# Information Theory, Inference, and Learning Algorithms

David J.C. MacKay

Draft 2.3.5 February 19, 2002

## Information Theory, Pattern Recognition and Neural Networks

Handout number 3.

Do not be disturbed by missing pagenumbers: the handouts do not include all the book's chapters.

Because the handouts are based on different drafts of the book, there may be occasional mismatches between pagereferences, etc., where cross-references occur between this and the other handouts.

Nonexaminable material is indicated by the symbol $^*$.

## Approximate roadmap for the course

| | |
|---|---|
| Lecture 1 | Introduction to Information Theory. Chapter 1. |
| Before lecture 2 | Please work on exercise 4.4 (p. 76). |
| About now | Read chapters 2 and 5 and work on exercises in chapter 2. |
| Lecture 2–3 | Information content & typicality. Chapter 5. |
| Lecture 4 | Symbol codes. Chapter 6. |
| Lecture 5 | Arithmetic codes. Chapter 7. |
| About now | Read chapter 9 and do the exercises. |
| Lecture 6 | Noisy channels. Definition of mutual information and capacity. Chapter 10. |
| Lecture 7-8 | The noisy channel coding theorem. Chapter 11. |
| Lecture 9 | Clustering. Bayesian inference. Chapter 3, 22, 24. |
| About now | Read chapter 32 (Ising models). |
| Lecture 10 | Monte Carlo methods. Chapter 30, 31. |
| Lecture 12 | Variational methods. Chapter 34. |
| Lecture 13 | Neural networks – the single neuron. Chapter 41. |
| Lecture 14 | Capacity of the single neuron. Chapter 42. |
| Lecture 15 | Learning as inference. Chapter 43. |
| Lecture 16 | The Hopfield network. Content-addressable memory. Chapter 44. |

# *Contents*

# *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.

Unfamiliar notation? See appendix A, p. 592.

## *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.

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

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

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

$$\begin{aligned} \mathrm{var}[r] &\equiv \mathcal{E}\left[(r-\mathcal{E}[r])^2\right] && (0.3) \\ &= \mathcal{E}[r^2] - (\mathcal{E}[r])^2 = \sum_{r=0}^{N} P(r|f,N) r^2 - (\mathcal{E}[r])^2 . && (0.4) \end{aligned}$$

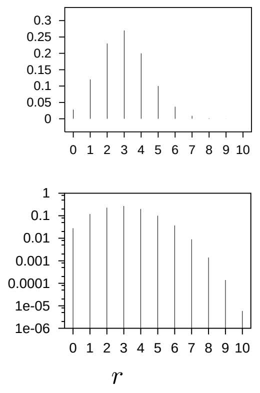


Figure 0.1. The binomial distribution $P(r|f{=}0.3,\, N{=}10)$, on a linear scale (top) and a logarithmic scale (bottom).

Rather than evaluating the sums over $r$ (0.2,0.4) 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 (which is either zero or one) in the first toss, the number of heads in the second toss, and so forth. In general,

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

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:

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

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

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

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

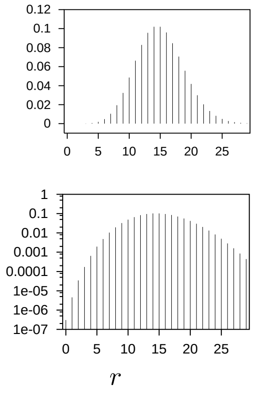


Figure 0.2. The Poisson distribution $P(r|\lambda{=}15)$, on a linear scale (top) and a logarithmic scale (bottom).

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

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

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

$$\begin{aligned} e^{-\lambda}\frac{\lambda^\lambda}{\lambda!} &\simeq \frac{1}{\sqrt{2\pi\lambda}} && (0.10)\\ \Rightarrow \lambda! &\simeq \lambda^\lambda e^{-\lambda}\sqrt{2\pi\lambda}. && (0.11)\end{aligned}$$

This is Stirling's approximation for the factorial function, 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 $\binom{N}{r} \equiv \frac{N!}{(N-r)!\,r!}$.

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

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'.

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

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

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

$$\log\binom{N}{r} \simeq NH_2(r/N), \tag{0.15}$$

or, equivalently,

$$\binom{N}{r} \simeq 2^{NH_2(r/N)}. \tag{0.16}$$

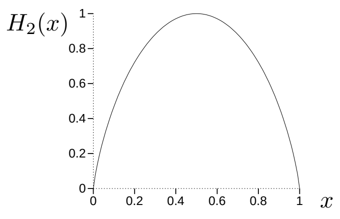


Figure 0.3. The binary entropy function

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

$$\log\binom{N}{r} \simeq NH_2(r/N) - \tfrac{1}{2}\log\left[2\pi N\,\frac{N-r}{N}\,\frac{r}{N}\right]. \tag{0.17}$$

# 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)*

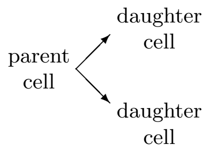
In the first half of this book we study how to measure information content; we learn how to compress data; and we learn how to communicate perfectly over imperfect communication channels.

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.



computer memory
disc drive
computer memory

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 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).

$$x \;\begin{tikzcd} 0 \arrow[r] \arrow[dr] & 0 \\ 1 \arrow[r] \arrow[ur] & 1 \end{tikzcd}\; y \qquad \begin{array}{lcl lcl} P(y{=}0|x{=}0) & = & 1-f; & P(y{=}0|x{=}1) & = & f; \\ P(y{=}1|x{=}0) & = & f; & P(y{=}1|x{=}1) & = & 1-f. \end{array}$$

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 $f$.

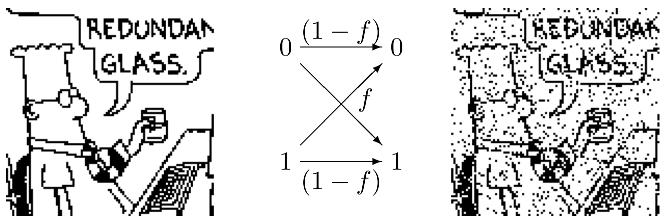


Figure 1.2. A binary data sequence of length 10000 transmitted over a binary symmetric channel with noise level $f = 0.1$. [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

As an example, let's imagine that $f = 0.1$, that is, ten per cent 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 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.

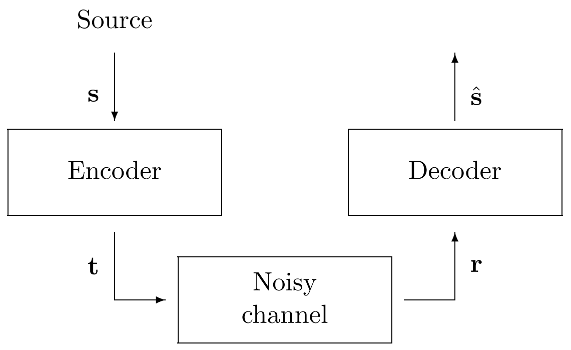


Figure 1.3. The 'system' solution for achieving reliable communication over a noisy channel. The encoding system introduces systematic redundancy into the transmitted vector **t**. The decoding system uses this known redundancy to deduce from the received vector **r** *both* the original source vector *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 channels 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? [To make the rules of the game clear: we want to be able to detect *and* correct errors; and retransmission is not an option. We get only one chance to encode, transmit, and decode.]

*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* '$R_3$'.

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

Figure 1.4. The 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 $\mathbf{n}$ to the transmitted vector – adding in modulo 2 arithmetic, i.e., the binary algebra in which $\mathtt{1}+\mathtt{1}=\mathtt{0}$). A possible noise vector $\mathbf{n}$ and received vector $\mathbf{r} = \mathbf{t} + \mathbf{n}$ are shown in figure 1.5.

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| $\mathbf{s}$ | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| $\mathbf{t}$ | $\overbrace{000}$ | $\overbrace{000}$ | $\overbrace{111}$ | $\overbrace{000}$ | $\overbrace{111}$ | $\overbrace{111}$ | $\overbrace{000}$ |
| $\mathbf{n}$ | 000 | 001 | 000 | 000 | 101 | 000 | 000 |
| $\mathbf{r}$ | 000 | 001 | 111 | 000 | 010 | 111 | 000 |

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

$$P(s \,|\, r_1 r_2 r_3) = \frac{P(r_1 r_2 r_3 \,|\, s) P(s)}{P(r_1 r_2 r_3)}. \tag{1.1}$$

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

$$P(s=\mathtt{1} \,|\, r_1 r_2 r_3) = \frac{P(r_1 r_2 r_3 \,|\, s=\mathtt{1}) P(s=\mathtt{1})}{P(r_1 r_2 r_3)}; \tag{1.2}$$

$$P(s=\mathtt{0} \,|\, r_1 r_2 r_3) = \frac{P(r_1 r_2 r_3 \,|\, s=\mathtt{0}) P(s=\mathtt{0})}{P(r_1 r_2 r_3)}. \tag{1.3}$$

This posterior probability is determined by two factors: the *prior probability* $P(s)$, and the data-dependent term $P(r_1 r_2 r_3 \,|\, s)$, which is called the *likelihood* of $s$. The normalizing constant $P(r_1 r_2 r_3)$ is irrelevant to the optimal decoding decision, which is to guess $\hat{s}=\mathtt{0}$ if $P(s=\mathtt{0} \,|\, \mathbf{r}) > P(s=\mathtt{1} \,|\, \mathbf{r})$, and $\hat{s}=\mathtt{1}$ otherwise.

To find $P(s = \mathtt{0} \,|\, \mathbf{r})$ and $P(s = \mathtt{1} \,|\, \mathbf{r})$, we must make an assumption about the prior probabilities of the two hypotheses $s = \mathtt{0}$ and $s = \mathtt{1}$, and we must make an assumption about the probability of $\mathbf{r}$ given $s$. We assume that the prior probabilities are equal: $P(s=\mathtt{0}) = P(s=\mathtt{1}) = 0.5$; then maximizing the posterior probability $P(s \,|\, \mathbf{r})$ is equivalent to maximizing the likelihood $P(\mathbf{r} \,|\, s)$. And we 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

$$P(r_n \,|\, t_n) = \begin{cases} (1-f) & \text{if} \quad r_n = t_n \\ f & \text{if} \quad r_n \neq t_n. \end{cases} \tag{1.5}$$

Thus the likelihood ratio for the two hypotheses is

$$\frac{P(\mathbf{r} \,|\, s=\mathtt{1})}{P(\mathbf{r} \,|\, s=\mathtt{0})} = \prod_{n=1}^{N} \frac{P(r_n \,|\, t_n(\mathtt{1}))}{P(r_n \,|\, t_n(\mathtt{0}))}; \tag{1.6}$$

each factor $\frac{P(r_n \,|\, t_n(\mathtt{1}))}{P(r_n \,|\, t_n(\mathtt{0}))}$ 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 $\gamma$ in the likelihood ratio.

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

Algorithm 1.1. Majority-vote decoding algorithm for $R_3$. Also shown are the likelihood ratios, assuming the channel is a binary symmetric channel; $\gamma \equiv (1-f)/f$.

Thus the majority-vote decoder shown in algorithm 1.1 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.

We now apply the majority vote decoder to the received vector of figure 1.5. The first three received bits are all 0, so we decode this triplet as a 0. In the second triplet of figure 1.5, there are two 0s and one 1, so we decode this triplet 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 1.6.

$$
\begin{array}{r|ccccccc}
\mathbf{s} & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\mathbf{t} & \overbrace{000} & \overbrace{000} & \overbrace{111} & \overbrace{000} & \overbrace{111} & \overbrace{111} & \overbrace{000} \\
\mathbf{n} & 000 & 001 & 000 & 000 & 101 & 000 & 000 \\
\hline
\mathbf{r} & \underbrace{000} & \underbrace{001} & \underbrace{111} & \underbrace{000} & \underbrace{010} & \underbrace{111} & \underbrace{000} \\
\hat{\mathbf{s}} & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
\text{corrected errors} & & \star & & & & & \\
\text{undetected errors} & & & & & \star & & \\
\end{array}
$$

Figure 1.6. Decoding the received vector from figure 1.5.

Solutions on p.22

Exercise 1.1:$^{A2}$ Show that the error probability is reduced by the use of $R_3$ by computing the error probability of this code 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 1.7 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

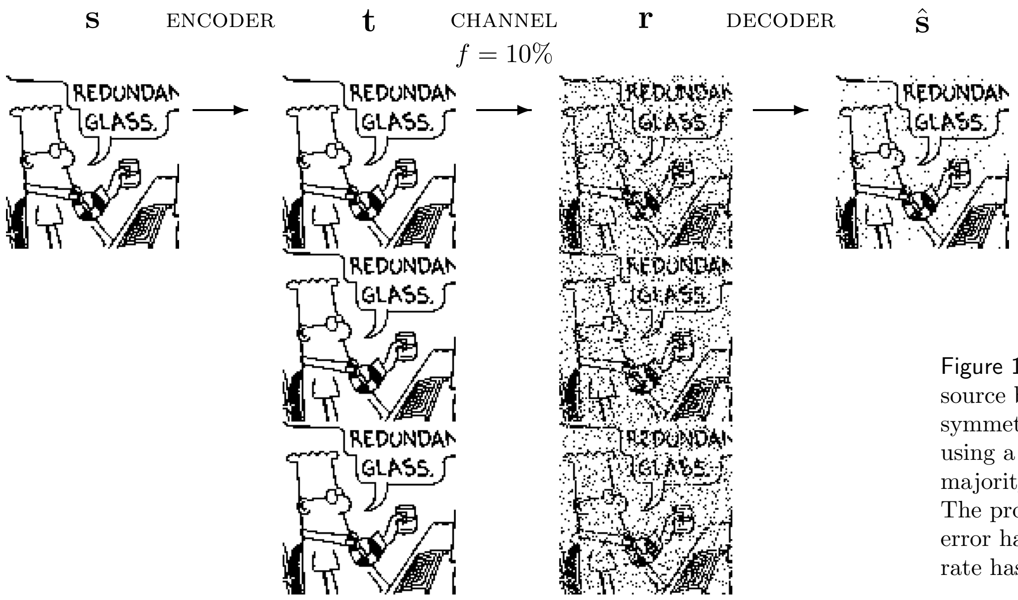


Figure 1.7. Transmitting 10000 source bits over a binary symmetric channel with $f = 10\%$ using a repetition code and the majority vote decoding algorithm. The probability of decoded bit error has fallen to about 3%; the rate has fallen to 1/3.

phone call. Similarly, we would need three of the original noisy gigabyte disc drives in order to create a one-gigabyte disc drive with $p_\mathrm{b} = 0.03$.

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.

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 $N$,

$$p_\mathrm{b} = \sum_{n=(N+1)/2}^{N} \binom{N}{n} f^n (1-f)^{N-n}. \tag{1.7}$$

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

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

(d) Assuming $f = 0.1$, 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 1.8.

## *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

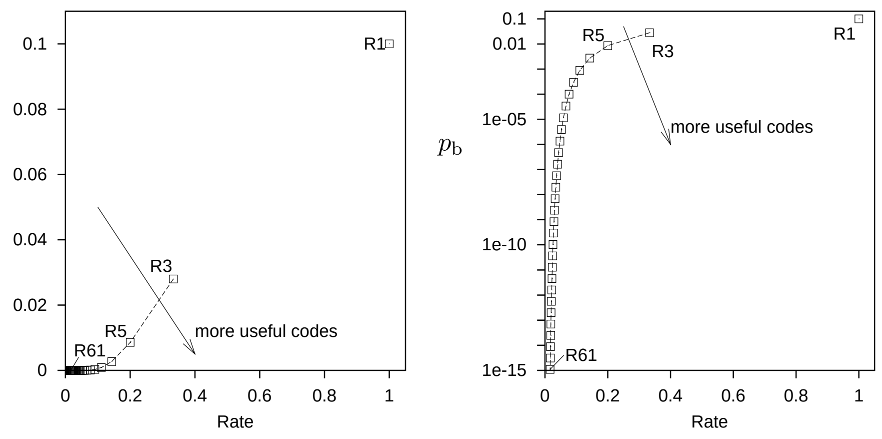


Figure 1.8. Error probability $p_b$ versus rate for repetition codes over a binary symmetric channel with $f = 0.1$. The right hand figure shows $p_b$ on a logarithmic scale. We would like the rate to be large and $p_b$ to be small.

*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 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.

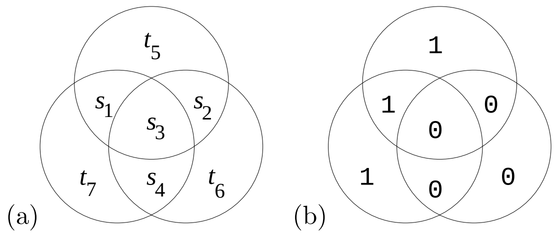


Figure 1.9. Pictorial representation of encoding for the Hamming (7,4) code.

The encoding operation for the code is shown pictorially in figure 1.9. 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 1.9b shows the transmitted codeword for the case $\mathbf{s} = $ `1000`.

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 $\mathbf{t}$ is obtained from the source sequence

| **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 1.10. The sixteen codewords $\{\mathbf{t}\}$ of the (7,4) Hamming code. Any pair of codewords differ from each other in at least three bits.

$\mathbf{s}$ by a linear operation,

$$\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{1.8}$$

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

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \tag{1.9}$$

and the encoding equation (1.8) uses modulo-2 arithmetic [$1 + 1 = 0$, $0 + 1 = 1$, etc.].

In the encoding operation (1.8) I have assumed that $\mathbf{s}$ and $\mathbf{t}$ are column vectors. If instead they are row vectors, then this equation is replaced by

$$\mathbf{t} = \mathbf{s}\mathbf{G}, \tag{1.10}$$

where

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.11}$$

I find it easier to relate to the right-multiplication (1.8) than the left-multiplication (1.10). Many coding theory texts use the left-multiplying conventions (1.10–1.11), however.

The rows of the generator matrix (1.11) can be viewed as defining four basis vectors lying in a seven-dimensional binary space. The sixteen codewords are obtained by making all possible linear combinations of these vectors.

### *Decoding the (7,4) Hamming code*

When we invent a more complex encoder $\mathbf{s} \rightarrow \mathbf{t}$, 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 (1.6) 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 1.10 then picking the closest. 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, based on the encoding picture, figure 1.9.

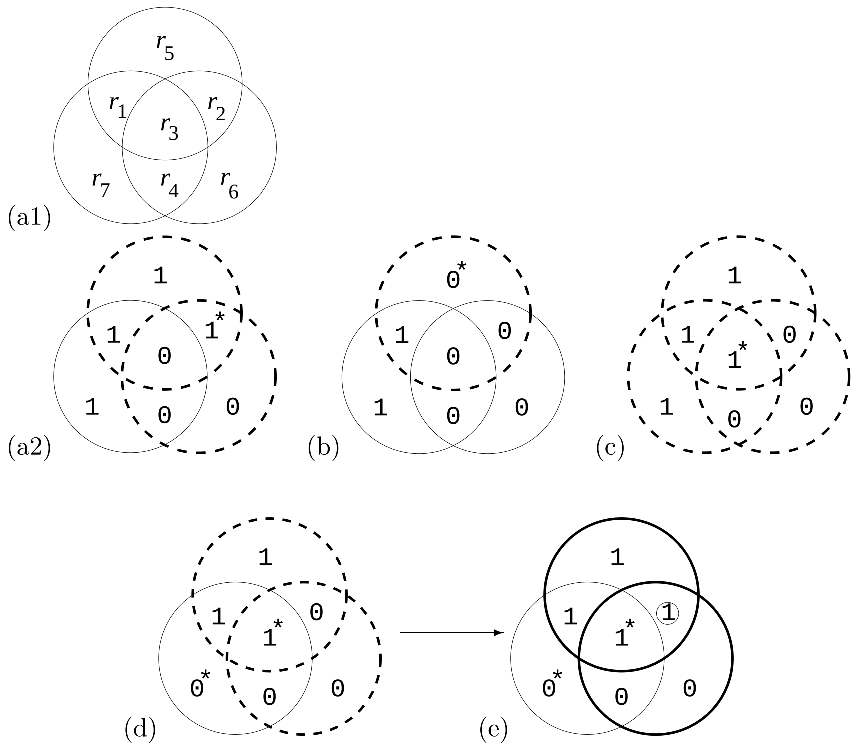


Figure 1.11. Pictorial representation of decoding of the Hamming (7,4) code. The bits that are flipped relative to figure 1.9(b) are labelled by $\star$. The violated parity checks are highlighted by dashed circles. One of the seven bits is the most probable suspect to account for each 'syndrome', i.e., 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.

As a first example, let's assume the transmission was $\mathbf{t} = \texttt{1000101}$ and the noise flips the second bit, so the received vector is $\mathbf{r} = \texttt{1000101} \oplus \texttt{0100000} = \texttt{1100101}$. We write the received vector into the three circles as shown in figure 1.11(a), and look at each of the three circles to see whether its parity is even. The circles whose parity is *not* even are shown by dashed lines. The decoding task is to find the smallest set of flipped bits that can account for these violations of the parity rules. [The pattern of violations of the parity checks is called the *syndrome*, and can be written as a binary vector – for example, in figure 1.11(a), the syndrome is $\mathbf{z} = (\texttt{1}, \texttt{1}, \texttt{0})$, because the first two circles are 'unhappy' (parity 1) and the third circle is 'happy' (parity 0).]

To solve this decoding task, we ask the question: can we find a unique bit that lies *inside* all the 'unhappy' circles and *outside* all the 'happy' circles? If so, the flipping of that bit could account for the observed syndrome. In the case shown in figure 1.11(a), the bit $r_2$ lies inside the two 'unhappy' circles and outside the third circle; no other single bit has this property, so $r_2$ is the only single bit capable of explaining the syndrome.

Let's work through a couple more examples. Figure 1.11(b) shows what happens if one of the parity bits, $t_5$, is flipped by the noise. Just one of the checks is violated. Only $r_5$ lies inside this unhappy circle and outside the other two happy circles, so $r_5$ is identified as the only single bit capable of explaining the syndrome.

If the central bit $r_3$ is received flipped, figure 1.11(c) shows that all three checks are violated; Only $r_3$ lies inside all three circles, so $r_3$ is identified as the suspect bit.

| Syndrome $\mathbf{z}$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| Unflip this bit | *none* | $r_7$ | $r_6$ | $r_4$ | $r_5$ | $r_1$ | $r_2$ | $r_3$ |

Algorithm 1.2. Actions taken by the optimal decoder for the (7,4) Hamming 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 $r_7$.

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 algorithm 1.2. 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 1.11(d) shows the situation when two bits, $r_3$ and $r_7$, are received flipped. The syndrome, 110, 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 1.11(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 are happy – 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 $3 \times 4$ matrix $\mathbf{P}$ such that the matrix of equation (1.9) is

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_4 \\ \mathbf{P} \end{bmatrix}, \tag{1.12}$$

where $\mathbf{I}_4$ is the $4 \times 4$ identity matrix, then the syndrome vector is $\mathbf{z} = \mathbf{Hr}$, 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

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.13}$$

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

$$\mathbf{Ht} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.14}$$

Solutions on p.23

Exercise 1.3:$^{B2}$ Prove that this is so by evaluating the $3 \times 4$ matrix $\mathbf{HG}^{\mathsf{T}}$.

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

$$\mathbf{Hn} = \mathbf{z}. \tag{1.15}$$

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

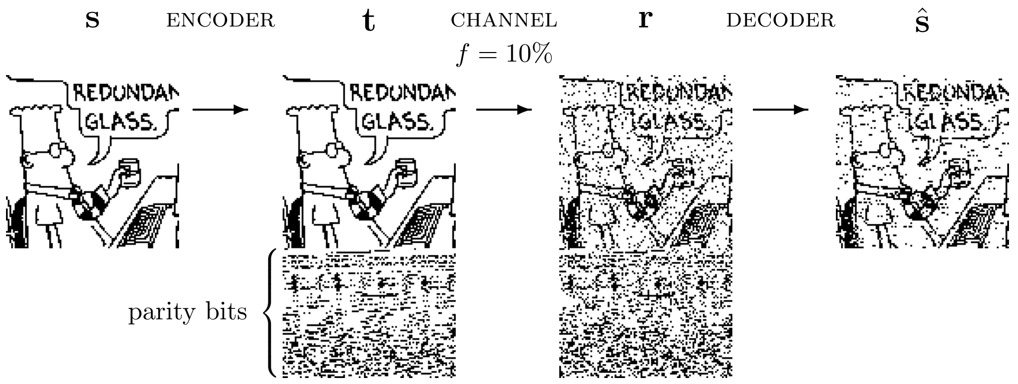


Figure 1.12. Transmitting 10,000 source bits over a binary symmetric channel with $f = 10\%$ using a (7,4) Hamming code. The probability of decoded bit error is about 7%.

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

Every possible received vector of length 7 bits is either a codeword, or it's one flip away from a codeword.

Since there are three parity constraints, each of which might or might not be violated, there are $2 \times 2 \times 2 = 8$ 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.

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,

$$p_{\rm B} = P(\hat{\mathbf{s}} \neq \mathbf{s}). \tag{1.16}$$

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

$$p_{\rm b} = \frac{1}{K} \sum_{k=1}^{K} P(\hat{s}_k \neq s_k). \tag{1.17}$$

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 $\mathrm{R}_3$. But notice that the Hamming code communicates at a greater rate, $R = 4/7$.

Figure 1.12 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 1.4:$^{A1}$ This exercise and the next three refer to the (7,4) Hamming code. Decode the received strings:

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

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

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

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

Exercise 1.5:$^{A2}$ (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) $^{B3}$ Show that to leading order the probability of bit error $p_{\rm b}$ goes as $9f^2$.

Exercise 1.6:$^{A2}$ 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 1.7:$^{B2}$ 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 1.8:$^{B2}$ Consider the repetition code $\mathrm{R}_9$. One way of viewing this code is as a *concatenation* of $\mathrm{R}_3$ with $\mathrm{R}_3$. We first encode the source stream with $\mathrm{R}_3$, then encode the resulting output with $\mathrm{R}_3$. We could call this code '$\mathrm{R}_3^2$'. This idea motivates an alternative decoding algorithm, in which we decode the bits three at a time using the decoder for $\mathrm{R}_3$; then decode the decoded bits from that first decoder using the decoder for $\mathrm{R}_3$.

Evaluate the probability of error for this decoder and compare it with the probability of error for the optimal decoder for $\mathrm{R}_9$.

Do the concatenated encoder and decoder for $\mathrm{R}_3^2$ have advantages over those for $\mathrm{R}_9$?

*Summary of codes' performances*

Figure 1.13 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.
This figure shows that we can, using linear block codes, achieve better performance than repetition codes; but the asymptotic situation still looks grim.

Exercise 1.9:$^{A5}$ Design an error-correcting code and a decoding algorithm for it, compute its probability of error, and add it to figure 1.13. [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

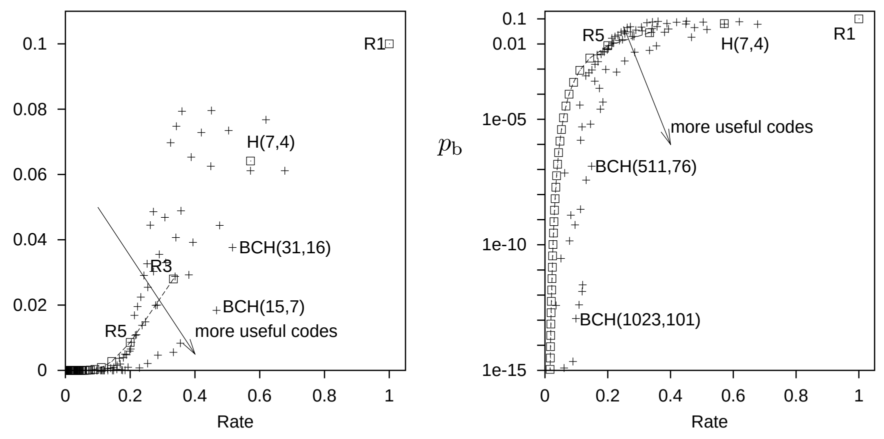


Figure 1.13. Error probability $p_b$ versus rate $R$ for repetition codes, the (7,4) Hamming code and BCH codes with block lengths up to 1023 over a binary symmetric channel with $f = 0.1$. The righthand figure shows $p_b$ on a logarithmic scale.

$(R, p_b)$ 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 $(R, p_b)$ plane was a curve passing through the origin $(R, p_b) = (0, 0)$; if this were so, then, in order to achieve a vanishingly small error probability $p_b$, 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 1.14. For any channel, there exist codes that

☆☆☆!

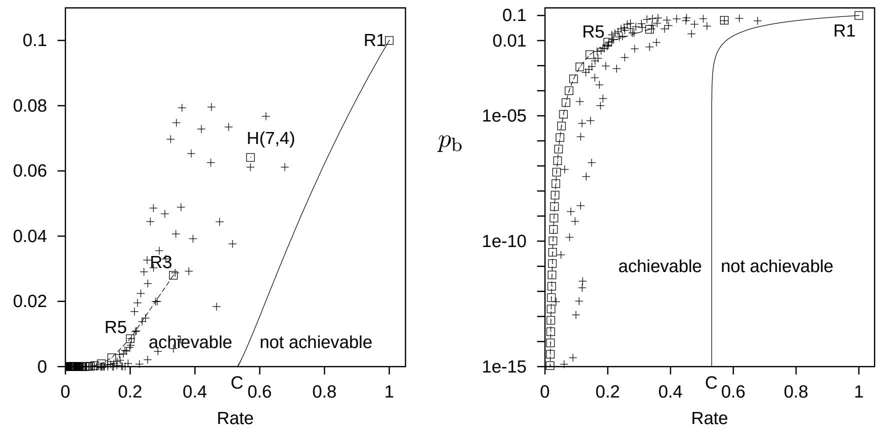


Figure 1.14. Shannon's noisy-channel coding theorem. The solid curve shows the Shannon limit on achievable values of $(R, p_b)$ for the binary symmetric channel with $f = 0.1$. Rates up to $R = C$ are achievable with arbitrarily small $p_b$. The points show the performance of some textbook codes, as in figure 1.13.
The equation defining the Shannon limit (the solid curve) is $R = C/(1 - H_2(p_b))$, where $C$ and $H_2$ are defined in equation (1.18).

make it possible to communicate with *arbitrarily small* probability of error $p_b$ 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

$$C(f) = 1 - H_2(f) = 1 - \left[ f \log_2 \frac{1}{f} + (1-f) \log_2 \frac{1}{1-f} \right]; \tag{1.18}$$

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. The repetition code $\mathrm{R}_3$ 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 (since 1/2 is less than 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, since, as we shall see, Shannon proved his noisy-channel coding theorem 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, you 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 11, 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): An error is made by $\mathrm{R}_3$ if two or more bits are flipped in a block of three. So the error probability of $\mathrm{R}_3$ is a sum of two terms: the probability of all three bits's being flipped, $f^3$; and the probability of exactly two bits's being flipped, $3f^2(1-f)$. [If these expressions are not obvious, see example 0.1 (p.6): the expressions are $P(r=3|f,N=3)$ and $P(r=2|f,N=3)$.]

$$p_b = p_B = 3f^2(1-f) + f^3 = 3f^2 - 2f^3. \tag{1.19}$$

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

See exercise 2.4 (p.31) for further discussion of this problem.

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

$$\binom{N}{\lceil N/2 \rceil} f^{(N+1)/2}(1-f)^{(N-1)/2}. \tag{1.20}$$

Notation: $\lceil N/2 \rceil$ denotes the smallest integer greater than or equal to $N/2$.

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

$$\frac{1}{N+1} 2^{NH_2(K/N)} \leq \binom{N}{K} \leq 2^{NH_2(K/N)} \Rightarrow \binom{N}{K} \simeq 2^{NH_2(K/N)}, \tag{1.21}$$

where this approximation introduces an error of order $\sqrt{N}$ – as shown in equation (0.17). So

$$p_b = p_B \simeq 2^N (f(1-f))^{N/2} = (4f(1-f))^{N/2}. \tag{1.22}$$

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:

$$\binom{N}{N/2} \simeq 2^N \frac{1}{\sqrt{2\pi N/4}}. \tag{1.23}$$

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

$$1 = \sum_K \binom{N}{K} 2^{-N} \simeq 2^{-N} \binom{N}{N/2} \sum_{r=-N/2}^{N/2} e^{-r^2/2\sigma^2} \simeq 2^{-N} \binom{N}{N/2} \sqrt{2\pi}\sigma, \tag{1.24}$$

where $\sigma = \sqrt{N/4}$, from which equation (1.23) 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

$$p_b \simeq \binom{N}{(N+1)/2} f^{(N+1)/2}(1-f)^{(N-1)/2} \quad (1.25)$$

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

$$\simeq \frac{1}{\sqrt{\pi N/8}} f[4f(1-f)]^{(N-1)/2} \quad (1.27)$$

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

$$(N-1)/2 \simeq \frac{\log 10^{-15} + \log \frac{\sqrt{\pi N/8}}{f}}{\log 4f(1-f)} \quad (1.28)$$

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

$$(\hat{N}_2 - 1)/2 \simeq \frac{-15 + 1.7}{-0.44} = 29.9 \Rightarrow \hat{N}_2 \simeq 60.9 \quad (1.29)$$

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.17): The matrix $\mathbf{HG}^{\mathsf{T}} \bmod 2$ is equal to the all-zero $3 \times 4$ matrix, so for any codeword $\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}$, $\mathbf{Ht} = \mathbf{HG}^{\mathsf{T}}\mathbf{s} = (0,0,0)^{\mathsf{T}}$.

Solution to exercise 1.4 (p.18): (a) `1100` (b) `0100` (c) `0100` (d) `1111`.

Solution to exercise 1.5 (p.19):

(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.

$$p_B = \sum_{r=2}^{7} \binom{7}{r} f^r (1-f)^{7-r}. \quad (1.30)$$

To leading order, this goes as

$$p_B \simeq \binom{7}{2} f^2 = 21 f^2. \quad (1.31)$$

(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. That means, if we average over all seven bits, the probability of one of them's being flipped is 3/7 times the block error probability, to leading order. Now, what we really care about is the probability of a source bit's being flipped. 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 bit 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 (for the source bits) is simply three sevenths of the probability of block error.

$$p_b \simeq \frac{3}{7} p_B \simeq 9f^2. \tag{1.32}$$

Solution to exercise 1.6 (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 1.7 (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 1.8 (p.19): The probability of error of $\mathrm{R}_3^2$ is, to leading order,

$$p_b(\mathrm{R}_3^2) \simeq 3\,[p_b(\mathrm{R}_3)]^2 = 3(3f^2)^2 + \ldots = 27f^4 + \ldots, \tag{1.33}$$

whereas the probability of error of $\mathrm{R}_9$ is dominated by the probability of five flips,

$$p_b(\mathrm{R}_9) \simeq \binom{9}{5} f^5(1-f)^4 \simeq 126f^5 + \ldots. \tag{1.34}$$

The $\mathrm{R}_3^2$ 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 simple code illustrates an important concept. Concatenated codes are widely used in practice because concatenation allows large codes to be implemented using simple encoding and decoding hardware. Some of the best known practical codes are concatenated codes.

## *Graphs corresponding to codes*

Solution to exercise 1.9 (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. There are many ways to design codes, and what follows is just one possible train of thought.

Here is an example of a linear block code that is similar to the (7,4) Hamming code, but bigger.

Many codes can be conveniently expressed in terms of graphs. In figure 1.9, 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 1.15. The 7 circles are the 7 transmitted

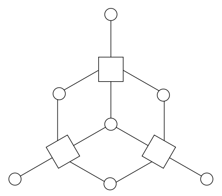


Figure 1.15. The graph of the (7,4) Hamming code. The 7 circles are the bit nodes and the 3 squares are the parity check nodes.

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 (1.13) are simply related to each other: each parity check node corresponds to a 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.

Having noticed this connection between linear codes and graphs, one simple way to invent linear codes is to think of a bipartite graph. For example, 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 `1`s it contains.]

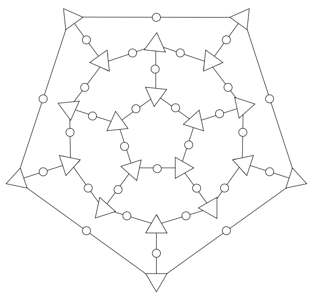


Figure 1.16. The graph defining the (30,11) dodecahedron code. The circles are the 30 transmitted bits and the triangles are the 20 parity checks, One parity check is redundant.

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, that is, if 19 constraints are satisfied, then the 20th is automatically satisfied; 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. If we flip all the bits surrounding one face of the original dodecahedron, then all the parity checks will be satisfied; so the code has 12 codewords of weight 5, one for each face. Since the lowest-weight codewords have weight 5, we say that the code has distance $d = 5$; the (7,4) Hamming code had distance 3 and could correct all single bit-flip errors. A code with distance 5 can correct all double bit-flip errors, but there are some triple bit-flip errors that it cannot correct. So the error probability of this code, assuming a binary symmetric channel, will be dominated, at least for low noise levels $f$, by a term of order $f^3$, perhaps something like

$$12 \binom{5}{3} f^3 (1-f)^{27}. \tag{1.35}$$

Of course, there is no obligation to make codes whose graphs can be represented on a plane, as this one can; the best linear codes, which have simple graphical descriptions, have graphs that are more tangled, as illustrated by the tiny (16,4) code of figure 1.17.

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 the encoding and decoding of a nonlinear code are even trickier tasks.

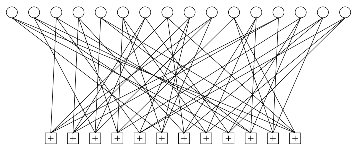


Figure 1.17. Graph of a rate 1/4 low-density parity-check code (Gallager code) with blocklength $N = 16$, and $M = 12$ constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by $\boxed{+}$ squares. The edges between nodes were placed at random. (See chapter 53 for more.)

# *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 (1.18).

# 2

# *Probability, Entropy, and Inference*

This chapter, and its sibling, chapter 9, 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 a 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. For example, if something is 'true with probability 1', I will usually simply say that it is 'true'.

## 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 '`-`'.

| $i$ | $a_i$ | $p_i$ |
|---|---|---|
| 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 |

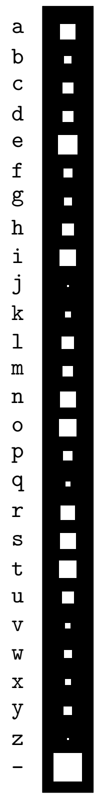


Figure 2.1. Probability distribution over the 27 outcomes for a randomly selected letter in an English language document (estimated from *The Frequently Asked Questions Manual for Linux*). The picture shows the probabilities by the areas of white squares.

