$\frac{\partial}{\partial V_{ji}} \log\det \mathbf{G} = V_{ij}^{-1} = W_{ij} \tag{43.12}$$

$$\frac{\partial}{\partial V_{ji}} V_{lm}^{-1} = -V_{lj}^{-1} V_{im}^{-1} = -W_{lj} W_{im} \tag{43.13}$$

$$\frac{\partial}{\partial W_{ij}} f = -V_{jm} \left(\frac{\partial}{\partial V_{lm}} f \right) V_{li}. \tag{43.14}$$

Let us define $a_i \equiv W_{ij} x_j$, $\phi_i(a_i) \equiv d \log p_i(a_i)/da_i$ and $z_i = \phi_i(a_i)$, which indicates in which direction a_i needs to change to make the probability of the data greater. We may then obtain the gradient with respect to V_{ji} using equations (43.12) and (43.13):

$$\frac{\partial}{\partial V_{ji}} \log P(\mathbf{x}^{(n)}|\mathbf{G}, \mathcal{H}) = -W_{ij} - a_i z_{i'} W_{i'j}. \tag{43.15}$$

Or alternatively, the derivative with respect to W_{ij}:

$$\frac{\partial}{\partial W_{ij}} \log P(\mathbf{x}^{(n)}|\mathbf{G}, \mathcal{H}) = V_{ji} + x_j z_i. \tag{43.16}$$

If we choose to change $\mathbf{W}$ so as to ascend this gradient, we obtain precisely the learning algorithm in Bell and Sejnowski (1995) (equation 43.3).

Examples

To help explain the generative modelling viewpoint, figures 43.1a-c illustrate typical distributions generated by the independent components model when the components have $1/\cosh$ and Cauchy distributions. Figure 43.1d shows some samples from the Cauchy model. The Cauchy distribution, being the more heavy–tailed, gives the clearest picture of how the predictive distribution depends on the assumed generative parameters $\mathbf{G}$.

The two best known special cases are:

1. No nonlinearity. If $\phi_i(a_i) = -\kappa a_i$, then implicitly we are assuming a Gaussian distribution on the latent variables. It is well known that the H-J algorithm only works because of its non-linearities; if the algorithm has linear output z then all that is obtained is second-order decorrelation. Equivalently, the Gaussian distribution on the latent variables is invariant under rotation of the latent variables, so there can be no evidence favouring any particular alignment of the latent variable space.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

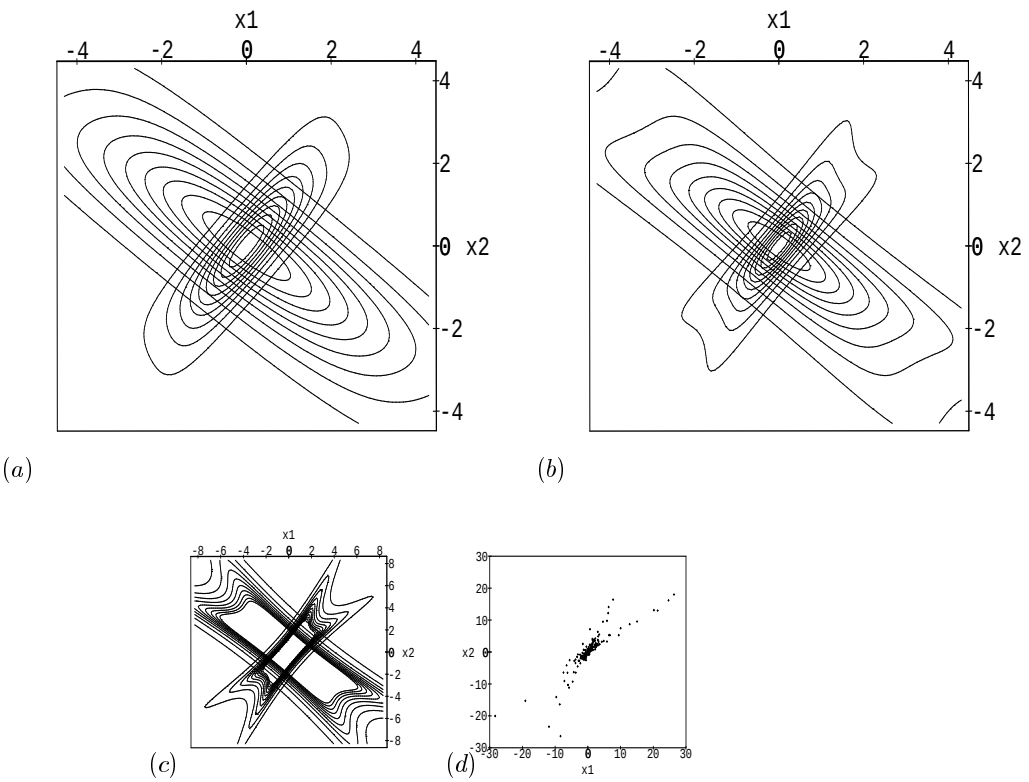


Figure 43.1. Illustration of the generative models implicit in the learning algorithm. (a) Distributions over two observables generated by $1/\cosh$ distributions on the latent variables, for $\mathbf{G} = \begin{bmatrix} 3/4 & 1/2 \\ 1/2 & 1 \end{bmatrix}$ (compact distribution) and $\mathbf{G} = \begin{bmatrix} 2 & -1 \\ -1 & 3/2 \end{bmatrix}$ (broader distribution). (b) Contours of the generative distributions when the latent variables have Cauchy distributions. The learning algorithm fits this amoeboid object to the empirical data in such a way as to maximize the likelihood. The contour plot in (b) does not adequately represent this heavy-tailed distribution. (c) Part of the tails of the Cauchy distribution, giving the contours $0.01 \ldots 0.1$ times the density at the origin. (d) Some data from one of the generative distributions illustrated in (b) and (c). Can you tell which? 200 samples were created, of which 196 fell in the plotted region.

2. A tanh nonlinearity. If $\phi_i(a_i) = -\tanh(a_i)$ then implicitly we are assuming $p_i(s_i) \propto 1/\cosh(s_i) \propto \frac{1}{e^{s_i}+e^{-s_i}}$. This is a heavier-tailed distribution for the latent variables than the Gaussian distribution. But heavier tails still would be a possibility.

3. We could also use a tanh nonlinearity with gain β, that is, $\phi_i(a_i) = -\tanh(\beta a_i)$. As β varies the implied probabilistic model changes, and is given by $p_i(s_i) \propto 1/[\cosh(\beta s_i)]^{1/\beta}$. In the limit of large β, the non-linearity becomes a step function and the probability distribution $p_i(s_i)$ becomes a biexponential distribution, $p_i(s_i) \propto \exp(-|s|)$. In the limit $\beta \to 0$ $p_i(s_i)$ approaches a Gaussian with mean zero and variance $1/\beta$.

43.3 A covariant, simpler and faster learning algorithm

We have thus derived a learning algorithm which performs steepest descents on the likelihood function.

Some designers of learning algorithms advocate the principle of covariance, which says, colloquially, that a consistent algorithm should give the same results independent of the units in which quantities are measured (Knuth, 1968).

A prime example of a *non*-covariant algorithm is the popular steepest descents rule. A dimensionless objective function $L(\mathbf{w})$ is defined, its derivative with respect to some parameters $\mathbf{w}$ is computed, and then $\mathbf{w}$ is changed by the rule

$$\Delta w_i = \eta \frac{\partial L}{\partial w_i}. \tag{43.17}$$

This popular equation is dimensionally inconsistent: the left hand side of this equation has dimensions of $[w_i]$ and the right hand side has dimensions $1/[w_i]$. The behaviour of the learning algorithm (43.17) is not covariant with respect to linear rescaling of the vector $\mathbf{w}$. Dimensional inconsistency is not the end of the world, as the success of the backpropagation algorithm has demonstrated, and indeed if η decreases with n (during on-line learning) as $1/n$ then the Munro-Robbins theorem (Bishop, 1992:p.41) shows that the parameters will asymptotically converge to the maximum likelihood parameters. But the non-covariant algorithm may take a very large number of iterations to achieve this convergence; indeed many former users of steepest descents algorithms prefer to use algorithms such as conjugate gradients that adaptively figure out the curvature of the objective function. The defense of equation (43.17) that points out η could be a dimensional constant is untenable if not all the parameters w_i have the same dimensions.

The algorithm would be covariant if it had the form

$$\Delta w_i = \eta \sum_{i'} M_{ii'} \frac{\partial L}{\partial w_i}, \tag{43.18}$$

where $\mathbf{M}$ is a matrix whose i, i' element has dimensions $[w_i w_{i'}]$. From where can we obtain such a matrix? Two sources of such matrices are *metrics* and *curvatures*.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Metrics and curvatures

If there is a natural metric that defines *distances* in our parameter space $\mathbf{w}$, then a matrix $\mathbf{M}$ can be obtained from the metric. There is often a natural choice. In the special case where there is a known quadratic metric defining the length of a vector $\mathbf{w}$, then the matrix can be obtained from the quadratic form. For example if the length is $\mathbf{w}^2$ then the natural matrix is $\mathbf{M} = \mathbf{I}$, and steepest descents is appropriate.

Another way of finding a metric is to look at the curvature of the objective function, defining $\mathbf{A} \equiv -\nabla\nabla L$ (where $\nabla \equiv \partial/\partial\mathbf{w}$). Then the matrix $\mathbf{M} = \mathbf{A}^{-1}$ will give a covariant algorithm; what is more, this algorithm is the Newton algorithm, so we recognize that it will alleviate one of the principle difficulties with steepest descents, namely its slow convergence to a minimum when the objective function is at all ill-conditioned. The Newton algorithm converges to the minimum in a single step if L is quadratic.

In some problems it may be that the curvature $\mathbf{A}$ consists of both data-dependent terms and data-independent terms; in this case, one might choose to define the metric using the data-independent terms only (Gull, 1989). The resulting algorithm will still be covariant but it will not implement an exact Newton step. Obviously there are many covariant algorithms; there is no unique choice. But covariant algorithms are a small subset of the set of *all* algorithms!

For the present maximum likelihood problem we have evaluated the gradient with respect to $\mathbf{G}$ and the gradient with respect to $\mathbf{W} = \mathbf{G}^{-1}$. Bell and Sejnowski (1995) chose to perform steepest ascents in $\mathbf{W}$, a procedure which is not covariant. Let us construct an alternative algorithm that is covariant with the help of the curvature of the log likelihood. Taking the second derivative of the log likelihood with respect to $\mathbf{W}$ we obtain two terms, the first of which is data-independent:

$$\frac{\partial V_{ji}}{\partial W_{kl}} = -V_{jk}V_{li}, \tag{43.19}$$

and the second of which is data-dependent:

$$\frac{\partial (z_i x_j)}{\partial W_{kl}} = x_j x_l \delta_{ik} z_i', \text{(no sum over } i) \tag{43.20}$$

where z' is the derivative of z. It is tempting to drop the data-dependent term and define the matrix $\mathbf{M}$ by $[M^{-1}]_{(ij)(kl)} = [V_{jk}V_{li}]$. However, this matrix is not positive definite (it has at least one non-positive eigenvalue), so it is a poor approximation to the curvature of the log likelihood, which must be positive definite in the neighbourhood of a maximum likelihood solution. We must therefore consult the data-dependent term for inspiration. The aim is to find a convenient approximation to the curvature and to obtain a covariant algorithm, not necessarily to implement an exact Newton step. What is the average value of $x_j x_l \delta_{ik} z_i'$? If the true value of $\mathbf{G}$ is $\mathbf{G}^*$, then

$$\left\langle x_j x_l \delta_{ik} z_i' \right\rangle = \left\langle V^*_{jm} s_m s_n V^*_{ln} \delta_{ik} z_i' \right\rangle . \tag{43.21}$$

We now make several severe approximations: we replace $\mathbf{G}^*$ by the present value of $\mathbf{G}$, and replace the correlated average $\langle s_m s_n z_i' \rangle$ by $\langle s_m s_n \rangle \langle z_i' \rangle \equiv \Sigma_{mn} D_i$. Here Σ is the variance–covariance matrix of the latent variables (which is assumed to exist), and D_i is the typical value of the curvature

$d^2 \log p_i(a)/da^2$. Given that the sources are assumed to be independent, Σ and D are both diagonal matrices. These approximations motivate the matrix $\mathbf{M}$ given by:

$$[M^{-1}]_{(ij)(kl)} = V_{jm}\Sigma_{mn}V_{ln}\delta_{ik}D_i, \tag{43.22}$$

that is,

$$M_{(ij)(kl)} = W_{mj}\Sigma^{-1}_{mn}W_{nl}\delta_{ik}D_i^{-1} \tag{43.23}$$

For simplicity, we further assume that the sources are similar to each other so that Σ and D are both homogeneous and that $\Sigma D = 1$. This will ead us to an algorithm that is covariant with respect to linear rescaling of the data x, but not with respect to linear rescaling of the latent variables. For problems where these assumptions do not hold, it will be straightforward to retain inhomogeneous Σ and D. We thus use:

$$M_{(ij)(kl)} = W_{mj}W_{ml}\delta_{ik} \tag{43.24}$$

Multiplying this matrix by the gradient in equation (43.16) we obtain the following covariant learning algorithm:

$$\Delta W_{ij} = \eta \left(W_{ij} + W_{i'j}a_{i'}z_i\right) \tag{43.25}$$

Notice that this expression does not require any inversion of the matrix $\mathbf{W}$. The only additional computation once $\mathbf{z}$ has been computed is a single backward pass through the weights to compute the quantity

$$x'_j = W_{i'j}a_{i'} \tag{43.26}$$

in terms of which the covariant algorithm reads:

$$\Delta W_{ij} = \eta \left(W_{ij} + x'_j z_i\right). \tag{43.27}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

44

Helmholtz machines

And other advanced methods for discovering latent variables such as the Becker–Hinton algorithm and the rectified Gaussian belief network.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

45

A single neuron as an unsupervised learning element

Oja's single neuron, from the viewpoint of latent variable modelling, P.C.A., and information maximization. Could put in more material on unsupervised Hebbian learning here, but probably the Oja neuron is enough.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part VII

Possible chapters, uncertain where to put

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

46

Valiant, PAC

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

47

NP completeness

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

48

More about Graphical models and belief propagation

Is there anything more to say?

Exact probability methods for probabilistic models related to directed acyclic graphs.

Decoding of error correcting codes on trellises.

The forward–backward algorithm for hidden Markov models.

Possibly some discussion of the junction tree algorithm, and contrasts between graphical representations of Markov networks, belief networks, and Tanner graphs.

The trick of putting a child to turn an UD into a DAG.

The primary reference for the fundamental theory of graphical models is Lauritzen (1996). A very readable introduction to Bayesian networks is given by Jensen (1996).

48.1 Markov graphs and Bayesian belief networks

In statistical data modelling and statistical physics we are often interested in a collection of variables whose interrelationships are compactly described in terms of a graph. In Markov graphs and Bayesian belief networks, the variables are represented by vertices in the graph, and dependencies between them are represented by edges. We already met several such graphs. The noise channels of chapters 1 and 9 are very simple *directed graphs* (figure 48.2b). And the Ising model of chapter 24 is a Markov graph. We also encountered a directed graph corresponding to a Markov process when we discussed the data processing inequality.

In a Markov graph, the edges in the graph describe the *conditional independences* between variables. In a Bayesian belief network, the edges are *directed*, *i.e.*, they have arrows on them; these arrows specify how the joint probability of all variables can be factorized. In the next section we will discuss Tanner graphs which give an alternative graphical representation of relationships between variables.

Notes: in a Markov graph the pdf can be written as exp(sum functions of cliques) where a clique is a collection of nodes that are fully connected. In a Tanner graph, it's a sum of functions one for each function node.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Definition of conditional independence

There are several equivalent definitions of conditional independence. We first review the definition of independence which we gave in chapter 1.

Independence. Two random variables X and Y are *independent* (sometimes written $X \perp Y$) if and only if:

$$P(x, y) = P(x)P(y). \tag{48.1}$$

Exercise 48.1: Prove that this definition of independence is equivalent to the following statement about the conditional probability of x given y:

X and Y are *independent* if and only if:

$$P(x|y) = P(x). \tag{48.2}$$

Conditional independence. Two random variables X and Y are *conditionally independent* given Z (written $X \perp Y|Z$) if and only if:

$$P(x, y|z) = P(x|z)P(y|z) \text{for all values of } z. \tag{48.3}$$

Exercise 48.2: Show that we could equivalently define conditional independence in the following ways:

- X and Y are conditionally independent given Z if and only if:

$$P(x|y, z) = P(x|z) \text{for all values of } z. \tag{48.4}$$

- X and Y are conditionally independent given Z if and only if the joint probability of x, y and z can be written:

$$P(x, y, z) = P(z)P(x|z)P(y|z). \tag{48.5}$$

The directed graphical model corresponding to this joint probability factorization is `x <- z -> y`.

Exercise 48.3: If the joint probability of x, y and z can be written

$$P(x, y, z) = P(x)P(z|x)P(y|z), \tag{48.6}$$

are X and Y conditionally independent given Z? The directed graphical model corresponding to this joint probability factorization is `x -> z -> y`.