**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, $V = \{\texttt{a}, \texttt{e}, \texttt{i}, \texttt{o}, \texttt{u}\}$, 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.

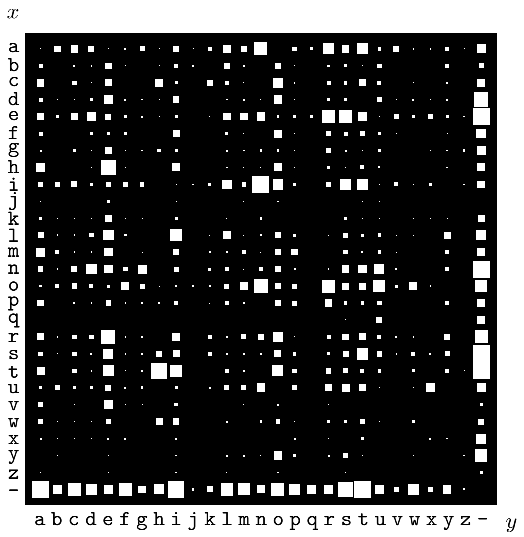


Figure 2.2. The probability distribution over the 27×27 possible bigrams $xy$ in an English language document, *The Frequently Asked Questions Manual for Linux.*

**Marginal probability.** We can obtain the marginal probability $P(x)$ 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, the marginal probability of $y$ is:

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

**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 $P(x, y)$ we can obtain conditional distributions, $P(y|x)$ and $P(x|y)$, by normalizing the rows and columns, respectively

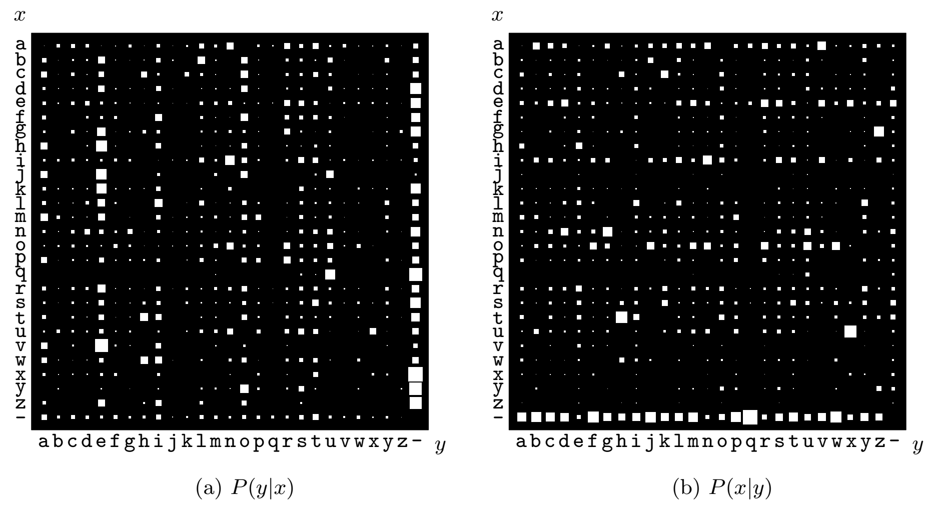


Figure 2.3. Conditional probability distributions. (a) $P(y|x)$: Each *row* shows the conditional distribution of the second letter, $y$, given the first letter, $x$, in a bigram $xy$. (b) $P(x|y)$: Each *column* shows the conditional distribution of the first letter, $x$, given the second letter, $y$.

(figure 2.3). The probability $P(y|x = \mathtt{q})$ is the probability distribution of the second letter given that the first letter is a `q`. As you can see in figure 2.3(a), the two most probable values for the second letter $y$ given the first letter $x = \mathtt{q}$ are `u` and `-`. (The space is common after `q` because the source document makes heavy use of the word `FAQ`.)

The probability $P(x|y = \mathtt{u})$ is the probability distribution of the first letter $x$ given that the second letter $y$ is a `u`. As you can see in figure 2.3(b) the two most probable values for $x$ given $y = \mathtt{u}$ 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:

$$\begin{aligned} P(x|\mathcal{H}) &= \sum_y P(x, y|\mathcal{H}) &(2.7)\\ &= \sum_y P(x|y, \mathcal{H})P(y|\mathcal{H}). &(2.8)\end{aligned}$$

**Bayes's theorem** – obtained from the product rule:

$$\begin{aligned} P(y|x, \mathcal{H}) &= \frac{P(x|y, \mathcal{H})P(y|\mathcal{H})}{P(x|\mathcal{H})} &(2.9)\\ &= \frac{P(x|y, \mathcal{H})P(y|\mathcal{H})}{\sum_{y'} P(x|y', \mathcal{H})P(y'|\mathcal{H})}. &(2.10)\end{aligned}$$

**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?

I said that we often define an ensemble in terms of a collection of conditional probabilities. The following example illustrates this idea.

Example 2.3: Jo has a test for a nasty disease. We denote Jo's state of health by the variable $a$ and the test result by $b$.

$$\begin{array}{ll} a=1 & \text{Jo has the disease} \\ a=0 & \text{Jo does not have the disease} \end{array} \tag{2.12}$$

The result of the test is either 'positive' ($b=1$) or 'negative' ($b=0$); the test is 95% reliable: in 95% of cases of people who really have the disease, a positive result is returned, and in 95% of cases of people who do not have the disease, a negative result is obtained. The final piece of background information is that 1% of people of Jo's age and background have the disease.

OK – Jo has the test, and the result was positive. What is the probability that Jo has the disease?

Solution:We write down all the provided probabilities. The test reliability specifies the conditional probability of $b$ given $a$:

$$\begin{array}{ll} P(b{=}1|a{=}1){=}0.95 & P(b{=}1|a{=}0){=}\,0.05 \\ P(b{=}0|a{=}1){=}0.05 & P(b{=}0|a{=}0){=}0.95; \end{array} \tag{2.13}$$

and the disease prevalence tells us about the marginal probability of $a$:

$$P(a{=}1){=}0.01 \quad P(a{=}0){=}0.99 \tag{2.14}$$

From the marginal $P(a)$ and the conditional probability $P(b|a)$ we can deduce the joint probability $P(a,b) = P(a)P(b|a)$ and any other probabilities we are interested in. For example, by the sum rule, the marginal probability of $b{=}1$ – the probability of getting a positive result – is

$$P(b{=}1) = P(b{=}1|a{=}1)P(a{=}1) + P(b{=}1|a{=}0)P(a{=}0). \tag{2.15}$$

Jo has received a positive result $b{=}1$ and is interested in how plausible it is that she has the disease (i.e., that $a=1$). The man in the street might be duped by the statement 'the test is 95% reliable, so Jo's positive result implies that there is a 95% chance that Jo has the disease', but this is incorrect. The correct solution to an inference problem is found using Bayes's theorem.

$$\begin{aligned} P(a{=}1|b{=}1) &= \frac{P(b{=}1|a{=}1)P(a{=}1)}{P(b{=}1|a{=}1)P(a{=}1) + P(b{=}1|a{=}0)P(a{=}0)} && (2.16) \\ &= \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.05 \times 0.99} && (2.17) \\ &= 0.16 && (2.18) \end{aligned}$$

So in spite of the positive result, the probability that Jo has the disease is only 16%. □

Exercise 2.4:$^{B2}$ Compare two ways of computing the probability of the repetition code $R_3$, assuming a binary symmetric channel (you did this once for exercise 1.1 (p.12)) and confirm that they give the same answer.

**Binomial distribution method** Add the probability of all three bits's being flipped to the probability of exactly two bits's being flipped.

**Sum rule method** Using the sum rule, compute the marginal probability that $\mathbf{r}$ takes on each of the eight possible values, $P(\mathbf{r})$. [$P(\mathbf{r}) = \sum_s P(s)P(\mathbf{r}|s)$.] Then compute the posterior probability of $s$ for each of the eight values of $\mathbf{r}$. [In fact, only two example cases $\mathbf{r} = (\mathtt{0}, \mathtt{0}, \mathtt{0})$ and $\mathbf{r} = (\mathtt{0}, \mathtt{0}, \mathtt{1})$ need be considered.] Notice that some of the inferred bits are better determined than others. From the posterior probability $P(s|\mathbf{r})$ you can read out the case-by-case error probability, the probability that the more probable hypothesis is not correct, $P(\text{error}|\mathbf{r})$. Find the average error probability using the sum rule,

Equation (1.1) gives the posterior probability of the input $s$, given the received vector $\mathbf{r}$.

$$P(\text{error}) = \sum_{\mathbf{r}} P(\mathbf{r})P(\text{error}|\mathbf{r}). \tag{2.19}$$

## 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 continental drift has turned New Guinea through 180 degrees relative to Australia'; or 'the probability that 2050 will be the warmest year on record, assuming people do not change their lifestyle'.

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 de-

**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

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

scribe only random variables. The big difference between the two approaches is that Bayesians also use probabilities to describe *inferences*.

## 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 2.5:$^{A2}$ 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 or expectation of some quantity that depends on the data. Here is another example of a forward probability problem:

Exercise 2.6:$^{A2}$ 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 2.5, obtaining $n_B$ blacks, and computes the quantity

$$z = \frac{(n_B - f_B N)^2}{N f_B (1 - f_B)}. \tag{2.20}$$

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 (2.21).

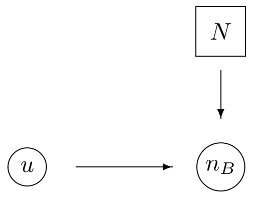


Figure 2.5. Graphical model for Bill and Fred's urn problem.

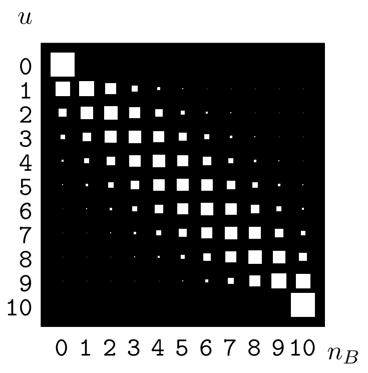


Figure 2.6. Joint probability of $u$ and $n_B$ for Bill and Fred's urn problem, after $N = 10$ draws.

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 distribution of some quantity *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's theorem.

Example 2.7: 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. If after $N = 10$ draws $n_B = 3$ blacks have been drawn, 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

$$P(u, n_B|N) = P(n_B|u, N)P(u). \tag{2.21}$$

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

$$P(u|n_B, N) = \frac{P(u, n_B|N)}{P(n_B|N)} \tag{2.22}$$

$$= \frac{P(n_B|u, N)P(u)}{P(n_B|N)}. \tag{2.23}$$

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 2.5 (p.32). [You *are* doing the exercises that are marked 'A', aren't you?] If we define $f_u \equiv u/10$ then

$$P(n_B|u, N) = \binom{N}{n_B} f_u^{n_B}(1 - f_u)^{N-n_B}. \tag{2.24}$$

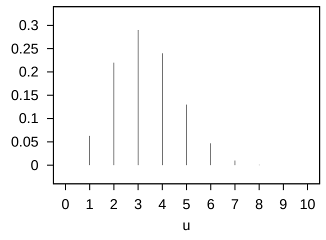
What about the denominator, $P(n_B|N)$? This is the marginal probability of $n_B$, which we can obtain using the sum rule:

$$P(n_B|N) = \sum_u P(u, n_B|N) = \sum_u P(u)P(n_B|u, N). \tag{2.25}$$

So the conditional probability of $u$ given $n_B$ is

$$P(u|n_B, N) = \frac{P(u)P(n_B|u, N)}{P(n_B|N)} \tag{2.26}$$

$$= \frac{1}{P(n_B|N)} \frac{1}{11} \binom{N}{n_B} f_u^{n_B}(1 - f_u)^{N-n_B}. \tag{2.27}$$



| $u$ | $P(u\|n_B = 3, N)$ |
|---|---|
| 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 $n_B = 3$ and $N = 10$.

This conditional distribution can be found by normalizing column 3 of figure 2.6 and is shown in figure 2.7. The normalizing constant, the marginal probability of $n_B$, is $P(n_B = 3|N = 10) = 0.083$. The posterior probability (2.27) is correct for all $u$, including the end-points $u = 0$ and $u = 10$, where $f_u = 0$ and $f_u = 1$ respectively. 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. □

## *Terminology of inverse probability*

In inverse probability problems it is convenient to give names to the probabilities appearing in Bayes's theorem. In equation (2.26), 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 $n_B$. For fixed $n_B$, $P(n_B|u, N)$ defines the *likelihood* of $u$.

Never say 'the likelihood of the data'. Always say 'the likelihood of the parameters'.
The likelihood function is not a probability distribution.

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 its value is not important if we simply wish to evaluate the relative probablities of the alternative hypotheses $u$. However, in more data modelling problems of any complexity, this quantity becomes important, and it is given various names: $P(n_B|N)$ is known as the *evidence* or the *marginal likelihood*.

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

$$P(\theta|D,\mathcal{H}) = \frac{P(D|\theta,\mathcal{H})P(\theta|\mathcal{H})}{P(D|\mathcal{H})} \tag{2.28}$$

is written:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}. \tag{2.29}$$

### *Inverse probability and prediction*

Example 2.7 (continued): Assuming again 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.]

Solution: By the sum rule,

$$P(\text{ball } N{+}1 \text{ is black}|n_B, N) = \sum_u P(\text{ball } N{+}1 \text{ is black}|u, n_B, N)P(u|n_B, N). \tag{2.30}$$

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

$$P(\text{ball } N{+}1 \text{ is black}|n_B, N) = \sum_u f_u P(u|n_B, N). \tag{2.31}$$

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

$$P(\text{ball } N{+}1 \text{ is black}|n_B{=}3, N{=}10) = 0.333. \tag{2.32}$$

Comment: Notice the difference between this prediction obtained using probability theory, and the widespread practice in statistics of making predictions by first selecting the most plausible hypothesis (which here would be that the urn is urn $u = 3$) and then making the predictions assuming that hypothesis to be true (which would give a probability of 0.3 for the next ball's being black). The correct prediction is the one that takes into account the uncertainty by *marginalizing* over the possible values of the hypothesis $u$. Marginalization here leads to slightly more moderate, less extreme predictions.

### *Inference as inverse probability*

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

Example 2.8: 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 2.7 (p.33), this problem has a subjective element. Given a restricted definition of probability that says 'probabilities are the frequencies of random variables', this example is 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 2.7 and 2.8 seem to have an essential similarity? [Especially when $N = 10$ and $n_H = 3$!]

To solve example 2.8, 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.*

Here $P(f)$ denotes a probability density, rather than a probability distribution.

Exercise 2.9:$^{B2}$ Assuming a uniform prior on $f_H$, $P(f_H) = 1$, solve the problem posed in example 2.8 (p.35). 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:

$$\int_0^1 dp_a \, p_a^{F_a}(1-p_a)^{F_b} = \frac{\Gamma(F_a+1)\Gamma(F_b+1)}{\Gamma(F_a+F_b+2)} = \frac{F_a!F_b!}{(F_a+F_b+1)!}. \tag{2.33}$$

You may also find it instructive to look back at example 2.7 (p.33) and equation (2.32).

By the way, we are now using $P()$ to denote probability *densities* over continuous variables as well as probabilities over discrete variables and probabilities of logical propositions. The probability that a continuous variable $v$ lies between values $a$ and $b$ (where $b > a$) is defined to be $\int_a^b \mathrm{d}v \, P(v)$. The density $P(v)$ is a dimensional quantity, having dimensions inverse to the dimensions of $v$ – in contrast to discrete probabilities, which are dimensionless. Conditional and joint probability densities are defined in just the same way as conditional and joint probabilities.

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 about $\frac{1}{2}$, 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

$$P(x_1 = \text{head}) = \int df_H \, P(f_H) P(x_1 = \text{head} | f_H) = \int df_H \, P(f_H) f_H = 1/2. \tag{2.34}$$

But the prediction for subsequent tosses will depend on the whole prior distribution, not just its mean.

*Data compression and inverse probability*

Consider the following task.

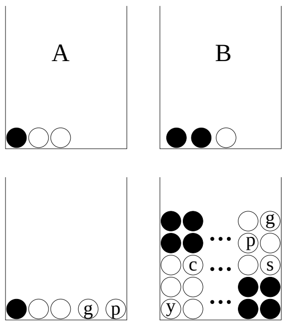
Example 2.10: Write a computer program capable of compressing binary files like this one:

```
0000000000000000000010010001000000100000010000000000000000000000000000000000001010000000000000110000
1000000000010000100000000010000000000000000000000100000000000000000100000000011000001000000011000100
0000000001001000000000001000100000000000000000110000000000000000000000000000010000000000000000100000000
```

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 of the file 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 2.8 (p.35)? 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 2.7, using inverse probability. Example 2.10 is solved in chapter 7.



*The likelihood principle*

Please solve the following two exercises.

Example 2.11: 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 2.12: 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 fifth of A's balls are black; two-fifths of B's are black.] 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? Does each answer depend on the detailed contents of each urn?

The details of the other possible outcomes and their probabilities are irrelevant. All that matters is the probability of the outcome that actually happened (here, that the ball drawn was black) given the different hypotheses. We need only to know the *likelihood*, i.e., how the probability of the data that happened varies with the hypothesis. This simple rule about inference is known as the *likelihood principle*. [And, in spite of the simplicity of this principle, many classical statistical methods violate it.]

The likelihood principle: given a generative model for data $d$ given parameters $\theta$, $P(d|\theta)$, and having observed a particular outcome $d_1$, all inferences and predictions should depend only on the function $P(d_1|\theta)$.

## 2.4 Definition of entropy and related functions

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

$$h(x) = \log_2 \frac{1}{P(x)}. \tag{2.35}$$

It is measured in bits. [The word 'bit' is also used to denote a variable whose value is 0 or 1; I hope context will always make clear which of the two meanings is intended.]

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 $x = \texttt{z}$ has a Shannon information content of 10.4 bits, and $x = \texttt{e}$ has an information content of 3.5 bits.

| $i$ | $a_i$ | $p_i$ | $\log_2 \frac{1}{p_i}$ |
|---|---|---|---|
| 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`.

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

$$H(X) \equiv \sum_{x \in \mathcal{A}_X} P(x) \log \frac{1}{P(x)}, \tag{2.36}$$

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

Like the information content, entropy is measured in bits.

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 $X$.

Example 2.13: The entropy of a randomly selected letter in an English document is about 4.11 bits, assuming its probability is as given in figure 2.8. 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:

$$H(X) \leq \log(|\mathcal{A}_X|) \text{ with equality iff } p_i{=}1/|X| \text{ for all } i. \tag{2.37}$$

Notation: the vertical bars '$|\cdot|$' have two meanings. $|\mathcal{A}_X|$ 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 *redundancy* measures the fractional difference between $H(X)$ and its maximum possible value, $\log(|\mathcal{A}_X|)$.

**The redundancy of** $X$ is:

$$1 - \frac{H(X)}{\log |\mathcal{A}_X|}. \tag{2.38}$$

We won't make use of 'redundancy' in this book, so I have not assigned a symbol to it – it would be redundant.

**The joint entropy of** $X, Y$ is:

$$H(X,Y) = \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x,y)}. \tag{2.39}$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \text{ iff } P(x,y) = P(x)P(y). \tag{2.40}$$

Our definitions for information content so far apply only 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 12.3. Further important definitions and exercises to do with entropy, will come along in section 9.1.

## 2.5 Gibbs's inequality

**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

$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}. \tag{2.41}$$

The relative entropy satisfies *Gibbs' inequality*

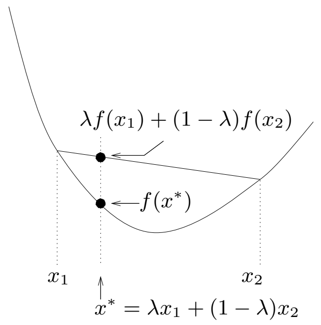
$$D_{\mathrm{KL}}(P||Q) \geq 0 \tag{2.42}$$

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

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

## 2.6 Jensen's inequality for convex functions

The words 'convex $\smile$' and 'concave $\frown$' may be pronounced 'convex-smile' and 'concave-frown'. This terminology has useful redundancy: while one may forget which way up 'convex' and 'concave' are, it is harder to confuse a smile with a frown.



$\lambda f(x_1) + (1-\lambda) f(x_2)$
$f(x^*)$
$x_1$
$x_2$
$x^* = \lambda x_1 + (1-\lambda)x_2$

Figure 2.9. Definition of convexity.

**Convex $\smile$ functions.** A function $f(x)$ is *convex* $\smile$ 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 $0 \leq \lambda \leq 1$,

$$f(\lambda x_1 + (1-\lambda)x_2) \ \leq \ \lambda f(x_1) + (1-\lambda) f(x_2). \tag{2.43}$$

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

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

Some strictly convex⌣ functions are

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

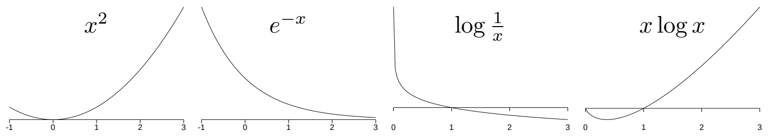


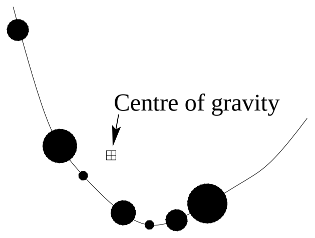
Figure 2.10. Convex⌣ functions.

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

$$\mathcal{E}\left[f(x)\right] \geq f(\mathcal{E}[x]), \tag{2.44}$$

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

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



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

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

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

Exercise 2.14:$^{C2}$ Prove Jensen's inequality.

Example 2.15: 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? [Use Jensen's inequality.]

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 $\mathcal{E}\left[x\right] = 10$ and $\mathcal{E}\left[f(x)\right] = 100$, where $f(x) = x^2$ is the function mapping lengths to areas. This is a strictly convex⌣ function. We notice that the equality $\mathcal{E}\left[f(x)\right] = f(\mathcal{E}[x])$ 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⌢ and there exists a point at which

$$\frac{\partial f}{\partial x_k} = 0 \text{ for all } k, \tag{2.45}$$

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

The converse does not hold: if a concave⌢ $f(\mathbf{x})$ 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 a probability $p \in (0, 1)$, is maximized on the boundary of the range, at $p = 1$, where $f'(p) = 1$.

## 2.7 Exercises

*Expectations and entropies*

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

$$\mathcal{E}\left[f(x)\right] = \langle f(x) \rangle = \sum_x P(x) f(x). \tag{2.46}$$

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.47

Exercise 2.16:$^{A1}$ 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 2.17:$^{A2}$ For an arbitrary ensemble, what is $\mathcal{E}\left[1/P(x)\right]$?

Exercise 2.18:$^{B2}$ 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 2.19:$^{B1}$ 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 2.20:$^{A3}$ 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 $\mathcal{A}_X$.) [Hint: use Jensen's inequality (2.44); if your first attempt to use Jensen does not succeed, remember that Jensen involves both a random variable and a function, and you have quite a lot of freedom in choosing these; think about whether your chosen function $f$ should be convex or concave; further hint: try $u = 1/p_i$ as the random variable.]

Exercise 2.21:$^{B3}$ The probability $p_n$ of the $n$th most frequent word in English is roughly approximated by

$$p_n \simeq \begin{cases} \frac{0.1}{n} & \text{for } n \in 1 \ldots 12367. \\ 0 & n > 12367. \end{cases} \tag{2.47}$$

[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 (per word)?

Solutions on p.48

Exercise 2.22:$^{B3}$ Prove that the relative entropy (equation (2.41)) satisfies $D_{\mathrm{KL}}(P||Q) \geq 0$ (Gibbs's inequality) with equality only if $P{=}Q$.

*Forward probability*

Exercise 2.23:$^{C2}$ 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 $\hat{f} \equiv h/(h+t)$, where $h$ and $t$ are the numbers of heads and tails tossed. Compute and sketch the probability distribution of $\hat{f}$.

Exercise 2.24:$^{B2}$ 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 bus-stop 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. There, as you can confirm, 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 2.25:$^{A3}$ (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 orindary 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.50

Exercise 2.26:$^{A2}$ If $q = 1 - p$ and $a = \log p/q$, show that

$$p = \frac{1}{1 + \exp(-a)}. \tag{2.48}$$

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

Exercise 2.27:$^{B2}$ Let $x$ and $y$ be correlated random variables with $x$ a binary variable taking values in $\mathcal{A}_X = \{0, 1\}$. Use Bayes's theorem to show that the log posterior probability ratio for $x$ given $y$ is

$$\log \frac{P(x=1|y)}{P(x=0|y)} = \log \frac{P(y|x=1)}{P(y|x=0)} + \log \frac{P(x=1)}{P(x=0)}. \tag{2.49}$$

Exercise 2.28:$^{B2}$ 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's theorem to show that the posterior probability ratio for $x$ given $\{d_i\}$ is

$$\frac{P(x=1|\{d_i\})}{P(x=0|\{d_i\})} = \frac{P(d_1|x=1)}{P(d_1|x=0)}\frac{P(d_2|x=1)}{P(d_2|x=0)}\frac{P(x=1)}{P(x=0)}. \tag{2.50}$$

## *Life in high-dimensional spaces*

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

Exercise 2.29:$^{A2}$ Consider a sphere of radius $r$ in an $N$-dimensional real space. Show that the fraction of the volume of the sphere that is in in the surface shell lying at values of the radius between $r-\epsilon$ and $r$, where $0<\epsilon<r$, is:

$$f = 1 - \left(1 - \frac{\epsilon}{r}\right)^N . \tag{2.51}$$

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 5.

Solutions on p.51

Exercise 2.30:$^{B2}$ 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:

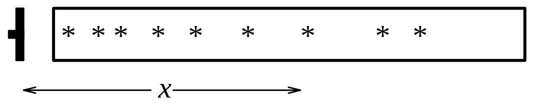
$$5, 3, 9, 3, 8, 4, 7.$$

What is the probability that the die is die A?

Exercise 2.31:$^{B2}$ 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 2.32:$^{A3}$ 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 $\lambda$. Decay events can only be observed if they occur in a window extending from $x = 1\,\text{cm}$ to $x = 20\,\text{cm}$. $N$ decays are observed at locations $\{x_1, \ldots, x_N\}$. What is $\lambda$?



Exercise 2.33:$^{B2}$ You move into a new house; the phone is connected, and you're pretty sure that the phone number is `740511`, but not as sure as you would like to be. As an experiment, you pick up the phone and dial `740511`; you obtain a 'busy' signal. Are you now more sure of your phone number? If so, how much?

Exercise 2.34:$^{B3}$ Forensic evidence

Two people have left traces of their own blood at the scene of a crime. A suspect, Oliver, 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 Oliver was one of the two people present at the crime?

# *Solutions to Chapter 2's exercises*

Solution to exercise 2.2 (p.30): No, they are not independent. If they were then all the conditional distributions $P(y|x)$ would be identical functions of $y$, regardless of $x$ (*c.f.* figure 2.3).

Solution to exercise 2.4 (p.31):

**Binomial distribution method** From the solution to exercise 1.1, $p_B = 3f^2(1-f) + f^3$.

**The sum rule method** The marginal probabilities of the eight values of $\mathbf{r}$ are illustrated by:

$$P(\mathbf{r}{=}\mathtt{0,0,0}) = {}^1\!/\!_2(1-f)^3 + {}^1\!/\!_2 f^3, \tag{2.52}$$

$$P(\mathbf{r}{=}\mathtt{0,0,1}) = {}^1\!/\!_2 f(1-f)^2 + {}^1\!/\!_2 f^2(1-f) = {}^1\!/\!_2 f(1-f). \tag{2.53}$$

The posterior probabilities are represented by

$$P(s{=}\mathtt{1}|\mathbf{r}{=}\mathtt{0,0,0}) = \frac{f^3}{(1-f)^3 + f^3} \tag{2.54}$$

and

$$P(s{=}\mathtt{1}|\mathbf{r}{=}\mathtt{0,0,1}) = \frac{(1-f)f^2}{f(1-f)^2 + f^2(1-f)} = f. \tag{2.55}$$

The probabilities of error in these representative cases are thus

$$P(\text{error}|\mathbf{r}{=}\mathtt{0,0,0}) = \frac{f^3}{(1-f)^3 + f^3} \tag{2.56}$$

and

$$P(\text{error}|\mathbf{r}{=}\mathtt{0,0,1}) = f. \tag{2.57}$$

Notice that while the average probability of error of $\mathrm{R}_3$ is about $3f^2$, the probability that any *particular* bit is wrong is either about $f^3$ or $f$.

The average error probability, using the sum rule, is

$$\begin{aligned} P(\text{error}) &= \sum_{\mathbf{r}} P(\mathbf{r})P(\text{error}|\mathbf{r}) \\ &= 2[{}^1\!/\!_2(1-f)^3 + {}^1\!/\!_2 f^3]\frac{f^3}{(1-f)^3+f^3} + 6[{}^1\!/\!_2 f(1-f)]f. \end{aligned}$$

So

$$P(\text{error}) = f^3 + 3f^2(1-f).$$

The first two terms are for the cases $\mathbf{r} = \mathtt{000}$ and $\mathtt{111}$; the remaining 6 are for the other outcomes, which share the same probability of occuring and identical error probability, $f$.

Solution to exercise 2.5 (p.32): We define the fraction $f_B \equiv B/K$.

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

$$P(n_B \,|\, f_B, N) = \binom{N}{n_B} f_B^{n_B}(1-f_B)^{N-n_B} \tag{2.58}$$

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

$$\mathcal{E}[n_B] = N f_B \tag{2.59}$$

$$\text{var}[n_B] = N f_B(1-f_B). \tag{2.60}$$

These results were derived in example 0.1 (p.6). The standard deviation of $n_B$ is $\sqrt{\text{var}[n_B]} = \sqrt{N f_B(1-f_B)}$.

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 2.6 (p.32): The numerator of the quantity

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

can be recognised as $(n_B - \mathcal{E}[n_B])^2$; the denominator is equal to the variance of $n_B$ (2.60), which is by definition the expectation of the numerator. So the expectation of $z$ is 1. [A random variable like $z$, which measures the deviation of data from the average value, is sometimes called $\chi^2$ (chi-squared).]

In the case $N = 5$ and $f_B = 1/5$, $N f_B$ is 1, and 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 2.14 (p.40): We wish to prove, given the property

$$f(\lambda x_1 + (1-\lambda)x_2) \;\leq\; \lambda f(x_1) + (1-\lambda) f(x_2), \tag{2.61}$$

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

$$\sum_{i=1}^{I} p_i f(x_i) \geq f\left(\sum_{i=1}^{I} p_i x_i\right). \tag{2.62}$$

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 (2.61) 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.

$$\begin{aligned} f\left(\sum_{i=1}^{I} p_i x_i\right) &= f\left(p_1 x_1 + \sum_{i=2}^{I} p_i x_i\right) \\ &\leq\; p_1 f(x_1) + \left[\sum_{i=2}^{I} p_i\right]\left[f\left(\sum_{i=2}^{I} p_i x_i \Big/ \sum_{i=2}^{I} p_i\right)\right] \\ &\leq\; p_1 f(x_1) + \left[\sum_{i=2}^{I} p_i\right]\left[\frac{p_2}{\sum_{i=2}^{I} p_i} f(x_2) + \frac{\sum_{i=3}^{I} p_i}{\sum_{i=2}^{I} p_i} f\left(\sum_{i=3}^{I} p_i x_i \Big/ \sum_{i=3}^{I} p_i\right)\right], \end{aligned} \tag{2.63}$$

and so forth. □

Solution to exercise 2.16 (p.41): $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. $f(a) = 10$, $f(b) = 5$, and $f(c) = 10/7$.

$$\mathcal{E}\left[f(x)\right] = 0.1 \times 10 + 0.2 \times 5 + 0.7 \times 10/7 = 3. \tag{2.64}$$

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

$$\mathcal{E}\left[1/P(x)\right] = \mathcal{E}\left[f(x)\right] = 3. \tag{2.65}$$

Solution to exercise 2.17 (p.41): For general $X$,

$$\mathcal{E}\left[1/P(x)\right] = \sum_{x \in \mathcal{A}_X} P(x) 1/P(x) = \sum_{x \in \mathcal{A}_X} 1 = |\mathcal{A}_X|. \tag{2.66}$$

Solution to exercise 2.18 (p.41): $p_a = 0.1$, $p_b = 0.2$, $p_c = 0.7$. $g(a) = 0$, $g(b) = 1$, and $g(c) = 0$.

$$\mathcal{E}\left[g(x)\right] = p_b = 0.2. \tag{2.67}$$

Solution to exercise 2.19 (p.41):

$$P\left(P(x) \in [0.15, 0.5]\right) = p_b = 0.2. \tag{2.68}$$

$$P\left(\left|\log \frac{P(x)}{0.2}\right| > 0.05\right) = p_a + p_c = 0.8. \tag{2.69}$$

Solution to exercise 2.20 (p.41): 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, 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 (not the recommended method):

$$H(X) = \sum_i p_i \log \frac{1}{p_i} \tag{2.70}$$

$$\frac{\partial H(X)}{\partial p_i} = \log \frac{1}{p_i} - 1 \tag{2.71}$$

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

$$G(\mathbf{p}) \equiv H(X) + \lambda \left(\sum_i p_i - 1\right) \tag{2.72}$$

$$\frac{\partial G(\mathbf{p})}{\partial p_i} = \log \frac{1}{p_i} - 1 + \lambda. \tag{2.73}$$

At a maximum,

$$\log \frac{1}{p_i} - 1 + \lambda = 0 \tag{2.74}$$

$$\Rightarrow \log \frac{1}{p_i} = 1 - \lambda, \tag{2.75}$$

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

$$\frac{\partial^2 G(\mathbf{p})}{\partial p_i \partial p_j} = -\frac{1}{p_i}\delta_{ij}, \tag{2.76}$$

which is negative definite. □

Proof using Jensen's inequality (recommended method): First a reminder of the inequality.

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

$$\mathcal{E}\left[f(x)\right] \geq f\left(\mathcal{E}[x]\right). \tag{2.77}$$

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

$$f(u) = \log\frac{1}{u} = -\log u \tag{2.78}$$

(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

$$u = 1/P(x) \tag{2.79}$$

then we find:

$$H(X) = -\mathcal{E}\left[f(1/P(x))\right] \leq -f\left(\mathcal{E}[1/P(x)]\right); \tag{2.80}$$

now we know from exercise 2.17 (p.41) that $\mathcal{E}[1/P(x)] = |\mathcal{A}_X|$, so

$$H(X) \leq -f\left(|\mathcal{A}_X|\right) = \log|\mathcal{A}_X|. \tag{2.81}$$

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 2.21 (p.41): The entropy is 9.7 bits per word.

Solution to exercise 2.22 (p.41):

$$D_{\rm KL}(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}. \tag{2.82}$$

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

$$\begin{aligned} D_{\rm KL}(P||Q) &= \mathcal{E}[f(Q(x)/P(x))] & (2.83)\\ &\geq f\left(\sum_x P(x)\frac{Q(x)}{P(x)}\right) = \log\left(\frac{1}{\sum_x Q(x)}\right) = 0, & (2.84)\end{aligned}$$

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

$$D_{\mathrm{KL}}(P||Q) = \sum_x Q(x) \frac{P(x)}{Q(x)} \log \frac{P(x)}{Q(x)} = \sum_x Q(x) f\left(\frac{P(x)}{Q(x)}\right) \tag{2.85}$$

$$\geq f\left(\sum_x Q(x) \frac{P(x)}{Q(x)}\right) = f(1) = 0, \tag{2.86}$$

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

Solution to exercise 2.23 (p.41): The probability of the number of tails $t$ is

$$P(t) = \left(\frac{1}{2}\right)^t \frac{1}{2} \text{ for } t \geq 0. \tag{2.87}$$

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

$$\mathcal{E}[t] = \sum_{t=0}^{\infty} t \left(\frac{1}{2}\right)^t \frac{1}{2}, \tag{2.88}$$

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

$$\mathcal{E}[t] = \frac{1}{2}(1 + \mathcal{E}[t]) + \frac{1}{2} 0 \;\Rightarrow\; \mathcal{E}[t] = 1. \tag{2.89}$$

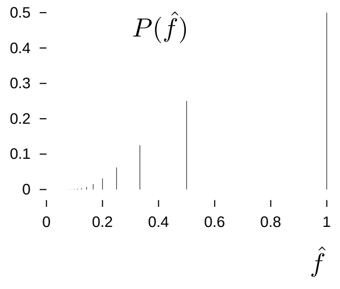


Figure 2.11. The probability distribution of the estimator $\hat{f} = 1/(1+t)$, given that $f = 1/2$.

The probability distribution of the 'estimator' $\hat{f} = 1/(1+t)$, given that $f = 1/2$, is plotted in figure 2.11. The probability of $\hat{f}$ simply the probability of the corresponding value of $t$.

Solution to exercise 2.24 (p.42):

(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 that the next six occurs on the $r$th roll is the probability of *not* getting a six for $r-1$ rolls multiplied by the probability of then getting a six:

$$P(r_1 = r) = \left(\frac{5}{6}\right)^{r-1} \frac{1}{6}. \tag{2.90}$$

This probability distribution of the number of rolls, $r$, may be called an exponential distribution, since

$$P(r_1 = r) = e^{-\alpha r}/Z, \tag{2.91}$$

where $\alpha = \ln(6/5)$.

(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?

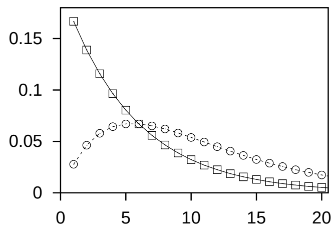


Figure 2.12. The probability distribution of the number of rolls $r_1$ from one 6 to the next (solid line),

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

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 of $r_{\rm tot}$ is 11.

Solution to exercise 2.25 (p.42):

(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 ordinary die and one with faces 6,6,6,0,0,0.

## *Inference problems*

Solution to exercise 2.26 (p.42):

$$a = \log \frac{p}{q} \tag{2.92}$$

$$\Rightarrow \quad \frac{p}{q} = e^a \tag{2.93}$$

And $q = 1 - p$ gives

$$\frac{p}{1-p} = e^a \tag{2.94}$$

$$\Rightarrow p = \frac{e^a}{e^a + 1} = \frac{1}{1 + \exp(-a)}. \tag{2.95}$$

The hyperbolic tangent is

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{2.96}$$

so

$$\begin{aligned} f(a) &\equiv \frac{1}{1+\exp(-a)} = \frac{1}{2}\left(\frac{1-e^{-a}}{1+e^{-a}} + 1\right) \\ &= \frac{1}{2}\left(\frac{e^{a/2} - e^{-a/2}}{e^{a/2} + e^{-a/2}} + 1\right) = \frac{1}{2}(\tanh(a/2) + 1). \end{aligned} \tag{2.97}$$

Solution to exercise 2.27 (p.42):

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.98}$$

$$\Rightarrow \frac{P(x=1|y)}{P(x=0|y)} = \frac{P(y|x=1)}{P(y|x=0)}\frac{P(x=1)}{P(x=0)} \tag{2.99}$$

$$\Rightarrow \log\frac{P(x=1|y)}{P(x=0|y)} = \log\frac{P(y|x=1)}{P(y|x=0)} + \log\frac{P(x=1)}{P(x=0)}. \tag{2.100}$$

Solution to exercise 2.28 (p.43): The conditional independence of $d_1$ and $d_2$ given $x$ means

$$P(x, d_1, d_2) = P(x)P(d_1|x)P(d_2|x). \tag{2.101}$$

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

$$\frac{P(x=1|\{d_i\})}{P(x=0|\{d_i\})} = \frac{P(\{d_i\}|x=1)}{P(\{d_i\}|x=0)}\frac{P(x=1)}{P(x=0)} \tag{2.102}$$

$$= \frac{P(d_1|x=1)}{P(d_1|x=0)}\frac{P(d_2|x=1)}{P(d_2|x=0)}\frac{P(x=1)}{P(x=0)}. \tag{2.103}$$

## *Life in high-dimensional spaces*

Solution to exercise 2.29 (p.43): The volume of a hypersphere of radius $r$ in $N$ dimensions is

$$V(r, N) = \frac{\pi^{N/2}}{(N/2)!}r^N. \tag{2.104}$$

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

$$\frac{r^N - (r-\epsilon)^N}{r^N} = 1 - \left(1 - \frac{\epsilon}{r}\right)^N. \tag{2.105}$$

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 $\epsilon$ is, for large enough $N$ essentially all the probability mass is in the surface shell of thickness $\epsilon$.

## *Solutions to further inference problems*

Solution to exercise 2.30 (p.43): Let the data be $D$. Assuming equal prior probabilities,

$$\frac{P(A|D)}{P(B|D)} = \frac{1}{2}\frac{3}{2}\frac{1}{1}\frac{3}{2}\frac{1}{2}\frac{2}{2}\frac{1}{2} = 9/32. \tag{2.106}$$

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

Solution to exercise 2.31 (p.43): The probability of the data given each hypothesis is:

$$P(D|A) = \frac{3}{20}\frac{1}{20}\frac{2}{20}\frac{1}{20}\frac{3}{20}\frac{1}{20}\frac{1}{20} = \frac{18}{20^7}; \quad (2.107)$$

$$P(D|B) = \frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{2}{20}\frac{1}{20}\frac{2}{20} = \frac{64}{20^7}; \quad (2.108)$$

$$P(D|C) = \frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20}\frac{1}{20} = \frac{1}{20^7}. \quad (2.109)$$

So

$$P(A|D) = \frac{18}{18+64+1} = \frac{18}{83}; \qquad P(B|D) = \frac{64}{83}; \qquad P(C|D) = \frac{1}{83}. \quad (2.110)$$

Solution to exercise 2.33 (p.44): There are two hypotheses. $\mathcal{H}_0$: your number is `740511`; $\mathcal{H}_1$: it is another number. The data, $D$, are 'when I dialed `740511`, I got a busy signal'. What is the probability of $D$, given each hypothesis? If your number is `740511`, then we expect a busy signal with certainty:

$$P(D|\mathcal{H}_0) = 1.$$

On the other hand, if $\mathcal{H}_1$ is true, then the probability that the number dialled returns a busy signal is smaller than 1, since various other outcomes were also possible (a ringing tone, or a number-unobtainable signal, for example). The value of this probability $P(D|\mathcal{H}_1)$ will depend on the probability $\alpha$ that a random phone number similar to your own phone number would be a valid phone number, and on the probability $\beta$ that you get a busy signal when you dial a valid phone number.

I estimate from the size of my phone book that Cambridge has about 75,000 valid phone numbers, all of length six digits. The probability that a random six-digit number is valid is therefore about $75,000/10^6 = 0.075$. If we exclude numbers beginning with 0, 1, and 9 from the random choice, the probability $\alpha$ is about $75,000/700,000 \simeq 0.1$. If we assume that telephone numbers are clustered then a misremembered number might be more likely to be valid than a randomly chosen number; so the probability, $\alpha$, that our guessed number would be valid, assuming $\mathcal{H}_1$ is true, might be bigger than 0.1. Anyway, $\alpha$ must be somewhere between 0.1 and 1. We can carry forward this uncertainty in the probability and see how much it matters at the end.

The probability $\beta$ that you get a busy signal when you dial a valid phone number is equal to the fraction of phones you think are in use or off-the-hook when you make your tentative call. This fraction varies from town to town and with the time of day. In Cambridge, during the day, I would guess that about 1% of phones are in use. At 4am, maybe 0.1%, or fewer.

The probability $P(D|\mathcal{H}_1)$ is the product of $\alpha$ and $\beta$, that is, about $0.1 \times 0.01 = 10^{-3}$. According to our estimates, there's about a one-in-a-thousand chance of getting a busy signal when you dial a random number; or one-in-a-hundred, if valid numbers are strongly clustered; or one-in-$10^4$, if you dial in the wee hours.

What does the data do to your beliefs about your phone number? The posterior probability ratio is the likelihood ratio times the prior probability ratio:

$$\frac{P(\mathcal{H}_0|D)}{P(\mathcal{H}_1|D)} = \frac{P(D|\mathcal{H}_0)}{P(D|\mathcal{H}_1)}\frac{P(\mathcal{H}_0)}{P(\mathcal{H}_1)} \quad (2.111)$$

The likelihood ratio is about 100-to-1 or 1000-to-1, so the posterior probability ratio is swung by a factor of 100 or 1000 in favour of $\mathcal{H}_0$. If the prior probability of $\mathcal{H}_0$ was 0.5 then the posterior probability is

$$P(\mathcal{H}_0|D) = \frac{1}{1 + \frac{P(\mathcal{H}_1|D)}{P(\mathcal{H}_0|D)}} \simeq 0.99 \text{ or } 0.999. \tag{2.112}$$

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 to chapter 5. 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 7.

Before reading chapter 3, you should have worked on exercises 2.26–2.32 (pp. 42–44).

# 3

# *More about Inference*

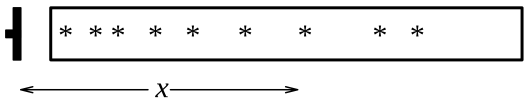
It is not a controversial statement that Bayes's theorem provides the correct language for describing the inference of a message communicated over a noisy channel, as we used it in chapter 1 (p. 11). But strangely, when it comes to other inference problems, the use of Bayes's theorem is not so widespread.

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 2.32):

> 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 $\lambda$. Decay events can only be observed if they occur in a window extending from $x = 1\,\text{cm}$ to $x = 20\,\text{cm}$. $N$ decays are observed at locations $\{x_1, \ldots, x_N\}$. What is $\lambda$?



I had scratched my head over this for some time. My education had provided me with a couple of approaches to solving such inference problems: contructing 'estimators' of the unknown parameters; or 'fitting' the model to the data, or a processed version of the data.

Since the mean of an unconstrained exponential distribution is $\lambda$, it seemed reasonable to examine the sample mean $\bar{x} = \sum_n x_n/N$ and see if an estimator $\hat{\lambda}$ could be obtained from it. It was evident that the estimator $\hat{\lambda} = \bar{x} - 1$ would be appropriate for $\lambda \ll 20\,\text{cm}$, 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

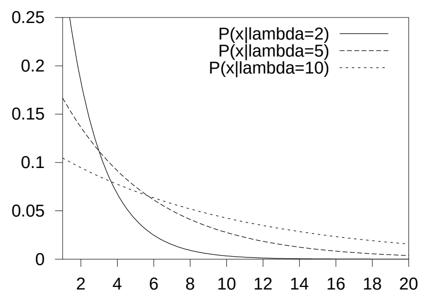


Figure 3.1. The probability density $P(x|\lambda)$ as a function of $x$.

constructed. But there was no obvious estimator that would work under all conditions.

Nor could I find a satisfactory approach based on fitting the density $P(x|\lambda)$ to a histogram derived from the data. I was stuck.

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

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 *Bayes's theorem*:

$$\begin{aligned} P(\lambda|\{x_1, \ldots, x_N\}) &= \frac{P(\{x\}|\lambda)P(\lambda)}{P(\{x\})} & (3.3) \\ &\propto \frac{1}{\left(\lambda Z(\lambda)\right)^N} \exp\left( -\textstyle\sum_1^N x_n/\lambda \right) P(\lambda). & (3.4) \end{aligned}$$

Suddenly, the straightforward distribution $P(\{x_1, \ldots, x_N\}|\lambda)$, defining the probability of the data given the hypothesis $\lambda$, 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 $\lambda$ (figure 3.1). Each curve was an innocent exponential, normalized to have area 1. Plotting the same function as a function of $\lambda$ for a fixed value of $x$, something remarkable happened: a peak emerges (figure 3.2). To help understand these two points of view of the one function, figure 3.3 shows a surface plot of $P(x|\lambda)$ as a function of $x$ and $\lambda$.

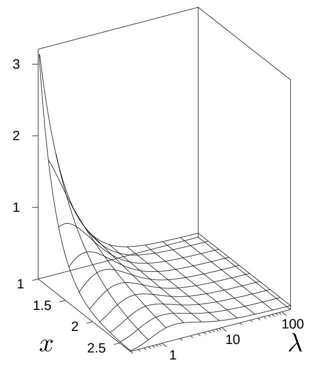


Figure 3.3. The probability density $P(x|\lambda)$ as a function of $x$ and $\lambda$.

For a dataset consisting of several points, e.g., the six points $\{x\}_{n=1}^N = \{1.5, 2, 3, 4, 5, 12\}$, the likelihood function $P(\{x\}|\lambda)$ is the product of the $N$ functions of $\lambda$, $P(x_n|\lambda)$ (figure 3.4).

[If you have any difficulty understanding this chapter I recommend ensuring you are happy with exercises 2.30 and 2.31 (p. 43) then noting their similarity to exercise 2.32.]

Steve summarised Bayes's theorem as embodying the fact that what you know about $\lambda$ 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 $\lambda$ which correctly describes the situation is *not* a stochastic variable, and the fact that the Bayesian uses a probability distribution $P$ does *not*

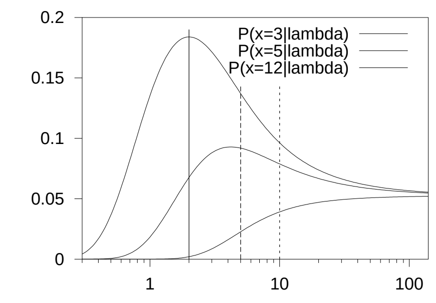


Figure 3.2. The probability density $P(x|\lambda)$ as a function of $\lambda$, for three different values of $x$. When plotted this way round, the function is known as the *likelihood* of $\lambda$. The marks indicate the three values of $\lambda$, $\lambda = 2, 5, 10$, that were used in the preceding figure.

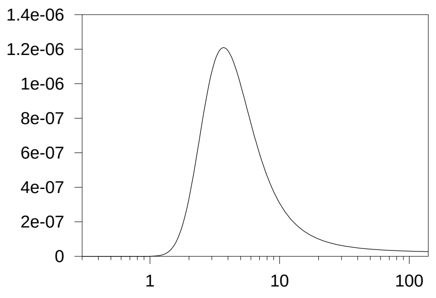


Figure 3.4. The likelihood function in the case of a six-point dataset, $P(\{x\} = \{1.5, 2, 3, 4, 5, 12\}|\lambda)$, as a function of $\lambda$.

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 $\lambda$), of which only one is actually true. That probabilities can denote degrees of belief, given assumptions, seemed intuitive to me.

The posterior probability distribution (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.

## *Assumptions in inference*

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 $\lambda$:

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

Second, when the assumptions are explicit, they are easier to criticize, and easier to modify – indeed, we can quantify the sensitivity of our inferences to the details of the assumptions. For example, 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 $x = 3$; the details of the prior $P(\lambda)$ become increasingly 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 $\lambda$, and don't give any information about the relative probabilities of large values of $\lambda$. So in this case, the details of the prior at the small $\lambda$ end of things are not important, but at the large $\lambda$ end, the 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's 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 at some 'significance level', 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 (which we'll denote by the symbols $a$ and $b$). 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 2.8 (p.35), and we will encounter it again in chapter 7, when we discuss adaptive data compression. It is also the original inference problem studied by Thomas Bayes in his essay published in 1763.

As in exercise 2.9 (p.36), 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

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

[For example, $P(\mathbf{s}=aaba|p_a, F=4, \mathcal{H}_1) = p_a p_a (1-p_a) p_a$.] This function of $p_a$ (3.8) defines the likelihood function. 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 $a$s and $F_b$ are $b$s 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's 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, $\mathcal{H}_1$, 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 as inference*

In order to perform model comparison, we write down Bayes's 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 total probability of getting the observed data. If $\mathcal{H}_1$ and $\mathcal{H}_0$ are the only models under consideration, this probability is given by the sum rule:

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

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 equation (3.10) where it appeared as the normalizing constant of the first inference we made – the inference of $p_a$ given the data.

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

We evaluated the normalizing constant for model $\mathcal{H}_1$ in (3.12). 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 some outcomes favour one model, and some 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, $\mathcal{H}_0$, 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.

| $F$ | Data $(F_a, F_b)$ | $\dfrac{P(\mathcal{H}_1 \mid \mathbf{s}, F)}{P(\mathcal{H}_0 \mid \mathbf{s}, F)}$ | |
|---|---|---|---|
| 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 $p_a = 1/6, p_b = 5/6$.

## 3.3 An example of legal evidence

The following example (exercise 2.34) 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, Oliver, 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 Oliver 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 with an equally carefully reasoned prior probability. [This is the view of many statisticians but learned British judges have recently disagreed and actually overturned the verdict of a trial, on appeal, because the jurors *had* been taught to use Bayes's 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:

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

(since given $S$, we already know that one trace will be of type O). The probability of the data given $\bar{S}$ is the probability that two unknown people drawn from the population have types O and AB:

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

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:

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

Thus the data in fact provide weak evidence *against* the supposition that Oliver 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, Alberto, 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:

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

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 Oliver, a suspect of type O and Alberto, a suspect of type AB. Intuitively, we still believe that the presence of the rare AB blood provides positive evidence that Alberto was there. But does the fact that type O blood was detected at the scene favour the hypothesis that Oliver 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 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, since we know that 10 out of the 100 suspects were present. 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, 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 $n_{\rm O}$ blood stains of individuals of type $O$ are found, and $n_{\rm AB}$ of type AB, a total of $N$ individuals in all, and unknown people come from a large population with fractions $p_{\rm O}, p_{\rm AB}$. (There may be other blood types too.) The task is to evaluate the likelihood ratio for the two hypotheses: $S$, 'the type O suspect (Oliver) 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 $n_{\rm O}, n_{\rm AB}$ individuals of the two types when $N$ individuals are drawn at random from the population:

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

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

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

The likelihood ratio is:

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

This is an instructive result. The likelihood ratio, i.e. the contribution of these data to the question of whether Oliver was present, depends simply on a comparison of the frequency of his blood type in the observed data with the background frequency 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 number expected under hypothesis $\bar{S}$, then the data give 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 $n_{\rm O}/N = p_{\rm O}$, 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:$^{B2}$ Another example in which the emphasis is not on priors. You visit a family whose three children are all at the local school. You don't know anything about the sexes of the children. While walking clumsily round the home, you stumble through one of the three unlabelled bedroom doors that you know belong, one each, to the three children, and find that the bedroom contains girlie stuff in sufficient quantities to convince you that the child who lives in that bedroom is a girl. Later, you sneak a look at a letter addressed to the parents, which reads "From the Headmaster: we are sending this letter to all parents who have male children at the school to inform them about the following boyish matters. . . ".

These two sources of evidence establish that at least one of the children is a girl, and that at least one of the children is a boy. What are the probabilities that there are (a) two girls and one boy; (b) two boys and one girl?

Exercise 3.2:$^{B2}$ 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 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 slimy trickster? If the latter, what is wrong with his argument?

Exercise 3.3:$^{A2}$ 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?

Exercise 3.4:$^{A2}$ 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 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.

[A similar alternative scenario is a gameshow whose *confused host* forgets the rules, and where the prize is, and opens one of the unchosen doors at random. He opens door 3, and the prize is not revealed. Does the optimal decision for the contestant depend on the contestant's beliefs about whether the gameshow host is confused or not?]

[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, 1,326 women (28%) were slain by husbands and boyfriends.
(Sources: `http://www.umn.edu/mincava/papers/factoid.htm`, `http://www.gunfree.inter.net/vpc/womenfs.htm`)

Exercise 3.5:$^{B2}$ A statistical statement appeared in *The Guardian* on Friday January 4, 2002:

> When spun on edge 250 times, a Belgian one-euro coin came up heads 140 times and tails 110. “It looks very suspicious to me,” said Barry Blight, a statistics lecturer at the London School of Economics. “If the coin were unbiased the chance of getting a result as extreme as that would be less than 7%.”

But *do* these data give evidence that the coin is biased rather than fair? [Hint: see equation (3.18).]

# *Solutions to Chapter 3's exercises*

Solution to exercise 3.2 (p.63): The statistic quoted indicates the probability that a randomly selected wife-beater will also 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 further 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 64. Let $m = 1$ denote the proposition that a woman has been murdered; $h = 1$, the proposition that the husband did it; and $b = 1$, the proposition that he beat her in the year preceding the murder. The statement 'someone else did it' is denoted by $h = 0$. We need to define $P(h|m = 1)$, $P(b|h = 1, m = 1)$, and $P(b = 1|h = 0, m = 1)$ in order to compute the posterior probability $P(h = 1|b = 1, m = 1)$. From the statistics, we can read out $P(h = 1|m = 1) = 0.28$. And if two million women out of 100 million are beaten, then $P(b = 1|h = 0, m = 1) = 0.02$. Finally, we need a value for $P(b|h = 1, m = 1)$: 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 $P(b = 1|h = 1, m = 1)$ is 0.9 or larger.

By Bayes's theorem, then,

$$P(h = 1|b = 1, m = 1) = \frac{.9 \times .28}{.9 \times .28 + .02 \times .72} \simeq 95\%. \tag{3.26}$$

One way to make obvious the dishonesty of the slimy 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 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 3.3 (p.64): 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.,

$$P(\mathcal{H}_1) = P(\mathcal{H}_2) = P(\mathcal{H}_3) = \frac{1}{3}. \tag{3.27}$$

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.

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

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

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

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

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

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

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 thirty times with a friend and keep track of the frequency with which switching gets the prize. Alternatively, you 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 *contestant's* 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 3.4 (p.64): 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 section 2.3.] 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, these two quantities have to be equal, by symmetry. 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:

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

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.

Solution to exercise 3.5 (p.65): We compare the models $\mathcal{H}_0$ – the coin is fair – and $\mathcal{H}_1$ – the coin is biased, with the prior on its bias set to the uniform distribution $P(p|\mathcal{H}_1) = 1$. [The use of a uniform prior seems reasonable to me, since I know that some coins, such as American pennies, have severe biases when spun on edge.] The likelihood ratio is:

$$\frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_0)} = \frac{\frac{140!110!}{251!}}{1/2^{250}} = 0.48. \tag{3.33}$$

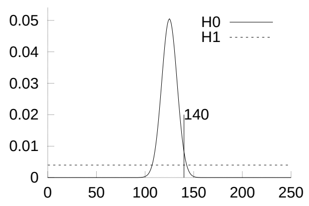


Figure 3.5. The probability distribution of the number of heads given the two hypotheses, that the coin is fair, and that it is biased, with the prior distribution of the bias being uniform. The outcome ($D = 140$ heads) gives weak evidence in favour of $\mathcal{H}_0$, the hypothesis that the coin is fair.

Thus the data give scarcely any evidence either way; in fact they give weak evidence (two to one) in favour of $\mathcal{H}_0$!

'No, no', objects the believer in bias, 'your silly uniform prior doesn't represent *my* prior beliefs about the bias of biased coins – I was *expecting* a small bias'. To be as generous as possible to the $\mathcal{H}_1$, let's see how well it could fare if the prior were presciently set. Let us allow a prior of the form

$$P(p|\mathcal{H}_1, \alpha) = \frac{1}{Z(\alpha)} p^{\alpha-1}(1-p)^{\alpha-1}, \quad \text{where } Z(\alpha) = \Gamma(\alpha)^2/\Gamma(2\alpha) \tag{3.34}$$

(a Beta distribution, with the original uniform prior reproduced by setting $\alpha = 1$). By tweaking $\alpha$, the likelihood ratio for $\mathcal{H}_1$ over $\mathcal{H}_0$,

$$\frac{P(D|\mathcal{H}_1, \alpha)}{P(D|\mathcal{H}_0)} = \frac{\Gamma(140+\alpha)\,\Gamma(110+\alpha)\,\Gamma(2\alpha)2^{250}}{\Gamma(250+2\alpha)\,\Gamma(\alpha)^2}, \tag{3.35}$$

can be increased a little. It is shown for several values of $\alpha$ in figure 3.6. Even the most favourable choice of $\alpha$ ($\alpha \simeq 50$) can yield a likelihood ratio of only two to one in favour of $\mathcal{H}_1$.

| $\alpha$ | $\frac{P(D\|\mathcal{H}_1, \alpha)}{P(D\|\mathcal{H}_0)}$ |
|---|---|
| .37 | .25 |
| 1.0 | .48 |
| 2.7 | .82 |
| 7.4 | 1.3 |
| 20 | 1.8 |
| 55 | 1.9 |
| 148 | 1.7 |
| 403 | 1.3 |
| 1096 | 1.1 |

Figure 3.6. Likelihood ratio for various choices of the prior distribution's hyperparameter $\alpha$.

In conclusion, the data are not 'very suspicious'. They can be construed as giving at most two-to-one evidence in favour of one or other of the two hypotheses.

Are these wimpy likelihood ratios the fault of over-restrictive priors? Is there any way of producing a 'very suspicious' conclusion? The prior that is best-matched to the data, in terms of likelihood, is the prior that sets $p$ to $f \equiv 140/250$ with probability one. Let's call this model $\mathcal{H}_*$. The likelihood ratio is $P(D|\mathcal{H}_*)/P(D|\mathcal{H}_0) = 2^{250} f^{140}(1-f)^{110} = 6.1$. So the strongest evidence that these data can possibly muster against the hypothesis that there is no bias is six-to-one.

While we are noticing the absurdly misleading answers that 'sampling theory' statistics produces, such as the P-value of 7% in the exercise we just solved, let's stick the boot in. If we make a tiny change to the data set, increasing the number of heads in 250 tosses from 140 to 141, we find that the P-value goes below the mystical value of 0.05 (the P-value is 0.0497). The classical statistician would happily squeak 'the probability of getting a result as extreme as 141 heads is smaller than 0.05 – we thus reject the null hypothesis at a significance level of 5%'. The correct answer is shown for several values of $\alpha$ in figure 3.7. The values worth highlighting from this table are, first, the likelihood ratio when $\mathcal{H}_1$ uses the standard uniform prior, which is 1:0.61 in favour of the null hypothesis $\mathcal{H}_0$. Second, the most favourable choice of $\alpha$, from the point of view of $\mathcal{H}_1$, can only yield a likelihood ratio of about 2.3:1 in favour of $\mathcal{H}_1$.

Be warned! A P-value of 0.05 is often interpreted as implying that the odds are stacked about twenty-to-one against the null hypothesis. But the truth in this case is that the evidence either slightly *favours* the null hypothesis, or disfavours it by at most three to one, depending on the choice of prior.

The P-values and 'significance levels' of classical statistics should be treated with *extreme caution*. Shun them!

| $\alpha$ | $\frac{P(D' \mid \mathcal{H}_1, \alpha)}{P(D' \mid \mathcal{H}_0)}$ |
|---|---|
| .37 | .32 |
| 1.0 | .61 |
| 2.7 | 1.0 |
| 7.4 | 1.6 |
| 20 | 2.2 |
| 55 | 2.3 |
| 148 | 1.9 |
| 403 | 1.4 |
| 1096 | 1.2 |

Figure 3.7. Likelihood ratio for various choices of the prior distribution's hyperparameter $\alpha$, when the data are $D' = 141$ heads in 250 trials.

# 4

# *Message Passing*

One of the themes of this book is the idea of doing complicated calculations using simple distributed hardware. It turns out that quite a few interesting problems can be solved by *message-passing* algorithms, in which simple messages are passed locally among simple processors whose operations lead, after some time, to the solution of a global problem.

## 4.1 Counting

As an example, consider a line of soldiers walking in the mist. The commander wishes to perform the complex calculation of counting the number of soldiers in the line. This problem could be solved in two ways.

First there is a solution that uses expensive hardware: the loud booming voices of the commander and his men. The commander could shout 'all soldiers report back to me within one minute!', then he could listen carefully as the men respond 'Molesworth here sir!', 'Fotherington-Thomas here sir!', and so on. This solution relies on several advanced pieces of hardware: there must be a reliable communication channel to and from every soldier; the commander must be able to listen to all the incoming messages – even when there are hundreds of soldiers – and be able to count; and all the soldiers must be well-fed if they are to be able to shout back across the possibly-large distance separating them from the commander.

The second way of finding this global function, the number of soldiers, does not require global communication hardware, high IQ, or good food; we simply require that each soldier can communicate single integers with the two adjacent soldiers in the line, and that the soldiers are capable of adding one to a number. Each soldier follows these rules:

**Message-passing rule-set A**

1. If you are the front soldier in the line, say the number 'one' to the soldier behind you.
2. If you are the backmost soldier in the line, say the number 'one' to the soldier in front of you.
3. If a soldier ahead of or behind you says a number to you, add one to it, and say the new number to the soldier on the other side.

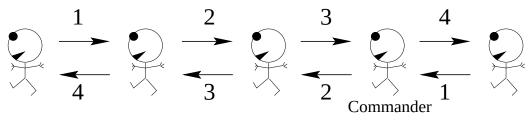


Figure 4.1. A line of soldiers counting themselves using rule-set A.

If the clever commander can not only add one to a number, but also add two numbers together, then he can find the global number of soldiers by simply adding together

| | | |
|---|---|---|
| | the number said to him by the soldier in front of him, | (which equals the total number of soldiers in front) |
| + | the number said to the commander by the soldier behind him, | (which is the number behind) |
| + | one | (to count the commander himself). |

This solution requires only local communication hardware and simple computations (storage and addition of integers).

## *Separation*

This clever trick makes use of a profound property of the total number of soldiers: that it can be written as the sum of the number of soldiers *in front of* a point and the number *behind* that point, two quantities which can be computed *separately.*

If the soldiers were not arranged in a line but were travelling in a jostling swarm, then it would not be easy to separate them into two groups in this way. The guerillas in figure 4.2 could not be counted using the above message-

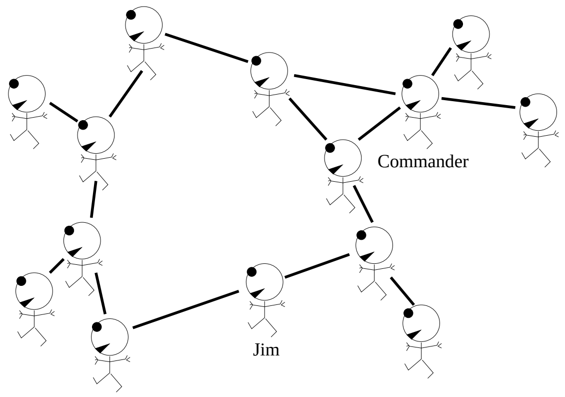


Figure 4.2. A swarm of guerillas.

passing rule-set A, because, while the guerillas do have neighbours (shown by lines), it is not clear who is in front and who is behind; furthermore, since the *graph* of connections between the guerillas contains cycles, it is not possible for a guerilla in a cycle (such as 'Jim') to *separate* the group into two groups, 'those in front', and 'those behind'.

A swarm of guerillas *can* be counted by a modified message-passing algorithm *if they are arranged in a graph that contains no cycles.*

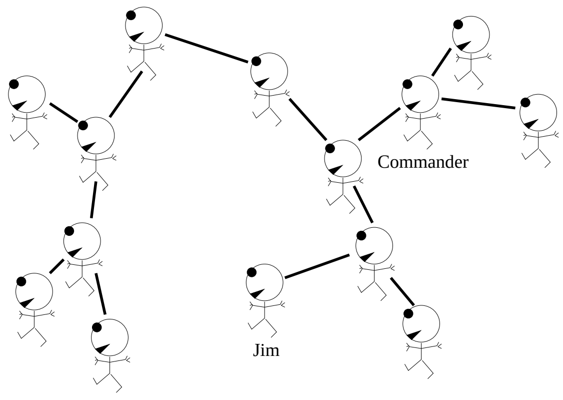


Figure 4.3. A swarm of guerillas whose connections form a tree.

Example 4.1: Describe a message-passing algorithm 'B' for counting a swarm of guerillas, like those in figure 4.3, whose connections form a *cycle-free graph*, also known as a *tree*.

Rule-set B produces messages from which any guerilla can deduce the total in the tree.

**Message-passing rule-set B**

1. Count your number of neighbours, $N$.
2. Keep count of the number of messages you have received from your neighbours, $m$, and of the values $v_1$, $v_2$, $\ldots v_N$ of each of those messages. Let $V$ be the running total of the messages you have received.
3. If the number of messages you have received, $m$, is equal to $N-1$, then identify the neighbour who has not sent you a message and tell them the number $V+1$.
4. If the number of messages you have received is equal to $N$, then:

   (a) the number $V+1$ is the required total.

   (b) for each neighbour $n$ {

   say to neighbour $n$ the number $V+1-v_n$

   }

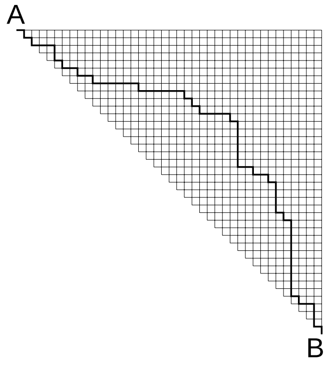


Figure 4.4. A triangular $41 \times 41$ grid. How many paths are there from A to B?

## 4.2 Path-counting

A more profound task than counting squaddies is the task of counting the number of paths through a grid, and finding how many paths pass through any given point in the grid.

Figure 4.4 shows a rectangular grid, and a path through the grid, connecting points A and B. The rules of the game are, starting from A, only rightward and downward moves are permitted.

1. How many such paths are there from A to B?

2. If a random path from A to B is selected, what is the probability that it passes through a particular node in the grid? [When we say 'random', we mean that all paths have exactly the the same probability of being selected.]

3. How can a random path from A to B be selected?

Counting all the paths from A to B doesn't seem straightforward. The number of paths is expected to be pretty big – even if the permitted grid was a diagonal strip only three nodes wide, there would still be about $2^{N/2}$ possible paths.

The computational breakthrough is to realise that to find the *number* of paths, we do not have to explicitly enumerate all the paths. Pick a point P in the grid and consider the number of paths from A to P. Every path from A to P must come in to P through one of its upstream neighbours ('upstream' meaning 'closer to A'). So the number of paths from A to P can be found by adding up the number of paths from A to each of those neighbours.

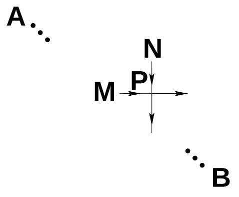


Figure 4.5. Every path from A to P enters P through an upstream neighbour of P; so we can find the number of paths from A to P by adding the number of paths from A to M and from A to N.

This message-passing algorithm is illustrated in figure 4.6 for a simple grid with ten vertices connected by twelve directed edges. We start by sending the '1' message from A. When any node has received messages from all its upstream neighbours, it sends the *sum* of them on to its downstream neighbours. At B, the number 5 emerges: we have counted the number of paths from A to B without enumerating them all. As a sanity-check, figure 4.7 shows the five distinct paths from A to B.

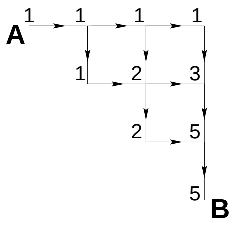


Figure 4.6. Messages sent in the forward pass.

Having counted all paths, we can now move on to more challenging problems: computing the probability that a random path goes through a given vertex, and creating a random path.

## *Probability of passing through a node*

By making a backward pass as well as the forward pass, we can deduce how many of the paths go through each node, which, if we divide by the total number of paths, is equal to the probability of a randomly selected path's passing through each node. Figure 4.8 shows the backward-passing messages in the lower-right corners of the tables, and the original forward-passing messages in the upper-left corners. By multiplying these two numbers at a given vertex, we find the total number of paths passing through that vertex. For example, four paths pass through the central vertex.

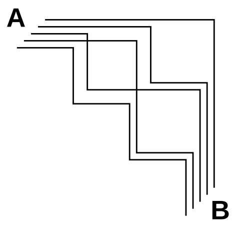


Figure 4.7. The five paths.

Figure 4.9 shows the result of this computation for the triangular $41 \times 41$ grid. The area of each blob is proportional to the probability of passing through each node.

## *Random path sampling*

Solutions on p.74 Exercise 4.2:$^{A1}$ If one creates a 'random' path from A to B by flipping a fair coin at every junction where there is a choice of two directions, is the resulting path a uniform random sample from the set of all paths? [Hint: imagine trying it for the grid of figure 4.6.]

There is a neat insight to be had here, and I'd like you to have the satisfaction of figuring it out.

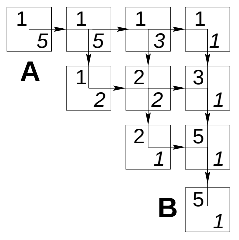


Figure 4.8. Messages sent in the forward and backward passes.

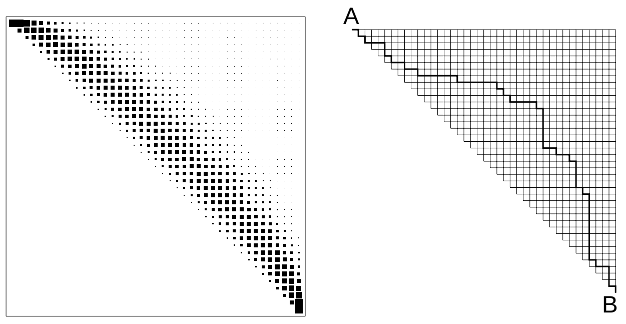


Figure 4.9. The probability of passing through each node, and a randomly chosen path.

Solutions on p.74 Exercise 4.3:$^{A1}$ Having run the forward and backward algorithms between points A and B on a grid, how can one draw one path from A to B *uniformly* at random?

Solution to exercise 4.2 (p.73): Since there are five paths through the grid of figure 4.6, they must all have probability 1/5. But a strategy based on fair coin-flips will produce paths whose probabilities are powers of 1/2.

Solution to exercise 4.3 (p.74): To make a uniform random walk, each step of the walk should be chosen using a different biased coin at each junction, with the biases chosen in proportion to the *backward* messages emanating from the two options. For example, at the first choice after leaving A, there is a '3' message coming from the East, and a '2' coming from South, so one should go East with probability 3/5 and South with probability 2/5. This is how the path in figure 4.9 was generated.

## 4.3 Summary and related ideas

Some global properties have a separability property. For example, the number of paths from A to P separates into the sum of the number of paths from A to M (the point to P's left) and the number of paths from A to N (the point above P). Many global properties do not have such separability properties, for example

1. The number of pairs of soldiers in a troop who share the same birthday.
2. The size of the largest group of soldiers who share a common height (rounded to the nearest centimetre).
3. The length of the shortest tour that a travellings salesman could take that visits every soldier in a troop.

We will make use of message-passing solutions to and other low-cost solutions to problems like these throughout this book.

# Part I

# Data Compression

# *About Chapter 5*

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, skim through them and keep going – you'll be able to enjoy chapters 6 and 7 without this chapter's tools.

Before reading chapter 5, you should have read chapter 2 and worked on exercises 2.16–2.20 and 2.25 (pp.41–42), and exercise 4.4 below.

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

| Notation | |
|---|---|
| $x \in \mathcal{A}$ | $x$ is a *member* of the set $\mathcal{A}$ |
| $\mathcal{S} \subset \mathcal{A}$ | $\mathcal{S}$ is a *subset* of the set $\mathcal{A}$ |
| $\mathcal{S} \subseteq \mathcal{A}$ | $\mathcal{S}$ is a subset of, or equal to, the set $\mathcal{A}$ |
| $\mathcal{V} = \mathcal{B} \cap \mathcal{A}$ | $\mathcal{V}$ is the *union* of the sets $\mathcal{B}$ and $\mathcal{A}$ |
| $\lvert\mathcal{A}\rvert$ | number of elements in set $\mathcal{A}$ |

Exercise 4.4: – *Please work on this problem before reading chapter 5.*

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 balls?

5

# The Source Coding Theorem

## 5.1 How to measure the information content of a random variable?

In the next few chapters, we'll be talking about probability distributions and random variables. Most of the time we can get by with sloppy notation, but occasionally, we will need precise notation. Here is the notation that we established in chapter 2.

**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

1. that the *Shannon information content*,

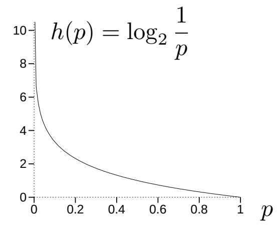
$$h(x{=}a_i) \equiv \log_2 \frac{1}{p_i}, \tag{5.1}$$

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

2. that the *entropy* of the ensemble,

$$H(X) = \sum_i p_i \log_2 \frac{1}{p_i}, \tag{5.2}$$

   is a sensible measure of the ensemble's average information content.



| $p$ | $h(p)$ | $H_2(p)$ |
|---|---|---|
| 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 |

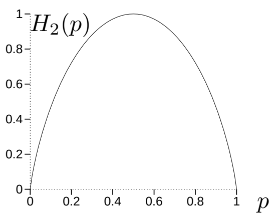


Figure 5.1. The Shannon information content $h(p) = \log_2 \frac{1}{p}$ and the binary entropy function $H_2(p) = H(p, 1{-}p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}$ as a function of $p$.