Markov graphs

In a Markov graph, variables are represented by nodes (white circles) and there are undirected edges between variables that describe the conditional independence relationships. It must be emphasised that the edges don't simply describe which variables are correlated. Take, for example, a thermally excited linear system of masses and springs (figure 48.1). The variables $\{x_i\}$ are

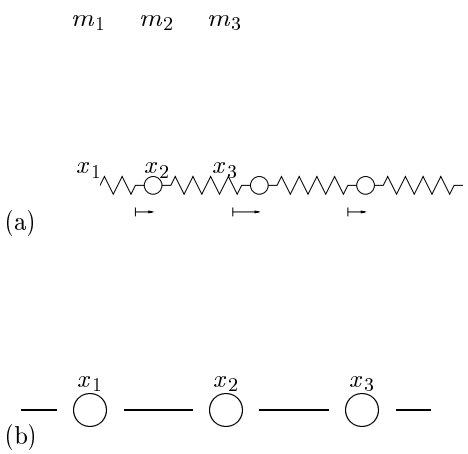


Figure 48.1. Masses and springs. (a) Physical structure. (b) Markov graph.

the displacements of the masses from equilibrium. The potential energy of a configuration is

$$E(\mathbf{x}) = \sum_i \frac{1}{2} k (x_i - x_{i+1})^2, \tag{48.7}$$

where k is the spring constant of the springs. The probability of a configuration $\mathbf{x}$ when the system is at temperature $1/\beta$ is

$$P(\mathbf{x}) = \frac{1}{Z} \exp[-\beta E(\mathbf{x})]. \tag{48.8}$$

In this system, all the variables $\{x_i\}$ are correlated with each other — if x_1 is positive, then each of the other x_i is also more likely to be positive — though the strength of this correlation decreases with distance. A graph describing correlations would thus be rather dull. The Markov graph describes conditional independences as follows. *If the graph can be separated into subsets of vertices A, B, C, such that C separates A from B, then A and B are independent given C.* The Markov graph for the masses and springs has the identical form to the physical structure.

Thus for example, if x_2 is given, then the probability of x_1 and x_3 is separable.

Graphs describing causality

In a second class of graphs, the edges represent *causual influences*. For example, if B is the 'burglar' variable such that $b = 1$ means there is a burglar in your house and $b = 0$ means there is not, and if A is the 'alarm' variable such that $a = 1$ means your burglar alarm is ringing, then (assuming your electrician is competent) there is a causal influence of B on A. The presence of the burglar ($b = 1$) makes it more likely that the alarm will ring ($a = 1$). Of course by Bayes' theorem is also true that if the alarm is ringing, then it becomes more likely that there is a burglar in your house, but there is an asymmetry between A and B which becomes evident when we include additional variables in the problem, as we will see in a moment. We represent the causal influence of B on A by a *directed edge*, as shown in figure 48.2.

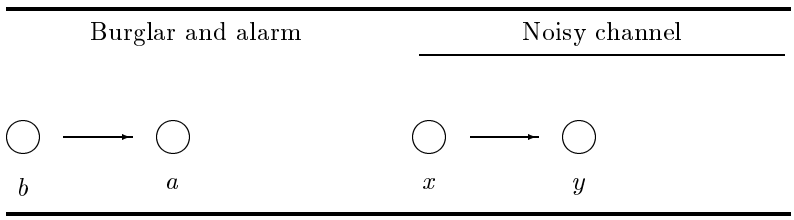


Figure 48.2. Examples of causal relationships.

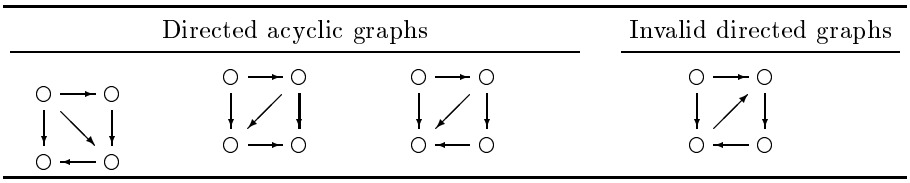


Figure 48.3. Directed graphs. The graphs on the left are directed acyclic graphs. The ones on the right are not, because there are directed cycles in them.

This figure also shows another familiar example of a causal relationship, between the input x and output y of a noisy channel (c.f. figures 1.1 etc.). If an arrow points from x to y then x is said to be a parent of y and y is said to be a child of x.

The formal meaning of a directed graph is as follows. First, for the graph to be valid, it must be a *directed acyclic graph*. This means that all the variables can be ordered in seniority, such that all the parents of a variable are more senior than it and all its children are more junior than it. Thus there are no directed cycles.

Bayesian belief networks start to get interesting when we have more than two variables in the graph. Some graphs with three variables are shown in figure 48.4.

In figure 48.4(a), variables a and b are parents of variable c. An example of such a graph is a symmetric noisy channel with a being the input of the channel, b being the noise level of the channel; the output of the channel c has a probability distribution that depends on the input and on the noise level. This graph has the following independence properties:

- the parents a and b are independent.
- once the child c is known, the parents a and b are *not* generally independent. For example, if we know the output c of the channel, then finding out the input a tells us something about the noise level.

In figure 48.4(a), an extra edge has been added from a to b. An example of a system with such a graph is as follows:

This graph has no special independence properties.

In figure 48.4(c), variables a and b are children of variable c. An example of such a graph is a broadcast channel: c is the input of the channel, and a

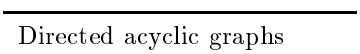


Figure 48.4. More directed graphs.

and b are the two received signals at two independent receivers, whose errors are assumed to be uncorrelated. This graph has the following independence properties:

- Given c, the children a and b are independent. For example, knowing what was transmitted, you may have a good idea of what the received signals a and b should be, but there is no correlation between them — finding out what the received signal a is doesn't make you any wiser about b.
- The children a and b are in general *not* marginally independent. For example, finding out a will usually help you make a better guess about what b is.

In figure 48.4(c), variables c only depends on a through b. An example of such a graph is a = which lectures a student attended; b = their answers to exam questions; c = the mark that they get. Your exam mark only depends on how you answer the questions. This graph has the following independence property:

- a and c are independent given b.

48.2 Tanner graphs / factor graphs

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part VIII

Tutti: unifying chapters

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

49

'Bits back'

A review of work on source coding by Hinton, Frey *et al.*. The bits back method is a practical method for source coding in cases where the probabilistic model is complex, having latent variables, etc. This topic brings together statistical data modelling, variational methods and source coding.

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part IX

Sparse Graph Codes

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

50

Introduction to sparse graph codes

In 1948, Claude Shannon posed and solved one of the fundamental problems of information theory. The question was whether it is possible to communicate reliably over noisy channels, and, if so, at what rate. He defined a theoretical limit, now known as the Shannon limit, up to which communication is possible, and beyond which communication is not possible. Since 1948, coding theorists have attempted to design error-correcting codes capable of achieving what Shannon proved is possible.

In the last decade remarkable progress has been made towards the Shannon limit, using codes that are defined in terms of sparse random graphs, and which are decoded by a simple probability–based message–passing algorithm.

This document outlines a proposed paper reviewing low density parity check codes, repeat–accumulate codes, and turbo codes, explaining how they work, and highlighting the significant steps of the last few years.

The central problem of communication theory is to construct an encoding and a decoding system that make it possible to communicate reliably over a noisy channel. The encoding system uses the source data to select a codeword from a set of codewords. The choice of this set of codewords, the 'code', is the first task. The decoding algorithm ideally infers, given the output of the channel, which codeword in the code is the most likely to have been transmitted; for an appropriate choice of distance, this is the 'closest' codeword to the received signal. Implementing this inference is the second task.

Designing a good error correcting code is difficult because (a) it is hard to find an explicit set of well–spaced codewords; and (b) for a generic code, decoding, *i.e.*, finding the closest codeword to a received signal, is intractable.

However, a simple method for designing codes, first pioneered by Gallager (1962), has recently been rediscovered and generalized. The practical performance of Gallager's codes and their modern cousins is vastly better than the performance of the codes with which textbooks have been filled in the intervening years.

In a **sparse graph code**, the nodes in the graph represent the transmitted bits and the constraints they satisfy. For a linear code with a codeword length N and rate $R = K/N$, the number of constraints is of order $M = N - K$ (there could be more constraints, if they happen to be redundant). Any linear code can be described by a graph, but what makes a sparse graph code special is that each constraint only involves a small number of variables in the graph: the number of edges in the graph scales roughly linearly with N.

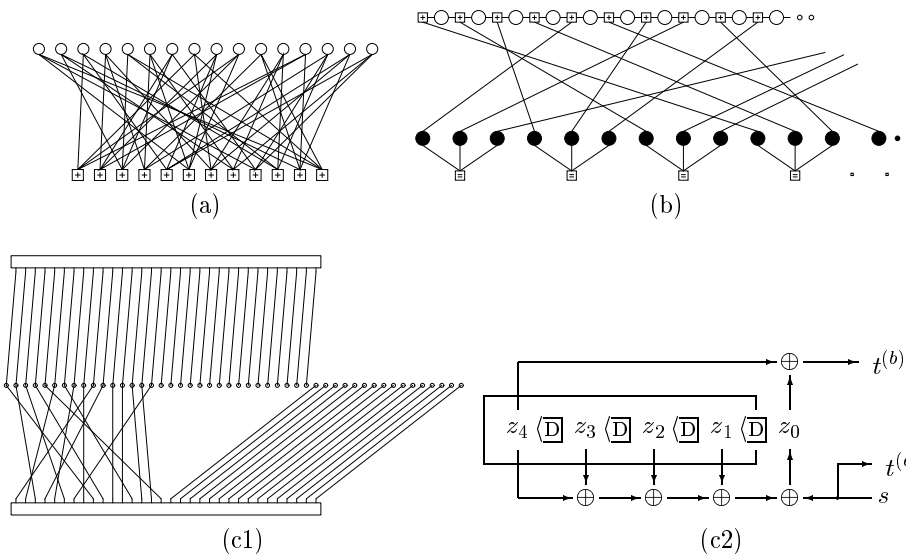


Figure 50.1. Graphs of three sparse graph codes.
(a) A rate 1/4 low density parity check code (Gallager code) with blocklength $N = 16$, and $M = 12$ constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by $\boxed{+}$ squares. Each $\boxed{+}$ constraint forces the sum of the $k = 4$ bits to which it is connected to be even.
(b) A repeat–accumulate code with rate 1/3. Each white circle represents a transmitted bit. Each black circle represents an intermediate binary variable. Each $\boxed{=}$ constraint forces the variables to which it is connected to be equal.
(c) A turbo code with rate 1/3. (c1) The circles represent the codeword bits. The two rectangles represent rate 1/2 convolutional codes, with the systematic bits occupying the left half of the rectangle and the parity bits occupying the right half. (c2) Each trellis is generated by a $(21/37)_8$ recursive filter which has a state space of 4 bits.

The graph defining a **low density parity check code** (Gallager code) contains two types of node: codeword bits, and parity constraints. In a regular (j, k) Gallager code, each codeword bit is connected to j parity constraints and each constraint is connected to k bits. The connections in the graph are made at random.

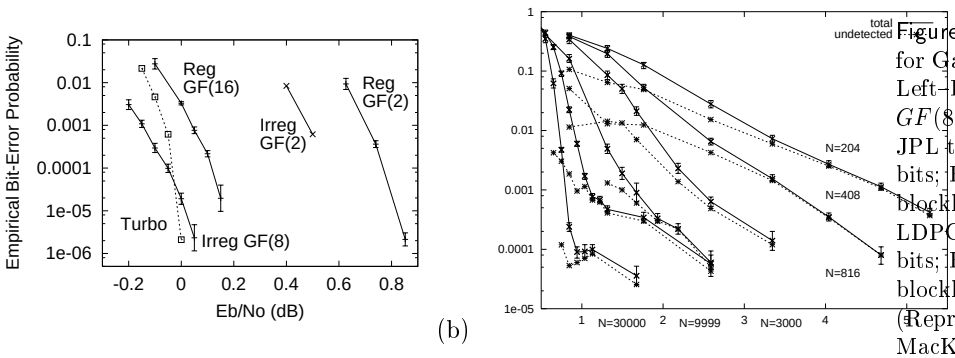
Repeat–accumulate codes (Divsalar *et al.*, 1998) can be represented by a graph with four types of node: equality constraints $\boxed{=}$, intermediate binary variables (black circles), parity constraints $\boxed{+}$, and the transmitted bits (white circles). The encoder sets each group of intermediate bits to values read from the source. These bits are put through a fixed random permutation. The $\boxed{+}$ constraints cause the transmitted stream, working from left to right, to be the accumulated sum (modulo 2) of the permuted intermediate bits.

In a **turbo code** (Berrou and Glavieux, 1996), the K source bits drive two linear feedback shift registers, which emit parity bits.

All these codes can be decoded by a local message-passing algorithm on the graph, the sum-product algorithm (MacKay and Neal, 1996; McEliece *et al.*, 1998; MacKay, 1999), and, while this algorithm is not the optimal decoder, the empirical results are record-breaking.

Which of the three types of sparse graph code is 'best' depends on the chosen rate and blocklength, the permitted encoding and decoding complexity, and the question of whether occasional undetected errors are acceptable (turbo codes and repeat–accumulate codes both typically make occasional undetected errors because they have a small number of low weight codewords; Gallager codes do not typically show such an error floor). Gallager codes are the most versatile; it's easy to make a competitive Gallager code with almost any rate and blocklength.

The best known binary Gallager codes are *irregular* codes whose parity check matrices have *nonuniform* weight per column (Luby *et al.*, 1998; Urbanke *et al.*, 1999). The carefully constructed codes of Urbanke, Richardson and Shokrollahi outperform turbo codes at long blocklengths. Turbo codes



(a) (b)

Figure 50.2. (a) Empirical results for Gaussian Channel, Rate 1/4 Left–Right : Irregular LDPC, $GF(8)$ blocklength 24000 bits; JPL turbo, blocklength 65536 bits; Regular LDPC, $GF(16)$, blocklength 24448 bits; Irregular LDPC , $GF(2)$, blocklength 64000 bits; Regular LDPC, $GF(2)$, blocklength 40000 bits. (Reproduced from (Davey and MacKay, 1998b).)
(b) Block error probability of repeat–accumulate codes with rate 1/3 and various blocklengths, shown as a function of the noise level of the binary symmetric channel. The dotted lines show the frequency of undetected redundant constraints outperform equivalent Gallager codes. Data on DSC code performance courtesy of R. Lucas and M. Fossorier. The table shows the N, M, K, distance d, and row weight k of some difference–set cyclic codes, highlighting the codes that have large d/N, small k, and large N/M. In the comparison the Gallager code had $(j,k) = (4,13)$, and rate identical to the DSC code.

DIFFERENCE SET CYCLIC CODES

N	7	21	73	273	1057	4161
M	4	**10**	**28**	**82**	**244**	**730**
K	3	11	45	191	813	3431
d	4	**6**	**10**	**18**	34	66
k	3	**5**	**9**	17	33	65

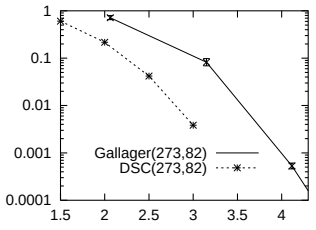


Figure 50.3. Low density parity check codes satisfying many redundant constraints outperform equivalent Gallager codes.

can also be beaten by irregular Gallager codes defined over finite fields $GF(q)$ (Davey and MacKay, 1998b). While there is a good theory for Gallager code design (Urbanke *et al.*, 1999), there is no comparable theory for the construction of irregular graphs with state variables. Given the ease with which simple repeat–accumulate codes achieve good results, it seems plausible that the best codes should make use of state variables.

The performance of Gallager codes can be enhanced by designing the code to have **redundant sparse constraints** (Tanner, 1981) (R. Lucas and M. Fossorier, personal communication). There is a difference–set cyclic code, for example, that has $N = 273$, and $K = 191$, but the code satisfies not $M = 82$ but N, *i.e.*, *273* low-weight constraints (figure 50.3). It is impossible to make random Gallager codes that have anywhere near this much redundancy among their checks.

An open problem is to discover codes sharing the remarkable properties of the difference–set cyclic codes but with different blocklengths and rates. I call this task **the Tanner challenge**.

51

Low density parity check codes

These codes can be decoded by belief propagation. Their performance is excellent. They lend themselves to theoretical analysis too: it is easy to prove that, given an optimal decoder, these codes are asymptotically 'very good', *i.e.*, approach the Shannon limit.

These codes can also be decoded by a variational free energy minimization method — this gives a connection to Hopfield networks for constraint satisfaction problems.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

52

Convolutional codes

This chapter, which follows tightly on from chapter 20, is a prerequisite for Turbo codes. It makes use of the ideas of codes and trellises and the forward–backward algorithm.

This chapter is in a very rough and ready state! Please don't trouble to send any feedback on it yet.

52.1 Introduction to Convolutional Codes

When we studied linear block codes, we described them in three ways:

1. The generator matrix describes how to turn a string of K arbitrary source bits into a transmission of N bits.

2. The parity chack matrix specifies the $M = N - K$ parity check constraints that a valid codeword satisfies.

3. The trellis of the code describes its valid codewords in terms of paths through a trellis with labelled edges.

   Trellises are a powerful way to describe codes. Some codes that can be generated by simple finite state machines have cumbersome generator matrices, but their trellises are simple. Trellises lend themselves to a description of decoding in terms of probability propagation. And some non–linear codes can be conveniently described by trellises too.

A fourth way of describing some block codes is the algebraic approach, but this approach is not covered in my book because it has been well covered by numerous other books CITE, and because, as we will see in chapters GALLAGER and TURBO, the state of the art in error correcting codes makes no use of algebraic coding theory.

We will now describe convolutional codes in two ways: first, in terms of an algorithm for generating transmissions $\mathbf{t}$ from source bits $\mathbf{s}$; and second in terms of trellises that describe the constraints satisfied by valid transmissions.

52.2 Linear feedback shift registers

We generate a transmission with a convolutional code by putting our source stream through a linear filter. This filter may make use of: a shift register; linear output functions; and linear feedback.

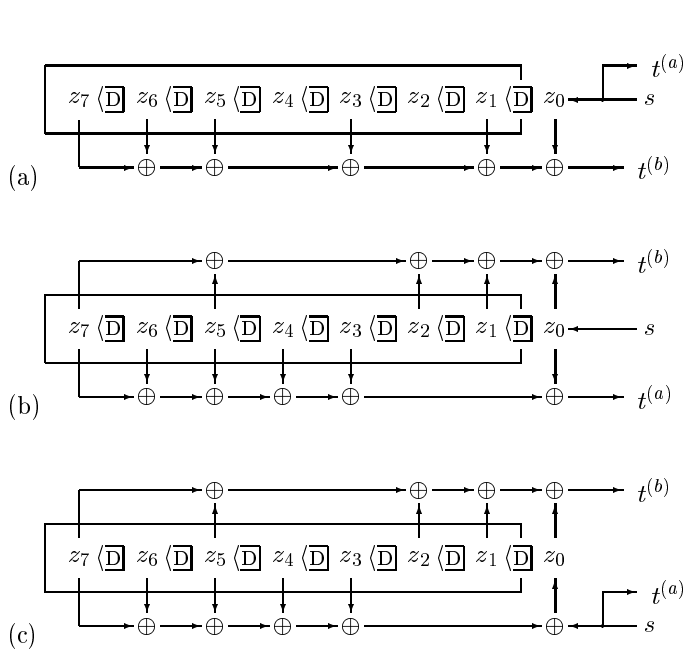


Figure 52.1. Linear feedback shift registers for generating convolutional codes with rate 1/2. After Frey (1998).

I will show the shift register in a right–to–left orientation so that bits roll from right to left as time goes on. This orientiation may be unconventional, but it has some notational advantages.

Figure 52.1 shows three linear feedback shift registers which could be used to define convolutional codes. The symbol $\boxed{\text{D}}$ indicates a copying with a delay of one clock cycle. The symbol $\oplus$ denotes linear addition modulo 2 with no delay. The rectangular box surrounding the bits $z_1 \ldots z_7$ indicate the *memory* of the filter, also known as its *state*. All three filters have one input and two outputs. On each clock cycle, the source supplies one bit, and the filter outputs two bits $t^{(a)}$ and $t^{(b)}$. By concatenating together these bits we can obtain from our source stream $s_1 s_2 s_3 \ldots$ a transmission stream $t_1^{(a)} t_1^{(b)} t_2^{(a)} t_2^{(b)} t_3^{(a)} t_3^{(b)} \ldots$. Because there are two transmitted bits for every source bit, the codes shown in figure 52.1 have rate 1/2. Because these filters have $K = 7$ bits of memory, the codes they define are known as a *constraint–length 7 codes*.

Systematic nonrecursive

The filter shown in figure 52.1(a) is a special case of linear feedback shift register with no feedback. It also has the property that one of the output bits, $t^{(a)}$ is identical to the source bit s. This encoder is thus called *systematic*, because the source bits are reproduced transparently in the transmitted stream, and *nonrecursive*, because it has no feedback. The other transmitted bit $t^{(b)}$ is a linear function of the state of the filter. One way of describing that function is as a dot product between two binary vectors of length $K+1$: a binary vector $\mathbf{g}^{(b)} = (FILLMEIN)$ and the state vector $\mathbf{z} = (z_K, z_{K-1}, \ldots, z_1, z_0)$. Note

that we have incorporated the bit z_0 which will be put into the first bit of the memory on the next cycle, into the state vector. The vector $\mathbf{g}^{(b)}$ is made by setting $g_k^{(b)} = 1$ if there is a tap (a downward pointing arrow) from state bit z_k into the transmitted bit $t^{(b)}$.

A convenient way to describe these binary tap vectors is in octal. Thus, this filter makes use of the tap vector $number_8$. ($FILLMEIN$). I have drawn the delay lines from right to left to make it easy to relate the figures to these octal numbers.

Nonsystematic nonrecursive

The filter shown in figure 52.1(b) also has no feedback. But it is not systematic. It makes use of two tap vectors $\mathbf{g}^{(a)}$ and $\mathbf{g}^{(b)}$ to create its two transmitted bits. This encoder is thus *nonsystematic* and *nonrecursive*. Because of their added complexity, nonsystematic codes can have error–correcting abilities superior to those of systematic nonrecursive codes with the same constraint length.

Systematic recursive

The filter shown in figure 52.1(c) is similar to the nonsystematic nonrecursive filter shown in figure 52.1(b), but it makes use of the taps that formerly made up $\mathbf{g}^{(a)}$ to make a linear signal that is fed back into the shift register along with the source bit. The output $t^{(b)}$ is a linear function of the state vector as before. The other output is $t^{(a)} = s$, so this filter is systematic.

Equivalence of systematic recursive and nonsystematic nonrecursive codes

These two filters are *code–equivalent* in that the set of valid codewords that they define are identical. This can be seen by showing that for every codeword of the nonsystematic nonrecursive code we can choose a source stream for the other code such that its output is identical.

Let's add a label to the diagram, denoting by p the quantity $\sum_{k=1}^{K} g_k^{(a)} z_k$, as shown in figure 52.2(a) and (b), which shows a pair of smaller but otherwise equivalent filters. Clearly if the two transmissions are to be equivalent — that is, the $t^{(a)}$s are equal in both figures and so are the $t^{(b)}$s — then on every cycle the source bit in the systematic code must be $s = t^{(a)}$. So now we must simply confirm that for this choice of s, the systematic code's shift register will follow the same state sequence as that of the nonsystematic code, assuming that the states match initially. In figure 52.2(a) we have

$$t^{(a)} = p \oplus z_0^{\text{nonrecursive}} \tag{52.1}$$

whereas is figure 52.2(b) we have

$$z_0^{\text{recursive}} = t^{(a)} \oplus p. \tag{52.2}$$

Substituting for $t^{(a)}$, and using $p \oplus p = 0$ we immediately find

$$z_0^{\text{recursive}} = z_0^{\text{nonrecursive}}. \tag{52.3}$$

Thus, any codeword of a nonsystematic nonrecursive code is a codeword of a systematic recursive code with the 'same' taps — the same, in the sense

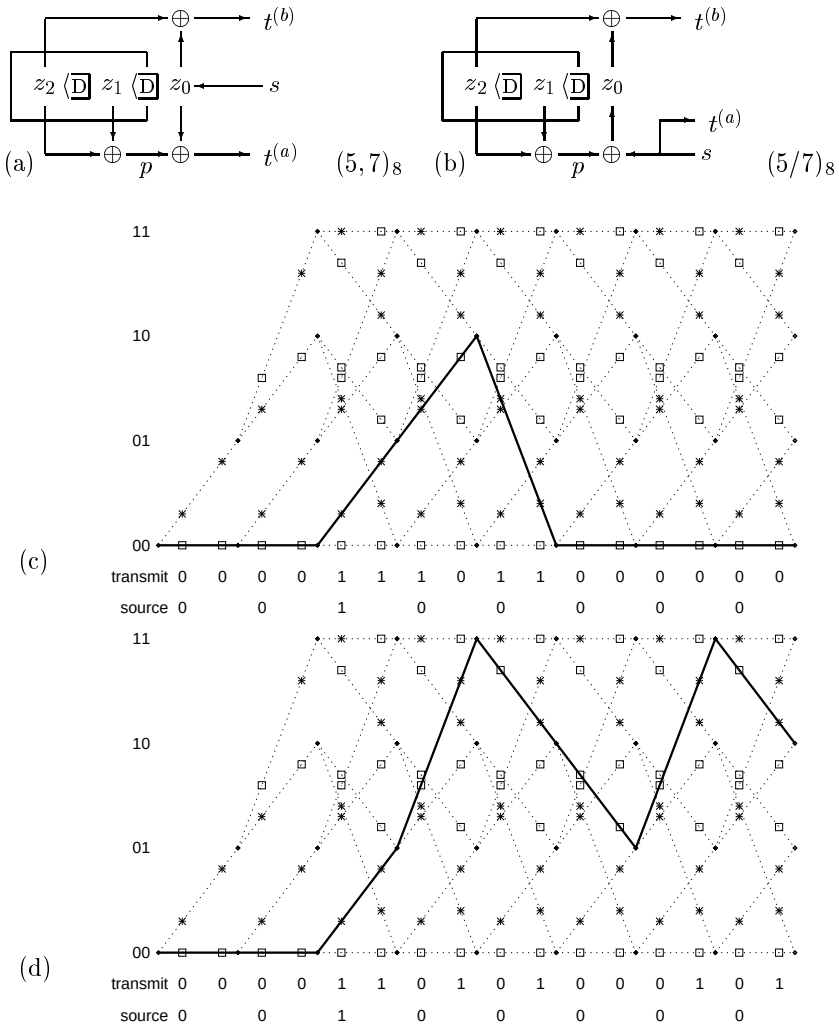


Figure 52.2. Trellis of $K = 2$ rate 1/2 convolutional codes. $K = 2$. These two are compared to explain how recursive and non–recursive are equivalent. Comparison of trellises for NN and SR. Paths taken through trellis when source = 00100000. NN first transmission = 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0. Note finite implulse response. SR second, transmission = 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1. But note that there is a path through either trellis corresponding to the other.

(b)

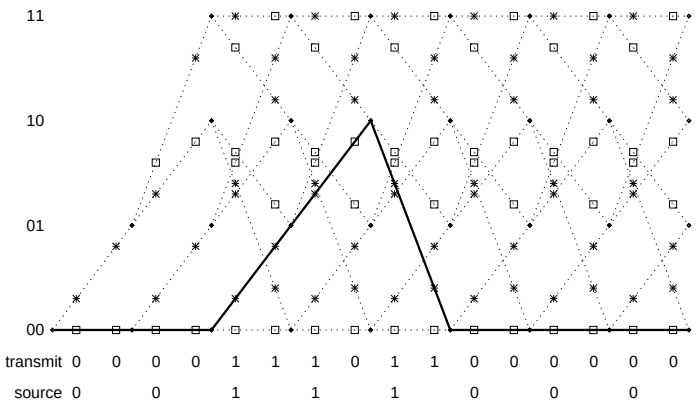


Figure 52.3. There exists a source sequence for the SR such that the path through the trellis is the same as in the NN case. Path taken through SR trellis when source = (00111000). SR transmission = 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,

that there are vertical arrows in all the same places in figure 52.2(a) and (b), though one of the arrows points up instead of down in (b).

Now, while these two codes are equivalent, they behave rather differently as generators. The nonrecursive code has a *finite impulse response*, that is, if one puts in a string that is all zeroes except for a single one, the resulting output stream contains a finite number of ones, as shown in figure 52.2(c). Once the one bit has passed through all the states of the memory, the delay line is back in the all–zero state and there is no record of that bit's passing. The figure shows the trellis of the nonrecursive code of figure 52.2(a), assuming that the initial state of the filter is $(z_2, z_1) = (0, 0)$. This figure shows time on the horizontal axis and the state of the filter at each time step by the vertical coordinate. The solid line shows the state sequence resulting from the source string $\mathbf{s} = (0, 0, 1, 0, 0, 0, 0, 0)$. On the line segements are shown the emitted symbols $t^{(a)}$ and $t^{(b)}$, with stars for '1' and boxes for '0'. The light dotted lines show the other state trajectories that are possible for other source sequences.

Figure 52.2(d) shows the trellis of the recursive code of figure 52.2(b), assuming that the initial state of the filter is $(z_2, z_1) = (0, 0)$, and it shows by a solid line the response of this filter to the same source string $\mathbf{s} = (0, 0, 1, 0, 0, 0, 0, 0)$. The filter has an *infinite impulse response*, which after the first '11' is periodic with period equal to three clock cycles.

So, where has the equivalence of the codes gone? Well, the two trellises in figure 52.2(c) and (d) are identical, and for any state sequence of the nonrecursive filter there are source sequences for the recursive filter such that its state sequence and its output are the same as those of the nonrecursive filter. For example, if we put $\mathbf{s} = (0, 0, 1, 1, 1, 0, 0, 0, \ldots)$ into the recursive filter, as shown in figure 52.3, the state sequence and transmission are identical to those shown in figure 52.2(c).

In general a linear feedback shift register with K bits of memory will have an impulse response that is periodic with a period that is at most $2^K - 1$, corresponding to the filter visiting every non–zero state in its state space. If $2^K - 1$ is factorizable into a product of smaller integers, then depending on the choice of the taps the period may be equal to one of those smaller integers. Note connection to cheap pseudorandom number generators.

Comment

Note with the trellis we can show more general codes than the generator description, for example, we can add an extra two bits to the choice of codeword by allowing any start state. These two bits aren't explicitly transmitted, but are implicit in the transmission since each path through the trellis corresponds to a unique transmission.

Indeed can go further down this route: can throw away the input and one of the outputs, and simply load the source bits into the memory, then let the LFSR roll. The sequence of values of z_0 is transmitted. This defines a cyclic code. *(check ADJECTIVES)*

For appropriate choice of taps, the period of the LFSR is large. This is used in crypto as a cheap way to turn a short key into a pseudorandom bit stream. Thus decoding and cryptanalysis closely related here.

A recursive code is conventionally described by the octal ratio eg 21/37.

52.3 The decoding problem

To keep things general, let's discuss the case where the initial start state is not known.

The receiver receives a bit stream, and wishes to infer the state sequence and thence the source stream. Let's first consider the noise–free decoding problem. Since it's systematic, can simply read out the relevant bits. What about recovering the initial condition? In principle there might be two paths through the trellis which have identical systematic bits but different parity bits and different initial condition. Figure 52.4(a) shows a selection of one path through the trellis of figure 52.4(b), corresponding to the transmission $\mathbf{t} = (0,0,0,0,1,1,1,0,1,1,0,0,0,0,0,0)$. Also highlighted are the other edges in the graph which match each pair of bits. For example, at the left hand side a downward sloping line corresponds to a '00'. None of these lines connect to each other, which confirms that no other path through the trellis produces the same transmission.

Figure 52.4(b) shows the situation when the transmission of figure 52.4(a) is received with one bit, the 8th, flipped, so that the received vector is $\mathbf{r} = (0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0)$. There are three line styles, depending on the value of the likelihood: thick solid lines show the edges in the trellis which match the corresponding two bits of the received string exactly; thick dotted lines show edges that match one bit but mismatch the other; and thin dotted lines show the edges that mismatch both bits. The Viterbi algorithm seeks the path through the trellis which uses as many solid lines as possible; more precisely, it minimizes the cost of the path, where the cost is zero for a solid line, one for a thick dotted line, and two for a thin dotted line.

There is only one path which has the minimum cost of one, and that is the path shown in figure 52.2(c). Thus the Viterbi algorithm can correct some single errors.

Now look at the case shown in figure 52.5. Figure 52.6 shows two hypotheses useful for discussing figure 52.5. They differ in the last source bit only. The a figure is the only good hypothesis for b2, but in b1, both hypotheses are equally good. This illustrates the unequal protection of bits and motivates *termination* of the trellis.

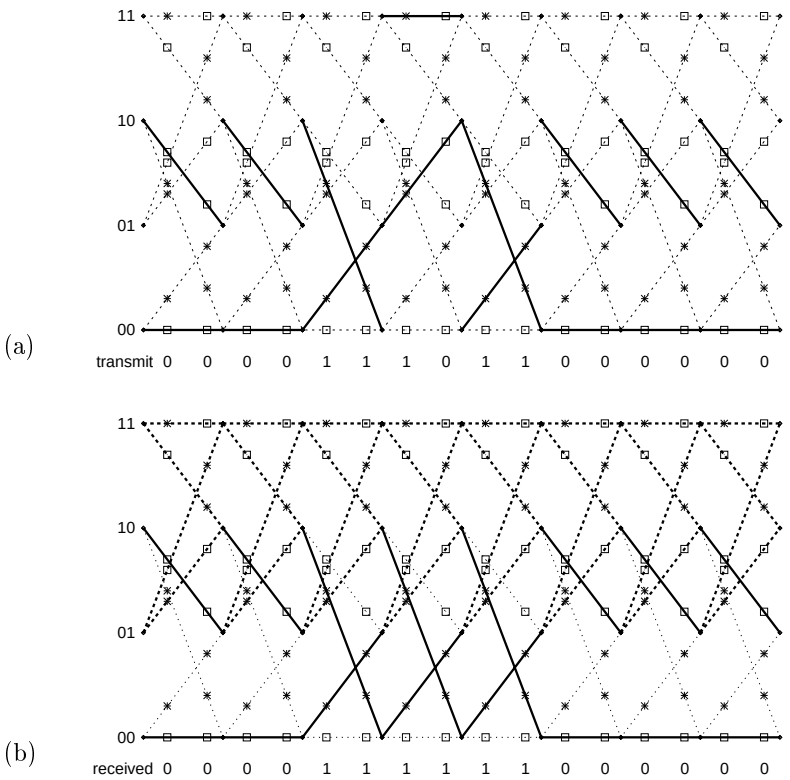
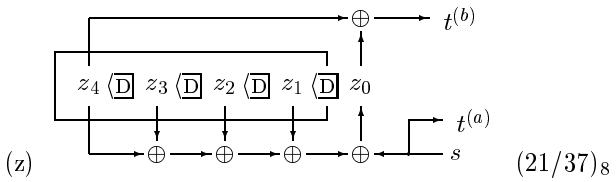


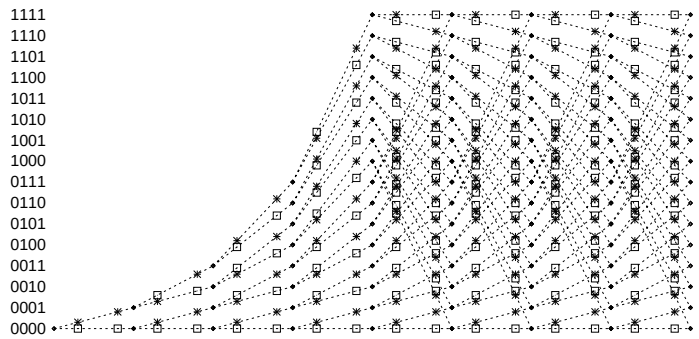
Figure 52.4. A transmission and a received vector, shown by the corresponding likelihoods for all the edges.