Figure 5.1 shows the Shannon information content of an outcome with probability $p$, as a function of $p$. The less probable an outcome is, the greater its Shannon information content. Figure 5.1 also shows the binary entropy function,

$$H_2(p) = H(p, 1-p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}, \tag{5.3}$$

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

## *Information content of independent random variables*

Why should $\log 1/p_i$ have anything to do with the information content? Why not some other function of $p_i$? We'll explore this question in detail shortly, but first, notice a nice property of this particular function $h(x) = \log 1/p(x)$.

Imagine learning the value of two *independent* random variables, $x$ and $y$. The definition of independence is that the probability distribution is separable into a *product*:

$$P(x, y) = P(x)P(y). \tag{5.4}$$

Intuitively, we might want any measure of the 'amount of information gained' to have the property of *additivity* – that is, for independent random variables $x$ and $y$, the information gained when we learn $x$ and $y$ should equal the sum of the information gained if $x$ alone were learned and the information gained if $y$ alone were learned.

The Shannon information content of the outcome $x, y$ is

$$h(x, y) = \log \frac{1}{P(x, y)} = \log \frac{1}{P(x, y)} = \log \frac{1}{P(x)} + \log \frac{1}{P(y)} \tag{5.5}$$

so it does indeed satisfy

$$h(x, y) = h(x) + h(y), \text{ if } x \text{ and } y \text{ are independent.} \tag{5.6}$$

Exercise 5.1:$^{A1}$ Show that, if $x$ and $y$ are independent, the entropy of the outcome $x, y$ satisfies

$$H(X, Y) = H(X) + H(Y). \tag{5.7}$$

In words, entropy is additive for independent variables.

We now explore these ideas with some examples; then, in section 5.4 and in chapters 6 and 7, 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 4.4, p. 76) 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 in 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. If you haven't found a strategy that always gets there in three weighings, I encourage you to think about exercise 4.4 some more.

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 heavier', 'right heavier', and 'balance') are *as close as possible to equiprobable.* An optimal solution is shown in figure 5.2.

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'. Such a binary outcome only rules out half of the possible hypotheses, so a strategy that uses such outcomes must sometimes take longer to find the right answer.

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 2.20 (p.41): the entropy of an ensemble $X$ is biggest if all the outcomes have equal probability $p_i{=}1/|\mathcal{A}_X|$.

## *Guessing games*

In the game of twenty questions, one player thinks of an object, and the other player attempts to guess what the object is by asking questions that have yes/no answers, for example, 'is it alive?', or 'is it human?' The aim is to identify the object with as few questions as possible. What is the best strategy for playing this game? For simplicity, imagine that we are playing the rather dull version of twenty questions called 'sixty–three'.

**Example 5.2: 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 $x \bmod 2 = 1$?

[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 $\{\mathtt{1}, \mathtt{0}\}$, give the binary expansion of $x$, for example $35 \Rightarrow$ `100011`. Six questions suffice.

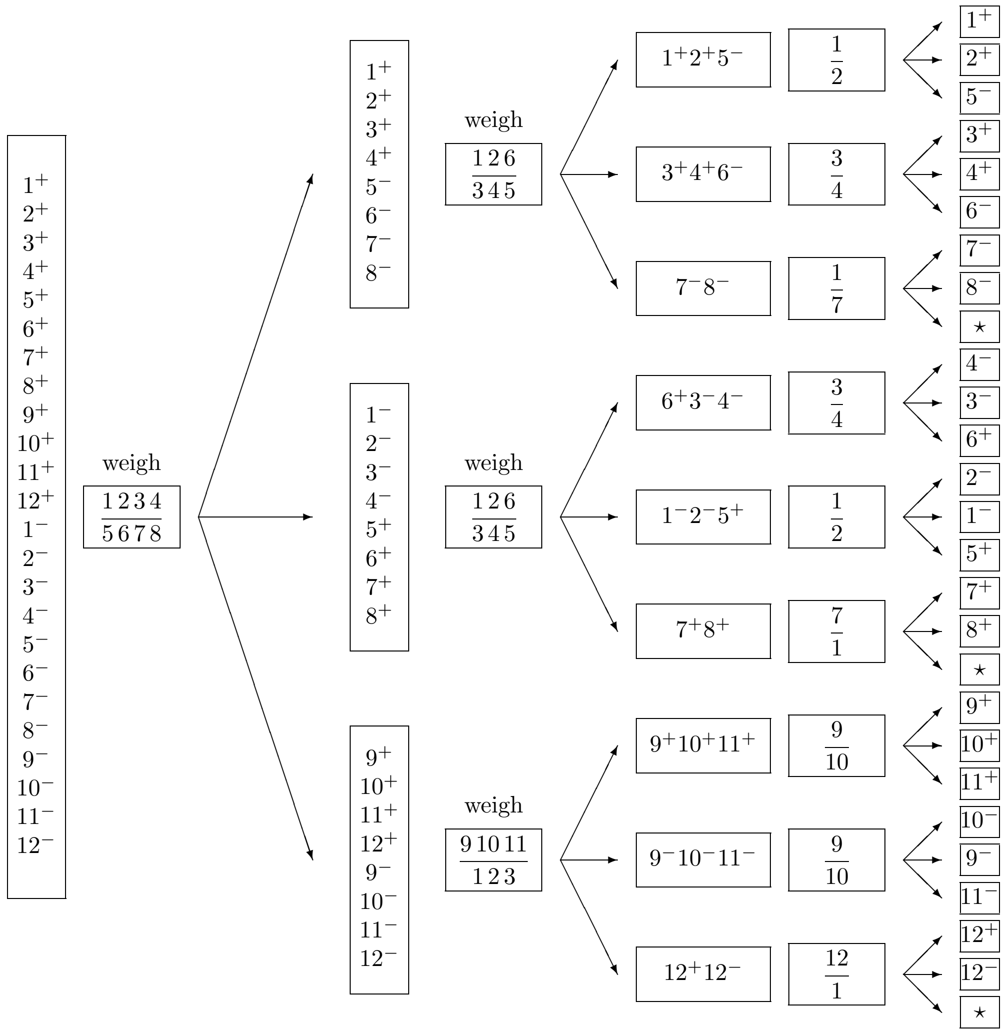
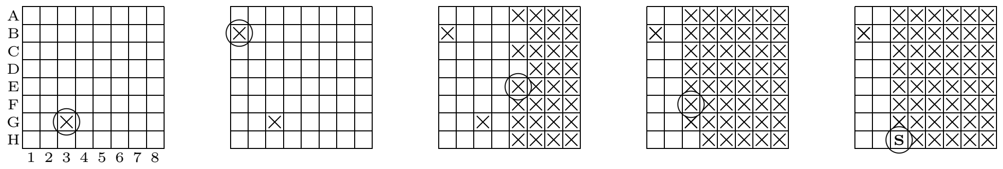


Figure 5.2. An optimal solution to the weighing problem. At each step there are two boxes: the left box shows which hypotheses are still possible; the right box shows the balls involved in the next weighing. The 24 hypotheses are written $1^+, \ldots, 12^-$, with, e.g., $1^+$ denoting that 1 is the odd 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 to the situation when the right side is heavier, and the lower arrow to the situation when the outcome is balanced. The three points labelled $\star$ correspond to impossible outcomes.



| | | | | | |
|---|---|---|---|---|---|
| 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 5.3. A game of submarine. The submarine is hit on the 49th attempt.

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 $\log_2(1/0.5) = 1$ 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 too?

## *The game of submarine: how many bits can one bit convey?*

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 5.3 shows a few pictures of this game in progress: the circle represents the square that is being fired at, and the $\times$s show squares in which the outcome was $x = \mathtt{n}$; the submarine is hit (outcome $x = \mathtt{y}$ shown by the symbol $\mathtt{s}$) 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

$$h(x) = h_{(1)}(\mathtt{y}) = \log_2 64 = 6 \, \text{bits}. \tag{5.8}$$

Now, it might seem a little strange that one binary outcome can convey six bits. But we have learnt the hiding place, which could have been any of 64 squares; so we have, by one lucky binary question, indeed learnt six bits.

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

$$h(x) = h_{(1)}(\mathtt{n}) = \log_2 \frac{64}{63} = 0.0227\,\text{bits}. \tag{5.9}$$

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

$$h_{(2)}(\mathtt{n}) = \log_2 \frac{63}{62} = 0.0230\,\text{bits}. \tag{5.10}$$

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

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{33}{32} \\ &= \; 0.0227 + 0.0230 + \cdots + 0.0430 \;=\; 1.0\,\text{bits}. \end{aligned} \tag{5.11}$$

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 79), 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

$$h_{(49)}(\mathtt{y}) = \log_2 16 = 4.0\,\text{bits}. \tag{5.12}$$

The total Shannon information content of all the outcomes is

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{17}{16} + \log_2 \frac{16}{1} \\ &= \; 0.0227 + 0.0230 + \cdots + 0.0874 + 4.0 \;=\; 6.0\,\text{bits}. \end{aligned} \tag{5.13}$$

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 (5.13) — then the total information gained is:

$$\begin{aligned} &\log_2 \frac{64}{63} + \log_2 \frac{63}{62} + \cdots + \log_2 \frac{n+1}{n} + \log_2 \frac{n}{1} \\ &= \; \log_2 \left[ \frac{64}{63} \times \frac{63}{62} \times \cdots \times \frac{n+1}{n} \times \frac{n}{1} \right] \;=\; \log_2 \frac{64}{1} \;=\; 6\,\text{bits}. \end{aligned} \tag{5.14}$$

What have we learned from the examples so far? I think the `submarine` example makes quite a convincing case for the claim that the Shannon information content is a sensible measure of information content. And the game of `sixty-three` shows that the Shannon information content can be intimately connected to the size of a file that encodes the outcomes of a random experiment.

In case you're not convinced, let's look at one more example.

### *The Wenglish language*

*Wenglish* is a language similar to English. Wenglish sentences consist of words drawn at random from the Wenglish dictionary, which contains $2^{15} = 32,768$ words, all of length 5 characters. The Wenglish dictionary was constructed by the Wenglish language committee, who created each of those 32,768 words at random by picking five letters from the probability distribution over `a`...`z` depicted in figure 2.1.

Some entries from the dictionary are shown in figure ??. Notice that the number of words in the dictionary (32,768) is much smaller than the total number of possible words of length 5 letters, $26^5 \simeq 12,000,000$.

Because the probability of the letter `z` is about 1/1000, only 32 of the words in the dictionary begin with the letter `z`. In contrast, the probability of the letter `a` is about 0.0625, and 2048 of the words begin with the letter `a`. Of those 2048 words, two start `az`, and 128 start `aa`.

Let's imagine that we are reading a Wenglish document, and let's discuss the Shannon information content of the characters as we acquire them. If we are given the text one word at a time, the Shannon information content of each five-character word is $\log 32768 = 15$ bits, since Wenglish uses all its words with equal probability. The average information content per character is 3 bits.

Now let's look at the information content if we read the document one character at a time. If, say, the first letter of a word is `a`, the Shannon information content is $\log 1/0.0625 \simeq 4$ bits. If the first letter is `z`, the Shannon information content is $\log 1/0.001 \simeq 10$ bits. The information content is thus highly variable at the first character. The total information content of the 5 character in a word, however, is exactly 15 bits, so the letters that follow an initial `z` have lower information content per character than the letters that follow an initial `a`.

Similarly, in English, if rare characters occur at the start of the word (e.g., `xyl...`, then often we can identify the whole word immediately; whereas words that start with common characters (e.g., `pro...`) require more characters before we can identify them.

| | |
|---|---|
| 1 | `aaail` |
| 2 | `aaaiu` |
| 3 | `aaald` |
| | ⋮ |
| 129 | `abati` |
| | ⋮ |
| 2047 | `azpan` |
| 2048 | `aztdn` |
| | ⋮ |
| | ⋮ |
| 16384 | `odrcr` |
| | ⋮ |
| | ⋮ |
| 32736 | `zatnt` |
| | ⋮ |
| 32768 | `zxast` |

Figure 5.4. The Wenglish dictionary.

## 5.2 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 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 $L$ bits per source symbol and recover the data reliably, then we will say that the average information content of that source is at most $L$ bits per symbol.

### *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

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

of the ASCII character set (decimal values 0 to 127). This character set uses only seven of the eight bits in a byte.

Solutions on p.97 Exercise 5.3:$^{B1}$ 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 number of *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

$$H_0(X) = \log_2 |\mathcal{A}_X|. \quad (5.15)$$

$H_0(X)$ 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

$$H_0(X, Y) = H_0(X) + H_0(Y). \quad (5.16)$$

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 5.4:$^{A2}$ 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]http://sunsite.org.uk/public/usenet/news-faqs/comp.compression/

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). We'll denote by $\delta$ the probability of the source string's being one of the confusable files, so a lossy compressor has a probability $\delta$ of failure. If $\delta$ can be made very small then a lossy compressor may be practically useful.

2. A *lossless* compressor maps all files to different encodings; if it 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, and the probability that it is shortened is large.

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

## 5.3 Information content defined in terms of lossy compression

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 the characters { `!`, `@`, `#`, `%`, `^`, `*`, `~`, `<`, `>`, `/`, `\`, `_`, `{`, `}`, `[`, `]`, `|` }, thereby reducing the size of the alphabet by seventeen. The larger the risk we are willing to take, the smaller our final alphabet becomes.

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

Example 5.5: Let

$$\begin{aligned} \mathcal{A}_X &= \{\, \mathtt{a}, \mathtt{b}, \mathtt{c},\ \mathtt{d},\ \ \mathtt{e},\ \ \mathtt{f},\ \ \mathtt{g},\ \ \mathtt{h}\ \}, \\ \text{and} \quad \mathcal{P}_X &= \{\, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{1}{4}, \tfrac{3}{16}, \tfrac{1}{64}, \tfrac{1}{64}, \tfrac{1}{64}, \tfrac{1}{64} \,\}. \end{aligned} \qquad (5.17)$$

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 5.5 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.

Let us now formalize this idea. To make a compression strategy with risk $\delta$, we make the smallest possible subset $S_\delta$ such that the probability that $x$ is not in $S_\delta$ is less than or equal to $\delta$, i.e., $P(x \notin S_\delta) \leq \delta$. For each value of $\delta$ we can then define a new measure of information content — the log of the size of this smallest subset $S_\delta$. [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.]

| $\delta = 0$ | | $\delta = 1/16$ | |
|---|---|---|---|
| $x$ | $c(x)$ | $x$ | $c(x)$ |
| 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 5.5. Binary names for the outcomes, for two failure probabilities $\delta$.

**The smallest $\delta$-sufficient subset** $S_\delta$ is the smallest subset of $\mathcal{A}_X$ satisfying

$$P(x \in S_\delta) \geq 1 - \delta. \tag{5.18}$$

The subset $S_\delta$ 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 is $\geq (1-\delta)$.

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:

$$H_\delta(X) = \log_2 |S_\delta| \tag{5.19}$$

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 5.1.]

Figure 5.6 shows $H_\delta(X)$ for the ensemble of example 5.5 as a function of $\delta$.

## *Extended ensembles*

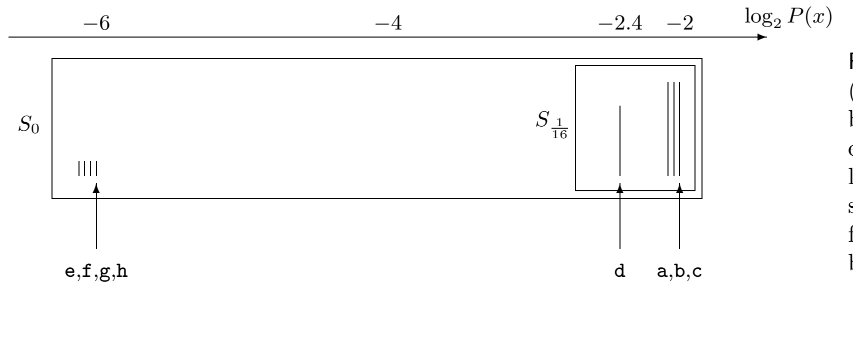
Is this compression method any more useful if we compress *blocks* of symbols from a source?

We now turn to examples where the outcome $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is a string of $N$ 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, (exercise 9.1 (p.168)) so $H(X^N) = NH(X)$.

Example 5.6: Consider a string of $N$ flips of a bent coin, $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, where $x_n \in \{0, 1\}$, 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

$$P(\mathbf{x}) = p_0^{N-r(\mathbf{x})} p_1^{r(\mathbf{x})}. \tag{5.20}$$

To evaluate $H_\delta(X^N)$ we must find the smallest sufficient subset $S_\delta$. 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 5.7 and 5.8 show graphs of $H_\delta(X^N)$ against $\delta$ for the cases $N = 4$ and $N = 10$. The steps are the values of $\delta$ 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.



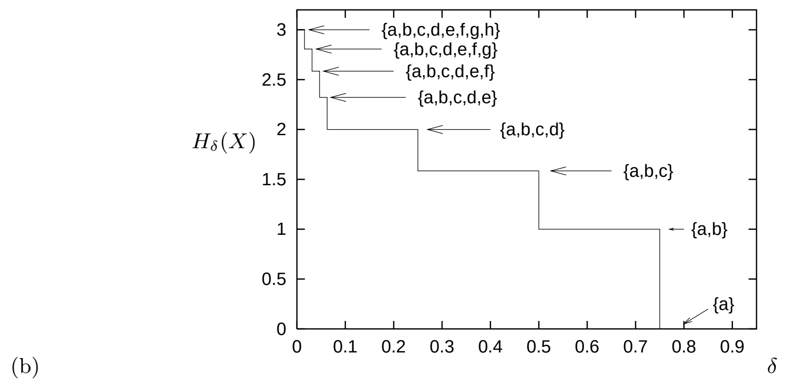
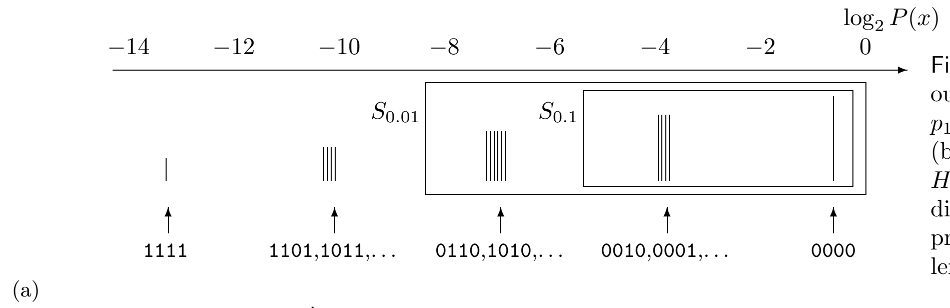


Figure 5.6. (a) The outcomes of $X$ (from example 5.5 (p.85)), ranked by their probability. (b) The essential bit content $H_\delta(X)$. The labels on the graph show the smallest sufficient set as a function of $\delta$. Note $H_0(X) = 3$ bits and $H_{1/16}(X) = 2$ bits.



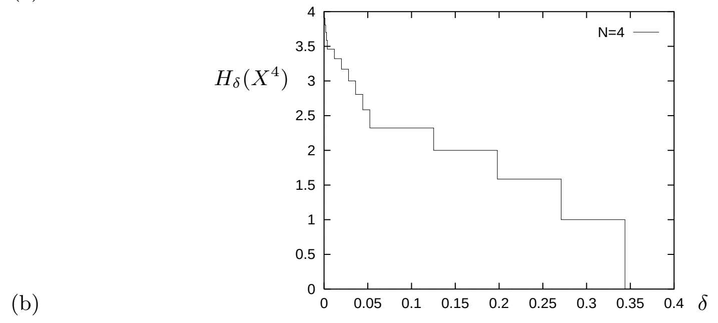


Figure 5.7. (a) The sixteen outcomes of the ensemble $X^4$ with $p_1 = 0.1$, ranked by probability. (b) The essential bit content $H_\delta(X^4)$. The upper schematic diagram indicates the strings's probabilities by the vertical lines's lengths (not to scale).

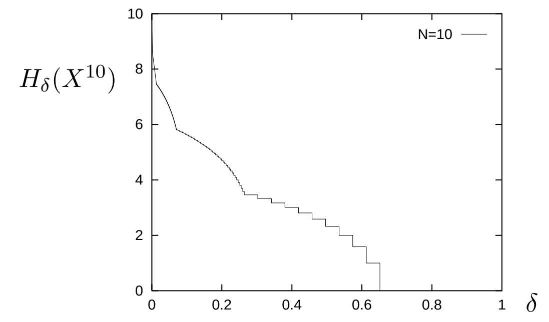


Figure 5.8. $H_\delta(X^N)$ for $N = 10$ binary variables with $p_1 = 0.1$.

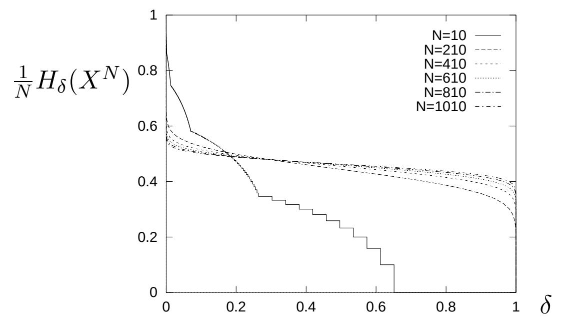


Figure 5.9. $\frac{1}{N}H_\delta(X^N)$ for $N = 10, 210, \ldots, 1010$ binary variables with $p_1 = 0.1$.

Exercise 5.7:$^{C2}$ What are the mathematical shapes of the curves between the cusps?

For the examples shown in figures 5.6–5.8, $H_\delta(X^N)$ depends strongly on the value of $\delta$, 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 $\delta$ — and for all $\delta$ it is very close to $NH(X)$, where $H(X)$ is the entropy of one of the random variables.

Figure 5.9 illustrates this asymptotic tendency for the binary ensemble of example 5.6. 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 $\delta$, 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 5.1** Shannon's Source Coding theorem. *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 $N > N_0$,*

$$\left|\frac{1}{N}H_\delta(X^N) - H\right| < \epsilon. \tag{5.21}$$

## 5.4 Typicality

Why does increasing $N$ help? Let's examine long strings from $X^N$. Table 5.10 shows fifteen samples from $X^N$ for $N = 100$ and $p_1 = 0.1$. The probability

| $\mathbf{x}$ | $\log_2(P(\mathbf{x}))$ |
|---|---|
| ...1...................1.....1....1.1.......1........1...........1.....................1.......11... | $-50.1$ |
| ......................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..1.11..... | $-56.4$ |
| ...11...........1...1.....1.1......1..........1....1...1.....1............1......................... | $-53.2$ |
| ..............1......1.........1.1.......1..........1............1...1......................1....... | $-43.7$ |
| .....1........1.......1...1............1............1...........1......1..11........................ | $-46.8$ |
| .....1..1..1...............111...................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. | $-43.7$ |
| 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......... | $-46.8$ |
| .................................................................................................... | $-15.2$ |
| 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 | $-332.1$ |

Figure 5.10. 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 log-probabilities of the random strings, which may be compared with the entropy $H(X^{100}) = 46.9$ bits.

of a string $\mathbf{x}$ that contains $r$ 1s and $N-r$ 0s is

$$p(\mathbf{x}) = p_1^r (1-p_1)^{N-r}. \tag{5.22}$$

The number of strings that contain $r$ 1s is

$$n(r) = \binom{N}{r}. \tag{5.23}$$

So the number of 1s, $r$, has a binomial distribution:

$$p(r) = \binom{N}{r} p_1^r (1-p_1)^{N-r}. \tag{5.24}$$

These functions are shown in figure 5.11. The mean of $r$ is $Np_1$, and its standard deviation is $\sqrt{Np_1(1-p_1)}$ (p. 6). If $N$ is 100 then

$$r \sim Np_1 \pm \sqrt{Np_1(1-p_1)} \simeq 10 \pm 3. \tag{5.25}$$

If $N = 1000$ then

$$r \sim 100 \pm 10. \tag{5.26}$$

Notice that as $N$ gets bigger, the probability distribution of $r$ becomes more concentrated in the sense that while the range of possible values of $r$ grows as $N$, 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*.

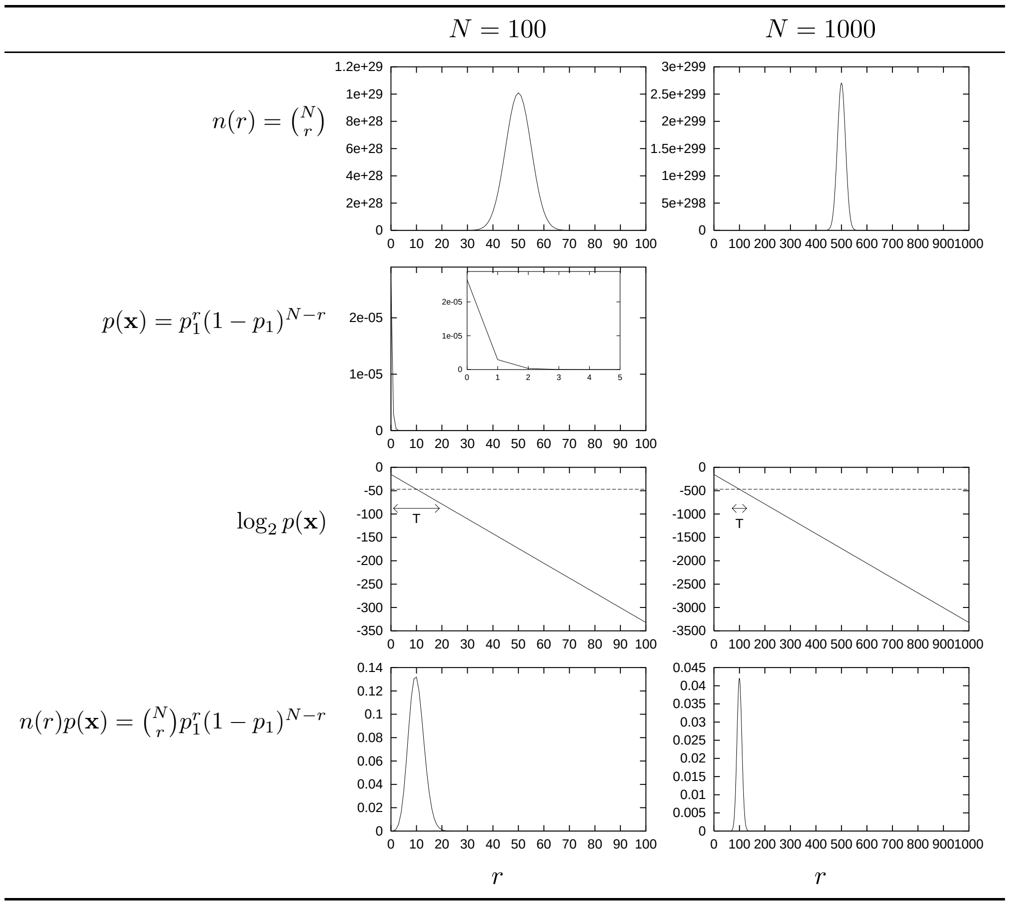


Figure 5.11. Anatomy of the typical set $T$. For $p_1 = 0.1$ and $N = 100$ and $N = 1000$, these graphs show $n(r)$, the number of strings containing $r$ 1s; the probability $p(\mathbf{x})$ of a single string that contains $r$ 1s; the same probability on a log scale; and the total probability $n(r)p(\mathbf{x})$ 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 $T_{N\beta}$ (as defined in section 5.4) for $N = 100$ and $\beta = 0.29$ (left) and $N = 1000$, $\beta = 0.09$ (right).

### Definition of the typical set

Let us define typicality for an arbitrary ensemble $X$ with alphabet $\mathcal{A}_X$. Our definition of a typical string will involve the string's probability. A long string of $N$ symbols will usually contain about $p_1 N$ occurrences of the first symbol, $p_2 N$ occurrences of the second, etc. Hence the probability of this string is roughly

$$p(\mathbf{x})_{\text{TYP}} = p(x_1)p(x_2)p(x_3)\ldots p(x_N) \simeq p_1^{(p_1 N)} p_2^{(p_2 N)} \ldots p_I^{(p_I N)} \tag{5.27}$$

so that the information content of a typical string is

$$\log_2 \frac{1}{p(\mathbf{x})} \simeq N \sum_i p_i \log_2 \frac{1}{p_i} \simeq NH. \tag{5.28}$$

So the random variable $\log_2 \frac{1}{p(\mathbf{x})}$, which is the information content of $\mathbf{x}$, is very likely to be close in value to $NH$. We build our definition of typicality on this observation.

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 $\beta$ 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, $T_{N\beta}$

$$T_{N\beta} \equiv \left\{ \mathbf{x} \in \mathcal{A}_X^N : \left| \frac{1}{N} \log_2 \frac{1}{P(\mathbf{x})} - H \right| < \beta \right\}. \tag{5.29}$$

We will show that whatever value of $\beta$ 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 94.]

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 (verbal statement).** $N$ 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.

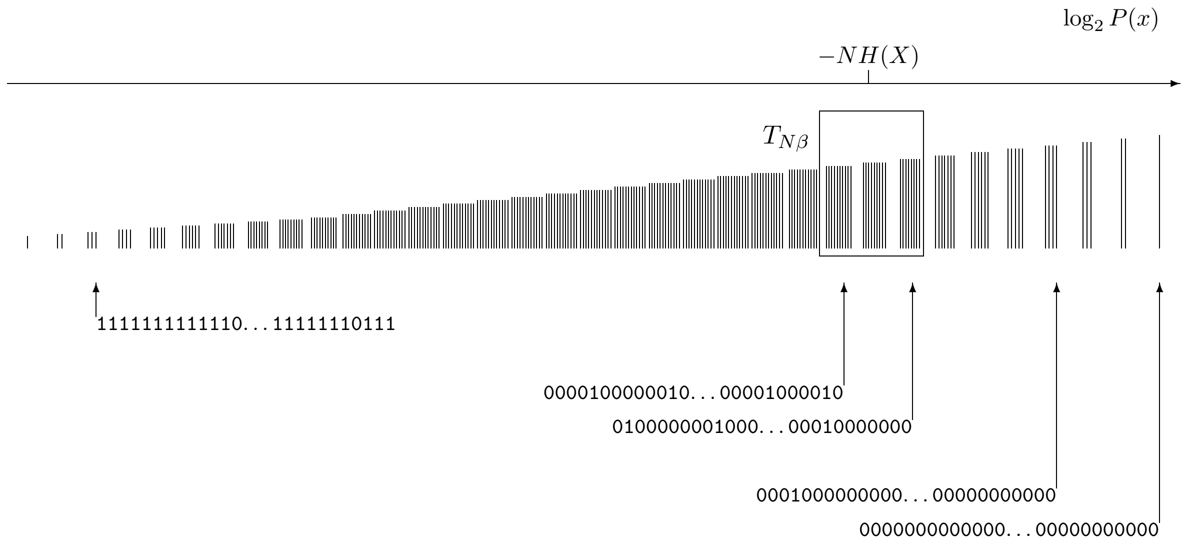


Figure 5.12. Schematic diagram showing all strings in the ensemble $X^N$ ranked by their probability, and the typical set $T_{N\beta}$.

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.

## 5.5 Proofs

This section may be skipped if found tough going.

### *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 are $\mathcal{E}[u] = \bar{u} = \sum_u P(u)u$ and $\text{var}(u) = \sigma_u^2 = \mathcal{E}[(u-\bar{u})^2] = \sum_u P(u)(u-\bar{u})^2$.

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 $P(u)$.

**Chebyshev's inequality 1.** Let $t$ be a non-negative real random variable, and let $\alpha$ be a positive real number. Then

$$P(t \geq \alpha) \;\leq\; \frac{\bar{t}}{\alpha}. \tag{5.30}$$

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 $\alpha$ be a positive real number. Then

$$P\left((x-\bar{x})^2 \geq \alpha\right) \;\leq\; \sigma_x^2/\alpha. \tag{5.31}$$

Proof: Take $t = (x-\bar{x})^2$ and apply the previous proposition.

**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 $\sigma_h^2$: $x = \frac{1}{N}\sum_{n=1}^{N} h_n$. Then

$$P((x-\bar{h})^2 \geq \alpha) \leq \sigma_h^2/\alpha N. \tag{5.32}$$

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 ($\alpha$ very small). No matter how large $\sigma_h^2$ is, and no matter how small the required $\alpha$ 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 5.1 (page 88)*

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 $N$ information contents $h_n = \log_2(1/P(x_n))$, each of which is a random variable with mean $H = H(X)$ and variance $\sigma^2 \equiv \text{var}[\log_2(1/P(x_n))]$. (Each term $h_n$ is in fact the Shannon information content of the $n$th outcome.)

We again define a typical set with parameters $N$ and $\beta$ thus:

$$T_{N\beta} = \left\{ \mathbf{x} \in \mathcal{A}_X^N : \left[\frac{1}{N}\log_2 \frac{1}{P(\mathbf{x})} - H\right]^2 < \beta^2 \right\}. \tag{5.33}$$

For all $\mathbf{x} \in T_{N\beta}$, the probability of $\mathbf{x}$ satisfies

$$2^{-N(H+\beta)} < P(\mathbf{x}) < 2^{-N(H-\beta)}. \tag{5.34}$$

And by the law of large numbers,

$$P(\mathbf{x} \in T_{N\beta}) \geq 1 - \frac{\sigma^2}{\beta^2 N}. \tag{5.35}$$

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 $\beta$. How does this result relate to source coding?

We must relate $T_{N\beta}$ to $H_\delta(X^N)$. We will show that for any given $\delta$ there is a sufficiently big $N$ such that $H_\delta(X^N) \simeq NH$.

### *Part 1:* $\frac{1}{N}H_\delta(X^N) < H + \epsilon$.

The set $T_{N\beta}$ is not the best subset for compression. So the size of $T_{N\beta}$ gives an upper bound on $H_\delta$. We show how *small* $H_\delta(X^N)$ must be by calculating how big $T_{N\beta}$ could possibly be. We are free to set $\beta$ 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

$$|T_{N\beta}|\, 2^{-N(H+\beta)} < 1, \tag{5.36}$$

that is, the size of the typical set is bounded by

$$|T_{N\beta}| < 2^{N(H+\beta)}. \tag{5.37}$$

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 $H_\delta(X^N) \leq \log_2 |T_{N\beta}| < N(H + \epsilon)$.

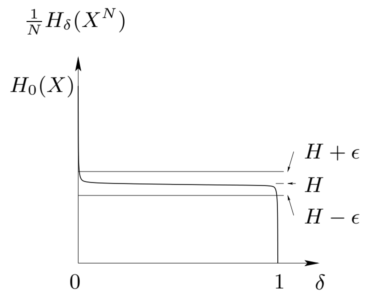


Figure 5.13. Schematic illustration of the two parts of the theorem. Given any $\delta$ and $\epsilon$, we show that for large enough $N$, $\frac{1}{N}H_\delta(X^N)$ lies (1) below the line $H + \epsilon$ and (2) above the line $H - \epsilon$.

*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 $\delta$-sufficient subset $S_\delta$ is smaller than the above inequality would allow. We can make use of our typical set to show that they must be mistaken. Remember that we are free to set $\beta$ 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

$$P(\mathbf{x} \in S' \cap T_{N\beta}) + P(\mathbf{x} \in S' \cap \overline{T_{N\beta}}), \tag{5.38}$$

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:

$$P(\mathbf{x} \in S') \leq 2^{N(H-2\beta)}\, 2^{-N(H-\beta)} + \frac{\sigma^2}{\beta^2 N} = 2^{-N\beta} + \frac{\sigma^2}{\beta^2 N}. \tag{5.39}$$

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 $S_\delta$. Thus *any* subset $S'$ with size $|S'| \leq 2^{N(H-\epsilon)}$ has probability less than $1 - \delta$, so by the definition of $H_\delta$, $H_\delta(X^N) > N(H - \epsilon)$.

Thus for large enough $N$, the function $\frac{1}{N}H_\delta(X^N)$ is essentially a constant function of $\delta$, for $0 < \delta < 1$, as was illustrated in figure 5.9. $\square$

## 5.6 Comments

The source coding theorem (page 88) 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 $\delta$ is extremely small, the number of bits per symbol $\frac{1}{N}H_\delta(X^N)$ needed to specify a long $N$-symbol string $\mathbf{x}$ with vanishingly small error probability does not have to exceed $H + \epsilon$ bits. We need to have only a tiny tolerance for 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 $\delta$ 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 $\beta$ 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 $\beta$ in terms of $N$ as $\alpha/\sqrt{N}$, for some constant $\alpha$, 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 $\beta$ decreases, $N$ increases, and this ratio $2^{\alpha\sqrt{N}}$ grows exponentially. Thus we have 'equipartition' only in a weak sense!

## 5.7 Exercises

*Weighing problems*

Solutions on p.97

Exercise 5.8:$^{B2}$ 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 5.9:$^{B2}$ 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.97

Exercise 5.10:$^{C4}$ (a) Is it possible to solve exercise 4.4 (p.76) (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 solution to the general 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.98

Exercise 5.11:$^{C3}$ You are given 12 balls and the three-outcome balance of exercise 4.4; 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 with a lossy compressor, with loss $\delta$*

Exercise 5.12:$^{B2}$ Let $\mathcal{P}_X = \{0.4, 0.6\}$. Sketch $\frac{1}{N} H_\delta(X^N)$ as a function of $\delta$ for $N = 1, 2$ and 100.

Exercise 5.13:$^{B2}$ Let $\mathcal{P}_Y = \{0.5, 0.5\}$. Sketch $\frac{1}{N} H_\delta(Y^N)$ as a function of $\delta$ for $N = 1, 2, 3$ and 100.

Solutions on p.99

Exercise 5.14:$^{B2}$ 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 5.15:$^{B3}$ Sketch the Cauchy distribution

$$P(x) = \frac{1}{Z}\frac{1}{x^2+1}, \quad x \in (-\infty, \infty). \tag{5.40}$$

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 distribution. 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?

## *Curious functions related to $p \log 1/p$*

Exercise 5.16:$^{E4}$ This exercise has no purpose at all; it's included for the enjoyment of those who like mathematical curiousities.

Sketch the function

$$f(x) = x^{x^{x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}}} \tag{5.41}$$

for $x \geq 0$. To be explicit about the order in which the powers are evaluated, here's another definition of $f$:

$$f(x) = x^{\left(x^{\left(x^{\cdot^{\cdot^{\cdot}}}\right)}\right)} \tag{5.42}$$

Hints:

(a) Consider $f(\sqrt{2})$: you might be able to persuade yourself that $f(\sqrt{2}) = 2$. You might also be able to persuade yourself that $f(\sqrt{2}) = 4$. What's going on? [Yes, a two-valued function.]

(b) For a given $x$, if $f(x) = y$, then we have $y = x^y$, so $y$ is found at the intersection of the curves $u_1(y) = x^y$ and $u_2(y) = y$.

(c) Work out the inverse function to $f$ – that is, the function $g(y)$ such that if $x = g(y)$ then $y = f(x)$ – hint: it's closely related to $p \log 1/p$.

# *Solutions to Chapter 5's exercises*

Solution to exercise 5.3 (p.84): 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 5.4 (p.84): 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 unique 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 5.7 (p.88): Between the cusps, all the changes in probability are equal, and the number of elements in $T$ changes by one at each step. So $H_\delta$ varies logarithmically with $(-\delta)$.

Solution to exercise 5.8 (p.95): 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 5.10 (p.95): This solution was found by Dyson and Lyness in 1946 and presented in the following elegant form by John Conway in 1999.[2] 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
```

[2]Posting to `geometry-puzzles@forum.swarthmore.edu` Thu, 28 Jan 1999.

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

$$N = (3^W - 3)/2 \tag{5.43}$$

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 $w$th weighing putting those whose $w$th letter is `A` in pan `A` and those whose $w$th letter is `B` in pan `B`.

Solution to exercise 5.11 (p.95):

(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 just which are the odd balls. Let us define the oddness of an odd ball to be the absolute value of the difference between its weight and the regular weight. There is a good chance that we will also learn something about the relative oddnesses of the two odd balls. If balls $m$ and $n$ are the odd balls, there is a good chance that the optimal weighing strategy will at some point put ball $m$ on one side of the balance and ball $n$ on the other, along with a load of regular balls; if $m$ and $n$ are both heavy balls, say, the outcome of this weighing will reveal, at the end of the day, whether $m$ was heavier than $n$, 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.

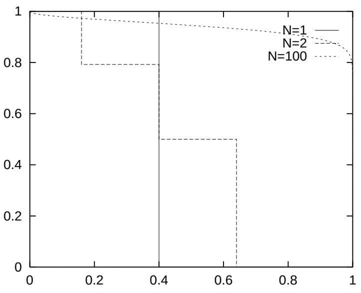
A more careful estimate takes this annoying possibility into account.

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. I don't know whether the original problem can be solved in 6 weighings.

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 $13 \times 1760$, 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 5.12 (p.95): The curves $\frac{1}{N} H_\delta(X^N)$ as a function of $\delta$ for $N = 1, 2$ and 100 are shown in figure 5.14. Note that $H_2(0.4) = 0.971$ bits.

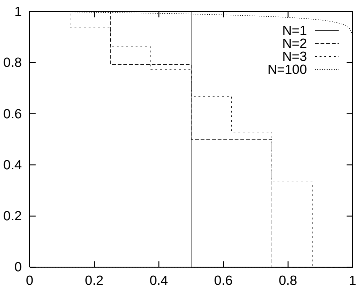


| $N=1$ | | |
|---|---|---|
| $\delta$ | $\frac{1}{N} H_\delta(\mathbf{X})$ | $2^{H_\delta(\mathbf{X})}$ |
| 0–0.4 | 1 | 2 |
| 0.4–1 | 0 | 1 |

| $N=2$ | | |
|---|---|---|
| $\delta$ | $\frac{1}{N} H_\delta(\mathbf{X})$ | $2^{H_\delta(\mathbf{X})}$ |
| 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 5.14. $\frac{1}{N} H_\delta(\mathbf{X})$ (vertical axis) against $\delta$ (horizontal), for $N = 1, 2, 100$ binary variables with $p_1 = 0.4$.

Solution to exercise 5.13 (p.95): The curves $\frac{1}{N} H_\delta(Y^N)$ as a function of $\delta$ for $N = 1, 2, 3$ and 100 are shown in figure 5.15. Note that $H_2(0.5) = 1$ bit.



| $N=2$ | | |
|---|---|---|
| $\delta$ | $\frac{1}{N} H_\delta(\mathbf{Y})$ | $2^{H_\delta(\mathbf{Y})}$ |
| 0–0.25 | 1 | 4 |
| 0.25–0.5 | 0.79248 | 3 |
| 0.5–0.75 | 0.5 | 2 |
| 0.75–1 | 0 | 1 |

| $N=3$ | | |
|---|---|---|
| $\delta$ | $\frac{1}{N} H_\delta(\mathbf{Y})$ | $2^{H_\delta(\mathbf{Y})}$ |
| 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 5.15. $\frac{1}{N} H_\delta(\mathbf{Y})$ (vertical axis) against $\delta$ (horizontal), for $N = 1, 2, 3, 100$ binary variables with $p_1 = 0.5$.

Solution to exercise 5.14 (p.95): The Gibbs entropy is $k_\mathrm{B} \sum_i p_i \log \frac{1}{p_i}$, where $i$ runs over all states of the system. This entropy is equivalent (apart from the factor of $k_\mathrm{B}$) 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 $\Omega$ 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 constrained ensemble. The Gibbs entropy of a microcanonical ensemble is trivially equal to the Boltzmann entropy.

We now consider a thermal distribution (the *canonical* ensemble), where the probability of a state $\mathbf{x}$ is

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{E(\mathbf{x})}{k_{\rm B}T}\right). \tag{5.44}$$

With this canonical ensemble we can associate a corresponding microcanonical ensemble, an ensemble with total energy fixed to the mean energy of the canonical ensemble (fixed to within some precision $\epsilon$). Now, fixing the total energy to a precision $\epsilon$ is equivalent to fixing the value of $\log 1/P(\mathbf{x})$ 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 5.15 (p.96): The normalizing constant of the Cauchy distribution

$$P(x) = \frac{1}{Z} \frac{1}{x^2+1}$$

is

$$Z = \int_{-\infty}^{\infty} dx \, \frac{1}{x^2+1} = \left[\tan^{-1} x\right]_{-\infty}^{\infty} = \frac{\pi}{2} - \frac{-\pi}{2} = \pi. \tag{5.45}$$

The mean and variance of this distribution are both undefined. (The distribution is 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

$$P(z) = \frac{1}{\pi^2} \int_{-\infty}^{\infty} dx_1 \, \frac{1}{x_1^2+1} \frac{1}{(z-x_2)^2+1}, \tag{5.46}$$

which after a considerable labour using standard methods gives

$$P(z) = \frac{1}{\pi^2} 2 \frac{\pi}{z^2+4} = \frac{2}{\pi} \frac{1}{z^2+2^2}, \tag{5.47}$$

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 (5.47) 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 (5.47).

Solution to exercise 5.16 (p.96): The function $f(x)$ has inverse function

$$g(y) = y^{1/y}. \tag{5.48}$$

Note

$$\log g(y) = 1/y \log y. \tag{5.49}$$

I obtained a tentative graph of $f(x)$ by plotting $g(y)$ with $y$ along the vertical axis and $g(y)$ along the horizontal axis. The resulting graph suggests that $f(x)$ is single valued for $x \in (0, 1)$, and looks surprisingly well-behaved and ordinary. For $x \in (1, e^{1/e})$, $f(x)$ is two-valued. $f(\sqrt{2})$ is equal both to 2 and 4. For $x > e^{1/e}$ (which is about 1.44), $f(x)$ is infinite. However, it might be argued that this approach to sketching $f(x)$ is not valid, if we define $f$ as the limit of the sequence of functions $x$, $x^x$, $x^{x^x}, \ldots$; this sequence does not have a limit for $0 \le x \le (1/e)^e \simeq 0.07$ on account of a pitchfork bifurcation at $x = (1/e)^e$; and for $x \in (1, e^{1/e})$, the sequence's limit is single-valued – the lower of the two values sketched in the figure.

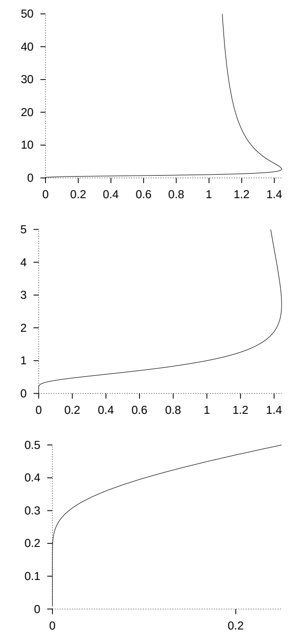


Figure 5.16. $f(x) = x^{x^{x^{x^{\cdot^{\cdot^{\cdot}}}}}}$, shown at three different scales.

# *About Chapter 6*

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 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.*

Imagine a rubber glove filled with water. If we compress two fingers of the glove, some other part of the glove has to expand, because the total volume of water is constant. [Water is essentially incompressible.] Similarly, when we shorten the codewords for some outcomes, there must be other codewords that get longer, if the scheme is not lossy. In this chapter we will discover the information-theoretic equivalent of water volume.

Before reading chapter 6, you should have worked on exercise 2.22 (p.41).

We will use the following notation for intervals:

$$\begin{array}{ll} x \in [1, 2) & \text{means that } x \geq 1 \text{ and } x < 2; \\ x \in (1, 2] & \text{means that } x > 1 \text{ and } x \leq 2. \end{array}$$

# 6

# 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.

The idea is that we can achieve compression, on average, by assigning *shorter* encodings to the more probable outcomes and *longer* encodings to the less probable.

The key issues are:

**What are the implications if a symbol code is *lossless*?** If some codewords are shortened, by how much do other codewords have to be lengthened?

**Making compression practical.** How can we ensure that a symbol code is easy to decode?

**Optimal symbol codes.** How should we assign codelengths to achieve the best compression, and what is the best achievable compression?

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 6.1: $\{0, 1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

Example 6.2: $\{0, 1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$.

## 6.1 Symbol codes

**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 $\{\mathtt{0},\mathtt{1}\}^+$. $c(x)$ 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 $\{\mathtt{0},\mathtt{1}\}^+$ obtained by concatenation, without punctutation, of the corresponding codewords:

$$c^+(x_1 x_2 \ldots x_N) = c(x_1)c(x_2)\ldots c(x_N). \tag{6.1}$$

[The term 'mapping' here is a synonym for 'function'.]

Example 6.3: 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 $C_0$:

$C_0$:

| $a_i$ | $c(a_i)$ | $l_i$ |
|---|---|---|
| a | 1000 | 4 |
| b | 0100 | 4 |
| c | 0010 | 4 |
| d | 0001 | 4 |

Using the extended code, we may encode `acdbac` as

$$c^+(\mathtt{acdbac}) = \mathtt{100000100001010010000010} \tag{6.2}$$

There are basic requirements for a useful symbol code. First, any encoded string must have a unique decoding. Second, the symbol code must be easy to decode. And third, the code should achieve as much compression as possible.

### *Any encoded string must have a unique decoding*

**A code $C(X)$ is uniquely decodeable** if, under the extended code $C^+$, no two distinct strings have the same encoding, i.e.,

$$\forall\, \mathbf{x},\mathbf{y} \in \mathcal{A}_X^+,\ \mathbf{x} \neq \mathbf{y} \ \Rightarrow\ c^+(\mathbf{x}) \neq c^+(\mathbf{y}). \tag{6.3}$$

The code $C_0$ defined above is an example of a uniquely decodeable code.

### *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 symbol 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* or *self-punctuating* 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 uniquely decodeable.

Prefix codes are also known as 'prefix-free codes' or 'prefix condition codes'.

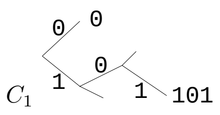
Prefix codes correspond to trees.

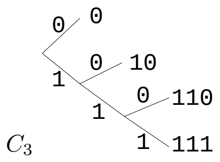
Example 6.4: The code $C_1 = \{\texttt{0}, \texttt{101}\}$ is a prefix code because `0` is not a prefix of `101`, nor is `101` a prefix of `0`.

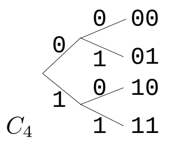
Example 6.5: Let $C_2 = \{\texttt{1}, \texttt{101}\}$. This code is not a prefix code because `1` is a prefix of `101`.

Example 6.6: The code $C_3 = \{\texttt{0}, \texttt{10}, \texttt{110}, \texttt{111}\}$ is a prefix code.

Example 6.7: The code $C_4 = \{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$ is a prefix code.







Prefix codes can be represented on binary trees. *Complete* prefix codes correspond to binary trees with no unused branches.

Solutions on p.119 Exercise 6.8:$^{A1}$ Is $C_2$ uniquely decodeable?

Example 6.9: Consider exercise 4.4 (p. 76) and figure 5.2 (p. 80). 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.

*The code should achieve as much compression as possible*

**The expected length** $L(C, X)$ of a symbol code $C$ for ensemble $X$ is

$$L(C, X) = \sum_{x \in \mathcal{A}_X} P(x)\, l(x). \tag{6.4}$$

We may also write this quantity as

$$L(C, X) = \sum_{i=1}^{I} p_i l_i \tag{6.5}$$

where $I = |\mathcal{A}_X|$.

Example 6.10: Let

$$\begin{array}{rrcl} & \mathcal{A}_X & = & \{\ \texttt{a},\ \texttt{b},\ \texttt{c},\ \texttt{d}\ \}, \\ \text{and} & \mathcal{P}_X & = & \{\ 1/2, 1/4, 1/8, 1/8\ \}, \end{array} \tag{6.6}$$

and consider the code $C_3$:

$C_3$:

| $a_i$ | $c(a_i)$ | $p_i$ | $\log_2 \frac{1}{p_i}$ | $l_i$ |
|---|---|---|---|---|
| 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 |

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`. $C_3$ 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 6.11: Consider the fixed length code for the same ensemble $X$, $C_4$:

$C_4$:

| $a_i$ | $c(a_i)$ | $p_i$ | $\log_2 \frac{1}{p_i}$ | $l_i$ |
|---|---|---|---|---|
| `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 6.12: Consider $C_5$:

$C_5$:

| $a_i$ | $c(a_i)$ | $p_i$ | $\log_2 \frac{1}{p_i}$ | $l_i$ |
|---|---|---|---|---|
| `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 |

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 6.13: Consider the code $C_6$:

$C_6$:

| $a_i$ | $c(a_i)$ | $p_i$ | $\log_2 \frac{1}{p_i}$ | $l_i$ |
|---|---|---|---|---|
| `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 (6.3).]

$C_6$ 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.

## 6.2 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 $\{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$, and shorten one of its codewords, for example $\texttt{00} \rightarrow \texttt{0}$, 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.

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 $\{\texttt{100}, \texttt{101}, \texttt{110}, \texttt{111}\}$. 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 that is $2^2$ 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,

$$\sum_i 2^{-l_i} \leq 1, \tag{6.7}$$

then the code may be uniquely decodeable. This inequality is the Kraft inequality.

**Kraft inequality.** For any uniquely decodeable code $C$ over the binary alphabet $\{0, 1\}$, the codeword lengths must satisfy:

$$\sum_{i=1}^{I} 2^{-l_i} \leq 1, \tag{6.8}$$

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; prefix codes are uniquely decodeable, and are easy to decode. 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.

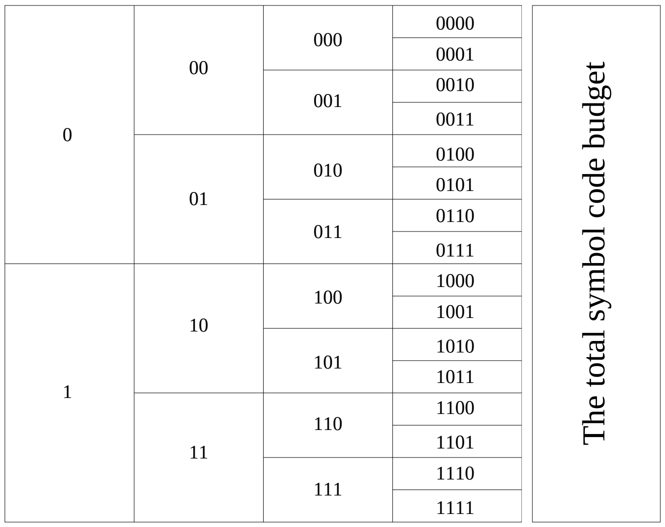


Figure 6.1. The symbol coding budget. The 'cost' $2^{-l}$ of each codeword (with length $l$) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.

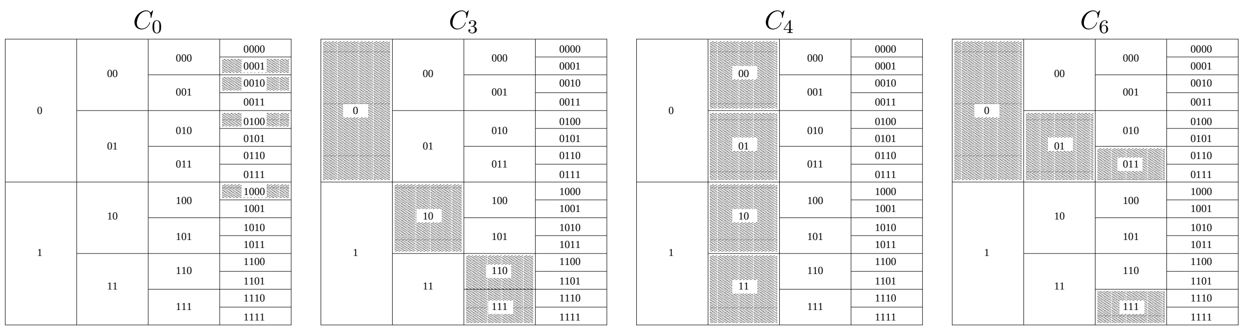


Figure 6.2. Selections of codewords made by codes $C_0, C_3, C_4$ and $C_6$ from section 6.1.

**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

$$S^N = \left[\sum_i 2^{-l_i}\right]^N = \sum_{i_1=1}^{I}\sum_{i_2=1}^{I}\ldots\sum_{i_N=1}^{I} 2^{-\left(l_{i_1}+l_{i_2}+\ldots l_{i_N}\right)} \tag{6.9}$$

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 $l_{\max} = \max_i l_i$:

$$S^N = \sum_{l=Nl_{\min}}^{Nl_{\max}} 2^{-l}A_l. \tag{6.10}$$

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

$$S^N = \sum_{l=Nl_{\min}}^{Nl_{\max}} 2^{-l}A_l \leq \sum_{l=Nl_{\min}}^{Nl_{\max}} 1 \ \leq \ Nl_{\max}. \tag{6.11}$$

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.119

Exercise 6.14:$^{B3}$ 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 6.1). 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 6.1 are illustrated in figure 6.2. 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 – because prefix codes correspond to trees. Notice that a *complete* prefix code corresponds to a *complete* tree having no unused branches.

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.

## 6.3 What's the most compression that we can hope for?

We wish to minimize the expected length of a code,

$$L(C, X) \quad = \quad \sum_i p_i l_i. \tag{6.12}$$

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's 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 $z \leq 1$:

$$\begin{aligned} L(C, X) \quad &= \quad \sum_i p_i l_i = \sum_i p_i \log 1/q_i - \log z & (6.13) \\ &\geq \quad \sum_i p_i \log 1/p_i - \log z & (6.14) \\ &\geq \quad H(X). & (6.15) \end{aligned}$$

The equality $L(C, X) = H(X)$ is achieved only if the Kraft equality $z = 1$ is satisfied, and if the codelengths satisfy $l_i = \log(1/p_i)$. $\square$

This is an important result so let's say it again:

**Optimal source codelengths.** The expected length is minimized and is equal to $H(X)$ only if the codelengths are equal to the *Shannon information contents*:

$$l_i = \log_2(1/p_i). \tag{6.16}$$

**Implicit probabilities defined by codelengths.** Conversely, any choice of codelengths $\{l_i\}$ *implicitly* defines a probability distribution $\{q_i\}$,

$$q_i \equiv 2^{-l_i}/z, \tag{6.17}$$

for which those codelengths would be the optimal codelengths. If the code is complete then $z = 1$ and the implicit probabilities are given by $q_i = 2^{-l_i}$.

## 6.4 How much can we compress?

So, we can't compress below the entropy. How close can we expect to get to the entropy?

**Theorem 6.1** Source coding theorem for symbol codes. *For an ensemble $X$ there exists a prefix code $C$ with expected length satisfying*

$$H(X) \leq L(C, X) < H(X) + 1. \tag{6.18}$$

Proof: We set the codelengths to integers slightly larger than the optimum lengths:

$$l_i = \lceil \log_2(1/p_i) \rceil \tag{6.19}$$

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.

$$\sum_i 2^{-l_i} = \sum_i 2^{-\lceil \log_2(1/p_i) \rceil} \leq \sum_i 2^{-\log_2(1/p_i)} = \sum_i p_i = 1. \tag{6.20}$$

Then we confirm

$$L(C, X) = \sum_i p_i \lceil \log(1/p_i) \rceil < \sum_i p_i (\log(1/p_i) + 1) = H(X) + 1. \tag{6.21}$$

□

*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.

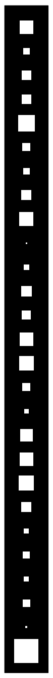
If the true probabilities are $\{p_i\}$ and we use a complete code with lengths $l_i$, we can view those lengths as defining implicit probabilities $q_i = 2^{-l_i}$. Continuing from equation (6.13), the average length is

$$L(C, X) = H(X) + \sum_i p_i \log p_i / q_i, \tag{6.22}$$

i.e., it exceeds the entropy by the Kullback–Leibler divergence $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q})$ (as defined on p.39).

## 6.5 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 figure 6.3?



| $x$ | $P(x)$ |
|---|---|
| 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 6.3. 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 root. This construction has the right spirit, as in the weighing problem, but it is not necessarily optimal; it achieves $L(C, X) \leq H(X) + 2$.

*The Huffman coding algorithm*

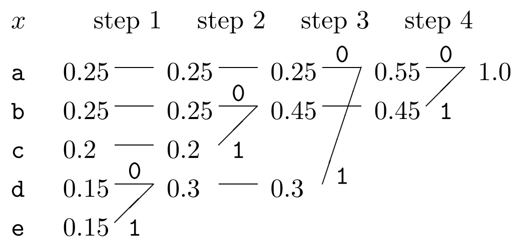
The trick is to construct the code *backwards* starting from the tails of the codewords; *we build the binary tree from its leaves.*

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 6.15: 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 binary digits 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 not depend on the choice.

Solutions on p.119

Exercise 6.16:$^{C3}$ Prove that there is no better symbol code for a source than the Huffman code.

Example 6.17: We can make a Huffman code for the probability distribution over the alphabet introduced in figure 2.1. The result is shown in figure 6.4. 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 |

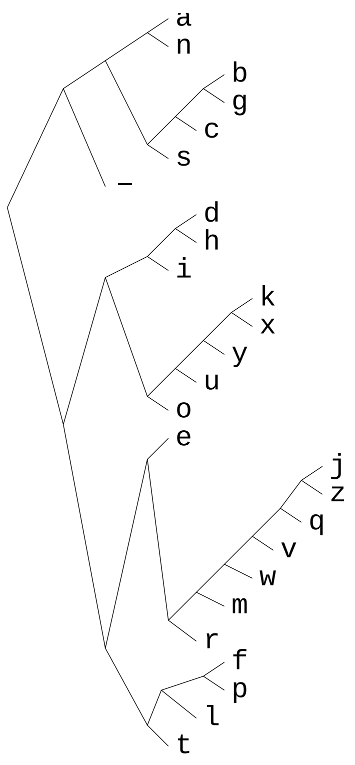


Figure 6.4. Huffman code for the English language ensemble (monogram statistics).

*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 6.18: Find the optimal binary symbol code for the ensemble:

$$\begin{array}{ll} \mathcal{A}_X = \{ & \texttt{a} \;,\; \texttt{b} \;,\; \texttt{c} \;,\; \texttt{d} \;,\; \texttt{e} \;,\; \texttt{f} \;,\; \texttt{g} \;\} \\ \mathcal{P}_X = \{ & 0.01, 0.24, 0.05, 0.20, 0.47, 0.01, 0.02 \} \end{array} . \tag{6.23}$$

Notice that a greedy top-down method can split this set into two subsets $\{\texttt{a},\texttt{b},\texttt{c},\texttt{d}\}$ and $\{\texttt{e},\texttt{f},\texttt{g}\}$ which both have probability 1/2, and that $\{\texttt{a},\texttt{b},\texttt{c},\texttt{d}\}$ can be divided into subsets $\{\texttt{a},\texttt{b}\}$ and $\{\texttt{c},\texttt{d}\}$, 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.

## 6.6 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.

### *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 6.1. 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 at least one bit per character, making a total cost of nine bits where virtually no information is being conveyed (0.13 bits in total, to be precise). 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 5. 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.

### *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.

## 6.7 Summary

**Kraft inequality.** If a code is *uniquely decodeable* its lengths must satisfy

$$\sum_i 2^{-l_i} \leq 1. \tag{6.24}$$

For any lengths satisfying the Kraft inequality, there exists a prefix code with those lengths.

**Optimal source codelengths for an ensemble** are equal to the Shannon information contents

$$l_i = \log_2 \frac{1}{p_i}, \tag{6.25}$$

and conversely, any choice of codelengths defines *implicit probabilities*

$$q_i = \frac{2^{-l_i}}{z}. \tag{6.26}$$

**The relative entropy** $D_{\mathrm{KL}}(\mathbf{p}||\mathbf{q})$ 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

$$H(X) \leq L(C, X) < H(X) + 1. \tag{6.27}$$

**The Huffman coding algorithm** generates an optimal symbol code iteratively. At each iteration, the two least probable symbols are combined.

## 6.8 Exercises

Solutions on p.120

Exercise 6.19:$^{B2}$ Is the code $\{\mathtt{00}, \mathtt{11}, \mathtt{0101}, \mathtt{111}, \mathtt{1010}, \mathtt{100100}, \mathtt{0110}\}$ uniquely decodeable?

Exercise 6.20:$^{B2}$ Is the ternary code $\{\mathtt{00}, \mathtt{012}, \mathtt{0110}, \mathtt{0112}, \mathtt{100}, \mathtt{201}, \mathtt{212}, \mathtt{22}\}$ uniquely decodeable?

Exercise 6.21:$^{A3}$ 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 6.22:$^{A2}$ 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 6.23:$^{C3}$ (Continuation of exercise 6.22.) 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

$$\mathbf{p} = \mu_1 \mathbf{q}^{(1)} + \mu_2 \mathbf{q}^{(2)} + \mu_3 \mathbf{q}^{(3)}, \tag{6.28}$$

where $\{\mu_i\}$ are positive.

Exercise 6.24:$^{B2}$ Make ensembles for which the difference between the entropy and the expected length of the Huffman code is as big as possible.

Exercise 6.25:$^{B2}$ 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 6.26:$^{B2}$ *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 6.27:$^{A2}$ Assume that a sequence of symbols from the ensemble $X$ introduced at the beginning of this chapter is compressed using 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. 123

Exercise 6.28:$^{B2}$ 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 6.29:$^{A3}$ Prove that this metacode is incomplete, and explain why this combined code is suboptimal.

# *Solutions to Chapter 6's exercises*

Solution to exercise 6.8 (p.105): Yes, $C_2 = \{\texttt{1}, \texttt{101}\}$ 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 $c(a_2) = \texttt{101}$ contains the symbol `0`.

Solution to exercise 6.14 (p.109): 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

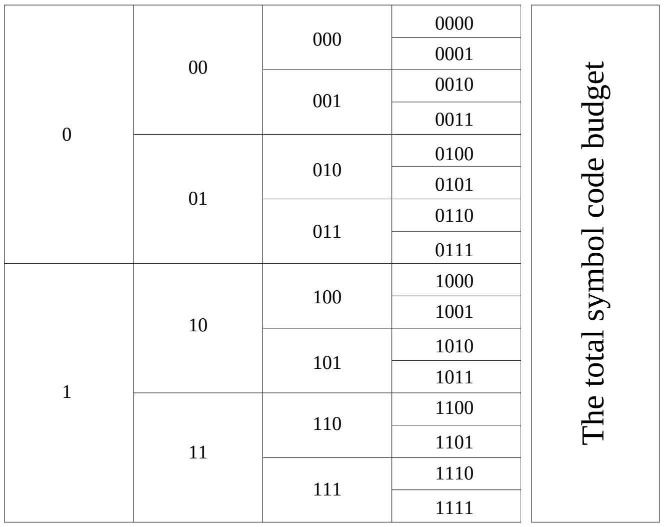


Figure 6.5. The codeword supermarket and the symbol coding budget. The 'cost' $2^{-l}$ of each codeword (with length $l$) is indicated by the size of the box it is written in. The total budget available when making a uniquely decodeable code is 1.

in figure 6.5 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 6.5. 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 $I$th 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 6.16 (p.112): 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. 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 of $a$ and $c$, so that $a$ 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. Therefore it is valid to give the two symbols with smallest probability equal encoded lengths. Huffman coding produces optimal symbol codes.

| symbol | probability | Huffman codewords | Rival code's codewords | Modified rival code |
|---|---|---|---|---|
| $a$ | $p_a$ ▭ | $\boxed{c_{\rm H}(a)}$ | $\boxed{c_{\rm R}(a)}$ | $\boxed{c_{\rm R}(c)}$ |
| $b$ | $p_b$ □ | $\boxed{c_{\rm H}(b)}$ | $\boxed{c_{\rm R}(b)}$ | $\boxed{c_{\rm R}(b)}$ |
| $c$ | $p_c$ ▭ | $\boxed{c_{\rm H}(c)}$ | $\boxed{c_{\rm R}(c)}$ | $\boxed{c_{\rm R}(a)}$ |

Figure 6.6. 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.

Solution to exercise 6.19 (p.116): 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 6.20 (p.116): The ternary code $\{$`00`, `012`, `0110`, `0112`, `100`, `201`, `212`, `22`$\}$ *is* uniquely decodeable because it is a prefix code.

Solution to exercise 6.21 (p.116): 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:

$$\{0000, 1000, 0100, 0010, 0001, 1100, 0110, 0011, 0101, 1010, 1001, 1110, 1101, 1011, 0111, 1111\} \rightarrow \{1, 3, 3, 3, 4, 6, 7, 7, 7, 7, 7, 9, 9, 9, 10, 10\}.$$

| $a_i$ | $p_i$ | $\log_2 \frac{1}{p_i}$ | $l_i$ | $c(a_i)$ |
|---|---|---|---|---|
| 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 6.7. Huffman code for $X^4$ when $p_0 = 0.6$. Column 4 shows the assigned codelengths and column 5 the codewords. Notice some strings whose probabilities are identical, e.g., the fourth and fifth, receive different codelengths.

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 6.7. The expected length is 3.92 bits, and the entropy is 3.88 bits.

Solution to exercise 6.22 (p.116): The set of probabilities $\{p_1, p_2, p_3, p_4\} = \{1/6, 1/6, 1/3, 1/3\}$ gives 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 $\{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$ or the code $\{\texttt{000}, \texttt{001}, \texttt{01}, \texttt{1}\}$. Both codes have expected length 2.

Another solution is $\{p_1, p_2, p_3, p_4\} = \{1/5, 1/5, 1/5, 2/5\}$.

And a third is $\{p_1, p_2, p_3, p_4\} = \{1/3, 1/3, 1/3, 0\}$.

Solution to exercise 6.23 (p.117): 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

$$p_1 = p_3 + p_4. \tag{6.29}$$

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 (6.29) for $\mathbf{p}$. Each choice of equalities gives rise to one of the set of three vectors

$$\{1/3, 1/3, 1/6, 1/6\},\ \{2/5, 1/5, 1/5, 1/5\} \text{ and } \{1/3, 1/3, 1/3, 0\}. \tag{6.30}$$

Solution to exercise 6.25 (p.117): 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 of length $N$ 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, ... with the last elements in the set being, say, two strings of equal maximum length 00 ... 01 and 00 ... 00. 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 6.26 (p.117): 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.

Solution to exercise 6.27 (p.117): 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

$$P(\text{bit is } 1) = \sum_i p_i f_i \tag{6.31}$$

$$= \frac{1}{2} \times 0 + \frac{1}{4} \times 1/2 + \frac{1}{8} \times 2/3 + \frac{1}{8} \times 1 = {}^1\!/\!_3.\text{"} \tag{6.32}$$

$C_3$:

| $a_i$ | $c(a_i)$ | $p_i$ | $l_i$ |
|---|---|---|---|
| a | 0 | 0.5 | 1 |
| b | 10 | 0.25 | 2 |
| c | 110 | 0.125 | 3 |
| d | 111 | 0.125 | 3 |

This answer is wrong because it falls for the busstop fallacy, which was introduced in exercise 2.24 (p.42): 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 and the next bus. The second 'average' is twice as big as the first 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 (6.31). 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

$$\sum_i p_i f_i l_i; \tag{6.33}$$

and the expected total number of bits added per symbol is

$$\sum_i p_i l_i. \tag{6.34}$$

So the fraction of `1`s in the transmitted string is

$$\begin{aligned} P(\text{bit is } \mathtt{1}) &= \frac{\sum_i p_i f_i l_i}{\sum_i p_i l_i} & (6.35) \\ &= \frac{\frac{1}{2}\times 0 + \frac{1}{4}\times 1 + \frac{1}{8}\times 2 + \frac{1}{8}\times 3}{7/4} = \frac{(7/8)}{(7/4)} = 1/2. \end{aligned}$$

For a general symbol code and a general ensemble, the expectation (6.35) 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 6.28 (p.118): 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 $r(q-1)+1$, for some integer $r$. 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 6.29 (p.118): 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

$$l'(x) = \log_2 K + \min_k l_k(x). \tag{6.36}$$

We compute the Kraft sum:

$$S = \sum_x 2^{-l'(x)} = \frac{1}{K}\sum_x 2^{-\min_k l_k(x)} \tag{6.37}$$

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 the metacode sends via code $k$. Then

$$S = \frac{1}{K}\sum_k \sum_{x\in\mathcal{A}_k} 2^{-l_k(x)}. \tag{6.38}$$

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

$$\sum_{x\in\mathcal{A}_k} 2^{-l_k(x)} \leq 1, \tag{6.39}$$

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

$$S \leq \frac{1}{K}\sum_{k=1}^K 1 = 1, \tag{6.40}$$

with equality only if equation (6.39) 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 (6.39) holding for *all* $k$. So

$$S < 1. \tag{6.41}$$

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 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 7*

Before reading chapter 7, 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).

# 7

# *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.

## 7.1 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 consists of 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, 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 $26 \times 27^L$ 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 now.

## 7.2 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 6.6).

These defects are rectified by *arithmetic codes*, which were invented by Elias (Abramson 1963), 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 game) to interpret the binary string.

Let the source alphabet be $\mathcal{A}_X = \{a_1, \ldots, a_I\}$, and let the $I$th 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 $P(x_n{=}a_i | x_1, \ldots, x_{n-1})$.

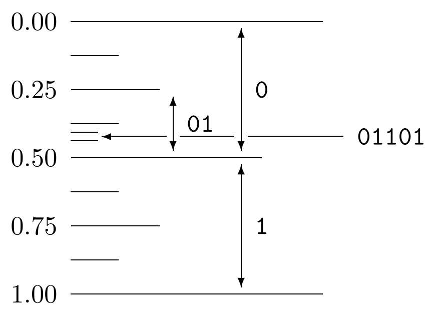


Figure 7.1. Binary strings define real intervals within the real line [0,1). We first encountered a picture like this when we discussed the symbol-code supermarket in chapter 6.

## 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 two million decimal places.

Two million decimal digits, because each byte translates into a little more than two decimal digits.

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 7.2.

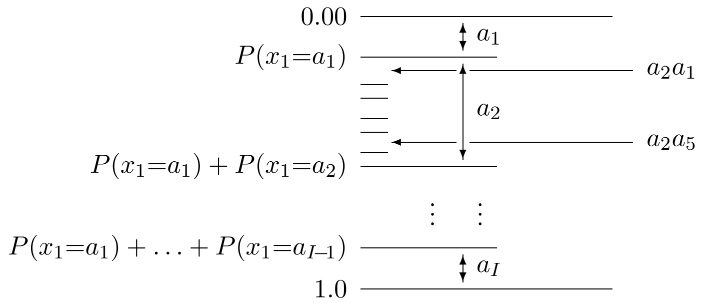

0.00
$a_1$
$P(x_1{=}a_1)$
$a_2a_1$
$a_2$
$a_2a_5$
$P(x_1{=}a_1) + P(x_1{=}a_2)$
$P(x_1{=}a_1) + \ldots + P(x_1{=}a_{I-1})$
$a_I$
1.0

Figure 7.2. 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