Figure 52.7 shows the likelihood function for the received vector equal to a random received vector, to illustrate how difficult the decoding problem gets when there is a lot of noise.

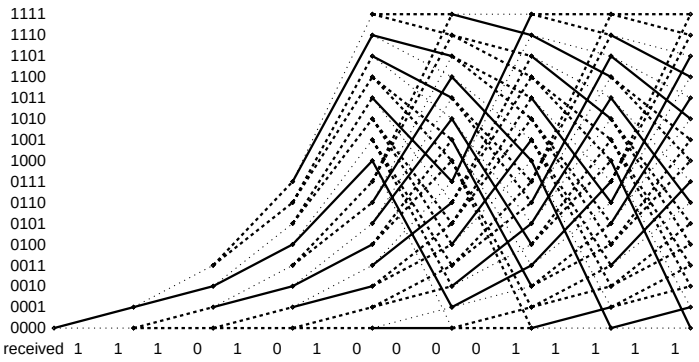
Termination is shown in figure 52.8. Note that terminating makes the protection uniform. Seems that the end bits get the best protection now.



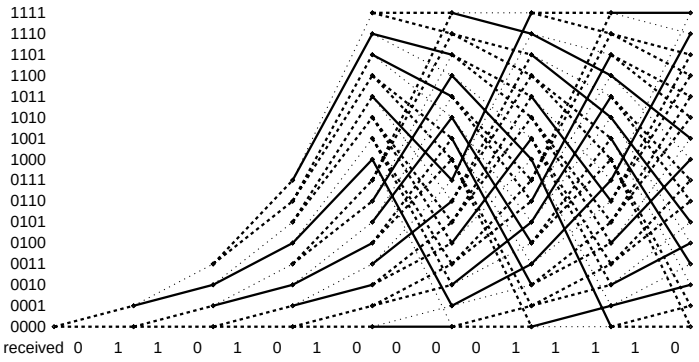
(z) $(21/37)_8$



(a)

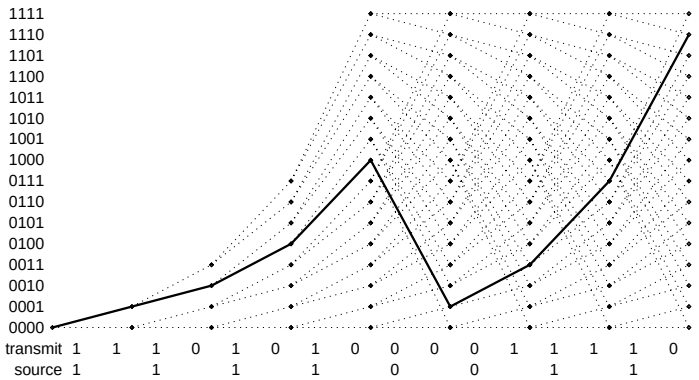


(b1)

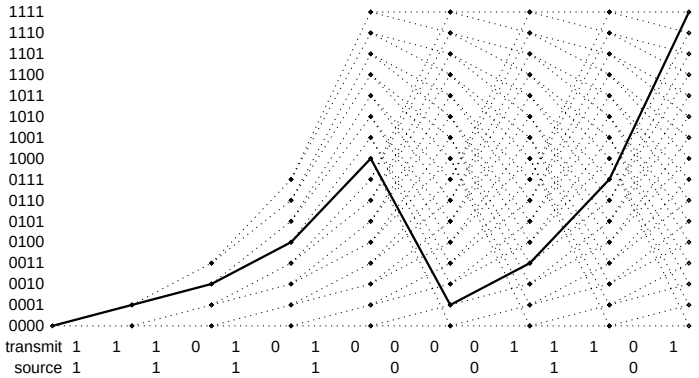


(b2)

Figure 52.5. The trellis for a $K = 4$ code and (b1) the likelihood function for the received vector equal to a codeword with just one bit flipped; (b2) the likelihood function for the received vector equal to a codeword with a different one bit flipped. Notice also in b1 that there are pairs of paths which are identical for many successive bits. This figure illustrates the unequal protection of bits and motivates termination of the trellis.

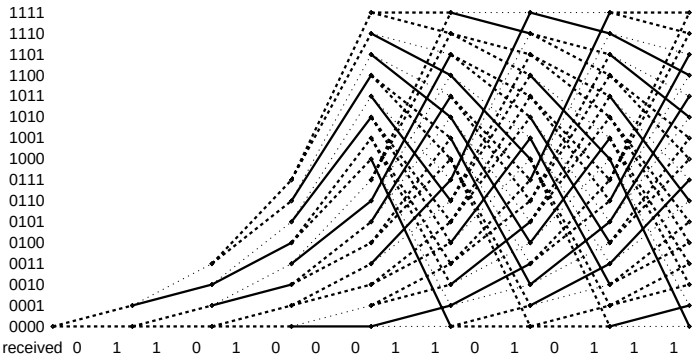


(a)



(b)

Figure 52.6. Two hypotheses useful for discussing figure 52.5. They differ in the last source bit only. The a figure is the only good hypothesis for b2, but in b1, both hypotheses are equally good.



(a)

Figure 52.7. The likelihood function for the received vector equal to a random received vector, to illustrate how difficult the decoding problem gets when there is a lot of noise.

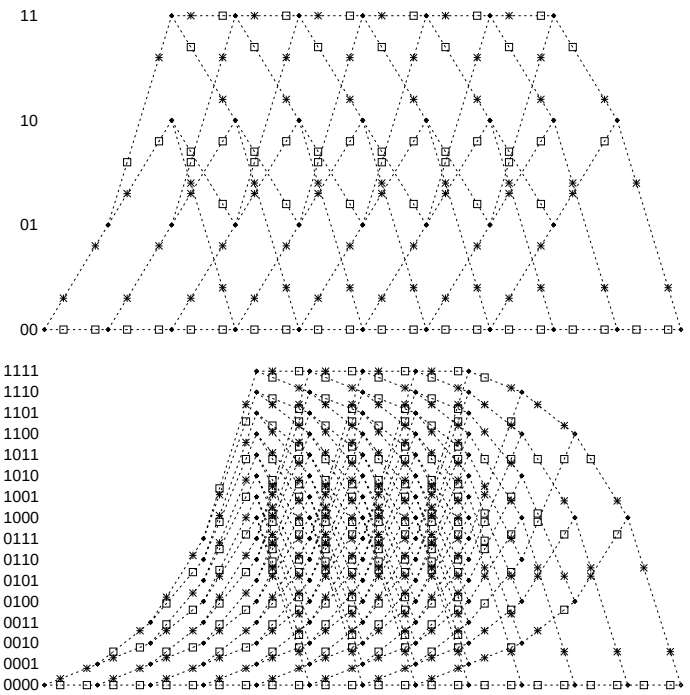


Figure 52.8. Terminated trellises

53

Turbo codes

The state of the art in error–correcting codes, these codes are decoded by belief propagation.

Furthermore, Turbo codes *are* low density parity check codes.

53.1 Draft material on Turbo codes

A low density parity check code is a block code which has a parity check matrix, $\mathbf{H}$, every row and column of which is 'sparse'.

A regular Gallager code is a low density parity check code in which every column of $\mathbf{H}$ has the same weight t and every row has the same weight t_r; regular Gallager codes are constructed at random subject to these constraints.

An (N, K) Turbo code is defined by a number of constituent encoders (often, two) and an equal number of 'interleavers' which are $K \times K$ permutation matrices. Without loss of generality, we take the first interleaver to be the identity matrix. The constituent encoders are often convolutional codes. A string of K source bits is encoded by feeding them into each constituent encoder in the order defined by the associated interleaver, and transmitting the bits that come out of each constituent encoder. For simplicity, let us concentrate on Turbo codes with two constituent codes that are both convolutional codes. Often the first constituent encoder is chosen to be a systematic encoder, and the second is a non–systematic one which emits parity bits only. The transmitted codeword then consists of K source bits followed by M_1 parity bits generated by the first convolutional code and M_2 parity bits from the second.

For the purposes of this paper we will not need to discuss the decoder for either of these codes.

One unifying viewpoint for these two code families is in terms of trellis constrained codes. A trellis constrained code is a code whose codewords satisfy a set of constraints, each constraint being compactly described by a trellis in which two or more of the codeword bits participate. Viewing these codes as trellis constrained codes, they appear rather different. The $M \times N$ parity check matrix of a regular Gallager code defines M trellises. Each trellis constrains the parity of t_r of the bits to be even, and each of the N bits participates in t trellises. We can think of a Turbo code as a trellis constrained code in which there are two trellises (figure **??**); the K source bits and the first M_1 parity bits participate in the first trellis and the K source bits and the last M_2 parity bits participate in the second trellis. Each codeword bit participates in either

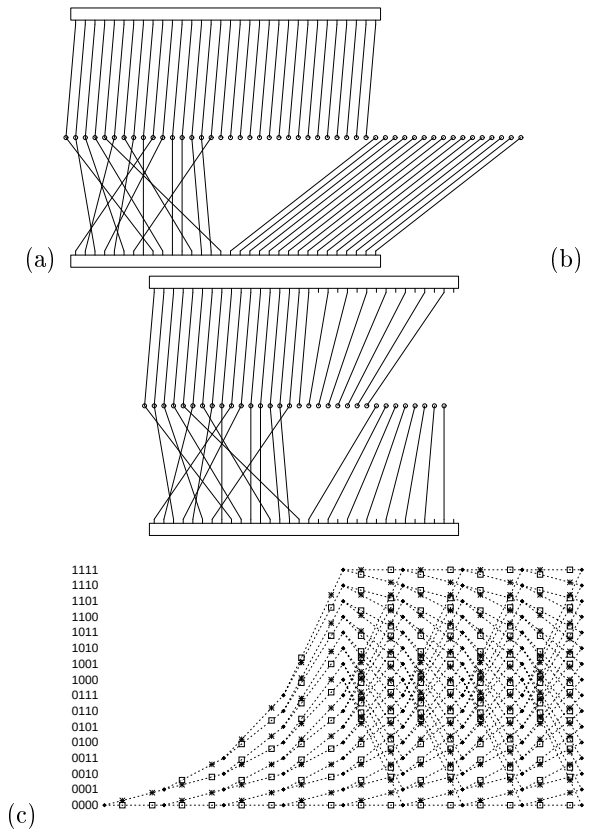


Figure 53.1. (a,b) A rate 1/3 (a) and rate 1/2 (b) Turbo code represented as trellis constrained codes. The circles represent the codeword bits. The two rectangles represent trellises of rate 1/2 convolutional codes, with the systematic bits occupying the left half of the rectangle and the parity bits occupying the right half. The puncturing of these constituent codes in the rate 1/2 Turbo code is represented by the lack of connections to half of the parity bits in each trellis. (c) The contents of the rectangles. Each trellis is generated by the recursive filter shown in figure 53.3 which has a state space of 4 bits. In each pair of adjacent bits, one is a systematic bit and one is a parity bit.

one or two trellises, depending on whether it is a parity bit or a source bit. See figure 53.2.

However, we will now see that from the point of view of their parity check matrices, Turbo codes are actually very similar to Gallager codes.

The parity check matrix of a single convolutional code

Note different meaning for parity check matrix from convolutional code literature. Here we are talking about the literal parity check matrix of the code

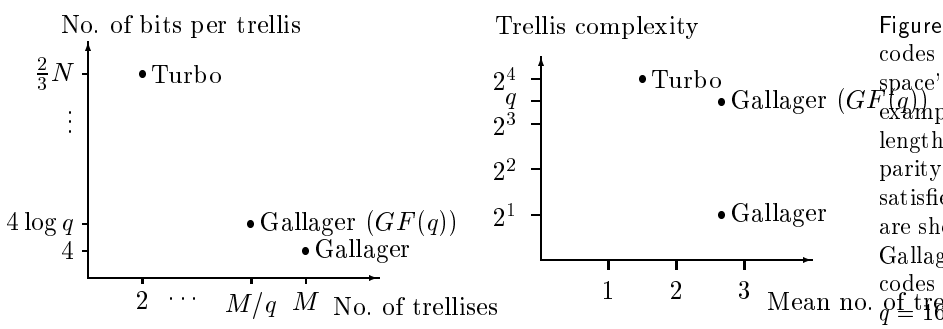


Figure 53.2. One view of Gallager codes and Turbo codes in 'code space', using rate 1/3 codes as an example. N denotes the block length and M is the number of parity constraints, $M = N - K$, satisfied by a codeword. Points are shown for ordinary binary Gallager codes and for Gallager codes over $GF(q)$ also ($q = 8$ and $q = 16$ have been found useful, cite Davey and MacKay). From this perspective, Gallager codes and Turbo codes seem quite different.

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

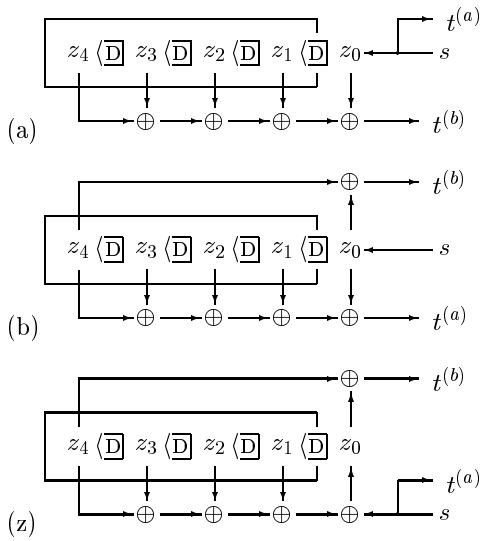


Figure 53.3. Linear feedback shift registers for generating convolutional codes. $K = 4$.

viewed as a linear block code.

A systematic recursive convolutional code, as used in Turbo codes, is equivalent (in the sense that its codewords are the same) to a nonsystematic nonrecursive convolutional code (figure **??**(b)). Now, what parity constraints are satisfied by the latter code? Well, if we pass stream b through the convolutional filter that generated stream a and vice versa, then the two resulting streams are identical. So the parity check matrix of a single convolutional code may be written as a low density parity check matrix as shown in figure **??**(b).

Issue neglected here: termination. Termination simply adds an extra k constraints, where k is the constraint length. Not a big deal.

The parity check matrix of a Turbo code

Note that for the standard constraint length 4 convolutional codes, the profile of the turbo code's parity check matrix is roughly K columns of weight about 4, and the remaining columns of weight about 5; the row weight is about 7 for all rows.

Here puncturing ignored. How to handle it: if the bit only participates in one check, remove that check. If it participates in more than one check, use row manipulations to create new higher weight checks that don't involve that bit.

Note that classic Turbo codes are punctured down to rate 1/2.

53.2 Consequences

Some of the theoretical results proved for Gallager codes carry over to Turbo codes. The positive ones (for example, that Gallager codes are 'very good') don't carry over since they rely on creating the whole matrix at random. But

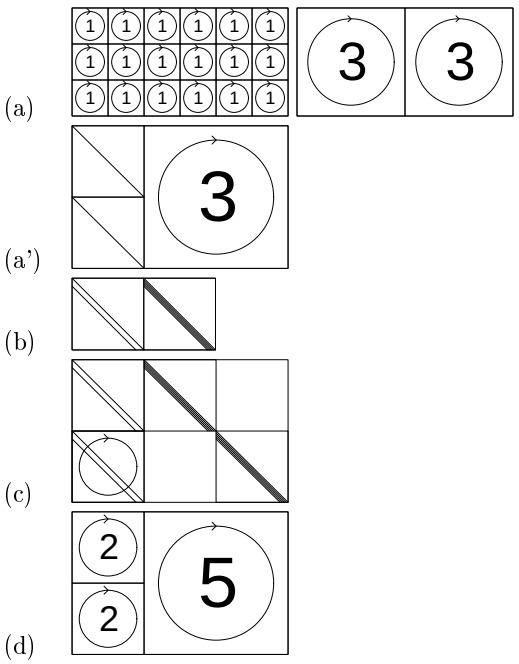


Figure 53.4. Schematic pictures of the parity check matrices of (a) a regular Gallager code, rate 1/2, (a') an almost regular Gallager code, rate 1/3, (b) a convolutional code, rate 1/2, and (c) a Turbo code, rate 1/3. Notation: A diagonal line represents an identity matrix. A band of diagonal lines represent a band of diagonal 1s. A circle inside a square represents the random permutation of all the columns in that square. A number inside a square represents the number of random permutation matrices superposed in that square. Horizontal and vertical lines indicate the boundaries of the blocks within the matrix. (d) shows another code with roughly the same profile as a Turbo code.

the negative ones do. One negative result is that Turbo codes definitely can't get to the Shannon limit unless the constraint length of the constituent codes grows — just as Gallager codes are only very good as the weight per column t increases.