```
u := 0.0
v := 1.0
p := v − u
for n = 1 to N (
        Compute $\{R_n(i|x_1,\ldots,x_{n-1})\}_{i=0}^{I}$ using equation (7.3)
        v := $u + pR_n(x_n|x_1,\ldots,x_{n-1})$
        u := $u + pQ_n(x_n|x_1,\ldots,x_{n-1})$
        p := v − u
)
```

Algorithm 7.1. Arithmetic coding. Iterative procedure to find the interval $[u, v)$ for the string $x_1x_2\ldots x_N$.

$P(x_2{=}a_j|x_1{=}a_i)$. Indeed the length of the interval $a_ia_j$ will be precisely the joint probability

$$P(x_1{=}a_i, x_2{=}a_j) = P(x_1{=}a_i)P(x_2{=}a_j|x_1{=}a_i). \tag{7.1}$$

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 7.2 can be written out explicitly as follows. The intervals are defined in terms of the lower and upper cumulative probabilities

$$Q_n(a_i|x_1,\ldots,x_{n-1}) \equiv \sum_{i'=1}^{i-1} P(x_n{=}a_{i'}|x_1,\ldots,x_{n-1}), \tag{7.2}$$

$$R_n(a_i|x_1,\ldots,x_{n-1}) \equiv \sum_{i'=1}^{i} P(x_n{=}a_{i'}|x_1,\ldots,x_{n-1}). \tag{7.3}$$

As the $n$th 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

$$a_1 \leftrightarrow [Q_1(a_1), R_1(a_1)) = [0, P(x_1{=}a_1)), \tag{7.4}$$

$$a_2 \leftrightarrow [Q_1(a_2), R_1(a_2)) = [P(x{=}a_1), P(x{=}a_1) + P(x{=}a_2)), \tag{7.5}$$

and

$$a_I \leftrightarrow [Q_1(a_I), R_1(a_I)) = [P(x_1{=}a_1) + \ldots + P(x_1{=}a_{I-1}), 1.0). \tag{7.6}$$

Algorithm 7.1 describes the general procedure.

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.

## *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 2.8 (p.35) and section 3.2 (p. 58).] 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, '□'. 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□`'. 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 | | |
|---|---|---|---|
| | $P(\texttt{a})$=0.425 | $P(\texttt{b})$=0.425 | $P(\square)$=0.15 |
| `b` | $P(\texttt{a}\|\texttt{b})$=0.28 | $P(\texttt{b}\|\texttt{b})$=0.57 | $P(\square\|\texttt{b})$=0.15 |
| `bb` | $P(\texttt{a}\|\texttt{bb})$=0.21 | $P(\texttt{b}\|\texttt{bb})$=0.64 | $P(\square\|\texttt{bb})$=0.15 |
| `bbb` | $P(\texttt{a}\|\texttt{bbb})$=0.17 | $P(\texttt{b}\|\texttt{bbb})$=0.68 | $P(\square\|\texttt{bbb})$=0.15 |
| `bbba` | $P(\texttt{a}\|\texttt{bbba})$=0.28 | $P(\texttt{b}\|\texttt{bbba})$=0.57 | $P(\square\|\texttt{bbba})$=0.15 |

The corresponding intervals are shown in figure 7.3. The interval `b` is the middle 0.425 of $[0, 1)$. The interval `bb` is the middle 0.567 of `b`, and so forth.

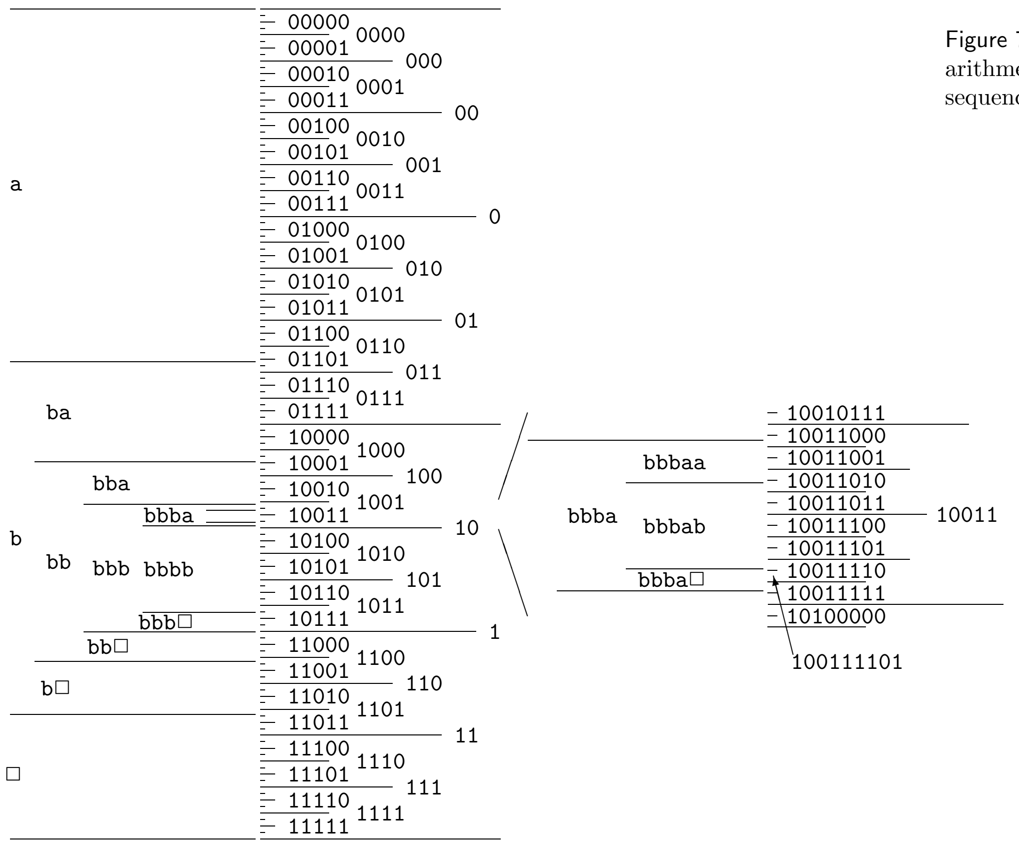


Figure 7.3. Illustration of the arithmetic coding process as the sequence `bbba□` is transmitted

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 '□' arrives, we need a procedure for terminating the encoding. Magnifying the interval '`bbba`□' (figure 7.3, right) we note that '`100111101`' is wholly contained by `bbba`□, so the encoding can be completed by appending '`11101`'.

Solutions on p.143

Exercise 7.1:$^{A2}$ 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 $P(\texttt{a}), P(\texttt{b}), P(\Box)$ are computed using the 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 $P(\texttt{a}|\texttt{b}), P(\texttt{b}|\texttt{b}), P(\Box|\texttt{b})$ 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 needed to compute only $N|\mathcal{A}|$ 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 6.6), 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 source alphabet and any encoded alphabet. The size of the source alphabet and the encoded 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 7.3 was generated using a simple model that always assigns a probability of 0.15 to $\Box$, and assigns the remaining 0.85 to $\mathtt{a}$ and $\mathtt{b}$, divided in proportion to probabilities given by Laplace's rule, which we encountered in exercise 2.9 (p.36),

$$P_{\rm L}(\mathtt{a}|x_1,\ldots,x_{n-1}) = (F_{\mathtt{a}}+1)/(F_{\mathtt{a}}+F_{\mathtt{b}}+2), \tag{7.7}$$

where $F_{\mathtt{a}}(x_1,\ldots,x_{n-1})$ is the number of times that $\mathtt{a}$ has occurred so far, and $F_{\mathtt{b}}$ is the count of $\mathtt{b}$s. These predictions corresponds to a simple Bayesian model that expects and adapts to a non-equal frequency of use of the source symbols $\mathtt{a}$ and $\mathtt{b}$ within a file.

Figure 7.4 displays the intervals corresponding to a large number of strings. Note that if the string so far has contained a large number of $\mathtt{b}$s then the probability of $\mathtt{b}$ relative to $\mathtt{a}$ is increased, and conversely if many $\mathtt{a}$s occur then $\mathtt{a}$s are made more probable. Larger intervals, remember, require fewer bits to encode.

### *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, $P(\mathtt{a}|\mathbf{s}{=}\mathtt{baa})$. An analogy for this model, as I indicated at the start, is the tossing of a bent coin. A bent coin labelled $\mathtt{a}$ and $\mathtt{b}$ is tossed some number of times $l$, which we don't know beforehand. The coin's probability of coming up $\mathtt{a}$ 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 $\mathbf{s} = \mathtt{baaba}\Box$ indicates that $l$ was 5 and the sequence of outcomes was $\mathtt{baaba}$.

1. It is assumed that the length of the string $l$ has an exponential probability distribution
$$P(l) = (1-p_{\Box})^l p_{\Box}. \tag{7.8}$$
This distribution corresponds to assuming a constant probability $p_{\Box}$ for the termination symbol '$\Box$' at each character.
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 $\mathtt{a}$ 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 $\mathbf{s}$ that contains $\{F_{\mathtt{a}}, F_{\mathtt{b}}\}$ counts of the two outcomes is the Bernoulli distribution
$$P(\mathbf{s}|p_{\mathtt{a}}, F) = p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}}. \tag{7.9}$$

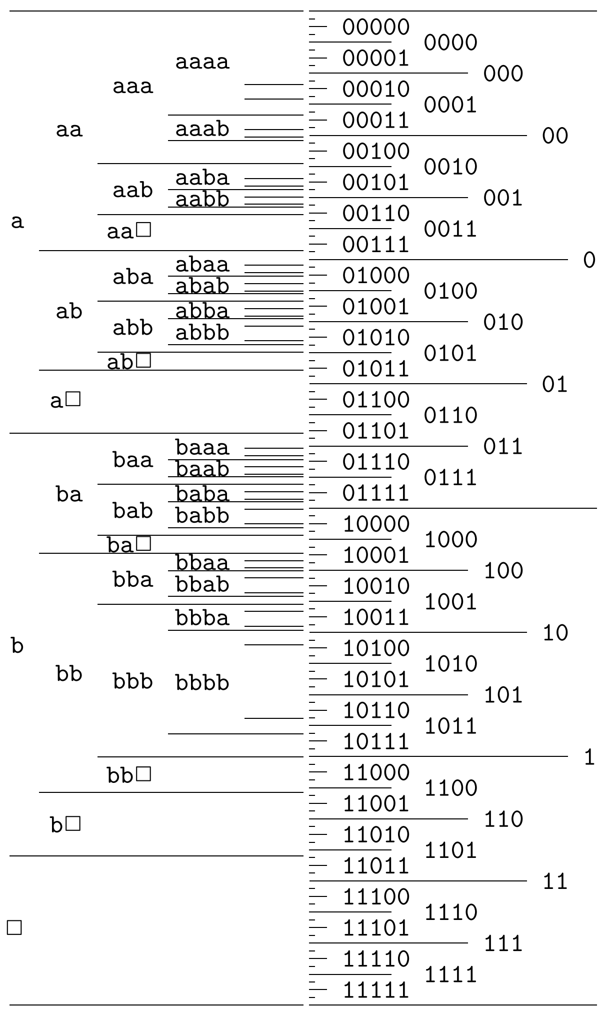


Figure 7.4. Illustration of the intervals defined by a simple Bayesian probabilistic model. The size of an intervals is proportional to the probability of the string. This model anticipates that the source is likely to be biased towards one of a and b, so sequences having lots of as or lots of bs have larger intervals than sequences of the same length that are 50:50 as and bs.

3. We assume a uniform prior distribution for $p_{\mathtt{a}}$,

   $$P(p_{\mathtt{a}}) = 1, \quad p_{\mathtt{a}} \in [0,1] \tag{7.10}$$

   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 as and $F_{\mathtt{b}}$ are bs, 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 '□'; this last assumption is just for convenience so we don't have factors of $(1 - p_{\square})$ sitting around).

The posterior probability of $p_{\mathtt{a}}$ given an unterminated string $\mathbf{s}$ of length $F$ that has counts $\{F_{\mathtt{a}}, F_{\mathtt{b}}\}$ is, by Bayes' theorem,

$$P(p_{\mathtt{a}}|\mathbf{s}, F) = \frac{P(\mathbf{s}|p_{\mathtt{a}}, F)P(p_{\mathtt{a}})}{P(\mathbf{s}|F)}. \tag{7.11}$$

The factor $P(\mathbf{s}|p_{\mathtt{a}}, F)$, which as a function of $p_{\mathtt{a}}$ is known as the likelihood function, was given in equation (7.9); the prior $P(p_{\mathtt{a}})$ was given in equation (7.10). Our inference of $p_{\mathtt{a}}$ is thus given by:

$$P(p_{\mathtt{a}}|\mathbf{s}, F) = \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}}}{P(\mathbf{s}|F)}. \tag{7.12}$$

The normalizing constant is given by the beta integral

$$P(\mathbf{s}|F) = \int_0^1 dp_{\mathtt{a}}\, p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}} P(p_{\mathtt{a}}) = \frac{\Gamma(F_{\mathtt{a}}+1)\Gamma(F_{\mathtt{b}}+1)}{\Gamma(F_{\mathtt{a}}+F_{\mathtt{b}}+2)} = \frac{F_{\mathtt{a}}!F_{\mathtt{b}}!}{(F_{\mathtt{a}}+F_{\mathtt{b}}+1)!}. \tag{7.13}$$

Exercise 7.2: 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,

$$P(\mathtt{a}|\mathbf{s}, F) = \int dp_{\mathtt{a}}\, P(\mathtt{a}|p_{\mathtt{a}})P(p_{\mathtt{a}}|\mathbf{s}, F). \tag{7.14}$$

The probability of an a given $p_{\mathtt{a}}$ is simply $p_{\mathtt{a}}$, so

$$P(\mathtt{a}|\mathbf{s}, F) = \int dp_{\mathtt{a}}\, p_{\mathtt{a}} \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}}}{P(\mathbf{s}|F)} \tag{7.15}$$

$$= \int dp_{\mathtt{a}}\, \frac{p_{\mathtt{a}}^{F_{\mathtt{a}}+1}(1-p_{\mathtt{a}})^{F_{\mathtt{b}}}}{P(\mathbf{s}|F)} \tag{7.16}$$

$$= \left[\frac{(F_{\mathtt{a}}+1)!F_{\mathtt{b}}!}{(F_{\mathtt{a}}+F_{\mathtt{b}}+2)!}\right] \Big/ \left[\frac{F_{\mathtt{a}}!F_{\mathtt{b}}!}{(F_{\mathtt{a}}+F_{\mathtt{b}}+1)!}\right] = \frac{F_{\mathtt{a}}+1}{F_{\mathtt{a}}+F_{\mathtt{b}}+2}, \tag{7.17}$$

which is Laplace's rule.

Exercise 7.3:$^{B3}$ Compare the expected message length when an ASCII file is compressed by the following three 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.**

$$P_{\mathrm{L}}(a|x_1,\ldots,x_{n-1}) = \frac{F_{\mathtt{a}}+1}{\sum_{\mathtt{a}'}(F_{\mathtt{a}'}+1)}. \tag{7.18}$$

**Arithmetic code using a Dirichlet model.** This model's predictions are:

$$P_{\mathrm{L}}(a|x_1,\ldots,x_{n-1}) = \frac{F_{\mathtt{a}}+\alpha}{\sum_{\mathtt{a}'}(F_{\mathtt{a}'}+\alpha)}, \tag{7.19}$$

where $\alpha$ 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.

## 7.3 Further applications of arithmetic coding

### *Efficient generation of random samples*

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 7.4:$^{A2}$ 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?

*Efficient data-entry devices*

When we enter text into a computer, we make gestures of some sort – maybe we tap a keyboard, or scribble with a pointer, or click with a mouse; an *efficient* text entry system is one where the number of gestures required to enter a given text string is *small*. Text entry can be viewed as an inverse process to data compression. In data compression, the aim is to map a given text string into a *small* number of bits. We have developed a which functions as an information-efficient text entry device driven by continuous pointing gestures (Ward *et al.* 2000). In this system, called Dasher, the user zooms in on the unit interval to locate the interval corresponding to their intended string, in the same style as figure 7.3. A language model (as used in text compression) controls the sizes of the intervals such that probable strings are quick and easy to identify. After an hour's practice, a typical novice user can write as fast with one finger driving Dasher as half their ten-finger QWERTY typing speed. You can download Dasher for various platforms from `www.inference.phy.cam.ac.uk/dasher/`.

## 7.4 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: $\lambda$,1,0,11,01,010,00,10,.... We include the empty substring $\lambda$ 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 $n$th 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 | $\lambda$ | 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 empty, because, given that there is only one substring on the stack – the string $\lambda$ – 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 7.5:$^{B2}$ 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 7.6:$^{B2}$ Encode the string 000000000000100000000000 using the basic Lempel-Ziv algorithm described above.

Exercise 7.7:$^{B2}$ 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, does not match 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 may, for many sources, be unfeasibly long, because the number of typical substrings

[1]Need to clarify this. It means the source is memoryless on sufficiently long timescales.

that need memorizing may be enormous. 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.

## 7.5 Demonstration

A demonstration arithmetic-coding software package written by Radford Neal[2] 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 7.2. The results using this Laplace model should be viewed as a basic benchmark since it is 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.[3] In fact it uses a carefully designed approximate arithmetic coder for binary alphabets called the Z-coder (Bottou *et al.* 1998), which is much faster than the arithmetic coding software described above. One of the neat tricks the Z-coder uses is the adaptive model only adapts occasionally (to save on computer time), with the decision about when to adapt being pseudo-randomly controlled by whether the arithmetic encoder emitted a bit.

The JBIG image compression standard for 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*, which makes use of a neat hack called the *Burrows-Wheeler transform* (Burrows and Wheeler 1994). This method is not based on an explicit probabilistic model, and it only works well for files larger than several thousand characters, but

[2]`ftp://ftp.cs.toronto.edu/pub/radford/www/ac.software.html`
[3]`DjVu` is described at `http://djvu.research.att.com/`

in practice it is a very effective compressor: it works well for files in which the context of a character is a good predictor for the character. Maybe I'll describe it in a future edition of this book.[4]

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 |

[4]There is a lot of information about the Burrows-Wheeler transform on the net. `http://dogma.net/DataCompression/BWT.shtml`

## 7.6 Summary

In the last three chapters we have studied three classes of data compression codes.

**Fixed-length block codes** (Chapter 5). 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 6). 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 $D_{\mathrm{KL}}$ 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 may be coded into a smaller number of bits. This property could only be obtained using a symbol code if the source stream were somehow chopped 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.

## 7.7 Problems

Solutions on p.144

Exercise 7.8:$^{A2}$ 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 7.9:$^{B2}$ 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 7.10:$^{B2}$ 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 7.11:$^{A2}$ Use a modified Lempel-Ziv algorithm in which, as discussed on p.138, 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 7.12:$^{B2}$ 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 7.13:$^{B3}$ Give examples of simple sources that have low entropy but would not be compressed well by the Lempel-Ziv algorithm.

# *Solutions to Chapter 7's exercises*

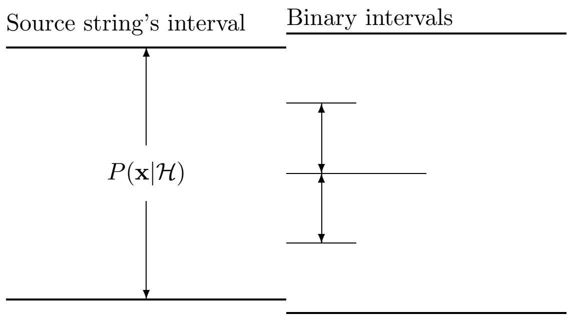


Figure 7.5. Termination of arithmetic coding in the worst case, where there is a two bit overhead. Either of the two binary intervals marked on the right hand side may be chosen. These binary intervals are no smaller than $P(\mathbf{x}|\mathcal{H})/4$.

Solution to exercise 7.1 (p.132): 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 7.5. 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.

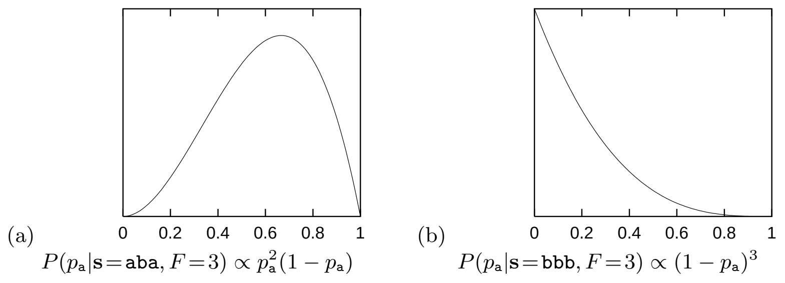


Figure 7.6. Posterior probability for the bias $p_a$ of a bent coin given two different data sets.

Solution to exercise 2.9 (p.36):

(a) $P(p_{\mathsf{a}}|\mathbf{s}{=}\mathtt{aba}, F{=}3) \propto p_{\mathsf{a}}^2(1-p_{\mathsf{a}})$. The most probable value of $p_{\mathsf{a}}$ (i.e., the value that maximizes the posterior probability density) is 2/3. The mean value of $p_{\mathsf{a}}$ is 3/5.

See figure 7.6(a).

(b) $P(p_{\mathtt{a}}|\mathbf{s}=\mathtt{bbb}, F=3) \propto (1-p_{\mathtt{a}})^3$. The most probable value of $p_{\mathtt{a}}$ (i.e., the value that maximizes the posterior probability density) is 0. The mean value of $p_{\mathtt{a}}$ is 1/5.

See figure 7.6(b).

Solution to exercise 7.4 (p.136): 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 7.5 (p.138): 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 7.6 (p.138): The encoding is `010100110010110001100`, which comes from the parsing

$$\mathtt{0, 00, 000, 0000, 001, 00000, 000000} \tag{7.20}$$

which is encoded thus

$$\mathtt{(,0), (1,0), (10,0), (11,0), (010,1), (100,0), (110,0).} \tag{7.21}$$

Solution to exercise 7.7 (p.138): The decoding is `0100001000100010101000001`.

Solution to exercise 7.8 (p.142): Figure 7.7 shows the left hand side of 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 7.7. Arithmetic encoder for binary strings of length $N=5$ with fixed weight $K=2$. (The right hand side, a regular binary scale, has been omitted.)

Solution to exercise 7.9 (p.142): 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 7.10 (p.142): 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 7.8.

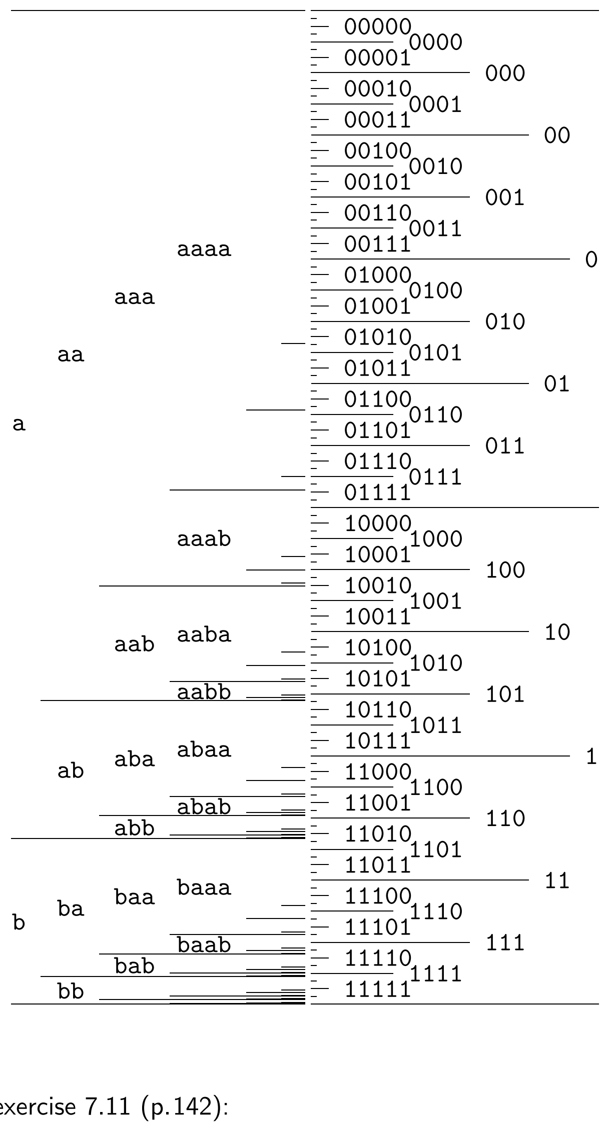


Figure 7.8. Arithmetic encoder for a sparse source with $f = 1/6$.

Solution to exercise 7.11 (p.142): The encoding is 00**1**10101101**0**00001101001010000**1**1, coming from the following parsed message:

$$(,0),(0,\mathbf{1}),(1,0),(10,1),(10,\mathbf{0}),(00,0),(011,0),(100,1),(010,0),(001,\mathbf{1}) \tag{7.22}$$

The highlighted symbols would be omitted in the further improved coding system.

Solution to exercise 7.12 (p.142): 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.

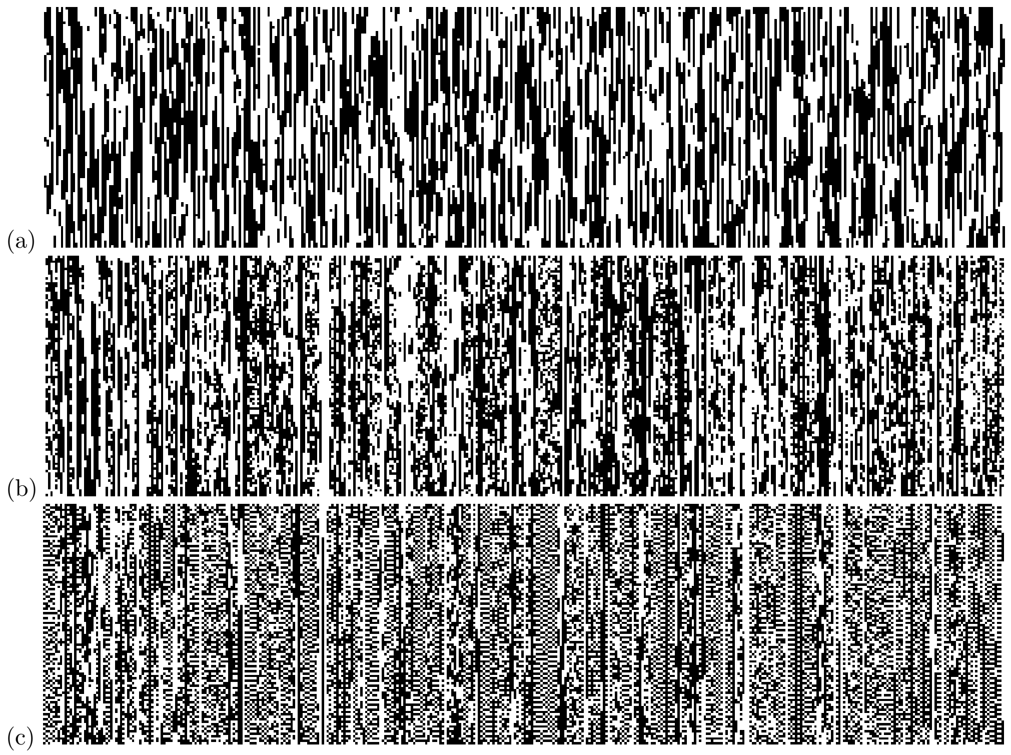


Figure 7.9. Three sources with low entropy that are not well compressed by Lempel-Ziv. The bit sequence is read from left to right. The image width is 400 pixels in each case.
(a) Each line differs from the line above in $p$ =5% of its bits.
(b) Each column $c$ has its own transition probability $p_c$ such that successive vertical bits are identical with probability $p_c$. The probabilities $p_c$ are drawn from a uniform distribution over $[0, 0.5]$.
(c) As in b, but the probabilities $p_c$ are drawn from a uniform distribution over $[0, 1]$.

Solution to exercise 7.13 (p.142): 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

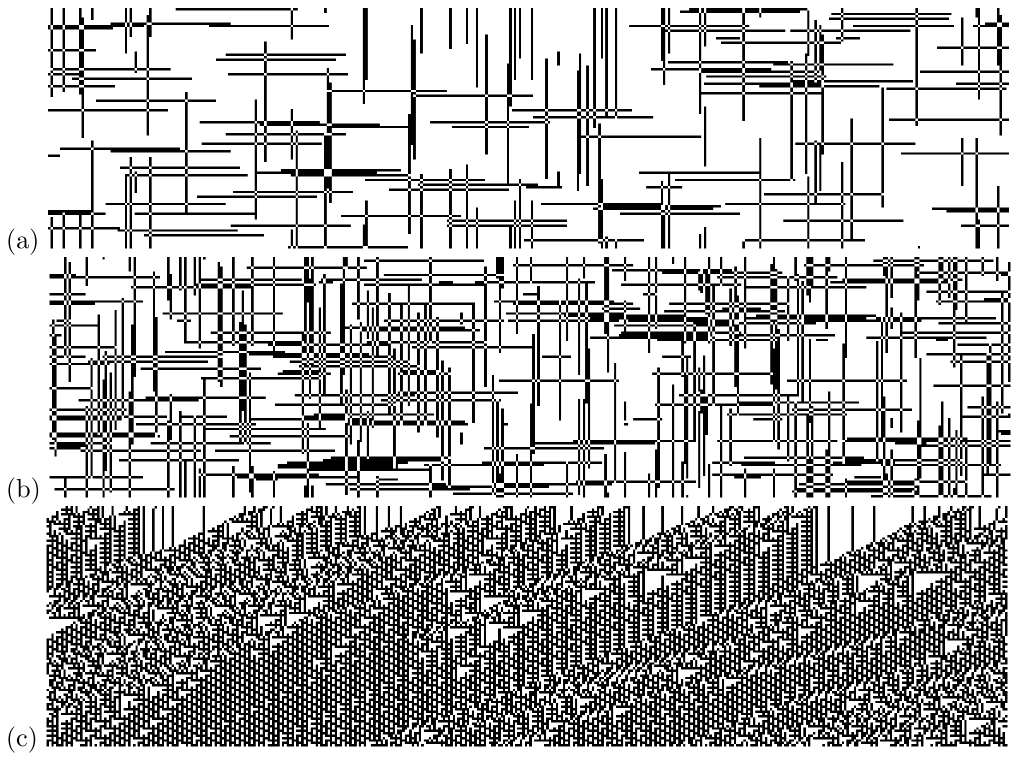


Figure 7.10. Three more sources with low entropy that are not optimally compressed by Lempel-Ziv. (a,b) Textures consisting of horizontal and vertical pins dropped at random on the plane. (c) The 100-step time-history of a cellular automaton with 400 cells.

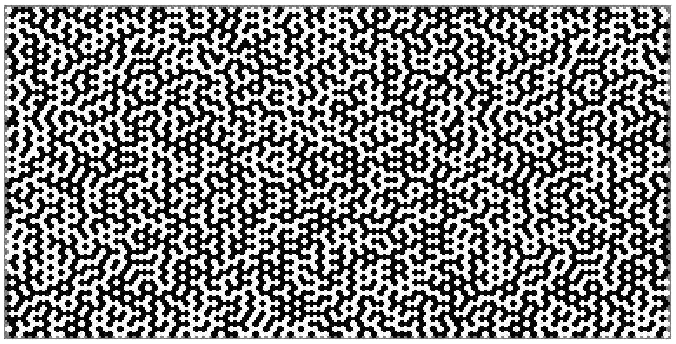


Figure 7.11. Frustrated triangular Ising model in one of its ground states.

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 $\{0, 1, \ldots, 9, -, :, \Box\}$. 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 :*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.

(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 7.9). 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 7.10a,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 7.11), which we will discuss in chapter 32. Like figure 7.10, this binary image has interesting correlations in two directions.

(f) Cellular automata – figure 7.10(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 about $2^{100}$ iterations.

In contrast, the JBIG compression method, which models the probability of a pixel given its local context and uses arithmetic coding, would do a good job on these images.

(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.

# 8

# Further exercises on data compression

The following exercises may be skipped by the reader who is eager to learn about noisy channels.

Exercise 8.1:$^{A3}$ Consider a Gaussian distribution in $N$ dimensions,

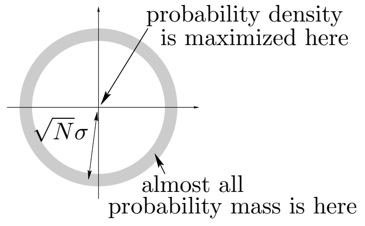
$$P(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{\sum_n x_n^2}{2\sigma^2}\right). \tag{8.1}$$

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, $r^2 = \left(\sum_n x_n^2\right)$.

You may find helpful the integral

$$\int dx \, \frac{1}{(2\pi\sigma^2)^{1/2}} \, x^4 \exp\left(-\frac{x^2}{2\sigma^2}\right) = 3\sigma^4, \tag{8.2}$$

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 (8.1) 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 8.2:$^{B3}$ 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 average information contents of the reversed language and the forward language?

Exercise 8.3:$^{A2}$ 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 8.4:$^{A2}$ 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 8.5:$^{A2}$ (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 8.6:$^{B3}$ (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 $\{\mathtt{0}, \mathtt{1}\}$ 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 $n$th 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

$$L(\mathbf{p}) = \sum_n p_n l_n \tag{8.3}$$

and the entropy *per symbol* to be

$$H(\mathbf{p}) = \sum_n p_n \log_2 \frac{1}{p_n}, \tag{8.4}$$

show that for the average information rate *per second* to be maximized, the symbols must be used with probabilties of the form

$$p_n = \frac{1}{Z} 2^{-\beta l_n} \tag{8.5}$$

where $Z = \sum_n 2^{-\beta l_n}$ and $\beta$ satisfies the implicit equation

$$\beta = \frac{H(\mathbf{p})}{L(\mathbf{p})}, \tag{8.6}$$

that is, $\beta$ is the rate of communication. Show that these two equations (8.5, 8.6) imply that $\beta$ must be set such that

$$\log Z = 0. \tag{8.7}$$

Assuming that the channel has the property

$$l_n = n \text{ seconds}, \tag{8.8}$$

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 (8.5, 8.7) 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 (8.8)?

Exercise 8.7:$^{B1}$ How many bits does it take to shuffle a pack of cards?

[In case this is not clear, here's the long-winded version: imagine using a random number generator to generate perfect shuffles of a deck of cards. What is the smallest number of random bits needed per shuffle?]

Exercise 8.8:$^{B2}$ 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 8.9:$^{B2}$ 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

$$\texttt{123}\Box, \tag{8.9}$$

and the roman sequence is

$$\texttt{CXXIII}\Box. \tag{8.10}$$

Each of these keypads defines a code mapping the 3599 cooking times from 0:01 to 59:59 into a string of symbols.

| Arabic | | | Roman | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | M | X | |
| 4 | 5 | 6 | C | I | □ |
| 7 | 8 | 9 | | | |
| | 0 | □ | | | |

(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 cooking-time-encoding system for a microwave oven.

Exercise 8.10:$^{C3}$ 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 8's exercises

Solution to exercise 8.1 (p.150): For a one–dimensional Gaussian, the variance of $x$, $\mathcal{E}[x^2]$, is $\sigma^2$. So the mean value of $r^2$ in $N$ dimensions, since the components of $\mathbf{x}$ are independent random variables, is

$$\mathcal{E}[r^2] = N\sigma^2. \tag{8.11}$$

The variance of $r^2$, similarly, is $N$ times the variance of $x^2$, where $x$ is a one-dimensional Gaussian variable.

$$\mathrm{var}(x^2) = \int dx \, \frac{1}{(2\pi\sigma^2)^{1/2}} x^4 \exp\left(-\frac{x^2}{2\sigma^2}\right) - \sigma^4. \tag{8.12}$$

The integral is found to be $3\sigma^4$ (equation (8.2)), 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 $\mathbf{x}_{\text{shell}}$ where $r = \sqrt{N}\sigma$ is

$$P(\mathbf{x}_{\text{shell}}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N\sigma^2}{2\sigma^2}\right) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2}\right). \tag{8.13}$$

Whereas the probability density at the origin is

$$P(\mathbf{x}{=}0) = \frac{1}{(2\pi\sigma^2)^{N/2}}. \tag{8.14}$$

Thus $P(\mathbf{x}_{\text{shell}})/P(\mathbf{x}{=}0) = \exp(-N/2)$. 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 8.2 (p.150): If we write down a language model for strings in forward-English, the same model defines a probability distribution over strings of backward English. The probability distributions have identical entropy, so the average information contents of the reversed language and the forward language are equal.

Solution to exercise 8.3 (p.150): Using the Huffman coding algorithm, we arrive at the answer shown, which is unique (apart from 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 8.5 (p.151): $470 \pm 30$.

Solution to exercise 8.9 (p.152):

(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 □.

The roman code is incomplete because

i. The key □ is an illegal first symbol.

ii. Some times can be produced by several symbol-strings. A time such as 1:23 can be entered as `CXXIII`□ or as `IICIXX`□.

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.

(c) 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.

(d) 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 $s = \sum_i x_i$.

(e) 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 | 0.01 | 0.1 | 0.01 | 0.1 | 0.5 | 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 | $\epsilon$ |

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 (1:30 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.

(f) 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 $\boxed{\texttt{--}}$□, $\boxed{\texttt{-}}$□, □, $\boxed{\texttt{+}}$□, and $\boxed{\texttt{++}}$□ respectively.

Solution to exercise 8.7 (p.152): There are 52! orderings of a pack of cards, so the minimum number of bits required to make a perfect shuffle is $\log_2(52!) \simeq 226$ bits.

Solution to exercise 8.8 (p.152): (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

$$\log_2 \binom{39}{13} \simeq 33\,\text{bits}. \tag{8.15}$$

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. Discussed in a later chapter.]

(b) The maximum number of bits is equal to 35, the number of distinct bids in the list $1♣ \ldots 7NT$. 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 that 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 8.4 (p.151): 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* *

## 8.1 Solution to exercise 8.10 (p.153)

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., $c_b(5) =$ `101`, $c_b(45) =$ `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., $c_B(5) =$ `01`, $c_B(45) =$ `01101` and $c_B(1) = \lambda$ (where $\lambda$ 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 $n-1$ `0`s followed 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 $C_\alpha$.** 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 $n$.

$$c_\alpha(n) = c_U[l_b(n)]c_B(n) \tag{8.16}$$

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 | $\overline{\texttt{1}}$ |
| 2 | `10` | 2 | $\overline{\texttt{01}}\texttt{0}$ |
| 3 | `11` | 2 | $\overline{\texttt{01}}\texttt{1}$ |
| 4 | `100` | 3 | $\overline{\texttt{001}}\texttt{00}$ |
| 5 | `101` | 3 | $\overline{\texttt{001}}\texttt{01}$ |
| 6 | `110` | 3 | $\overline{\texttt{001}}\texttt{10}$ |
| ⋮ | | | |
| 45 | `101101` | 6 | $\overline{\texttt{000001}}\texttt{01101}$ |

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 $C_\alpha$ is separable into the product of a probability distribution over the length $l$,

$$P(l) = 2^{-l}, \tag{8.17}$$

and a uniform distribution over integers having that length,

$$P(n|l) = \begin{cases} 2^{-l+1} & l_b(n) = l \\ 0 & \text{otherwise} \end{cases} . \tag{8.18}$$

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 $C_\alpha$.

**Code $C_\beta$.** The length of the standard binary representation is a positive integer $l_b(n)$. We send this length using $C_\alpha$, followed by the headless binary representation of $n$.

$$c_\beta(n) = c_\alpha[l_b(n)]c_B(n) \tag{8.19}$$

| $n$ | $c_b(n)$ | $l_b(n)$ | $c_\beta(n) = c_\alpha[l_b(n)]c_B(n)$ |
|---|---|---|---|
| 1 | 1 | 1 | $\overline{1}$ |
| 2 | 10 | 2 | $\overline{010}0$ |
| 3 | 11 | 2 | $\overline{010}1$ |
| 4 | 100 | 3 | $\overline{011}00$ |
| 5 | 101 | 3 | $\overline{011}01$ |
| 6 | 110 | 3 | $\overline{011}10$ |
| ⋮ | | | |
| 45 | 101101 | 6 | $\overline{00110}01101$ |

Iterating this procedure, we can define a sequence of codes.

**Code $C_\gamma$.**

$$c_\gamma(n) = c_\beta[l_b(n)]c_B(n) \tag{8.20}$$

| $n$ | $c_b(n)$ | $l_b(n)$ | $c_\gamma(n) = c_\beta[l_b(n)]c_B(n)$ |
|---|---|---|---|
| 1 | 1 | 1 | $\overline{1}$ |
| 2 | 10 | 2 | $\overline{0100}0$ |
| 3 | 11 | 2 | $\overline{0100}1$ |
| 4 | 100 | 3 | $\overline{0101}00$ |
| 5 | 101 | 3 | $\overline{0101}01$ |
| 6 | 110 | 3 | $\overline{0101}10$ |
| ⋮ | | | |
| 45 | 101101 | 6 | $\overline{01110}01101$ |

**Code $C_\delta$.**

$$c_\delta(n) = c_\gamma[l_b(n)]c_B(n) \tag{8.21}$$

### *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.

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. Generalizing this idea, we can make similar byte-based codes for integers in bases 3 and 7, and in any base of the form $2^n - 1$.

These codes are almost complete. (Recall that a code's being 'complete' means that it satisfies the Kraft inequality with equality.) The codes's remaining inefficiency is that they provide the ability to encode the integer zero and the empty string, neither of which was required.

Exercise 8.11**:** 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 kilobytes 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,

$$\boxed{\texttt{Claude Shannon}},$$

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:

$$c_b(n) \;=\; \begin{array}{l}\texttt{10000111101100110000111101011100100110010101000001010011110}\\ \texttt{1000110000111011101101110110111111101110}.\end{array}$$

- The self-delimiting representations of $n$ are:

$$c_\alpha(n) \;=\; \begin{array}{l}\overline{\texttt{000000000000000000000000000000000000000000000000000000000}}\\ \overline{\texttt{00000000000000000000000000000000000000}}\\ \overline{\texttt{1}}\texttt{0000111101100110000111101011100100110010101000001010011110}\\ \texttt{1000110000111011101101110110111111101110}.\end{array}$$

$$\begin{array}{rcl} c_{\beta}(n) & = & \mathtt{000000} \\ & & \mathtt{1}\overline{\mathtt{100010}} \\ & & \mathtt{00001111011001100001111010111001001100101010000010100111 10} \\ & & \mathtt{10001100001110111011011101101111101110.} \end{array}$$

$$\begin{array}{rcl} c_{\gamma}(n) & = & \overline{\mathtt{001}}\mathtt{11} \\ & & \overline{\mathtt{100010}} \\ & & \mathtt{0000111101100110000111101011100100110010101000001010011110} \\ & & \mathtt{10001100001110111011011101101111101110.} \end{array}$$

$$\begin{array}{rcl} c_{\delta}(n) & = & \mathtt{01}\overline{\mathtt{1}} \\ & & \mathtt{11} \\ & & \overline{\mathtt{100010}} \\ & & \mathtt{0000111101100110000111101011100100110010101000001010011110} \\ & & \mathtt{10001100001110111011011101101111101110.} \end{array}$$

- Byte based, base 3.

  0100101001101010100110100101100000101010010101010101100010100 10
  1100110000100000100101001010000101010100101001010011000011000 11

- base 7

  011000101000101000100011100110010100001100000110011001101100000
  10000010000000101010000110010111001000101001 11

- base 15

  100111101010010000100110010000101011100011101011101000001110111
  000111011101010101110011111011101000011 11

## *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 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 6.29 (p.118).

Another way to compare codes for integers is to consider a sequence of probability distributions, such as monotonic probability distributions over $n \geq 1$, 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

| $n$ | $c_\omega(n)$ | $n$ | $c_\omega(n)$ |
|---|---|---|---|
| 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 8.1. Elias's 'universal' code for integers. Examples from 1 to 1024.

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 by a single bit whether or not that message is the final integer (in its standard binary representation). Because a length is a positive integer and all positive integers begin with '1', all the leading 1s can be omitted. Table 8.1 shows the resulting codes.

The encoder of $C_\omega$ operates on the integer $n$ as follows. 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 $C_\omega$; then the encoder does not need to transmit all those initial 1's and the final zero. Maybe this code should be called $C_{\omega^\omega}$?

## *Solutions*

Solution to exercise 8.11 (p.161): The use of 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.

(a) The expected number of characters is $q + 1 = 256$, so the expected length of the integer is $256 \times 8 \simeq 2000$ bits.

(b) We wish to find $q$ such that $q \log q \simeq 800,000$ bits. A value of $q$ between $2^{15}$ and $2^{16}$ satisfies this constraint, so 16-bit blocks are roughly the optimal size, assuming there is one EOF character.

# Part II

# Noisy-Channel Coding

# 9

# *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 in which the random variables are correlated. This material has two motivations. First, data from the real world have interesting correlations, 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.

## 9.1 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:

$$H(X,Y) = \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x,y)}. \tag{9.1}$$

Entropy is additive for independent random variables:

$$H(X,Y) = H(X) + H(Y) \text{ iff } P(x,y) = P(x)P(y). \tag{9.2}$$

**The conditional entropy of** $X$ **given** $y = b_k$ is the entropy of the probability distribution $P(x \mid y{=}b_k)$.

$$H(X \mid y{=}b_k) \equiv \sum_{x \in \mathcal{A}_X} P(x \mid y{=}b_k) \log \frac{1}{P(x \mid y{=}b_k)}. \tag{9.3}$$

**The conditional entropy of** $X$ **given** $Y$ is the average, over $y$, of the conditional entropy of $X$ given $y$.

$$\begin{aligned} H(X \mid Y) &\equiv \sum_{y \in \mathcal{A}_Y} P(y) \left[ \sum_{x \in \mathcal{A}_X} P(x \mid y) \log \frac{1}{P(x \mid y)} \right] \\ &= \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x \mid y)}. \end{aligned} \tag{9.4}$$

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:

$$\log \frac{1}{P(x,y)} \quad = \quad \log \frac{1}{P(x)} + \log \frac{1}{P(y|x)} \tag{9.5}$$

so

$$h(x,y) = h(x) + h(y|x). \tag{9.6}$$

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:

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y). \tag{9.7}$$

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

$$I(X;Y) \quad \equiv \quad H(X) - H(X|Y), \tag{9.8}$$

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 $x$ conveys about $y$.

**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 $P(x,y|z = c_k)$,

$$I(X;Y|z = c_k) = H(X|z = c_k) - H(X|Y, z = c_k). \tag{9.9}$$

**The conditional mutual information between** $X$ **and** $Y$ **given** $Z$ is the average over $z$ of the above conditional mutual information.

$$I(X;Y|Z) = H(X|Z) - H(X|Y,Z). \tag{9.10}$$

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 9.1 shows how the total entropy $H(X,Y)$ of a joint ensemble can be broken down. **This figure is important.**

☆☆☆!

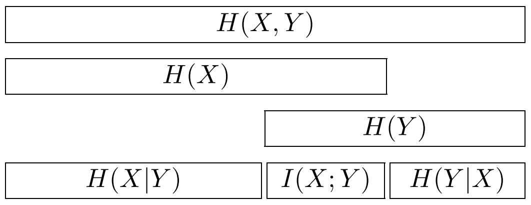


Figure 9.1. The relationship between joint information, marginal entropy, conditional entropy and mutual entropy.

## 9.2 Exercises

Exercise 9.1:$^{A2}$ Prove the assertion that entropy is additive for independent random variables (equation (9.2)).

Exercise 9.2:$^{B2}$ 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 9.3:$^{B3}$ Referring to the definitions of conditional entropy (9.3–9.4), 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 9.4:$^{B2}$ Prove the chain rule for entropy, equation (9.7). $[H(X, Y) = H(X) + H(Y|X)]$.

Exercise 9.5:$^{A2}$ 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 2.22 (p.41) and note that

$$I(X;Y) = D_{\rm KL}(P(x,y)||P(x)P(y)).] \tag{9.11}$$

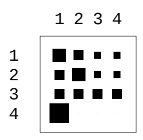
Exercise 9.6:$^{D4}$ The 'entropy distance' between two random variables can be defined to be the difference between their joint entropy and their mutual information:

$$D_H(X,Y) \equiv H(X,Y) - I(X;Y). \tag{9.12}$$

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 9.7:$^{A2}$ A joint ensemble $XY$ has the following joint distribution.

| $P(x,y)$ | | $x$ | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 |
| | 1 | 1/8 | 1/16 | 1/32 | 1/32 |
| $y$ | 2 | 1/16 | 1/8 | 1/32 | 1/32 |
| | 3 | 1/16 | 1/16 | 1/16 | 1/16 |
| | 4 | 1/4 | 0 | 0 | 0 |



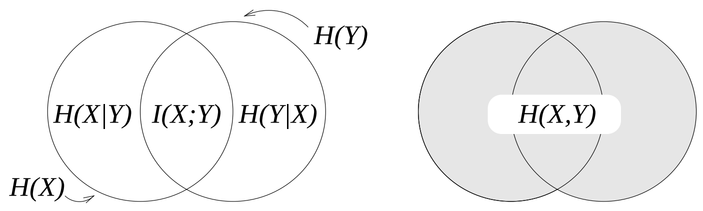


Figure 9.2. A misleading representation of entropies (contrast with figure 9.1).

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.173

Exercise 9.8:$^{A2}$ 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

$$z = x + y \bmod 2. \tag{9.13}$$

(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 $I(Z;X)$? Notice that this ensemble is related to the binary symmetric channel, with $x =$ input, $y =$ noise, and $z =$ output.

## *Three term entropies*

Exercise 9.9:$^{C3}$ Many texts draw figure 9.1 in the form of a Venn diagram (figure 9.2). 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 9.10:$^{A3}$ 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*

$$w \rightarrow d \rightarrow r, \tag{9.14}$$

that is, the probability $P(w,d,r)$ can be written as

$$P(w,d,r) = P(w)P(d|w)P(r|d). \tag{9.15}$$

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!

# *Solutions to Chapter 9's exercises*

Solution to exercise 9.5 (p.168): Symmetry of mutual information:

$$\begin{aligned}
I(X;Y) &= H(X) - H(X|Y) && (9.16)\\
&= \sum_x P(x)\log\frac{1}{P(x)} - \sum_{xy} P(x,y)\log\frac{1}{P(x|y)} && (9.17)\\
&= \sum_{xy} P(x,y)\log\frac{P(x|y)}{P(x)} && (9.18)\\
&= \sum_{xy} P(x,y)\log\frac{P(x,y)}{P(x)P(y)} && (9.19)
\end{aligned}$$

This expression is symmetric in $x$ and $y$ so

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X). \quad (9.20)$$

We can prove that mutual information is positive two ways. One is to continue from

$$I(X;Y) = \sum_{x,y} P(x,y)\log\frac{P(x,y)}{P(x)P(y)} \quad (9.21)$$

which is a relative entropy and use Gibbs's inequality (proved on 48), which asserts that this relative entropies is $\geq 0$, with equality 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

$$-\sum_{x,y} P(x,y)\log\frac{P(x)P(y)}{P(x,y)} \geq -\log\sum_{x,y}\frac{P(x,y)}{P(x,y)}P(x)P(y) = \log 1 = 0. \quad (9.22)$$

Solution to exercise 9.1 (p.168): Let $P(x,y) = P(x)P(y)$. Then

$$\begin{aligned}
H(X,Y) &= \sum_{xy} P(x)P(y)\log\frac{1}{P(x)P(y)} && (9.23)\\
&= \sum_{xy} P(x)P(y)\log\frac{1}{P(x)} + \sum_{xy} P(x)P(y)\log\frac{1}{P(y)} && (9.24)\\
&= \sum_x P(x)\log\frac{1}{P(x)} + \sum_y P(y)\log\frac{1}{P(y)} && (9.25)\\
&= H(X) + H(Y). && (9.26)
\end{aligned}$$

Solution to exercise 9.2 (p.168):

$$H(X,Y) = H(U,V,V,W) = H(U,V,W) = H_u + H_v + H_w. \quad (9.27)$$

$$H(X|Y) = H_u. \quad (9.28)$$

$$I(X;Y) = H_v. \quad (9.29)$$

Solution to exercise 9.3 (p.168): See exercise 9.7 (p.168) 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's theorem) and invoking Gibbs's inequality (exercise 2.22 (p.41)):

$$\begin{aligned} H(X|Y) &\equiv \sum_{y \in \mathcal{A}_Y} P(y) \left[ \sum_{x \in \mathcal{A}_X} P(x|y) \log \frac{1}{P(x|y)} \right] \\ &= \sum_{xy \in \mathcal{A}_X \mathcal{A}_Y} P(x,y) \log \frac{1}{P(x|y)} && (9.30) \\ &= \sum_{xy} P(x)P(y|x) \log \frac{P(y)}{P(y|x)P(x)} && (9.31) \\ &= \sum_x P(x) \log \frac{1}{P(x)} + \sum_x P(x) \sum_y P(y|x) \log \frac{P(y)}{P(y|x)}. && (9.32) \\ & && (9.33) \end{aligned}$$

The last expression is a sum of relative entropies between the distributions $P(y|x)$ and $P(y)$. So

$$H(X|Y) \leq H(X) + 0, \tag{9.34}$$

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 9.4 (p.168): The chain rule for entropy follows from the decomposition of a joint probability:

$$\begin{aligned} H(X,Y) &= \sum_{xy} P(x,y) \log \frac{1}{P(x,y)} && (9.35) \\ &= \sum_{xy} P(x)P(y|x) \left[ \log \frac{1}{P(x)} + \log \frac{1}{P(y|x)} \right] && (9.36) \\ &= \sum_x P(x) \log \frac{1}{P(x)} + \sum_x P(x) \sum_y P(y|x) \log \frac{1}{P(y|x)} && (9.37) \\ &= H(X) + H(Y|X). && (9.38) \end{aligned}$$

Solution to exercise 9.7 (p.168): See chapter 10.

## *Three term entropies*

Solution to exercise 9.9 (p.169): 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 9.2, 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 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

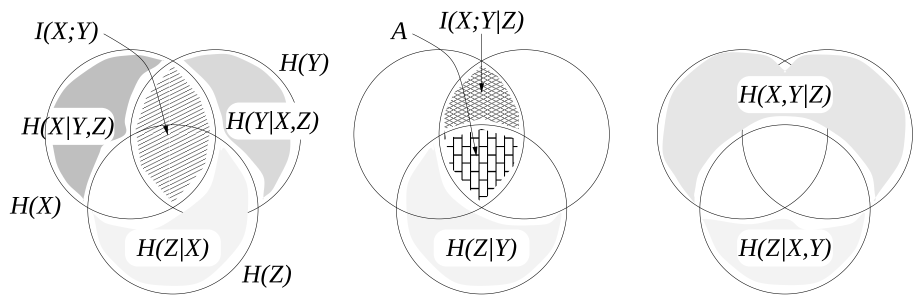


Figure 9.3. A misleading representation of entropies, continued.

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 9.3 correctly shows relationships such as

$$H(X) + H(Z|X) + H(Y|X,Z) = H(X,Y,Z). \tag{9.39}$$

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.

The above example is not at all a capricious or exceptional illustration. The binary symmetric channel with input $X$, noise $Y$, and output $Z$ situation in which $I(X;Y) = 0$ (input and noise are uncorrelated) but $I(X;Y|Z) > 0$ (once you see the output, the unknown input and the unknown noise are intimately related!).

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 9.10 (p.169): For any joint ensemble $XYZ$, the following chain rule for mutual information holds.

$$I(X;Y,Z) = I(X;Y) + I(X;Z|Y). \tag{9.40}$$

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:

$$I(W;D,R) = I(W;D) \tag{9.41}$$

and

$$I(W;D,R) = I(W;R) + I(W;D|R), \tag{9.42}$$

so

$$I(W;R) - I(W;D) \leq 0. \tag{9.43}$$

Solution to exercise 9.8 (p. 169):

$$z = x + y \bmod 2. \tag{9.44}$$

(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 10*

Before reading chapter 10, you should have read chapter 1 and worked on exercises 9.3–9.5 (pp.168–168), 2.22–9.7 (pp.41–168), and 9.8 (p.169).

**Information Theory, Pattern Recognition and Neural Networks**

**Handout number 2.**

Do not be disturbed by missing pagenumbers: the handouts do not include all the book's chapters.

Because the handouts are based on different drafts of the book, there may be occasional mismatches between pagereferences, etc., where cross-references occur between this and the other handouts.

Nonexaminable material is indicated by the symbol $^*$.

**Information Theory, Pattern Recognition and Neural Networks**

Handout number 3.

Do not be disturbed by missing pagenumbers: the handouts do not include all the book's chapters.

Because the handouts are based on different drafts of the book, there may be occasional mismatches between pagereferences, etc., where cross-references occur between this and the other handouts.

Nonexaminable material is indicated by the symbol $^*$.

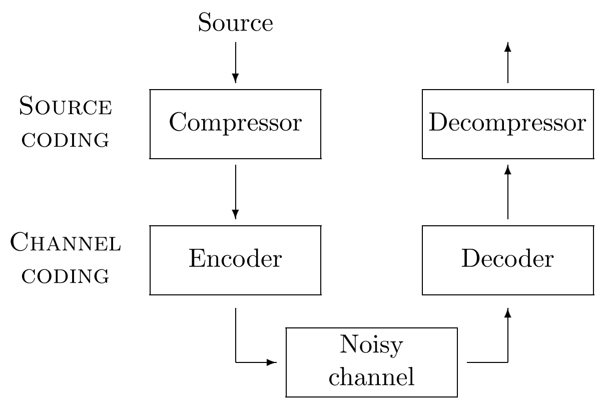
## Approximate roadmap for the course

| | |
|---|---|
| Lecture 1 | Introduction to Information Theory. Chapter 1. |
| Before lecture 2 | Please work on exercise 4.4 (p. 76). |
| About now | Read chapters 2 and 5 and work on exercises in chapter 2. |
| Lecture 2–3 | Information content & typicality. Chapter 5. |
| Lecture 4 | Symbol codes. Chapter 6. |
| Lecture 5 | Arithmetic codes. Chapter 7. |
| About now | Read chapter 9 and do the exercises. |
| Lecture 6 | Noisy channels. Definition of mutual information and capacity. Chapter 10. |
| Lecture 7-8 | The noisy channel coding theorem. Chapter 11. |
| Lecture 9 | Clustering. Bayesian inference. Chapter 3, 22, 24. |
| About now | Read chapter 32 (Ising models). |
| Lecture 10 | Monte Carlo methods. Chapter 30, 31. |
| Lecture 12 | Variational methods. Chapter 34. |
| Lecture 13 | Neural networks – the single neuron. Chapter 41. |
| Lecture 14 | Capacity of the single neuron. Chapter 42. |
| Lecture 15 | Learning as inference. Chapter 43. |
| Lecture 16 | The Hopfield network. Content-addressable memory. Chapter 44. |

# 10

# *Communication over a Noisy Channel*

## 10.1 The big picture



In chapters 5–7, 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 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.

## 10.2 Review of probability and information

As an example, we take the joint distribution $XY$ from exercise 9.7 (p.168). The marginal distributions $P(x)$ and $P(y)$ are shown in the margins.

| $P(x,y)$ | | $x$ | | | | $P(y)$ |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | |
| | 1 | $1/8$ | $1/16$ | $1/32$ | $1/32$ | $1/4$ |
| $y$ | 2 | $1/16$ | $1/8$ | $1/32$ | $1/32$ | $1/4$ |
| | 3 | $1/16$ | $1/16$ | $1/16$ | $1/16$ | $1/4$ |
| | 4 | $1/4$ | 0 | 0 | 0 | $1/4$ |
| $P(x)$ | | $1/2$ | $1/4$ | $1/8$ | $1/8$ | |

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 | $1/2$ | $1/4$ | $1/8$ | $1/8$ | $7/4$ |
| $y$ | 2 | $1/4$ | $1/2$ | $1/8$ | $1/8$ | $7/4$ |
| | 3 | $1/4$ | $1/4$ | $1/4$ | $1/4$ | 2 |
| | 4 | 1 | 0 | 0 | 0 | 0 |
| | | | | | | $H(X\|Y) = 11/8$ |

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.

## 10.3 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

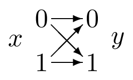
$$Q_{j|i} = P(y{=}b_j|x{=}a_i). \tag{10.1}$$

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:

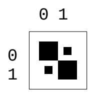
$$\mathbf{p}_Y = \mathbf{Q}\mathbf{p}_X. \tag{10.2}$$

Some useful model channels are:

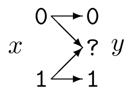
**Binary symmetric channel.** $\mathcal{A}_X{=}\{0,1\}$. $\mathcal{A}_Y{=}\{0,1\}$.



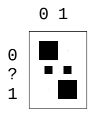
$P(y{=}0|x{=}0){=}1-f$; $P(y{=}0|x{=}1){=}f$;
$P(y{=}1|x{=}0){=}f$; $P(y{=}1|x{=}1){=}1-f$.



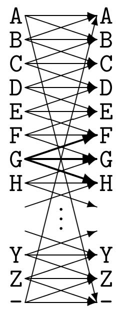
**Binary erasure channel.** $\mathcal{A}_X{=}\{0,1\}$. $\mathcal{A}_Y{=}\{0,?,1\}$.



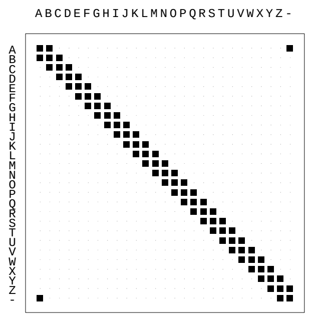
$P(y{=}0|x{=}0){=}1-f$; $P(y{=}0|x{=}1){=}0$;
$P(y{=}?|x{=}0){=}f$; $P(y{=}?|x{=}1){=}f$;
$P(y{=}1|x{=}0){=}0$; $P(y{=}1|x{=}1){=}1-f$.



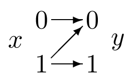
**Noisy typewriter.** $\mathcal{A}_X = \mathcal{A}_Y =$ the 27 letters {A, B, ..., Z, -}. The letters are arranged in a circle, and when the typist attempts to type B, what comes out is either A, B or C, with probability 1/3 each; when the input is C, the output is B, C or D; and so forth, with the final letter '-' adjacent to the first letter A.



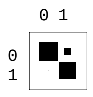
$\vdots$
$P(y{=}\mathtt{F}|x{=}\mathtt{G}){=}1/3$;
$P(y{=}\mathtt{G}|x{=}\mathtt{G}){=}1/3$;
$P(y{=}\mathtt{H}|x{=}\mathtt{G}){=}1/3$;
$\vdots$



**Z channel.** $\mathcal{A}_X{=}\{0,1\}$. $\mathcal{A}_Y{=}\{0,1\}$.



$P(y{=}0|x{=}0){=}1$; $P(y{=}0|x{=}1){=}f$;
$P(y{=}1|x{=}0){=}0$; $P(y{=}1|x{=}1){=}1-f$.



## 10.4 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:

$$P(x, y) = P(y|x)P(x). \tag{10.3}$$

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's theorem:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} = \frac{P(y|x)P(x)}{\sum_{x'} P(y|x')P(x')} \tag{10.4}$$

Example 10.1: 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.

$$\begin{aligned} P(x{=}1|y{=}1) &= \frac{P(y{=}1|x{=}1)P(x{=}1)}{\sum_{x'} P(y|x')P(x')} \\ &= \frac{0.85 \times 0.1}{0.85 \times 0.1 + 0.15 \times 0.9} \\ &= \frac{0.085}{0.22} \;=\; 0.39. \end{aligned} \tag{10.5}$$

Thus '$x$=1' is still less probable than '$x$=0', although it is not as improbable as it was before.

Exercise 10.2: Now assume we observe $y$=0. Compute the probability of $x$=1 given $y$=0.

Example 10.3: 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.

$$\begin{aligned} P(x{=}1|y{=}1) &= \frac{0.85 \times 0.1}{0.85 \times 0.1 + 0 \times 0.9} \\ &= \frac{0.085}{0.085} \;=\; 1.0. \end{aligned} \tag{10.6}$$

So given the output $y$=1 we become certain of the input.

Exercise 10.4: Alternatively, assume we observe $y$=0. Compute $P(x{=}1|y{=}0)$.

## 10.5 Information conveyed by a channel

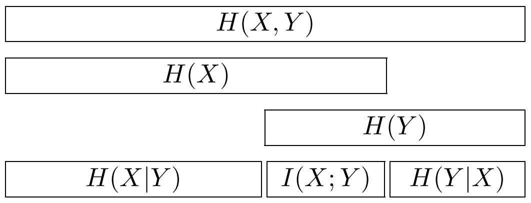
We now consider how much information can be communicated through a channel. In operational terms, we are interested in finding ways of using the channel such that all the bits that are communicated are recovered with negligible probability of error. In mathematical terms, assuming a particular input ensemble $X$, we can measure how much information the output conveys about the input by the mutual information:

$$I(X;Y) \equiv H(X) - H(X|Y) = H(Y) - H(Y|X). \tag{10.7}$$

Our aim is to establish the connection between these two ideas. Let us evaluate $I(X;Y)$ for some of the channels above.

*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 10.5: 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:

$$I(X;Y) = H(Y) - H(Y|X) \tag{10.8}$$

What is $H(Y|X)$? It is defined to be the weighted sum over $x$ of $H(Y|x)$; but $H(Y|x)$ is the same for each value of $x$: $H(Y|x{=}0)$ is $H_2(0.15)$, and $H(Y|x{=}1)$ is $H_2(0.15)$. So

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(0.22) - H_2(0.15) \\ &= 0.76 - 0.61 \;=\; 0.15 \text{ bits.} \end{aligned} \tag{10.9}$$

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 10.6: And now the Z channel, with $\mathcal{P}_X$ as above. $P(y{=}1){=}0.085$.

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(0.085) - [0.9 H_2(0) + 0.1 H_2(0.15)] \\ &= 0.42 - (0.1 \times 0.61) = 0.36 \text{ bits.} \end{aligned} \tag{10.10}$$

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 10.7: 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 10.8: 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:

$$C(Q) = \max_{\mathcal{P}_X} I(X;Y). \tag{10.11}$$

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 11 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 10.9: 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

$$C(Q_{\rm BSC}) \;=\; H_2(0.5) - H_2(0.15) \;=\; 1.0 - 0.61 \;=\; 0.39\,\text{bits}. \tag{10.12}$$

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 $I(X;Y)$,

$$I(X;Y) = H_2((1-f)p_1 + (1-p_1)f) - H_2(f) \text{ (figure 10.1)}. \tag{10.13}$$

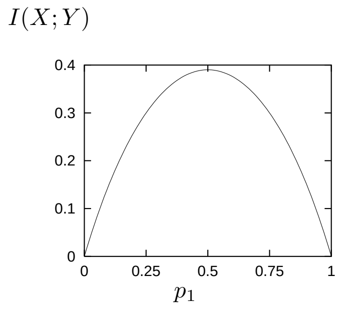


Figure 10.1. The mutual information $I(X;Y)$ for a binary symmetric channel with $f = 0.15$ as a function of the input distribution.

Example 10.10: The noisy typewriter. The optimal input distribution is a uniform distribution over $x$, and gives $C = \log_2 9$ bits.

Example 10.11: 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:

$$P(y{=}1) \;=\; p_1(1-f). \tag{10.14}$$

Then the mutual information is:

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(p_1(1-f)) - (p_0 H_2(0) + p_1 H_2(f)) \\ &= H_2(p_1(1-f)) - p_1 H_2(f). \end{aligned} \tag{10.15}$$

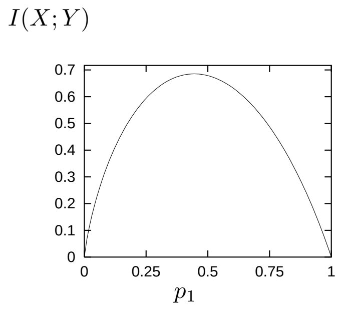


Figure 10.2. The mutual information $I(X;Y)$ for a Z channel with $f = 0.15$ as a function of the input distribution.

This is a non-trivial function of $p_1$, shown in figure 10.2. 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`.

Solutions on p.189

Exercise 10.12: What is the capacity of the binary symmetric channel for general $f$?

Exercise 10.13: 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.

## 10.6 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.

**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 per channel use.

[This definition holds for any channels, not only binary channels.]

**A decoder** for an $(N, K)$ block code is a mapping from the set of length-$N$ strings of channel outputs, $\mathcal{A}_Y^N$ to a codeword label $\hat{s} \in \{0, 1, 2, \ldots, 2^K\}$.

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:

$$p_{\text{B}} = \sum_{s_{\text{in}}} P(s_{\text{in}}) P(s_{\text{out}} \neq s_{\text{in}} | s_{\text{in}}) \tag{10.16}$$

**The maximal probability of block error** is

$$p_{\text{BM}} = \max_{s_{\text{in}}} P(s_{\text{out}} \neq s_{\text{in}} | s_{\text{in}}) \tag{10.17}$$

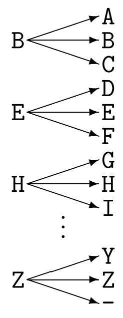
**The optimal decoder** for a channel code is the one that minimizes the probability of block error. It decodes an output $\mathbf{y}$ as the input $s$ that has maximum posterior probability $P(s|\mathbf{y})$.

$$P(s|\mathbf{y}) = \frac{P(\mathbf{y}|s)P(s)}{\sum_{s'} P(\mathbf{y}|s')P(s')} \tag{10.18}$$

$$\hat{s}_{\text{optimal}} = \text{argmax}\, P(s|\mathbf{y}). \tag{10.19}$$

A uniform prior distribution on $s$ 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 $\mathbf{s}_{\text{in}}$ (averaging over all $K$ bits).



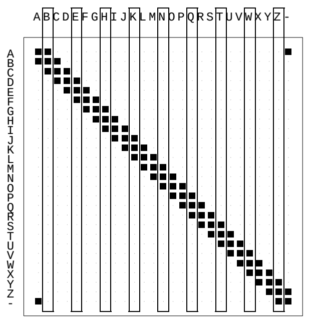


Figure 10.3. A non-confusable subset of inputs for the noisy typewriter.

**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 $< \epsilon$.

*Confirmation of the theorem for the noisy typewriter channel*

In the case of the noisy typewriter, we can easily confirm 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 10.3). 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 the capacity $C$, which we evaluated in example 10.10 (p.181).

How does this translate into the terms of the theorem?

| The theorem | How it applies to the noisy typewriter |
|---|---|
| *Associated with each discrete memoryless channel, there is a nonnegative number $C$.* | The capacity $C$ is $\log_2 9$. |
| *For any $\epsilon > 0$ and $R < C$, for large enough $N$,* | No matter what $\epsilon$ and $R$ are, we set the block length $N$ to 1. |
| *there exists a block code of length $N$ and rate $\geq R$* | The block code is $\{\texttt{B}, \texttt{E}, \ldots, \texttt{Z}\}$. 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 $R$. |
| *and a decoding algorithm,* | The decoding algorithm maps the received letter to the nearest letter in the code, |
| *such that the maximal probability of block error is $< \epsilon$.* | and its maximal probability of block error is zero, which is less than the given $\epsilon$. |

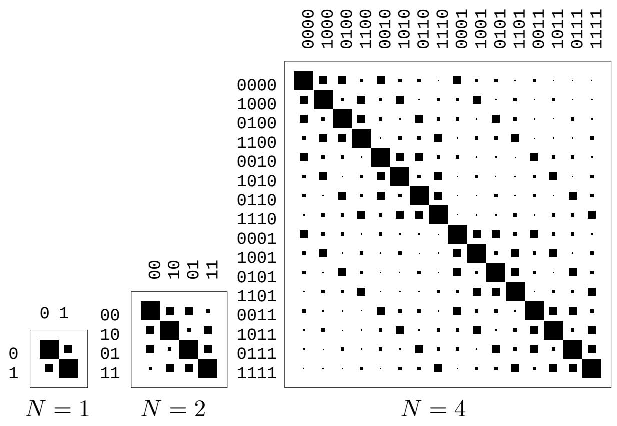


Figure 10.4. Extended channels obtained from a binary symmetric channel with transition probability 0.15.

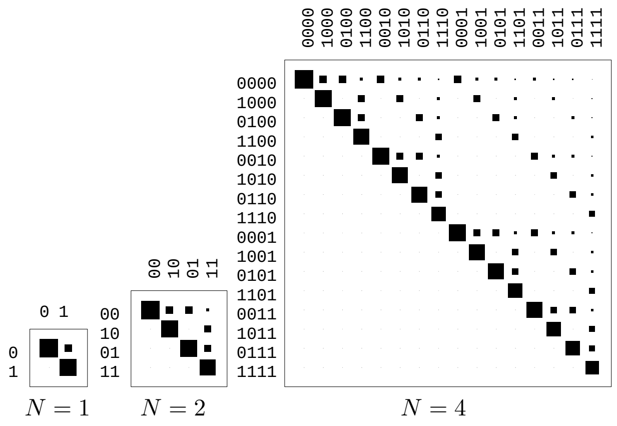


Figure 10.5. Extended channels obtained from a Z channel with transition probability 0.15. Each column corresponds to an input, and each row is a different output.

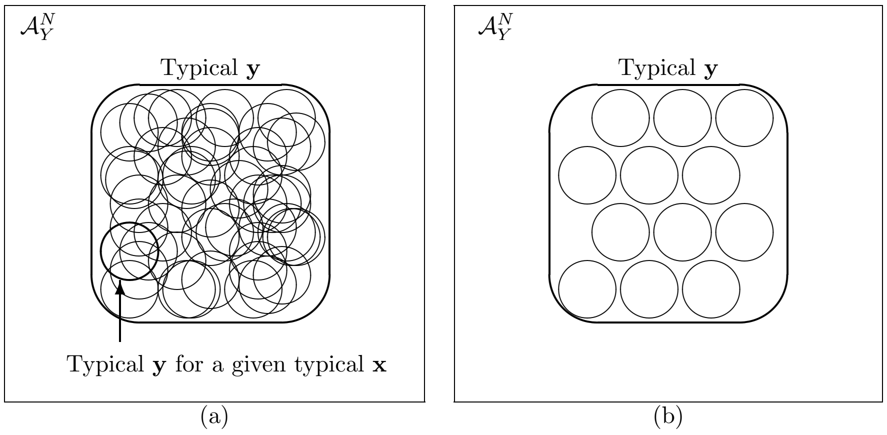


Figure 10.6. (a) Some typical outputs in $\mathcal{A}_Y^N$ corresponding to typical inputs **x**. (b) A subset of the typical sets shown in (a) that do not overlap each other. This picture can be compared with the solution to the noisy typewriter in figure 10.3.

## 10.7 Intuitive preview of proof

### *Extended channels*

To prove the theorem for a given channel, we consider the *extended channel*, corresponding to $N$ uses of the given channel. The extended channel has $|\mathcal{A}_X|^N$ possible inputs **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 10.4 and 10.5, with $N = 2$ and $N = 4$.

Exercise 10.14:$^{A1}$ Find the transition probability matrices **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 **x** is very likely to produce an output in a small subspace of the output alphabet – the typical output set, given that input. 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 **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 chapter 5, and consider the number of probable output sequences **y**. The total number of typical output sequences **y**, when **x** comes from the ensemble $X^N$, is $2^{NH(Y)}$, all having similar probability. For any particular typical input sequence **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 10.6a.

We now imagine restricting ourselves to a subset of the typical inputs **x** such that the corresponding typical output sets do not overlap, as shown in figure 10.6b. We can then bound the number of non-confusable inputs by dividing the size of the typical **y** set, $2^{NH(Y)}$, by the size of each typical-**y**-given-typical-**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 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.

## 10.8 Exercises

Solutions on p.191

Exercise 10.15:$^{A3}$ 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? One could argue that it is good to favour the 0 input, since it is transmitted without error – and also argue that it is good to favour the 1 input, since it often gives rise to the highly prized 1 output, which allows certain identification of the input!

(b) In the case of general $f$, show that the optimal input distribution is

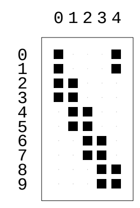
$$p_1^* = \frac{1/(1-f)}{1 + 2^{(H_2(f)/(1-f))}}. \tag{10.20}$$

(c) What happens to $p_1^*$ if the noise level $f$ is very close to 1?

Exercise 10.16:$^{A2}$ Sketch graphs of the capacity of the Z channel, the binary symmetric channel and the binary erasure channel as a function of $f$.

Exercise 10.17:$^{B2}$ What is the capacity of the five-input, ten-output channel whose transition probability matrix is

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0.25 & 0 & 0 & 0 \\ 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 \\ 0 & 0 & 0 & 0.25 & 0.25 \end{bmatrix}$$



? (10.21)

Exercise 10.18:$^{A2}$ Consider a Gaussian channel with binary input $x \in \{-1, +1\}$ and *real* output alphabet $\mathcal{A}_Y$, with transition probability density

$$Q(y|x, \alpha, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-x\alpha)^2}{2\sigma^2}}, \tag{10.22}$$

where $\alpha$ 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

$$P(x{=}1|y,\alpha,\sigma) = \frac{1}{1+e^{-a(y)}}. \tag{10.23}$$

Sketch the value of $P(x{=}1|y,\alpha,\sigma)$ as a function of $y$.

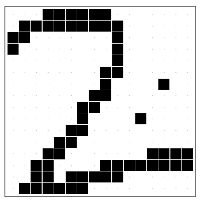
(b) Assume that a single bit is to be transmitted. What is the optimal decoder, and what is its probability of error? Express your answer in terms of the signal to noise ratio $\alpha^2/\sigma^2$ and the error function (the cumulative probability function of the Gaussian distribution),

$$\Phi(z) \equiv \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}\, dz. \tag{10.24}$$

[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 $\mathcal{A}_X = \{0,1,2,3,\ldots,9\}$; this selected message is the input to the channel. What comes out of the channel is a pattern of ink on paper. If the ink pattern is represented using 256 binary pixels, the channel $Q$ 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 10.19:$^{A2}$ 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's theorem to infer the probability of $x$ given $y$.

$$P(x|y) = \frac{P(y|x)P(x)}{\sum_{x'} P(y|x')P(x')} \tag{10.25}$$

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 10.20:$^{A2}$ 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 10.21:$^{B2}$ 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 $s$ from $\mathcal{A}_S = \{1, 2, \ldots, 24\}$, 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?

Exercise 10.22:$^{B2}$ 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 known to the receiver. When the building has selected a particular person from each room, the ordered string of days corresponding to that $K$-tuple of people 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 10's exercises*

Solution to exercise 10.2 (p.179): If we assume we observe $y{=}0$,

$$P(x{=}1|y{=}0) = \frac{P(y{=}0|x{=}1)P(x{=}1)}{\sum_{x'} P(y|x')P(x')} \tag{10.26}$$

$$= \frac{0.15 \times 0.1}{0.15 \times 0.1 + 0.85 \times 0.9} \tag{10.27}$$

$$= \frac{0.015}{0.78} = 0.019. \tag{10.28}$$

Solution to exercise 10.4 (p.179): If we observe $y = 0$,

$$P(x{=}1|y{=}0) = \frac{0.15 \times 0.1}{0.15 \times 0.1 + 1.0 \times 0.9} \tag{10.29}$$

$$= \frac{0.015}{0.915} = 0.016. \tag{10.30}$$

Solution to exercise 10.7 (p.180): The probability that $y = 1$ is 0.5, so the mutual information is:

$$I(X;Y) = H(Y) - H(Y|X) \tag{10.31}$$

$$= H_2(0.5) - H_2(0.15) \tag{10.32}$$

$$= 1 - 0.61 \ = \ 0.39 \text{ bits.} \tag{10.33}$$

Solution to exercise 10.8 (p.180): 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:

$$I(X;Y) = H(Y) - H(Y|X) \tag{10.34}$$

$$= H_2(0.575) - [0.5 \times H_2(0.15) + 0.5 \times 0] \tag{10.35}$$

$$= 0.98 - 0.30 \ = \ 0.679 \text{ bits.} \tag{10.36}$$

Solution to exercise 10.12 (p.181): By symmetry, the optimal input distribution is $\{0.5, 0.5\}$. Then the capacity is

$$C = I(X;Y) = H(Y) - H(Y|X) \tag{10.37}$$

$$= H_2(0.5) - H_2(f) \tag{10.38}$$

$$= 1 - H_2(f). \tag{10.39}$$

Would you like to find 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 $\{p_0, p_1\}$:

$$I(X;Y) = H(Y) - H(Y|X) \tag{10.40}$$

$$= H_2(p_0 f + p_1(1-f)) - H_2(f). \tag{10.41}$$

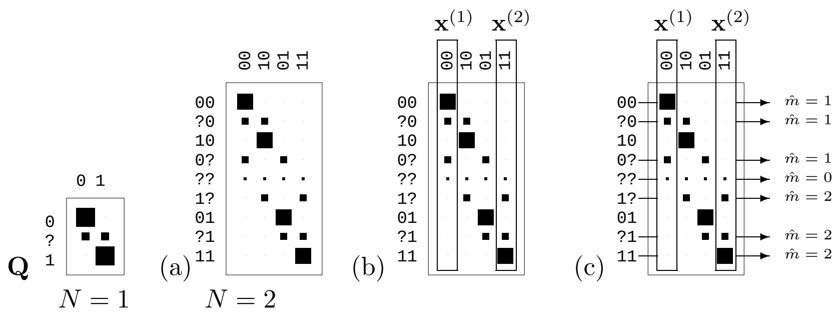


Figure 10.7. (a) The extended channel ($N = 2$) obtained from a binary erasure channel with erasure probability 0.15. (b) A block code consisting of the two codewords 00 and 11. (c) The optimal decoder for this code.

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 10.13 (p.181): Answer 1. 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 $fH_2(0.5)$.

$$\begin{aligned} C = I(X;Y) &= H(X) - H(X|Y) && (10.42)\\ &= H_2(0.5) - fH_2(0.5) && (10.43)\\ &= 1 - f. && (10.44)\end{aligned}$$

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 reliable. This result seems very reasonable, but it is far from obvious how to encode information so as to communicate *reliably* over this channel.

Answer 2. 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:

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) && (10.45)\\ &= H_2(p_1) - fH_2(p_1) && (10.46)\\ &= (1 - f)H_2(p_1). && (10.47)\end{aligned}$$

This mutual information achieves its maximum value of $(1-f)$ when $p_1 = 1/2$.

Solution to exercise 10.14 (p.185): The extended channel is shown in figure 10.7. 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 10.15 (p.186): In example 10.11 (p.181) we showed that the mutual information between input and output of the Z channel is

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H_2(p_1(1-f)) - p_1 H_2(f). \end{aligned} \tag{10.48}$$

We differentiate this expression with respect to $p_1$, taking care not to confuse $\log_2$ with $\log_e$:

$$\frac{d}{dp_1} I(X;Y) = (1-f)\log_2 \frac{1-p_1(1-f)}{p_1(1-f)} - H_2(f). \tag{10.49}$$

Setting this derivative to zero and rearranging using skills developed in exercise 2.26 (p.42), we obtain:

$$p_1^*(1-f) = \frac{1}{1+2^{H_2(f)/(1-f)}}, \tag{10.50}$$

so the optimal input distribution is

$$p_1^* = \frac{1/(1-f)}{1+2^{(H_2(f)/(1-f))}}. \tag{10.51}$$

As the noise level $f$ tends to 1, this expression tends to $1/e$ (as you can prove using L'Hôpital's rule).

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 $(p_1 - 1/2)$.

Solution to exercise 10.16 (p.186): The capacities of the three channels are shown in figure 10.8. For any $f < 0.5$, the BEC is the channel with highest capacity and the BSC the lowest.

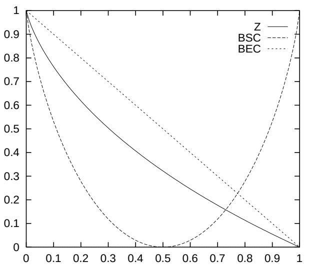


Figure 10.8. Capacities of the Z channel, binary symmetric channel, and binary erasure channel.

Solution to exercise 10.17 (p.186): 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

$$C = \max_{\mathcal{P}_X} H(Y) - H(Y|X) = \log \frac{10}{4} = \log {}^5\!/\!_2 \text{ bits}. \tag{10.52}$$

Solution to exercise 10.18 (p.186): The logarithm of the posterior probability ratio, given $y$, is

$$a(y) = \log \frac{P(x=1|y,\alpha,\sigma)}{Q(x=-1|y,\alpha,\sigma)} = \log \frac{Q(y|x=1,\alpha,\sigma)}{Q(y|x=-1,\alpha,\sigma)} = 2\frac{\alpha y}{\sigma^2}. \tag{10.53}$$

Using our skills picked up from exercise 2.26 (p.42), we rewrite this in the form

$$P(x=1|y,\alpha,\sigma) = \frac{1}{1+e^{-a(y)}}. \tag{10.54}$$

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

$$p_b = \int_{-\infty}^{0} dy\, Q(y|x=1,\alpha,\sigma) = \int_{-\infty}^{-x\alpha} dy\, \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} = \Phi\left(-\frac{x\alpha}{\sigma}\right) \quad (10.55)$$

where

$$\Phi(z) \equiv \int_{-\infty}^{z} \tfrac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}. \quad (10.56)$$

Solution to exercise 10.19 (p.187): 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 $6 \times 7$ pixels. This leaves 214 pixels that can be set arbitrarily, and there are also $12 \times 11$ 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 $127 \times 2^{-37}$ of them also contain another recognizable character. These double–hits decrease undetectably the above answer, $2^{219}$.

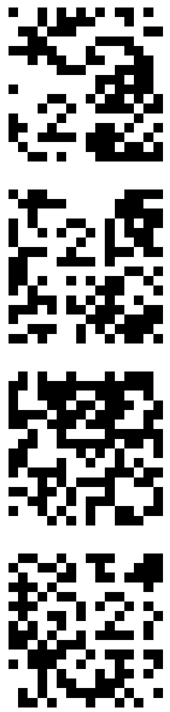


Figure 10.9. Four random samples from the set of $2^{219}$ 'miniature 2s' defined in the text.

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 $16 \times 16$ 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 10.20 (p.188): The probability that $q = 24$ people whose birthdays are drawn at random from $A = 365$ days all have *distinct* birthdays is

$$\frac{A(A-1)(A-2)\ldots(A-q+1)}{A^q}. \tag{10.57}$$

The probability that two (or more) people share a birthday is one minus this quantity, which, for $q = 24$ and $A = 365$, is about 0.5. This exact way of answering the question 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

$$\frac{q(q-1)}{2}\frac{1}{A}. \tag{10.58}$$

This answer is more instructive. The expected number of collisions is tiny if $q \ll \sqrt{A}$ and big if $q \gg \sqrt{A}$.

We can also approximate the probability that all birthdays are distinct, for small $q$, thus:

$$\begin{aligned}\frac{A(A-1)(A-2)\ldots(A-q+1)}{A^q} &= (1)(1-{}^{1}\!/\!_{\mathrm{A}})(1-{}^{2}\!/\!_{\mathrm{A}})\ldots(1-{}^{\mathrm{q}\text{-}1}\!/\!_{\mathrm{A}}) \\ &\simeq \exp(0)\exp(-{}^{1}\!/\!_{\mathrm{A}})\exp(-{}^{2}\!/\!_{\mathrm{A}})\ldots\exp(-{}^{\mathrm{q}\text{-}1}\!/\!_{\mathrm{A}}) \qquad (10.59) \\ &\simeq \exp\left(-\frac{1}{A}\sum_{i=1}^{q-1} i\right) = \exp\left(-\frac{q(q-1)/2}{A}\right). \qquad (10.60)\end{aligned}$$

Solution to exercise 10.21 (p.188): 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

$$1 - \left(\frac{A-1}{A}\right)^{q-1} = 1 - 0.939 = 0.061. \tag{10.61}$$

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 10.22 (p.188): 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,

$$1 - \left(\frac{A^K - 1}{A^K}\right)^{q^K - 1}. \qquad (10.62)$$

In the case $q = 364$ and $K = 1$ this probability of error is

$$1 - \left(1 - \frac{1}{A}\right)^{q-1} \simeq 1 - e^{-(q-1)/A} \simeq 1 - e = 0.63. \qquad (10.63)$$

[The exact answer found from equation (10.62) is 0.631.] Thus random coding is highly likely to lead to a communication failure.

As $K$ gets large, however, we can approximate

$$1 - \left(\frac{A^K - 1}{A^K}\right)^{q^K - 1} = 1 - \left(1 - \frac{1}{A^K}\right)^{q^K - 1} \simeq \frac{q^K - 1}{A^K} \simeq \left(\frac{q}{A}\right)^K. \qquad (10.64)$$

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 reliable, albeit bizarre, coding strategy.

# About Chapter 11

Before reading chapter 11, you should have read chapters 5 and 10 and worked on exercise 9.10 (p.169).

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 $s$th in the code |
| $s$ | the number of a chosen codeword (mnemonic: the *source* selects $s$) |
| $S = 2^K$ | the total number of codewords in the code |
| $K = \log_2 S$ | the number of bits conveyed by the choice of one codeword from $S$, assuming it is chosen with uniform probability |
| $\mathbf{s}$ | 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 $s$ |

# 11

# The Noisy-Channel Coding Theorem

## 11.1 The Theorem

- For every discrete memoryless channel, the channel capacity
$$C = \max_{\mathcal{P}_X} I(X;Y) \tag{11.1}$$
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
$$R(p_b) = \frac{C}{1 - H_2(p_b)}. \tag{11.2}$$
- For given $p_b$, rates greater than $R(p_b)$ are not achievable.

## 11.2 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 $\mathbf{x}^{(s)}$ and $\mathbf{y}$ 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 defined to be jointly typical (to tolerance $\beta$) with respect to the distribution $P(x, y)$ if

$$\begin{aligned}
\mathbf{x} \text{ is typical of } P(\mathbf{x}), \quad &\text{i.e.,} \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{x})} - H(X) \right| < \beta, \\
\mathbf{y} \text{ is typical of } P(\mathbf{y}), \quad &\text{i.e.,} \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{y})} - H(Y) \right| < \beta, \quad (11.3) \\
\text{and } \mathbf{x}, \mathbf{y} \text{ is typical of } P(\mathbf{x}, \mathbf{y}), \quad &\text{i.e.,} \quad \left| \frac{1}{N} \log \frac{1}{P(\mathbf{x}, \mathbf{y})} - H(X, Y) \right| < \beta.
\end{aligned}$$

**The jointly typical set** $J_{N\beta}$ is the set of all jointly typical sequence pairs of length $N$.

Example. Here is a jointly typical pair of length $N = 100$ for the ensemble $P(x,y)$ in which $P(x)$ has $(p_0, p_1) = (0.1, 0.9)$ and $P(x|y)$ corresponds to a binary symmetric channel with noise level 0.2.