53.3 Discussion

For those of us who thought that there was a considerable distance in 'code space' between Turbo codes and Gallager codes, this observation forces a shift in viewpoint. It also allows us to roll out a few simple theorems about the maximum likelihood performance of Turbo codes.

So, given that they are such similar codes, what are the differences? Why are regular Gallager codes worse than Turbo codes? We have caught up with Turbo codes only by (1) making them over GF(16); (2) making them irregular. But notice these Turbo codes are binary and their parity check matrices are pretty near regular!

I have some ideas about why Turbo are better.

Notice that the standard way of decoding a Gallager code is not how Turbo codes are decoded. Message passing different and would be more inaccurate in the Gallager style. Turbo sends messages all the way along trellises, so the within–trellis messages are correct.

Wasted checks

Consider a Gallager code with $t_r = 4$. Imagine that of the four bits that participate in one check, three of them happen to be well–determined given

the output of the channel. This check allows us instantly to correct the fourth bit, but then it plays no useful role. This is not the way a good code works!

In contrast, in a Turbo code, if some bits are well determined, the sole effect is to prune the trellis. Whatever bits are well–determined, the remaining bits participate in a trellis that looks (locally) just the same.

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Part X

Appendices

© David J.C. MacKay: Draft 2.0.7. February 14, 2000

A

Useful formulae, etc.

A.1 Statistical physics

A.2 About phase transitions

A system with states $\mathbf{x}$ in contact with a heat bath at temperature $T = 1/\beta$ has probability distribution

$$P(\mathbf{x}|\beta) = \frac{1}{Z(\beta)} \exp(-\beta E(\mathbf{x})). \tag{A.1}$$

The partition function is

$$Z(\beta) = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})). \tag{A.2}$$

The inverse temperature β can be interpreted as defining an exchange rate between entropy and energy.

Often, the system will be affected by some other parameters such as the volume of the box it is in, V, in which case Z is a function of V too, $Z(\beta, V)$.

For any system with a finite number of states, the function $Z(\beta)$ is evidently a continuous function of β. Moreover, all the derivatives of $Z(\beta)$ with respect to β are continuous too.

What phase transitions are all about, however, is this: phase transitions correspond to values of β and V at which Z has discontinuities or divergences in its derivatives.

Immediately we can deduce:

> Only systems with an infinite number of states can show phase transitions.

Often, we include a parameter N describing the size of the system. Phase transitions may appear in the limit $N \to \infty$. Real systems may have a value of N like 10^{23}.

If we make the system large simply by grouping together N independent systems whose partition function is $Z_{(1)}(\beta)$, then nothing interesting happens. The partition function is simply

$$Z_{(N)}(\beta) = [Z_{(1)}(\beta)]^N. \tag{A.3}$$

Now, while this function $Z_{(N)}(\beta)$ may be a very rapidly varying function of β, that doesn't mean it is showing phase transitions. The natural way to look

at the partition function is in the logarithm (which is roughly the same as the free energy)

$$\log Z_{(N)}(\beta) = N \log Z_{(1)}(\beta). \tag{A.4}$$

Duplicating the original system N times simply scales up all properties like the energy and heat capacity of the system by a factor of N. So if the original system showed no phase transitions then the scaled up system won't have any either.

Only systems with long–range correlations show phase transitions.

This is a vague assertion, sorry.... can it be tightened? Should emphasise that there do not have to be direct long–range couplings; short–range energetic couplings are enough to give rise to long–range correlations.

Why are points at which derivative diverge interesting?

The derivatives of $\log Z$ describe properties like the heat capacity of the system (that's the second derivative) or its fluctuations in energy. If the second derivative of $\log Z$ diverges at a temperature $1/\beta$, then the heat capacity of the system diverges there, which means it can absorb or release energy without changing temperature (think of ice melting in ice water) and the energy of the system fluctuates a lot at equilibrium, in contrast to the normal law–of–large–numbers behaviour, under which the energy only varies by one part in $\sqrt{N}$.

A toy system that shows a phase transition

Imagine a collection of N spins that have the following energy as a function of their state $\mathbf{x} \in \{0,1\}^N$.

$$E(\mathbf{x}) = \begin{cases} -N\epsilon & \mathbf{x} = (0,0,0,\ldots,0) \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

This energy function describes a ground state in which all the spins are aligned in the zero direction; the energy per spin in this state is $-\epsilon$. if any spin changes state then the energy is zero. This model is like an extreme version of a magnetic interaction, which encourages pairs of spins to be aligned.

We can contrast it with an ordinary system of N independent spins whose energy is:

$$E(\mathbf{x}) = \epsilon \sum_n (2x_n - 1) \tag{A.6}$$

The partition function is given by

$$Z(\beta) = e^{\beta N \epsilon} + 2^N - 1. \tag{A.7}$$

The function

$$\log Z(\beta) = \log \left(e^{\beta N \epsilon} + 2^N - 1 \right) \tag{A.8}$$

is sketched in figure A.1(a) along with its low temperature behaviour,

$$\log Z(\beta) \simeq N \beta \epsilon, \quad \beta \to \infty, \tag{A.9}$$

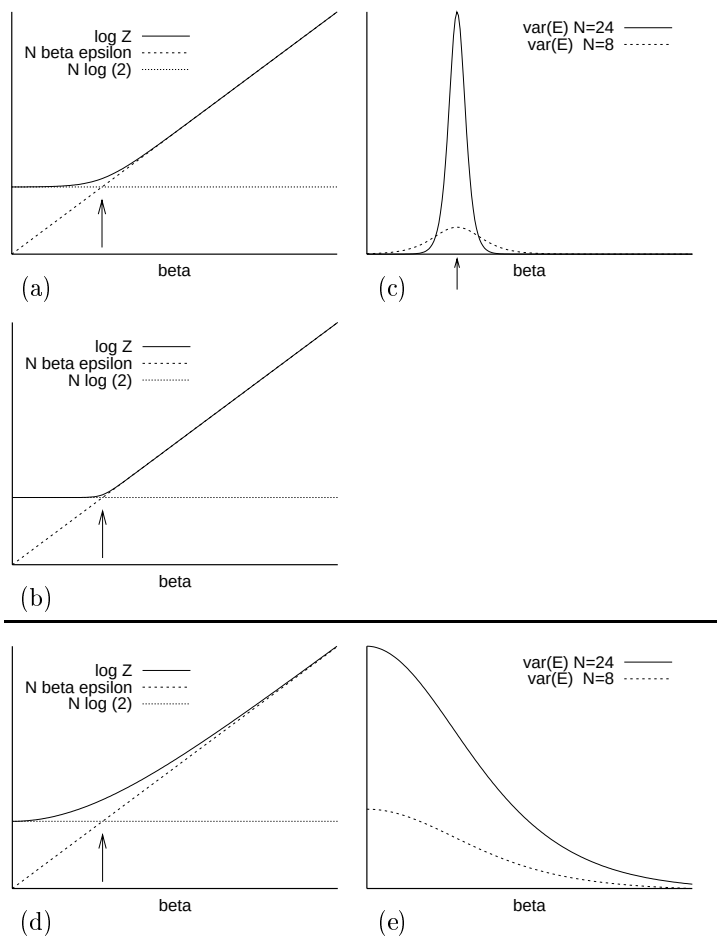


Figure A.1. (a) Partition function of toy system which shows a phase transition for large N. The arrow marks the point $\beta_c = \log 2/\epsilon$. (b) The same, for larger N. (c) The variance of the energy of the system as a function of β for two system sizes. As N increases the variance has an increasingly sharp peak at the critical point β_c. Contrast with (d,e) the partition function and energy–variance of a system consisting of N independent spins. The partition function changes gradually from one asymptote to the other, regardless of how large N is; the variance of the energy does not have a peak. The fluctuations are largest at high temperature and scale linearly with system size N.

and its high temperature behaviour,

$$\log Z(\beta) \simeq N \log 2, \quad \beta \rightarrow 0. \tag{A.10}$$

The arrow marks the point

$$\beta = \frac{\log 2}{\epsilon} \tag{A.11}$$

at which these two asymptotes intersect. In the limit $N \rightarrow \infty$, the graph of $\log Z(\beta)$ becomes more and more sharply bent at this point (figure A.1(b)).

The second derivative of $\log Z$, which describes the variance of the energy of the system, has a peak value, at $\beta = \log 2/\epsilon$, roughly equal to

$$\frac{N^2 \epsilon^2}{4}, \tag{A.12}$$

which corresponds to the system spending half of its time in the ground state and half its time in the other states. Figure A.1(b) show the heat capacity.

At this critical point, the heat capacity of this system is thus proportional to N^2; the heat capacity per spin is proportional to N, which, for infinite N, is infinite, in contrast to the behaviour of ordinary non–critical systems, whose capacity per atom is a finite number.

More generally

Phase transitions can be categorized by the degree of the derivative of $\log Z$ that diverges. In a first–order phase transition, the second derivative diverges, I think; in a second–order phase transition, the third derivative diverges? Have to check.

In the vicinity of a critical point, the concept of 'typicality' does not hold. For example, our toy system, at its critical point, has a 50% chance of being in a state with energy $-N\epsilon$, and roughly a $1/2^{N+1}$ chance of being in each of the other states with energy zero. It is thus not the case that $\log 1/P(\mathbf{x})$ is very likely to be close to the entropy of the system at this point, unlike a system with N i.i.d. components.

A.3 Finite field theory

Most linear codes are expressed in the language of Galois theory

Let me explain why Galois fields are an appropriate language. First, a definition and some examples.

A Field F is a set $F = \{0, F'\}$ such that

1. F forms a group[1] under an addition operation '+', with 0 being the identity;
2. F' forms a group[2] under a multiplication operation '$\cdot$'; multiplication of any element by 0 yields 0;
3. these operations satisfy the distributive rule $(a + b) \cdot c = a \cdot c + b \cdot c$.

[1] In fact an Abelian group, *i.e.*, one that satisfies commutativity — $a + b = b + a$.
[2] Also an Abelian group.

For example, the real numbers form a field, with '+' and '·' denoting ordinary addition and multiplication.

A Galois Field $GF(q)$ is a field with a finite number of elements q.

A unique Galois field exists for any $q = p^m$, where p is a prime number and m is a positive integer; there are no other finite fields.

$GF(2)$. The addition and multiplication tables for $GF(2)$ are as follows:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

These are the rules of addition and multiplication modulo 2.

$GF(p)$. For any prime number p, the addition and multiplication rules are those for ordinary addition and multiplication, modulo p.

$GF(4)$. The rules for $GF(p^m)$, with $m > 1$, are *not* those of ordinary addition and multiplication. The tables for $GF(4)$ are:

$+$	0	1	A	B
0	0	1	A	B
1	1	0	B	A
A	A	B	0	1
B	B	A	1	0

$\cdot$	0	1	A	B
0	0	0	0	0
1	0	1	A	B
A	0	A	B	1
B	0	B	1	A

These are *not* the rules of addition and multiplication modulo 4. Notice that 1+1=0, for example. So how can $GF(4)$ be described? It turns out that the elements can be related to *polynomials*. Consider polynomial functions of x of degree 1 and with coefficients that are elements of $GF(2)$.

Element	Polynomial	Bit pattern
0	0	00
1	1	01
A	x	10
B	$x+1$	11

These polynomials obey the addition and multiplication rules of $GF(4)$ *if* addition and multiplication are modulo the polynomial $x^2 + x + 1$, and the coefficients of the polynomials are from $GF(2)$. For example, $B \cdot B = x^2 + 2x + 1 = x = A$.

Each element may also be represented as a bit pattern as shown in the above table, with addition bitwise modulo 2, and multiplication defined with an appropriate carry operation.

Why are Galois fields relevant to linear codes? Imagine generalizing a binary generator matrix $\mathbf{G}$ and binary vector $\mathbf{s}$ to a matrix and vector with elements from a larger set, and generalizing the addition and multiplication operations that define the product $\mathbf{Gs}$. In order to produce an appropriate input for a symmetric channel, it would be convenient if, for random $\mathbf{s}$, the product $\mathbf{Gs}$ produced all elements in the enlarged set with equal probability. This is

easiest to ensure if these elements form a group under both addition and multiplication, because then these coding operations do not break the symmetry among the elements. When two random elements of a multiplicative group are multiplied together, all elements are produced with equal probability. This is not true of other sets such as the integers, for which the multiplication operation is more likely to give rise to some elements (the composite numbers) than others. Galois fields, by their definition, avoid such symmetry-breaking effects.

A.4 A code over $GF(8)$

We can denote the elements of $GF(8)$ by $\{0, 1, A, B, C, D, E, F\}$. Each element can be mapped onto a polynomial over $GF(2)$.

element	polynomial	binary representation
0	0	000
1	1	001
A	x	010
B	$x+1$	011
C	x^2	100
D	x^2+1	011
E	x^2+x	110
F	x^2+x+1	111

(A.13)

The multiplication and addition operations are given by multiplication and addition of the polynomials, modulo $x^3 + x + 1$.

Here is the multiplication table:

$\cdot$	0	1	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0
1	0	1	A	B	C	D	E	F
A	0	A	C	E	B	1	F	D
B	0	B	E	D	F	C	1	A
C	0	C	B	F	E	A	D	1
D	0	D	1	C	A	F	B	E
E	0	E	F	1	D	B	A	C
F	0	F	D	A	1	E	C	B

(A.14)

Here is a (9,2) code over $GF(8)$ generated by the generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & A & B & C & D & E & F \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{A.15}$$

000000000	011111111	0AAAAAAAA	0BBBBBBBB	0CCCCCCCC	0DDDDDDDD	0EEEEEEEE	0FFFFFFFF
101ABCDEF	110BADCFE	1AB01EFCD	1BA10FEDC	1CDEF01AB	1DCFE10BA	1EFCDAB01	1FEDCBA10
A0ACEB1FD	A1BDFA0EC	AA0EC1BDF	AB1FD0ACE	ACE0AFDB1	ADF1BECA0	AECA0DF1B	AFDB1CE0A
B0BED FC1A	B1AFCED0B	BA1CFDEB0	BB0DECFA1	BCFA1B0DE	BDEB0A1CF	BED0B1AFC	BFC1A0BED
C0CBFEAD1	C1DAEFBC0	CAE1D C0FB	CBF0CD1EA	CC0FBAE1D	CD1EABF0C	CEAD10CBF	CFBC01DAE
D0D1CAFBE	D1C0DBEAF	DAFBE0D1C	DBEAF1C0D	DC1D0EBFA	DD0C1FAEB	DEBFAC1D0	DFAEBD0C1
E0EF1DBAC	E1FE0CABD	EACDBF10E	EBDCAE01F	ECABD1FE0	EDBAC0EF1	EE01FBDCA	EF10EACDB
F0FDA1ECB	F1ECB0FDA	FADF0BCE1	FBCE1ADF0	FCB1EDA0F	FDA0FCB1E	FE1BCF0AD	FF0ADE1BC

Exercise A.1: Is this code a perfect code?

Exercise A.2: Is this code a maximum distance separable code?

Solution to exercise A.1 (p. 519): The (9,2) code has $M = 7$ parity checks, and its distance is $d = 8$. If the code were perfect, then all points would be at a distance of at most $d/2$ from the nearest codeword, and each point would only have one nearest codeword.

The (9,2) code is not a perfect code. Any code with even distance cannot be a perfect code because it must have vectors that are equidistant from the two nearest codewords, for example, `000001111` is at Hamming distance 4 from both `000000000` and `011111111`.

We can also find words that are at a distance greater than $d/2$ from all codewords, for example `111110000`, which is at a distance of five or more from all codewords.

Solution to exercise A.2 (p. 519): The (9,2) code is maximum distance separable. It has $M = 7$ parity checks, and when any 7 characters in a codeword are erased we can restore the others. Proof: any two by two submatrix of $\mathbf{G}$ is invertible.

(more here).

A.5 Eigenvectors

A.6 Eigenvectors and eigenvalues

A *right–eigenvector* of a square matrix $\mathbf{A}$ is a non–zero vector $\mathbf{e}_{\mathrm{R}}$ that satisfies

$$\mathbf{A}\mathbf{e}_{\mathrm{R}} = \lambda \mathbf{e}_{\mathrm{R}}, \tag{A.16}$$

where λ is the eigenvalue associated with that eigenvector. The eigenvalue may be a real number or complex number and it may be zero. Eigenvectors may be real or complex.

A *left–eigenvector* of a matrix $\mathbf{A}$ is a vector $\mathbf{e}_{\mathrm{L}}$ that satisfies

$$\mathbf{e}_{\mathrm{L}}^{\mathsf{T}}\mathbf{A} = \lambda \mathbf{e}_{\mathrm{L}}^{\mathsf{T}}. \tag{A.17}$$

The following statements for right–eigenvectors also apply to left–eigenvectors.

- If a matrix has two or more linearly independent right–eigenvectors with the same eigenvalue then that eigenvalue is called a degenerate eigenvalue of the matrix, or a repeated eigenvalue. Any linear combination of those eigenvectors is another right–eigenvector with the same eigenvalue.
- The principal right–eigenvector of a matrix is, by definition, the right–eigenvector with the largest associated eigenvalue.
- If a real matrix has a right–eigenvector with complex eigenvalue $\lambda = x + yi$ then it also has a right–eigenvector with the conjugate eigenvalue $\lambda^* = x - yi$. *(Check.)*

Symmetric matrices

If $\mathbf{A}$ is a real symmetric $N \times N$ matrix then

1. all the eigenvalues and eigenvectors of $\mathbf{A}$ are real;

2. every left eigenvector of $\mathbf{A}$ is also a right eigenvector of $\mathbf{A}$ with the same eigenvalue, and vice–versa;

3. a set of N eigenvectors and eigenvalues $\{\mathbf{e}^{(a)}, \lambda_a\}_{a=1}^{N}$ can be found that are orthonormal, that is,
$$\mathbf{e}^{(a)} \cdot \mathbf{e}^{(b)} = \delta_{ab}; \tag{A.18}$$
The matrix can be expressed as a weighted sum of outer products of the eigenvectors:
$$\mathbf{A} = \sum_a \lambda_a [\mathbf{e}^{(a)}][\mathbf{e}^{(a)}]^{\mathsf{T}}. \tag{A.19}$$

(Whereas I often use i and n as indices for sets of size I and N, I will use the indices a and b to run over eigenvectors, even if there are N of them. This is to avoid confusion with the components of the eigenvectors, which are indexed by n, *e.g.*, $e_n^{(a)}$.)

General square matrices

An $N \times N$ matrix can have up to N distinct eigenvalues. For each distinct eigenvalue that is non–zero, there is at least one left eigenvector and one right eigenvector. (CHECK.)

Left and right eigenvectors that have different eigenvalue are orthogonal, that is,
$$\text{if } \lambda_a \neq \lambda_b \text{ then } \mathbf{e}_{\mathrm{L}}^{(a)} \cdot \mathbf{e}_{\mathrm{R}}^{(b)} = 0. \tag{A.20}$$

Non–negative matrices

Definition

If all the elements of a matrix $\mathbf{C}$ satisfy $C_{mn} \geq 0$ then $\mathbf{C}$ is a non–negative matrix. (Further conditions? *e.g.*, $\mathbf{C} \neq 0$?) The statement $\mathbf{C} \geq 0$ means that $\mathbf{C}$ is a non–negative matrix. (Don't confuse with the statement that the determinant is ≥ 0 which would be written $|\mathbf{C}| \geq 0$, nor with the statement that $\mathbf{C}$ is positive definite, for which I have no symbol.)

Similarly, if all the elements of a vector $\mathbf{c}$ satisfy $c_i \geq 0$ then $\mathbf{c}$ is a non–negative vector.

Properties

A non–negative matrix has a principal eigenvector that is non–negative. It may also have other eigenvectors with the same eigenvalue that are not non–negative. But if the principal eigenvalue of a non–negative matrix is not degenerate, then the matrix has only one principal eigenvector $\mathbf{e}^{(1)}$, and it is non–negative. (Of course, any linear multiple of that vector, for example, $-\mathbf{e}^{(1)}$, is also an eigenvector; so the statement just made should really be worded..... (shall I be pedantic?).)

All the other eigenvalues are smaller in absolute magnitude. (Check; actually, there could be one with equal and opposite magnitude?) Yes, there can be multiple principal eigenvectors if the graph is not connected, and there can be eigenvectors with opposite eigenvalue if it is not ergodic in that it doesn't mix.

Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathrm{L}}, \mathbf{e}_{\mathrm{R}}$
$\begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	2.41: $\begin{bmatrix} .58 \\ .82 \\ 0 \end{bmatrix}$ $\begin{bmatrix} .82 \\ .58 \\ 0 \end{bmatrix}$; 1: $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$; -0.41: $\begin{bmatrix} -.58 \\ .82 \\ 0 \end{bmatrix}$ $\begin{bmatrix} -.82 \\ .58 \\ 0 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & 1 & 2 \\ 0 & 0 & 2 & 1 \\ 1 & 2 & 0 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$	3: $\begin{bmatrix} .5 \\ .5 \\ .5 \\ .5 \end{bmatrix}$ $\begin{bmatrix} .5 \\ .5 \\ .5 \\ .5 \end{bmatrix}$; 1: $\begin{bmatrix} -.5 \\ .5 \\ .5 \\ -.5 \end{bmatrix}$ $\begin{bmatrix} -.5 \\ .5 \\ .5 \\ -.5 \end{bmatrix}$; -1: $\begin{bmatrix} .5 \\ -.5 \\ .5 \\ -.5 \end{bmatrix}$ $\begin{bmatrix} .5 \\ -.5 \\ .5 \\ -.5 \end{bmatrix}$; -3: $\begin{bmatrix} -.5 \\ -.5 \\ .5 \\ .5 \end{bmatrix}$ $\begin{bmatrix} -.5 \\ -.5 \\ .5 \\ .5 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$	1.93: $\begin{bmatrix} .3 \\ .5 \\ .6 \\ .6 \end{bmatrix}$ $\begin{bmatrix} .1 \\ .2 \\ .4 \\ .9 \end{bmatrix}$; $-0.1-0.8i$: $\begin{bmatrix} -.3-.3i \\ .1-.7i \\ .5-.1i \\ -.2+.3i \end{bmatrix}$ $\begin{bmatrix} -.5+.4i \\ .3+.4i \\ .3-.3i \\ -.3-.2i \end{bmatrix}$; $-0.1+0.8i$: $\begin{bmatrix} -.3+.3i \\ .1+.7i \\ .5+.1i \\ -.2-.3i \end{bmatrix}$ $\begin{bmatrix} -.5-.4i \\ .3-.4i \\ .3+.3i \\ -.3+.2i \end{bmatrix}$; -0.77: $\begin{bmatrix} -.5 \\ .2 \\ -.7 \\ .4 \end{bmatrix}$ $\begin{bmatrix} -.7 \\ .5 \\ -.4 \\ .3 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	1.84: $\begin{bmatrix} .4 \\ .6 \\ .7 \end{bmatrix}$ $\begin{bmatrix} .3 \\ .5 \\ .9 \end{bmatrix}$; $-0.4-0.6i$: $\begin{bmatrix} -.5-.2i \\ .2-.7i \\ .1+.4i \end{bmatrix}$ $\begin{bmatrix} -.5+.5i \\ .5+.1i \\ -.2-.4i \end{bmatrix}$; $-0.4+0.6i$: $\begin{bmatrix} -.5+.2i \\ .2+.7i \\ .1-.4i \end{bmatrix}$ $\begin{bmatrix} -.5-.5i \\ .5-.1i \\ -.2+.4i \end{bmatrix}$
$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$	1.62: $\begin{bmatrix} .53 \\ .85 \end{bmatrix}$ $\begin{bmatrix} .53 \\ .85 \end{bmatrix}$; -0.62: $\begin{bmatrix} .85 \\ -.53 \end{bmatrix}$ $\begin{bmatrix} .85 \\ -.53 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	1.62: $\begin{bmatrix} .60 \\ .37 \\ .37 \\ .60 \end{bmatrix}$ $\begin{bmatrix} .60 \\ .37 \\ .37 \\ .60 \end{bmatrix}$; $0.5+0.9i$: $\begin{bmatrix} .1-.5i \\ -.3-.4i \\ .3+.4i \\ -.1+.5i \end{bmatrix}$ $\begin{bmatrix} .1-.5i \\ .3+.4i \\ -.3-.4i \\ -.1+.5i \end{bmatrix}$; $0.5-0.9i$: $\begin{bmatrix} .1+.5i \\ -.3+.4i \\ .3-.4i \\ -.1-.5i \end{bmatrix}$ $\begin{bmatrix} .1+.5i \\ .3-.4i \\ -.3+.4i \\ -.1-.5i \end{bmatrix}$; -0.62: $\begin{bmatrix} .37 \\ -.60 \\ -.60 \\ .37 \end{bmatrix}$ $\begin{bmatrix} .37 \\ -.60 \\ -.60 \\ .37 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	1.62: $\begin{bmatrix} .37 \\ .60 \\ .60 \\ .37 \end{bmatrix}$ $\begin{bmatrix} .37 \\ .60 \\ .60 \\ .37 \end{bmatrix}$; $-0.5+0.9i$: $\begin{bmatrix} -.2+.5i \\ .3+.4i \\ -.3-.4i \\ .2-.5i \end{bmatrix}$ $\begin{bmatrix} .2-.5i \\ .3+.4i \\ -.3-.4i \\ -.2+.5i \end{bmatrix}$; $-0.5-0.9i$: $\begin{bmatrix} -.2-.5i \\ .3-.4i \\ -.3+.4i \\ .2+.5i \end{bmatrix}$ $\begin{bmatrix} .2+.5i \\ .3-.4i \\ -.3+.4i \\ -.2-.5i \end{bmatrix}$; -0.62: $\begin{bmatrix} .60 \\ -.37 \\ -.37 \\ .60 \end{bmatrix}$ $\begin{bmatrix} .60 \\ -.37 \\ -.37 \\ .60 \end{bmatrix}$

Figure A.2. Some matrices and their eigenvectors

Examples

An important example of a non–negative matrix is a transition probability matrix $\mathbf{Q}$.

Transition probability matrices

Definition

A transition probability matrix $\mathbf{Q}$ has columns that are probability vectors, that is, it satisfies $\mathbf{Q} \geq 0$ and

$$\sum_i Q_{ij} = 1 \text{ for all } j. \tag{A.21}$$

 © David J.C. MacKay. Draft 2.0.7. February 14, 2000

Properties

This property can be rewritten in terms of the all–ones vector $\mathbf{n} = (1, 1, \ldots, 1)^{\mathsf{T}}$ thus:

$$\mathbf{n}^{\mathsf{T}}\mathbf{Q} = \mathbf{n}^{\mathsf{T}}. \tag{A.22}$$

So $\mathbf{n}$ is the principal left–eigenvector of $\mathbf{Q}$ with eigenvalue $\lambda_1 = 1$.

$$\mathbf{e}_{\mathrm{L}}^{(1)} = \mathbf{n}. \tag{A.23}$$

Because it is a non–negative matrix, $\mathbf{Q}$ has a principal right–eigenvector that is non–negative, $\mathbf{e}_{\mathrm{R}}^{(1)}$. This vector, if we normalize it such that $\mathbf{e}_{\mathrm{R}}^{(1)} \cdot \mathbf{n} = 1$, is called the invariant distribution of the transition probability matrix. It is the probability density which is left unchanged under $\mathbf{Q}$. Unlike the principal left–eigenvector, which we explicitly identified above, we can't usually identify the principal right–eigenvector without computation.

(To be rigorous I need to mention degenerate cases where there are multiple invariant distributions. Not ergodic. Give examples. If the $\mathbf{Q}$ is 'connected' (define) then it's ergodic and the invariant distribution is unique.)

The matrix may have up to $N-1$ other right eigenvectors all of which are orthogonal to the left eigenvector $\mathbf{n}$, that is, they are zero–sum vectors.

Examples

Random walk on hypercubes.

Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathrm{L}}, \mathbf{e}_{\mathrm{R}}$
$\begin{bmatrix} 0 & .5 & 0 \\ 1 & 0 & 1 \\ 0 & .5 & 0 \end{bmatrix}$	1: $\begin{bmatrix} .58 \\ .58 \\ .58 \end{bmatrix}$ $\begin{bmatrix} .41 \\ .82 \\ .41 \end{bmatrix}$; 0.00: $\begin{bmatrix} -.71 \\ -.00 \\ .71 \end{bmatrix}$ $\begin{bmatrix} -.71 \\ -.00 \\ .71 \end{bmatrix}$; -1: $\begin{bmatrix} .58 \\ -.58 \\ .58 \end{bmatrix}$ $\begin{bmatrix} .41 \\ -.82 \\ .41 \end{bmatrix}$
$\begin{bmatrix} 0 & .33 & 0 & 0 \\ 1 & 0 & .67 & 0 \\ 0 & .67 & 0 & 1 \\ 0 & 0 & .33 & 0 \end{bmatrix}$	1: $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$ $\begin{bmatrix} .22 \\ .67 \\ .67 \\ .22 \end{bmatrix}$; 0.33: $\begin{bmatrix} .67 \\ .22 \\ -.22 \\ -.67 \end{bmatrix}$ $\begin{bmatrix} .50 \\ .50 \\ -.50 \\ -.50 \end{bmatrix}$; -0.33: $\begin{bmatrix} .67 \\ -.22 \\ -.22 \\ .67 \end{bmatrix}$ $\begin{bmatrix} .50 \\ -.50 \\ -.50 \\ .50 \end{bmatrix}$; -1: $\begin{bmatrix} -.50 \\ .50 \\ -.50 \\ .50 \end{bmatrix}$ $\begin{bmatrix} -.22 \\ .67 \\ -.67 \\ .22 \end{bmatrix}$
$\begin{bmatrix} 0 & .25 & 0 & 0 & 0 \\ 1 & 0 & .50 & 0 & 0 \\ 0 & .75 & 0 & .75 & 0 \\ 0 & 0 & .50 & 0 & 1 \\ 0 & 0 & 0 & .25 & 0 \end{bmatrix}$	1: $\begin{bmatrix} .4 \\ .4 \\ .4 \\ .4 \\ .4 \end{bmatrix}$ $\begin{bmatrix} .1 \\ .5 \\ .7 \\ .5 \\ .1 \end{bmatrix}$; 0.50: $\begin{bmatrix} -.6 \\ -.3 \\ -.0 \\ .3 \\ .6 \end{bmatrix}$ $\begin{bmatrix} -.3 \\ -.6 \\ -.0 \\ .6 \\ .3 \end{bmatrix}$; 0: $\begin{bmatrix} .7 \\ 0 \\ -.2 \\ -.0 \\ .7 \end{bmatrix}$ $\begin{bmatrix} .4 \\ -.0 \\ -.8 \\ 0 \\ .4 \end{bmatrix}$; -0.50: $\begin{bmatrix} -.6 \\ .3 \\ -.0 \\ -.3 \\ .6 \end{bmatrix}$ $\begin{bmatrix} -.3 \\ .6 \\ -.0 \\ -.6 \\ .3 \end{bmatrix}$; -1: $\begin{bmatrix} .4 \\ -.4 \\ .4 \\ -.4 \\ .4 \end{bmatrix}$ $\begin{bmatrix} -.1 \\ .5 \\ -.7 \\ .5 \\ -.1 \end{bmatrix}$
$\begin{bmatrix} 0 & .2 & 0 & 0 & 0 & 0 \\ 1 & 0 & .4 & 0 & 0 & 0 \\ 0 & .8 & 0 & .6 & 0 & 0 \\ 0 & 0 & .6 & 0 & .8 & 0 \\ 0 & 0 & 0 & .4 & 0 & 1 \\ 0 & 0 & 0 & 0 & .2 & 0 \end{bmatrix}$	1: $\begin{bmatrix} .4 \\ .4 \\ .4 \\ .4 \\ .4 \\ .4 \end{bmatrix}$ $\begin{bmatrix} .1 \\ .3 \\ .6 \\ .6 \\ .3 \\ .1 \end{bmatrix}$; 0.60: $\begin{bmatrix} .6 \\ .4 \\ .1 \\ -.1 \\ -.4 \\ -.6 \end{bmatrix}$ $\begin{bmatrix} .2 \\ .6 \\ .4 \\ -.4 \\ -.6 \\ -.2 \end{bmatrix}$; 0.20: $\begin{bmatrix} .7 \\ .1 \\ -.1 \\ -.1 \\ .1 \\ .7 \end{bmatrix}$ $\begin{bmatrix} -.3 \\ -.3 \\ .6 \\ .6 \\ -.3 \\ -.3 \end{bmatrix}$; -0.20: $\begin{bmatrix} -.7 \\ .1 \\ .1 \\ -.1 \\ -.1 \\ .7 \end{bmatrix}$ $\begin{bmatrix} -.3 \\ .3 \\ .6 \\ -.6 \\ -.3 \\ .3 \end{bmatrix}$; -0.60: $\begin{bmatrix} .6 \\ -.4 \\ .1 \\ .1 \\ -.4 \\ .6 \end{bmatrix}$ $\begin{bmatrix} -.2 \\ .6 \\ -.4 \\ -.4 \\ .6 \\ -.2 \end{bmatrix}$; -1: $\begin{bmatrix} -.4 \\ .4 \\ -.4 \\ .4 \\ -.4 \\ .4 \end{bmatrix}$ $\begin{bmatrix} -.1 \\ .3 \\ -.6 \\ .6 \\ -.3 \\ .1 \end{bmatrix}$

Transition probability matrices for generating random paths through trellises.

Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathrm{L}}, \mathbf{e}_{\mathrm{R}}$
$\begin{bmatrix} 0 & .38 \\ 1 & .62 \end{bmatrix}$	1: $\begin{bmatrix} .71 \\ .71 \end{bmatrix}$ $\begin{bmatrix} .36 \\ .93 \end{bmatrix}$; -0.38: $\begin{bmatrix} -.93 \\ .36 \end{bmatrix}$ $\begin{bmatrix} -.71 \\ .71 \end{bmatrix}$
$\begin{bmatrix} 0 & .35 & 0 \\ 0 & 0 & .46 \\ 1 & .65 & .54 \end{bmatrix}$	1: $\begin{bmatrix} .58 \\ .58 \\ .58 \end{bmatrix}$ $\begin{bmatrix} .14 \\ .41 \\ .90 \end{bmatrix}$; $-0.2-0.3i$: $\begin{bmatrix} -.8+.1i \\ -.2-.5i \\ .2+.2i \end{bmatrix}$ $\begin{bmatrix} .2-.5i \\ -.6+.2i \\ .4+.3i \end{bmatrix}$; $-0.2+0.3i$: $\begin{bmatrix} -.8-.1i \\ -.2+.5i \\ .2-.2i \end{bmatrix}$ $\begin{bmatrix} .2+.5i \\ -.6-.2i \\ .4-.3i \end{bmatrix}$
$\begin{bmatrix} 0 & .34 & 0 & 0 \\ 0 & 0 & .44 & 0 \\ 0 & 0 & 0 & .48 \\ 1 & .66 & .56 & .52 \end{bmatrix}$	1: $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$ $\begin{bmatrix} .06 \\ .19 \\ .42 \\ .88 \end{bmatrix}$; $-0.0-0.4i$: $\begin{bmatrix} -.5-.4i \\ .1-.6i \\ .4-.1i \\ -.2+.2i \end{bmatrix}$ $\begin{bmatrix} .1-.4i \\ -.5-.1i \\ -.0+.5i \\ .5-.0i \end{bmatrix}$; $-0.0+0.4i$: $\begin{bmatrix} -.5+.4i \\ .1+.6i \\ .4+.1i \\ -.2-.2i \end{bmatrix}$ $\begin{bmatrix} .1+.4i \\ -.5+.1i \\ -.0-.5i \\ .5+.0i \end{bmatrix}$; -0.40: $\begin{bmatrix} -.75 \\ .14 \\ -.57 \\ .30 \end{bmatrix}$ $\begin{bmatrix} -.48 \\ .56 \\ -.52 \\ .43 \end{bmatrix}$

Illustrative transition probability matrices showing the two ways of being non–ergodic. (a) More than one principal eigenvector with eigenvalue 1. Corresponds to a not–fully connected space. (b) Other eigenvectors with complex or negative eigenvalues of magnitude 1. Corresponds to circulating density which doesn't relax to a single stationary distribution.

Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathrm{L}}, \mathbf{e}_{\mathrm{R}}$
$\begin{bmatrix} .90 & .20 & 0 & 0 \\ .10 & .80 & 0 & 0 \\ 0 & 0 & .90 & .20 \\ 0 & 0 & .10 & .80 \end{bmatrix}$	1: $\begin{bmatrix} 0 \\ 0 \\ .71 \\ .71 \end{bmatrix}$ $\begin{bmatrix} 0 \\ 0 \\ .89 \\ .45 \end{bmatrix}$; 1: $\begin{bmatrix} .71 \\ .71 \\ 0 \\ 0 \end{bmatrix}$ $\begin{bmatrix} .89 \\ .45 \\ 0 \\ 0 \end{bmatrix}$; 0.70: $\begin{bmatrix} .45 \\ -.89 \\ 0 \\ 0 \end{bmatrix}$ $\begin{bmatrix} .71 \\ -.71 \\ 0 \\ 0 \end{bmatrix}$; 0.70: $\begin{bmatrix} 0 \\ 0 \\ -.45 \\ .89 \end{bmatrix}$ $\begin{bmatrix} 0 \\ 0 \\ -.71 \\ .71 \end{bmatrix}$
$\begin{bmatrix} .90 & .20 & 0 & 0 \\ .10 & .79 & .02 & 0 \\ 0 & .01 & .88 & .20 \\ 0 & 0 & .10 & .80 \end{bmatrix}$	1: $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$ $\begin{bmatrix} .87 \\ .43 \\ .22 \\ .11 \end{bmatrix}$; 0.98: $\begin{bmatrix} -.18 \\ -.15 \\ .66 \\ .72 \end{bmatrix}$ $\begin{bmatrix} -.66 \\ -.28 \\ .61 \\ .33 \end{bmatrix}$; 0.70: $\begin{bmatrix} .20 \\ -.40 \\ -.40 \\ .80 \end{bmatrix}$ $\begin{bmatrix} .63 \\ -.63 \\ -.32 \\ .32 \end{bmatrix}$; 0.69: $\begin{bmatrix} -.19 \\ .41 \\ -.44 \\ .77 \end{bmatrix}$ $\begin{bmatrix} -.61 \\ .65 \\ -.35 \\ .30 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & .90 & .20 \\ 0 & 0 & .10 & .80 \\ .90 & .20 & 0 & 0 \\ .10 & .80 & 0 & 0 \end{bmatrix}$	1: $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$ $\begin{bmatrix} .63 \\ .32 \\ .63 \\ .32 \end{bmatrix}$; 0.70: $\begin{bmatrix} -.32 \\ .63 \\ -.32 \\ .63 \end{bmatrix}$ $\begin{bmatrix} .50 \\ -.50 \\ .50 \\ -.50 \end{bmatrix}$; -0.70: $\begin{bmatrix} .32 \\ -.63 \\ -.32 \\ .63 \end{bmatrix}$ $\begin{bmatrix} -.50 \\ .50 \\ .50 \\ -.50 \end{bmatrix}$; -1: $\begin{bmatrix} .50 \\ .50 \\ -.50 \\ -.50 \end{bmatrix}$ $\begin{bmatrix} .63 \\ .32 \\ -.63 \\ -.32 \end{bmatrix}$
$\begin{bmatrix} 0 & .01 & .88 & .20 \\ 0 & 0 & .10 & .80 \\ .90 & .20 & 0 & 0 \\ .10 & .79 & .02 & 0 \end{bmatrix}$	1: $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$ $\begin{bmatrix} .62 \\ .33 \\ .63 \\ .34 \end{bmatrix}$; 0.68: $\begin{bmatrix} .34 \\ -.62 \\ .33 \\ -.63 \end{bmatrix}$ $\begin{bmatrix} .50 \\ -.50 \\ .50 \\ -.50 \end{bmatrix}$; -0.70: $\begin{bmatrix} -.30 \\ .64 \\ .31 \\ -.64 \end{bmatrix}$ $\begin{bmatrix} .51 \\ -.49 \\ -.51 \\ .49 \end{bmatrix}$; -0.98: $\begin{bmatrix} -.50 \\ -.51 \\ .49 \\ .51 \end{bmatrix}$ $\begin{bmatrix} -.63 \\ -.31 \\ .64 \\ .30 \end{bmatrix}$

A.7 Some numbers

	2^{8192}	10^{2466}	Number of 1 kilobyte files
	2^{1024}	10^{308}	Number of states of a 2D Ising model with 32×32 spins
2^{1000}		10^{301}	Number of binary strings of length 1000
2^{500}		3×10^{150}	
	2^{469}	10^{141}	Number of binary strings of length 1000 having 100 `1`s and 900 `0`s
	2^{266}	10^{80}	Number of electrons in universe
2^{200}		1.6×10^{60}	
	2^{190}	10^{57}	Number of electrons in solar system
	2^{171}	3×10^{51}	Number of electrons neutrons in the earth
2^{100}		10^{30}	
	2^{98}	3×10^{29}	Age of universe / picoseconds
	2^{58}	3×10^{17}	Age of universe / seconds
2^{50}		10^{15}	
2^{40}		10^{12}	
		6×10^{9}	Number of bits to list one human genome
	2^{32}	5×10^{9}	Population of earth
2^{30}		10^{9}	
		2×10^{8}	Number of bits to list one *C. Elegans* genome
	2^{25}	3×10^{7}	One year / seconds
		2×10^{7}	Number of bits in the compressed postscript file that is this book
		2×10^{7}	Number of bits in `unix` kernel
		10^{7}	Number of bits to list one *E. Coli* genome
		6×10^{6}	Number of years since human/chimpanzee divergence
2^{20}		10^{6}	1048576
		2×10^{5}	Number of generations since human/chimpanzee divergence
		10^{5}	Number of genes in human genome
		1.5×10^{3}	Number of base pairs in a gene
2^{10}		10^{3}	1024
2^{0}		10^{0}	1
2^{-10}		10^{-3}	
2^{-20}		10^{-6}	
		3×10^{-8}	probability of error in transmission of coding DNA, per nucleotide, per generation
2^{-30}		10^{-9}	
2^{-60}		10^{-18}	probability of undetected error in a hard disc drive, after error correction and detection

Bibliography

Adler, S. L. (1981) Over-relaxation method for the Monte-Carlo evaluation of the partition function for multiquadratic actions. *Physical Review D — Particles and Fields* **23** (12): 2901–2904.

Aiyer, S. V. B., (1991) *Solving Combinatorial Optimization Problems Using Neural Networks*. Cambridge University Engineering Department PhD dissertation. CUED/F-INFENG/TR 89.

Amari, S., Cichocki, A., and Yang, H., (1996) A new learning algorithm for blind signal separation. In NIPS96.

Amit, D. J., Gutfreund, H., and Sompolinsky, H. (1985) Storing infinite numbers of patterns in a spin glass model of neural networks. *Phys. Rev. Lett.* **55**: 1530.

Angel, J. R. P., Wizinowich, P., Lloyd-Hart, M., and Sandler, D. (1990) Adaptive optics for array telescopes using neural-network techniques. *Nature* **348**: 221–224.

Bell, A. J., and Sejnowski, T. J. (1995) An information maximization approach to blind separation and blind deconvolution. *Neural Computation* **7** (6): 1129–1159.

Berlekamp, E. R. (1980) The technology of error–correcting codes. *IEEE Transactions on Information Theory*.

Berrou, C., and Glavieux, A. (1996) Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications* **44**: 1261–1271.

Bishop, C. M. (1992) Exact calculation of the Hessian matrix for the multilayer perceptron. *Neural Computation* **4** (4): 494–501.

Bishop, C. M. (1995) *Neural Networks for Pattern Recognition*. Oxford University Press.

Box, G. E. P., and Tiao, G. C. (1973) *Bayesian inference in statistical analysis*. Addison–Wesley.

Bretthorst, G. (1988) *Bayesian spectrum analysis and parameter estimation*. Springer. Also available at `bayes.wustl.edu`.

Bridle, J. S. (1989) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neuro-computing: algorithms, architectures and applications*, ed. by F. Fougelman-Soulie and J. Hérault. Springer–Verlag.

Bryan, R. (1990) Solving oversampled data problems by Maximum Entropy. In *Maximum Entropy and Bayesian Methods, Dartmouth, U.S.A., 1989*, ed. by P. Fougere, pp. 221–232. Kluwer.

Buntine, W., and Weigend, A. (1991) Bayesian back–propagation. *Complex Systems* **5**: 603–643.

Carroll, L. (1998) *Alice's adventures in Wonderland; and, Through the looking-glass: and what Alice found there*. London: Macmillan Children's Books.

Comon, P., Jutten, C., and Herault, J. (1991) Blind separation of sources .2. problems statement. *Signal Processing* **24** (1): 11–20.

Copas, J. B. (1983) Regression, prediction and shrinkage (with discussion). *J. R. Statist.Soc B* **45** (3): 311–354.

Cover, T. M. (1965) Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers* **14**: 326–334.

Cover, T. M., and Thomas, J. A. (1991) *Elements of Information Theory*. New York: Wiley.

Cowles, M. K., and Carlin, B. P. (1996) Markov-chain Monte-Carlo convergence diagnostics — a comparative review. *Journal of the American Statistical Association* **91** (434): 883–904.

Cox, R. (1946) Probability, frequency, and reasonable expectation. *Am. J. Physics* **14**: 1–13.

Davey, M. C., (1999) *Error-correction using Low-Density Parity-Check Codes*. University of Cambridge dissertation.

Davey, M. C., and MacKay, D. J. C. (1998a) Low density parity check codes over GF(q). *IEEE Communications Letters* **2** (6): 165–167.

Davey, M. C., and MacKay, D. J. C. (1998b) Low density parity check codes over GF(q). In *Proceedings of the 1998 IEEE Information Theory Workshop*, pp. 70–71. IEEE.

Davey, M. C., and MacKay, D. J. C., (1999) Watermark codes: Reliable communication over insertion/deletion channels. Submitted to ISIT 2000.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995) The Helmholtz machine. *Neural Computation* **7** (5): 889–904.

Dempster, A., Laird, N., and Rubin, D. (1977) Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B* **39**: 1–38.

Divsalar, D., Jin, H., and McEliece, R. J. (1998) Coding theorems for 'turbo–like' codes. In *Proceedings of the 36th Allerton Conference on Communication, Control, and Computing, Sept. 1998*, pp. 201–210, Monticello, Illinois. Allerton House.

Elias, P. (1975) Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* **21** (2): 194–203.

Everitt, B. S. (1984) *An Introduction to Latent Variable Models*. London: Chapman and Hall.

Ferreira, H., Clarke, W., Helberg, A., Abdel-Ghaffar, K. S., and Vinck, A. H. (1997) Insertion/deletion correction with spectral nulls. *IEEE Trans. Info. Theory* **43** (2): 722–732.

Feynman, R. P. (1972) *Statistical Mechanics*. New York: Addison–Wesley.

Gallager, R. G. (1962) Low density parity check codes. *IRE Trans. Info. Theory* **IT-8**: 21–28.

Gallager, R. G. (1968) *Information Theory and Reliable Communication*. New York: Wiley.

Gilks, W., and Wild, P. (1992) Adaptive rejection sampling for Gibbs sampling. *Applied Statistics* **41**: 337–348.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996) *Markov Chain Monte Carlo in Practice*. Chapman and Hall.

Green, P. J. (1995) Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **82**: 711–732.

Gull, S. F. (1988) Bayesian inductive inference and maximum entropy. In *Maximum Entropy and Bayesian Methods in Science and Engineering, vol. 1: Foundations*, ed. by G. Erickson and C. Smith, pp. 53–74, Dordrecht. Kluwer.

Gull, S. F. (1989) Developments in maximum entropy data analysis. In *Maximum Entropy and Bayesian Methods, Cambridge 1988*, ed. by J. Skilling, pp. 53–71, Dordrecht. Kluwer.

Gull, S. F., and Daniell, G. (1978) Image reconstruction from incomplete and noisy data. *Nature* **272**: 686–690.

Hanson, R., Stutz, J., and Cheeseman, P. (1991) Bayesian classification with correlation and inheritance. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia*, volume 2, pp. 692–698. Morgan Kaufmann.

Hendin, O., Horn, D., and Hopfield, J. J. (1994) Decomposition of a mixture of signals in a model of the olfactory bulb. *Proceedings of the National Academy of Sciences of the United States of America* **91** (13): 5942–5946.

Hertz, J., Krogh, A., and Palmer, R. G. (1991) *Introduction to the Theory of Neural Computation*. Addison-Wesley.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995) The wake-sleep algorithm for unsupervised neural networks. *Science* **268** (5214): 1158–1161.

Hinton, G. E., and Ghahramani, Z. (1997) Generative models for discovering sparse distributed representations. *Proc. Roy. Soc.*.

Hinton, G. E., and Sejnowski, T. J. (1986) Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, ed. by D. E. Rumelhart and J. E. McClelland, pp. 282–317. Cambridge Mass.: MIT Press.

Hinton, G. E., and van Camp, D. (1993) Keeping neural networks simple by minimizing the description length of the weights. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pp. 5–13. ACM Press, New York, NY.

Hinton, G. E., and Zemel, R. S. (1994) Autoencoders, minimum description length and Helmholtz free energy. In *Advances in Neural Information Processing Systems 6*, ed. by J. D. Cowan, G. Tesauro, and J. Alspector, San Mateo, California. Morgan Kaufmann.

Jaynes, E. T. (1983) Bayesian intervals versus confidence intervals. In *E.T. Jaynes. Papers on Probability, Statistics and Statistical Physics*, ed. by R. D. Rosenkrantz, p. 151. Kluwer.

Jensen, F. V. (1996) *An Introduction to Bayesian Networks*. London: UCL press.

JPL, (1996) Turbo codes performance. Available from `http://www331.jpl.nasa.gov/public/TurboPerf.html`.

Jutten, C., and Herault, J. (1991) Blind separation of sources .1. an adaptive algorithm based on neuromimetic architecture. *Signal Processing* **24** (1): 1–10.

Knuth, D. E. (1968) *The art of computer programming. Volume 1: fundamental algorithms*. Reading, Mass.: Addison Wesley.

Kschischang, F. R., and Sorokine, V. (1995) On the trellis structure of block codes. *IEEE Trans. in Inform. Theory* **41** (6): 1924–1937.

Lauritzen, S. L. (1996) *Graphical Models*. Number 17 in Oxford Statistical Science Series. Oxford: Clarendon Press.

Levenshtein, V. I. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics - Doklady* **10** (8): 707–710.

Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (1998) Improved low-density parity-check codes using irregular graphs and belief propagation. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, p. 117.

MacKay, D. J. C., (1991) *Bayesian Methods for Adaptive Models*. California Institute of Technology dissertation.

MacKay, D. J. C. (1992a) Bayesian interpolation. *Neural Computation* **4** (3): 415–447.

MacKay, D. J. C. (1992b) The evidence framework applied to classification networks. *Neural Computation* **4** (5): 698–714.

MacKay, D. J. C. (1992c) A practical Bayesian framework for backpropagation networks. *Neural Computation* **4** (3): 448–472.

MacKay, D. J. C. (1994) Bayesian non-linear modelling for the prediction competition. In *ASHRAE Transactions, V.100, Pt.2*, pp. 1053–1062, Atlanta Georgia. ASHRAE.

MacKay, D. J. C. (1995a) Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research, Section A* **354** (1): 73–80.

MacKay, D. J. C. (1995b) Developments in probabilistic modelling with neural networks—ensemble learning. In *Neural Networks: Artificial Intelligence and Industrial Applications. Proceedings of the 3rd Annual Symposium on Neural Networks, Nijmegen, Netherlands, 14-15 September 1995*, pp. 191–198, Berlin. Springer.

MacKay, D. J. C. (1995c) Free energy minimization algorithm for decoding and cryptanalysis. *Electronics Letters* **31** (6): 446–447.

MacKay, D. J. C. (1996a) Bayesian non-linear modelling for the 1993 energy prediction competition. In *Maximum Entropy and Bayesian Methods, Santa Barbara 1993*, ed. by G. Heidbreder, pp. 221–234, Dordrecht. Kluwer.

MacKay, D. J. C. (1996b) Density networks and their application to protein modelling. In *Maximum Entropy and Bayesian Methods, Cambridge 1994*, ed. by J. Skilling and S. Sibisi, pp. 259–268, Dordrecht. Kluwer.

MacKay, D. J. C. (1998) Choice of basis for Laplace approximation. *Machine Learning* **33** (1): 77–86.

MacKay, D. J. C. (1999) Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory* **45** (2): 399–431.

MacKay, D. J. C., and Neal, R. M. (1996) Near Shannon limit performance of low density parity check codes. *Electronics Letters* **32** (18): 1645–1646. Reprinted *Electronics Letters*, **33**(6):457–458, March 1997.

Marinari, E., and Parisi, G. (1992) Simulated tempering — a new Monte-Carlo scheme. *Europhysics Letters* **19** (6): 451–458.

McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998) Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communications* **16** (2): 140–152.

Mosteller, F., and Wallace, D. L. (1984) *Applied Bayesian and Classical Inference. The case of* The Federalist *papers*. Springer.

Neal, R. M. (1993a) Bayesian learning via stochastic dynamics. In *Advances in Neural Information Processing Systems 5*, ed. by C. L. Giles, S. J. Hanson, and J. D. Cowan, pp. 475–482, San Mateo, California. Morgan Kaufmann.

Neal, R. M. (1993b) Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG–TR–93–1, Dept. of Computer Science, University of Toronto.

Neal, R. M. (1995) Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation. Technical Report 9508, Dept. of Statistics, University of Toronto.

Neal, R. M. (1996) *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. New York: Springer.

Neal, R. M. (1997) Markov chain Monte Carlo methods based on 'slicing' the density function. Technical Report 9722, Dept. of Statistics, Univ. of Toronto.

Neal, R. M. (1998) Annealed importance sampling. Technical Report 9805, Dept. of Statistics, Univ. of Toronto.

Neal, R. M., and Hinton, G. E. (1993) A new view of the EM algorithm that justifies incremental, sparse, and other variants. *Biometrika*. submitted.

Neal, R. M., and Hinton, G. E. (1998) A new view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, ed. by M. I. Jordan, NATO Science Series, pp. 355–368. Kluwer Academic Press.

Pearlmutter, B. A., and Parra, L. C., (1996) A context-sensitive generalization of ica. To appear in ICONIP. Also available at `http://www.cnl.salk.edu/~bap/papers/iconip-96-cica.ps.gz`.

Propp, J. G., and Wilson, D. B. (1996) Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms* **9** (1-2): 223–252.

Rabiner, L. R., and Juang, B. H. (1986) An introduction to hidden Markov models. *IEEE ASSP Magazine* pp. 4–16.

Reif, F. (1965) *Fundamentals of Statistical and Thermal Physics*. McGraw–Hill.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back–propagating errors. *Nature* **323**: 533–536.

SAUL, L., and JORDAN, M. (1995) Boltzmann chains and hidden Markov models. In *Advances in Neural Information Processing Systems 7*, ed. by G. Tesauro, D. Touretzky, and T. Leen, pp. 435–442. M.I.T. Press.

SEJNOWSKI, T. J. (1986) Higher order Boltzmann machines. In *Neural networks for computing*, ed. by J. Denker, pp. 398–403, New York. American Institute of Physics.

SEJNOWSKI, T. J., and ROSENBERG, C. R. (1987) Parallel networks that learn to pronounce english text. *Journal of Complex Systems* **1** (1): 145–168.

SENETA, E. (1973) *Non-negative Matrices*. New York: Wiley.

SHANNON, C. E. (1993a) The best detection of pulses. In *Collected Papers of Claude Shannon*, ed. by N. J. A. Sloane and A. D. Wyner, pp. 148–150. New York: IEEE Press.

SHANNON, C. E. (1993b) *Collected Papers*. New York: IEEE Press. Edited by N. J. A. Sloane and A. D. Wyner.

SKILLING, J. (1989) Classic maximum entropy. In *Maximum Entropy and Bayesian Methods, Cambridge 1988*, ed. by J. Skilling, Dordrecht. Kluwer.

SKILLING, J. (1993) Bayesian numerical analysis. In *Physics and Probability*, ed. by W. T. Grandy, Jr. and P. Milonni, Cambridge. C.U.P.

STRAUSS, C. E. M., WOLPERT, D. H., and WOLF, D. R. (1993) Alpha, evidence, and the entropic prior. In *Maximum Entropy and Bayesian Methods, Paris 1992*, ed. by A. Mohammed-Djafari, Dordrecht. Kluwer.

TANNER, M. A. (1996) *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions*. Springer Series in Statistics. Springer Verlag, 3rd edition.

TANNER, R. M. (1981) A recursive approach to low complexity codes. *IEEE Transactions on Information Theory* **27** (5): 533–547.

THODBERG, H. H. (1996) Review of Bayesian neural networks with an application to near infrared spectroscopy. *IEEE Transactions on Neural Networks* **7** (1): 56–72.

THOMAS, A., SPIEGELHALTER, D. J., and GILKS, W. R. (1992) BUGS: A program to perform Bayesian inference using Gibbs sampling. In *Bayesian Statistics 4*, ed. by J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, pp. 837–842. Oxford: Clarendon Press.

URBANKE, R., RICHARDSON, T., and SHOKROLLAHI, A., (1999) Design of provably good low density parity check codes. Submitted.

WAHBA, G. (1975) A comparison of GCV and GML for choosing the smoothing parameter in the generalized spline smoothing problem. *Numer. Math.* **24**: 383–393.

WEIGEND, A. S., RUMELHART, D. E., and HUBERMAN, B. A. (1991) Generalization by weight–elimination with applications to forecasting. In *Advances in Neural Information Processing Systems 3*, ed. by R. P. L. et. al., pp. 875–882. Morgan Kaufmann.

WEIR, N. (1991) Applications of maximum entropy techniques to HST data. In *Proceedings of the ESO/ST–ECF Data Analysis Workshop, April 1991*, ed. by P. Grosbol and R. Warmels, pp. 115–129, Garching. European Southern Observatory/Space Telescope – European Coordinating Facility.

WITTEN, I. H., NEAL, R. M., and CLEARY, J. G. (1987) Arithmetic coding for data compression. *Communications of the ACM* **30** (6): 520–540.

WOLPERT, D. H. (1993) On the use of evidence in neural networks. In *Advances in Neural Information Processing Systems 5*, ed. by C. L. Giles, S. J. Hanson, and J. D. Cowan, pp. 539–546, San Mateo, California. Morgan Kaufmann.

YEUNG, R. W. (1991) A new outlook on shannon-information measures. *IEEE Transactions on Information Theory* **37** (3.1): 466–474.

ZIPF, G. K. (1949) *Human Behavior and the Principle of Least Effort*. Addison-Wesley.

Index

accumulator, 245
activation, 360
activation functions, 361
activity, 361
activity rule, 359, 360
adaptive models, 101
adaptive rejection sampling, 300
alchemists, 72
approximation
  of complex distribution by a simpler one, 203
arabic, 140
arithmetic codes, 114
arithmetic coding, 102, **113**
  software, 125
asymptotic equipartition
  why it is a misleading term, 81

balance, 65
ban (unit), 216
Banbury, 216
bandwidth, 196
battleships, 69
Bayes
  Rev. Thomas, 54
Bayesian, 31
Bayesian belief networks, 279
BCH codes, 19
bent coin, 54
Berrou, 204
biased, 274
binary entropy function, 7, 20
binary erasure channel, **165**, 168
binary representations, 146
binary symmetric channel, 9, **165**, 166–168
bipartite graph, 24
birthday problem, 228, 230
bit (unit), 216
bits back, 104, 111
bivariate Gaussian, 308
Bletchley Park, 217
block code, **169**
Bombes, 217
burglar alarm, 279
burglar alarm and earthquake, 161
burst error channels, 203
busstop fallacy, 40, 48, 108

canonical, 87
capacity, 20, **164**, **167**, 168
  symmetry argument, 168
cellular automaton, 136
centre of gravity, 38
channel
  binary erasure, **165**, 168
  binary symmetric, 163, **165**, 166, 167
  bursty, 203
  discrete memoryless, 165
  extended, 170
  fading, 203
  Gaussian, 173
  input ensemble, 167
  noisy, 163
    capacity, 167
  noisy typewriter, **165**, 170
  others, 173
  Z channel, **165**, 166, 167
Clockville, 40
coalesce, 337
code
  concatenated, 204
  data compression, 70
  dodecahedron, 25
  for integers, 146
  Hamming, 24
  hash, 224
  source code, 70, 73
    block code, 73
    symbol code, 90
  stream codes, 113–137
codebreakers, 217
codes
  concatenated, 203
  error-correcting, *see* error-correcting codes
  linear, *see* error-correcting codes

self-delimiting, 146
codeword, *see* source code or error-correcting code
of symbol code, **90**
compression
a difficult problem, 137
lossless, 72
lossy, 72
of already-compressed files, 72
of *any* file, 72
concatenated code, 203
concatenated codes, 203
concatenating, 303
concave∩, 37
conjugate gradient, 369
constraint satisfaction, 410
content addressable, 358
continuous channel, 196
conventions
error function, 173
logarithms, 7
matrices, 165
vectors, 165
convex hull, 103
convex∪, 37
Convex∪ functions, 37
convexity, 300
convolutional code, 204
convolutional codes, 202
cost function, 198
Cox axioms, 30
crib, 217
cryptography
digital signatures, 229
tamper detection, 229

data compression, 71
deciban (unit), 216
decibel, 203
decoder, 163, 169
degrees of belief, 30
dictionary, 123
differentiator, 245
difficulty of compression, 137
digital signature, 229
directory, 224
Dirichlet model, 122
disc drive, 8
discriminant function, 197
distance
D_{KL}, 37
entropy distance, 156
DNA, 231
dodecahedron code, 25
Donald Knuth, 179
doors
game show, 59

earthquake
during gameshow, 60
earthquake and burglar alarm, 161
ensemble
extended, 73
entropy distance, 156
equipartition, 78
ergodic, 125
error function, 173
error probability
block, 169
error-correcting codes
and the difficulty of compression, 137
estimator, 51
evidence, 33, 56, 276
exact sampling, *see* Monte Carlo methods
explaining away, 279
exponential distribution, 48
expurgation, 187
extended channel, 170
extended code, **90**
extended ensemble, 73
extra bit, **97**, 101

Fading channels, 203
fixed point, 403
forgery, 229
forward probability, 31
forward-backward algorithm, 264
free energy
minimization, 342
variational, 342
frustrations, 328
full probabilistic modelling, 174

Galileo, 204
Gallager codes, *see* low-density parity-check codes
Galois fields, 203
game show, 59
Gaussian channel, 173

© David J.C. MacKay. Draft 2.0.7. February 14, 2000

Gaussian distribution, **194**
general Huffman coding algorithm, 109
generative model, 31
generative modelling, 174
generator matrix, 15
genes, 231
genome, 231
Gibbs sampling, **300**, *see* Monte Carlo methods
Gibbs' inequality, 37
Glavieux, 204
gradient descent, 369
graph
  corresponding to code, 24
graphical model, 32
guessing game, 113

Hamming code, 19
  graph, 24
hash code, 224
heat bath, 300
Hertz, 196
hidden Markov models, 461
Hints
  how to compute mutual information, 167
Hopfield network, 400, 410
Huffman code, 90, **98**
  'optimality', 98, 102
  disadvantages, 100, 119
  general alphabet, 104
human-machine interfaces, design of, 140
hybrid Monte Carlo, 308, **308**, 388
hypersphere, 47

identical twin, 114
implicit assumptions, 203
implicit probabilities, **96**
importance sampling, *see* Monte Carlo methods
inference problems
  bent coin, 54
information, 65
information content, 70
  how to measure, 66
information retrieval, 224
inner code, 203
input ensemble, 167
interleaving, 203
inverse probability, 31
inversion
  of hash function, 229
Ising model, 136
Ising models, 320

Jensen's inequality, 38
Jet Propulsion Laboratory, 204
John Conway, 84
joint typicality, 183
joint typicality theorem, 184
juggling, 21
jurors, 57

K-L distance, 37
Kraft inequality, **94**
Kullback-Leibler divergence, 37

Langevin method, *see* Monte Carlo methods
Langevin Monte Carlo method, 388
Laplace model, 122
Laplace's rule, 122
lawyer, 59, 61
learning, 360
learning algorithms, 357
learning rule, 359
Lempel-Ziv, 123–137
  criticisms of, 133
Lempel-Ziv coding, 113
likelihood, 53, 169, 262
likelihood and probability, 33
likelihood principle, 35, 62
limit cycle, 403
linear, 191
linear block code, 14
linear code, **15**
linear feedback shift register, 202
logarithms, 7
long thin strip, 332
lossy compressor, 188
low density parity check codes, 204
Lyapunov function, 403

majority vote, 11
MAP, 11
mapping, 90
marginal likelihood, 33, 276
Markov chain, 157
Markov chain Monte Carlo, *see* Monte Carlo methods

Markov model, *see* Markov chain and hidden Markov model
maximum likelihood, 11
maximum likelihood decoder, 169
MCMC, *see* Monte Carlo methods
mean field, 344
memory, address–based, 358
memory, associative, 358
message-passing algorithms, 262
metacode, 104, 110
metric, 406
Metropolis method, 388, *see* Monte Carlo methods
microwave oven, 140
min-sum algorithm, 263
mixing, 303
mixture code, 130
mobile phone, 203
models of images, 416
momentum, 369
Monte Carlo methods
  Markov chain Monte Carlo, 296
  Gibbs sampling, **300**
  importance sampling, **291**
  Markov chain Monte Carlo, **295**
  Metropolis-Hastings method, **295**
  rejection sampling, **293**
Morse code, 247
multivariate Gaussian distribution, 194
murder, 59, 61
mutual information, 163, 167, 168
  how to compute, 167
myths
  compression, 72

nat (unit), 216
neural network, 357
neuron, 360
noisy channel
  capacity, 167
noisy channel coding theorem, 169, 183
noisy typewriter, **165**, 170, 172
noisy-channel coding theorem, 20
non-confusable inputs, 170
noninformative, 273
notation
  absolute value, 36
  conventions of this book, 165
  expectation, 39
  intervals, 89
  logarithms, 7
  matrices, 165
  set size, 36
  vectors, 165
NP-complete problem, 410

one-way hash function, 229
optimal decoder, 169
optimal input distribution, **167**
optimization, 369
optimization problems, 410
outer code, 203

parity, 14
parity check bits, 14
parity check matrix, 17
parity check nodes, 24
partial partition functions, 329
partition function
  partial, 329
phase transition, 290
phone directory, 224
phone number, 224
Poisson process, 40, 48
Poissonville, 40
power cost, 198
practical, 201
precision, **194**, 199
predictive distribution, 114
prefix, **91**
prefix code, **91**, 95
prefix codes, 95
probabilistic model, 114, 125
probability
  Bayesian, 52
probability of block error, 169
probability propagation, 204, 205
product code, 203
proposal density, **293**
protein, 231, 234

Radford Neal, 125
random coding, 186
random walk, 297
random–coding exponent, 191
random-walk Metropolis method, 308
rate, **169**
rectangular code, 203
redundancy

in channel code, 163
redundant
constraints in code, 25
Reed-Solomon code, 204
Reed-Solomon codes, 203
rejection sampling, *see* Monte Carlo methods
adaptive, 300
relative entropy, 37
repetition code, 21
repetition codes, 19
roman, 140
runlength–limited channels, 239

saddle-point approximation, 284
satellite communications, 204
Shannon, 20, 21, *see* noisy channel coding theorem, source coding theorem
noisy channel coding theorem, 169
shannon (unit), 217
Shannon information content, 70
Shannon's noisy channel coding theorem, 183
shifter ensemble, 416
Simpson, O.J., allusion to, 61
Simpson, O.J., possible allusion to, 59
slice sampling, *see* Monte Carlo methods
software
arithmetic coding, 125
source code
for integers, 146
optimal lengths, 96
prefix code, 95
source codes
uniquely decodeable, 93
source coding
with variable symbol durations, 247
span, 269
sphere–packing exponent, 192
statistical physics, 322
stereoscopic vision, 416
stream codes, 113–137
subjective, 31
submarine, 69
sum-product, 264
sum-product algorithm, 204, 205, 495
symbol code
disadvantages, 100
symbol codes, **90**
symbol coding budget, **94**
symbol-code supermarket, 94, 115
symmetry argument, 168
syndrome, 16, 17
syndrome decoding, 17

telephone numbers, 133
thermal distribution, 87
Thitimajshima, 204
three doors, 59
transfer matrix method, 329
transition probabilities, 165
translation-invariant, 329
travelling salesman problem, 410
trellis, 244
trellis section, 241
turbo codes, 204
Turing, Alan, 217
twenty questions, 67
twin, 114
typical set, **76**
typical set decoding, 186
typical sets, 172
for noisy channel, 172

uniquely decodeable, **92**
uniquely decodeable symbol codes, 93
universal, 125

variable duration symbols, 247
variational free energy, 342
variational free energy minimization, 342
variational methods, 341
Viterbi algorithm, 263
volume, 47

waiting for a bus, 40, 48
weighing problem, 65, 67
weight, 25
of noise pattern, 16
weight decay, 369
wife-beaters, 59, 61

Z channel, **165**, 166–168, 173