```
x   1111111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
y   0011111111000000000000000000000000000000000000000000000000000000000000000000000000111111111111111111
```

Notice that $\mathbf{x}$ has 10 1s, and so is typical of the probability $P(\mathbf{x})$ (at any tolerance $\beta$); and $\mathbf{y}$ has 26 1s, so it is typical of $P(\mathbf{y})$ [because $P(y=1) = 0.26$]; and $\mathbf{x}$ and $\mathbf{y}$ differ in 20 bits, which is the typical number of flips for this channel.

**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 (to tolerance $\beta$) 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,
$$|J_{N\beta}| \leq 2^{N(H(X,Y)+\beta)}. \tag{11.4}$$
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,
$$P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) \leq 2^{-N(I(X;Y)-3\beta)}. \tag{11.5}$$

Proof. The proof of parts 1 and 2 by the law of large numbers follows that of the source coding theorem in chapter 5. [For part 2, let the pair $x, y$ play the role of $x$ in the source coding theorem, replacing $P(x)$ there by the probability distribution $P(x,y)$.]

For the third part,

$$\begin{aligned}
P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) &= \sum_{(\mathbf{x},\mathbf{y}) \in J_{N\beta}} P(\mathbf{x}) P(\mathbf{y}) && (11.6)\\
&\leq |J_{N\beta}|\, 2^{-N(H(X)-\beta)}\, 2^{-N(H(Y)-\beta)} && (11.7)\\
&\leq 2^{N(H(X,Y)+\beta) - N(H(X)+H(Y)-2\beta)} && (11.8)\\
&= 2^{-N(I(X;Y)-3\beta)} && (11.9)
\end{aligned}$$

A cartoon of the jointly typical set is shown in figure 11.1. Two independent typical vectors are jointly typical with probability

$$P((\mathbf{x}', \mathbf{y}') \in J_{N\beta}) \simeq 2^{-N(I(X;Y))} \tag{11.10}$$

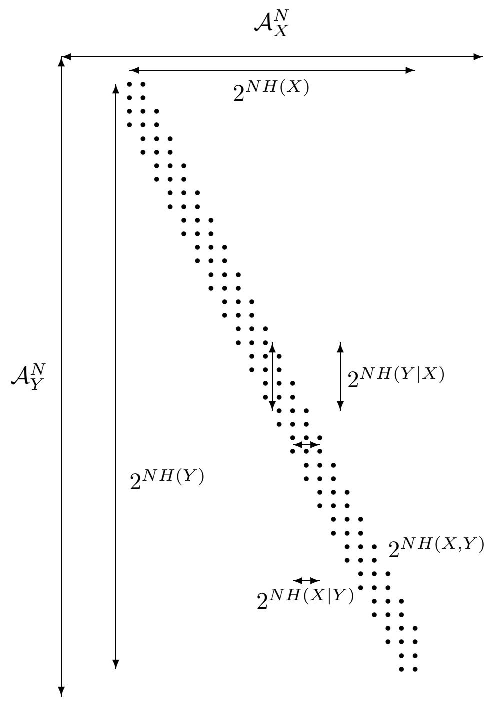


Figure 11.1. The jointly typical set. The horizontal direction represents $\mathcal{A}_X^N$, the set of all input strings of length $N$. The vertical direction represents $\mathcal{A}_Y^N$, the set of all output strings of length $N$. Each dot represents a jointly typical pair of sequences $(\mathbf{x}, \mathbf{y})$. The total number of jointly typical sequences is about $2^{NH(X,Y)}$. [Compare with figure 10.7a, page 190.]

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

$$2^{NH(X,Y)}/2^{NH(X)+NH(Y)} = 2^{-NI(X;Y)}. \tag{11.11}$$

## 11.3 Proof of the noisy-channel coding theorem

### *Analogy*

Imagine that we wish to prove that there is a child in a class of one hundred children who weighs less than 10 kg. Individual children are difficult to catch and weigh. Shannon's method of solving the task is to scoop up all the children in a net and weigh them all at once on a big weighing machine. If we find that their total weight is smaller than 1000 kg then the children's average weight must be smaller than 10 kg, so there must exist *at least one* child who weighs less than 10 kg – indeed there must be many! Shannon's method isn't guaranteed to reveal the existence of an underweight child, since it relies on there being no elephants in the class. But if we use his method and get a total weight smaller than 1000 kg then our task is solved.

### *From skinny children to fantastic codes*

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 this: instead of

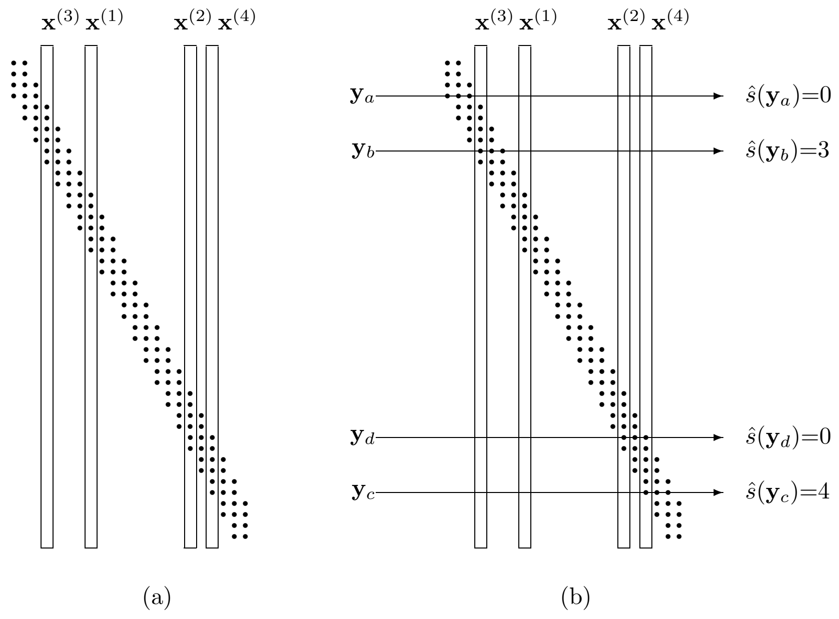

$\mathbf{x}^{(3)}$ $\mathbf{x}^{(1)}$ $\mathbf{x}^{(2)}$ $\mathbf{x}^{(4)}$
$\mathbf{y}_a$
$\mathbf{y}_b$
$\mathbf{y}_d$
$\mathbf{y}_c$
$\hat{s}(\mathbf{y}_a)$=0
$\hat{s}(\mathbf{y}_b)$=3
$\hat{s}(\mathbf{y}_d)$=0
$\hat{s}(\mathbf{y}_c)$=4
(a)
(b)

Figure 11.2. (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 $\mathbf{x}^{(3)}$ only, $\mathbf{y}_b$, is decoded as $\hat{s} = 3$. 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 $\hat{s} = 0$.

constructing a good coding and decoding system and evaluating its error probability, Shannon calculated the average probability of block error of *all* codes, and proved that this average is small. There must then exist individual codes that have small probability of block error.

### *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

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n). \tag{11.12}$$

   A random code is shown schematically in figure 11.2(a).

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

$$P(\mathbf{y}|\mathbf{x}^{(s)}) = \prod_{n=1}^{N} P(y_n|x_n^{(s)}). \tag{11.13}$$

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 11.2.

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,

$$p_{\mathrm{B}}(\mathcal{C}) \equiv P(\hat{s} \neq s|\mathcal{C}). \tag{11.14}$$

This is a difficult quantity to evaluate for any given code.

Second, there is the average over all codes of this block error probability,

$$\langle p_{\mathrm{B}} \rangle \equiv \sum_{\mathcal{C}} P(\hat{s} \neq s|\mathcal{C}) P(\mathcal{C}). \tag{11.15}$$

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 $\mathcal{C}$,

$$p_{\mathrm{BM}}(\mathcal{C}) \equiv \max_{s} P(\hat{s} \neq s|s, \mathcal{C}), \tag{11.16}$$

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.

We will get to this result by first finding the average block error probability, $\langle p_{\mathrm{B}} \rangle$. 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 block error probability is satisfactorily small but whose maximal block error probability is unknown (and could conceivably be enormous), can be modified to make a code of slightly smaller rate whose maximal block error probability is also guaranteed to be small. We modify the code by throwing away the worst 50% of its codewords.

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 in $\mathcal{C}$ 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 the input $\mathbf{x}^{(1)}$ and the output $\mathbf{y}$ are not jointly typical vanishes, by the joint typicality theorem's first part (page 197). We give a name, $\delta$, to the upper bound on this probability, satisfying $\delta \to 0$ as $N \to \infty$; for any desired $\delta$, we can find a blocklength $N(\delta)$ such that the $P((\mathbf{x}^{(1)}, \mathbf{y}) \notin J_{N\beta}) \leq \delta$.

(b) The probability that $\mathbf{x}^{(s')}$ and $\mathbf{y}$ are jointly typical, for a *given* $s' \neq 1$ is $\leq 2^{-N(I(X;Y)-3\beta)}$, by part 3. And there are $(2^{NR'} - 1)$ rival values of $s'$ to worry about.

Thus the average probability of error $\langle p_{\rm B} \rangle$ satisfies:

$$\langle p_{\rm B} \rangle \;\leq\; \delta + \sum_{s'=2}^{2^{NR'}} 2^{-N(I(X;Y)-3\beta)} \tag{11.17}$$

$$\leq\; \delta + 2^{-N(I(X;Y)-R'-3\beta)} \tag{11.18}$$

This can be made $< 2\delta$ by increasing $N$ if

$$R' < I(X;Y) - 3\beta. \tag{11.19}$$

The inequality that bounds the total probability of error $P_{\rm ERR}$ by the sum of the probabilities $P_{s'}$ of all sorts of events $s'$ each of which is sufficient to cause error,

$$P_{\rm TOT} \leq P_1 + P_2 + \ldots,$$

is called a 'union bound'. It is only an equality if the different events that cause error never occur at the same time as each other.

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 $4\delta$. We use these remaining codewords to define a new code. 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*. The resulting code may not be the best code of its rate and length, but it is still good enough to prove the noisy-channel coding theorem, which is what we are trying to do here.

This is a subtle step.

In conclusion, we can 'construct' 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.

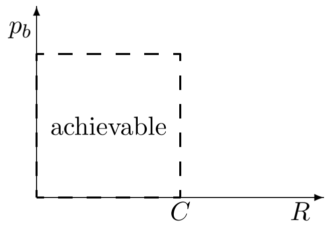


Figure 11.3. Portion of the $R, p_b$ plane proved achievable.

## 11.4 Communication (with errors) above capacity*

We have proved, for any discrete memoryless channel, the achievability of a portion of the $R, p_b$ plane shown in figure 11.3. 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

$$p_b = \frac{1}{2}(1 - 1/R). \tag{11.20}$$

The curve corresponding to this strategy is shown by the dashed line in figure 11.4.

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 11.4. So, how can this optimum be achieved?

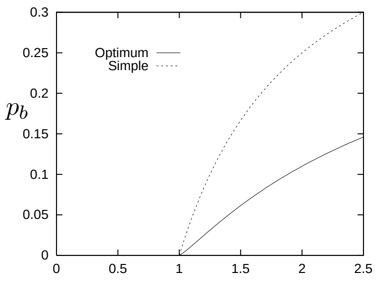


Figure 11.4. A simple bound on achievable points $(R, p_b)$, and Shannon's bound.

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 $K$). 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:

$$R = \frac{C}{1 - H_2(p_b)}. \tag{11.21}$$

## 11.5 The non-achievable region*

The source, encoder, noisy channel and decoder define a Markov chain:

$s \rightarrow \mathbf{x} \rightarrow \mathbf{y} \rightarrow \hat{s}$

$$P(s, \mathbf{x}, \mathbf{y}, \hat{s}) = P(s)P(\mathbf{x}|s)P(\mathbf{y}|\mathbf{x})P(\hat{s}|\mathbf{y}). \tag{11.22}$$

The data processing inequality (exercise 9.10, p.169) 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 11.1:$^{C2}$ 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?

## 11.6 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.?? Exercise 11.2:$^{B2}$ 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.?? Exercise 11.3:$^{C2}$ Show that $I(X;Y)$ is a concave $\frown$ function of the input probability vector $\mathbf{p}$.

Since $I(X;Y)$ is concave $\frown$ 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

$$\frac{\partial I(X;Y)}{\partial p_i} = \lambda \;\text{ for all } i, \tag{11.23}$$

where $\lambda$ 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.** $\mathcal{A}_X$=$\{0,?,1\}$. $\mathcal{A}_Y$=$\{0,1\}$.

$$\begin{tikzcd}[row sep=tiny] 0 \arrow[r] & 0 \\ ? \arrow[ur] \arrow[dr] & \\ 1 \arrow[r] & 1 \end{tikzcd}$$

$$\begin{array}{lll} P(y{=}0|x{=}0){=}1; & P(y{=}0|x{=}?){=}1/2; & P(y{=}0|x{=}1){=}0; \\ P(y{=}1|x{=}0){=}0; & P(y{=}1|x{=}?){=}1/2; & P(y{=}1|x{=}1){=}1. \end{array}$$

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 11.4:$^{B2}$ 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.207 Exercise 11.5:$^{B2}$ Describe the condition, similar to equation (11.23), 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 $\smile$ function of the channel parameters.

Reminder:
The term 'convex $\smile$' means 'convex', and the term 'concave $\frown$' means 'concave'; the little smile and frown symbols are included simply to remind you what convex and concave mean.

3. There may be several optimal input distributions, but they all look the same at the output.

Solutions on p.??

Exercise 11.6:$^{B2}$ 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 11.7:$^{C2}$ Prove that $I(X,Y)$ is a convex $\smile$ function of $Q(y|x)$.

Solutions on p.??

Exercise 11.8:$^{C2}$ 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 $\frown$ 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 11.9: This channel

$$\begin{array}{ll} P(y{=}0|x{=}0){=}0.7; & P(y{=}0|x{=}1){=}0.1; \\ P(y{=}?|x{=}0){=}0.2; & P(y{=}?|x{=}1){=}0.2; \\ P(y{=}1|x{=}0){=}0.1; & P(y{=}1|x{=}1){=}0.7. \end{array} \tag{11.24}$$

is a symmetric channel because its outputs can be partitioned into $(0, 1)$ and ?, so that the matrix can be rewritten:

$$\begin{array}{ll} \hline P(y{=}0|x{=}0){=}0.7; & P(y{=}0|x{=}1){=}0.1; \\ P(y{=}1|x{=}0){=}0.1; & P(y{=}1|x{=}1){=}0.7; \\ \hline P(y{=}?|x{=}0){=}0.2; & P(y{=}?|x{=}1){=}0.2. \\ \hline \end{array} \tag{11.25}$$

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.?? Exercise 11.10:$^{C2}$ Prove that for a symmetric channel with any number of inputs, the uniform distribution over the inputs is an optimal input distribution.

Solutions on p.208 Exercise 11.11:$^{B2}$ Are there channels that are not symmetric whose optimal input distributions are uniform? Find one, or prove there are none.

## 11.7 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 $\epsilon$ 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 $\epsilon$.

Presumably, the smaller $\epsilon$ is and the closer $R$ is to $C$, the larger $N$ has to be.

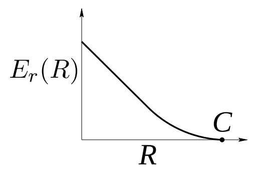


Figure 11.5. A typical random-coding exponent

### *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:

$$p_{\mathrm{B}} \leq \exp\left[-NE_r(R)\right] \tag{11.26}$$

where $E_r(R)$ is the *random-coding exponent* of the channel, a convex$\smile$, decreasing, positive function of $R$ for $0 \leq R < C$. The random-coding exponent is also known as the reliability function.

[By an expurgation argument it can also be shown that there exist block codes for which the *maximal* probability of error $p_{\mathrm{BM}}$ is also exponentially small in $N$.]

The definition of $E_r(R)$ is given in Gallager (1968), p. 139. $E_r(R)$ approaches zero as $R \rightarrow C$; the typical behaviour of this function is illustrated in figure 11.5. 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 $p_{\mathrm{B}}$ 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

$$p_B \gtrsim \exp[-NE_{\mathrm{sp}}(R)], \tag{11.27}$$

where the function $E_{\mathrm{sp}}(R)$, the *sphere-packing exponent* of the channel, is a convex$\smile$, 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 $p_{\mathrm{B}}$, we can, after working out the relevant functions $C$, $E_{\mathrm{r}}(R)$, and $E_{\mathrm{sp}}(R)$, advise him that there exists a solution to his problem using a particular blocklength $N$; indeed that almost any randomly chosen code with that blocklength should do the job. Unfortunately we have not found out how to implement these encoders and decoders in practice; 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 $E_{\mathrm{r}}(R)$ and $E_{\mathrm{sp}}(R)$ is diminished.

## 11.8 Exercises

Exercise 11.12:$^{A2}$ A binary erasure channel with input $x$ and output $y$ has transition probability matrix:

$$\mathbf{Q} = \begin{bmatrix} 1-q & 0 \\ q & q \\ 0 & 1-q \end{bmatrix}$$

$$\begin{tikzcd} 0 \arrow[r] \arrow[dr] & 0 \\ & ? \\ 1 \arrow[r] \arrow[ur] & 1 \end{tikzcd}$$

Find the *mutual information* $I(X;Y)$ between the input and output for general input distribution $\{p_0, p_1\}$, and show that the *capacity* of this channel is $C = 1 - q$ bits.

A 'Z channel' has transition probability matrix:

$$\mathbf{Q} = \begin{bmatrix} 1 & q \\ 0 & 1-q \end{bmatrix}$$

$$\begin{tikzcd} 0 \arrow[r] & 0 \\ 1 \arrow[r] \arrow[ur] & 1 \end{tikzcd}$$

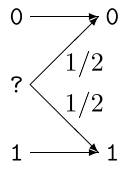
Show that, using a (2,1) code, **two** uses of a Z channel can be made to emulate **one** use of an erasure channel, and state the erasure probability of that erasure channel. Hence show that the capacity of the Z channel, $C_Z$, satisfies $C_Z \geq \frac{1}{2}(1-q)$ bits.

Explain why the result $C_Z \geq \frac{1}{2}(1-q)$ is an inequality rather than an equality.

# *Solutions to Chapter 11's exercises*

Solution to exercise 11.4 (p. 203): If the input distribution is $\mathbf{p} = (p_0, p_?, p_1)$, the mutual information is

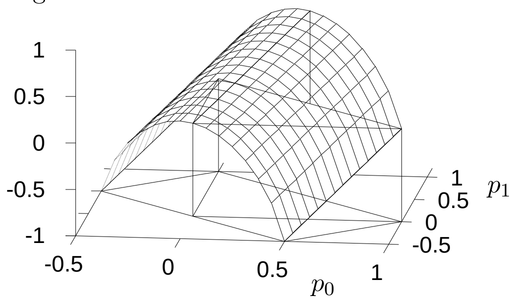
$$I(X;Y) = H(Y) - H(Y|X) = H_2(p_0 + p_?/2) - p_?. \qquad (11.28)$$

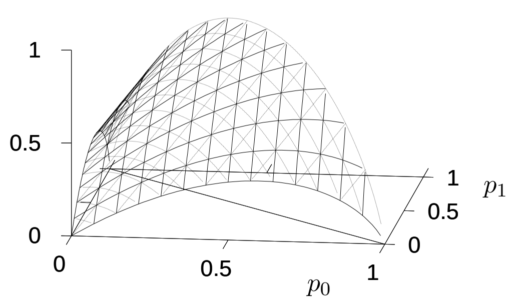


We can build a good sketch of this function in two ways: by careful inspection of the function, or by looking at special cases.

For the plots, the two-dimensional representation of $\mathbf{p}$ I will use has $p_0$ and $p_1$ as the independent variables, so that $\mathbf{p} = (p_0, p_?, p_1) = (p_0, (1-p_0-p_1), p_1)$.

By inspection: If we use the quantities $p_* \equiv p_0 + p_?/2$ and $p_?$ as our two degrees of freedom, the mutual information becomes very simple: $I(X;Y) = H_2(p_*) - p_?$. Converting back to $p_0 = p_* - p_?/2$ and $p_1 = 1 - p_* - p_?/2$, we obtain the sketch shown at the left below. This function is like a tunnel rising up the direction of increasing $p_0$ and $p_1$. To obtain the required plot of $I(X;Y)$ we have to strip away the parts of this tunnel that live outside the feasible simplex of probabilities; we do this by redrawing the surface, showing only the parts where $p_0 > 0$ and $p_1 > 0$. A full plot of the function is shown at the right.



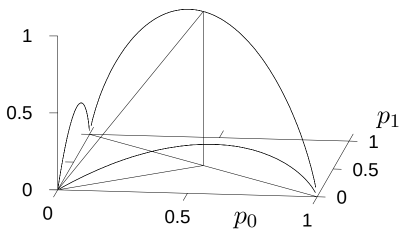


Special cases: In the special case $p_? = 0$, the channel is a noiseless binary channel, and $I(X;Y) = H_2(p_0)$.

In the special case $p_0 = p_1$, the term $H_2(p_0 + p_?/2)$ is equal to 1, so $I(X;Y) = 1 - p_?$.

In the special case $p_0 = 0$, the channel is a Z channel with error probability 0.5. We know how to sketch that, from the previous chapter (figure 10.2).

These special cases allow us to construct the skeleton shown in the margin.



Solution to exercise 11.5 (p. 203): Necessary and sufficient conditions for $\mathbf{p}$ to maximize $I(X;Y)$ are

$$\left.\begin{array}{rcll} \frac{\partial I(X;Y)}{p_i} & = & \lambda & \text{and} \quad p_i > 0 \\ \frac{\partial I(X;Y)}{p_i} & \leq & \lambda & \text{and} \quad p_i = 0 \end{array}\right\} \text{ for all } i, \qquad (11.29)$$

where $\lambda$ 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 (11.29) holds.

Solution to exercise 11.11 (p.205): 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 whereas the optimal input distribution is just $(I-1)$–dimensional; so in the $I(J-1)$-dimensional space of perturbations around a symmetric channel, we expect there to be a 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.

$$\mathbf{Q} = \left[ \begin{array}{cccc} 0.9585 & 0.0415 & 0.35 & 0.0 \\ 0.0415 & 0.9585 & 0.0 & 0.35 \\ 0 & 0 & 0.65 & 0 \\ 0 & 0 & 0 & 0.65 \end{array} \right] \tag{11.30}$$

Solution to exercise 11.12 (p.206): $I(X;Y) = H(X) - H(X|Y)$.

$I(X;Y) = H_2(p_0) - qH_2(p_0)$.

Maximize over $p_0$, get $C = 1 - q$.

The (2,1) code is $\{\texttt{01}, \texttt{10}\}$. With probability $q$, the `1` is lost, giving the output `00`, which is equivalent to the "?" output of the Binary Erasure Channel. With probability $(1-q)$ there is no error; the two input words and the same two output words are identified with the `0` and `1` of the BEC. The equivalent BEC has erasure probability $q$. Now, this shows the capacity of the Z channel is at least half that of the BEC.

This result is a bound, not an inequality, because our code constrains the input distribution to be 50:50, which is not necessariy optimal, and because we've introduced simple anticorrelations among successive bits, which optimal codes for the channel would not do.

# *About Chapter 12*

Before reading chapter 12, you should have read chapters 10 and 11.

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 $\mu$ and variance $\sigma^2$, which we write:

$$y \sim \text{Normal}(\mu, \sigma^2), \text{ or } P(y) = \text{Normal}(y; \mu, \sigma^2), \tag{11.31}$$

then the distribution of $y$ is:

$$P(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-(y-\mu)^2/2\sigma^2\right]. \tag{11.32}$$

[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

$$P(\mathbf{y}|\mathbf{x}, \mathbf{A}) = \frac{1}{Z(\mathbf{A})} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x})^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x})\right), \tag{11.33}$$

where $\mathbf{x}$ is the mean of the distribution, $\mathbf{A}$ is the inverse of the variance-covariance matrix, and the normalizing constant is $Z(\mathbf{A}) = (\det(\mathbf{A}/2\pi))^{-1/2}$.

This distribution has the property that the variance $\Sigma_{ii}$ of $y_i$, and the covariance $\Sigma_{ij}$ of $y_i$ and $y_j$ are given by

$$\Sigma_{ij} \equiv \mathcal{E}\left[(y_i - \bar{y}_i)(y_j - \bar{y}_j)\right] = A^{-1}_{ij}, \tag{11.34}$$

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.

# 12

# *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. What can Shannon tell us about these continuous channels? And 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?

## 12.1 The Gaussian Channel

The most popular model of a real-input, real-output channel 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:

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-(y-x)^2/2\sigma^2\right]. \qquad (12.1)$$

This channel has a continuous input and output but is discrete in time. We will show below that certain continuous-time channels are 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 in terms of a continuous-time channel* *

Consider a physical (electrical, say) channel with inputs and outputs that are continuous in time. We put in $x(t)$, 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:

$$\int_0^T dt\, [x(t)]^2/T \leq P. \qquad (12.2)$$

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* $N_0$.

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 $\phi_n(t)$,

$$x(t) = \sum_{n=1}^{N} x_n \phi_n(t), \tag{12.3}$$

where $\int_0^T dt\, \phi_n(t)\phi_m(t) = \delta_{nm}$. The receiver can then compute the scalars:

$$\begin{aligned} y_n \;\equiv\; \int_0^T dt\, \phi_n(t) y(t) \;&=\; x_n + \int_0^T dt\, \phi_n(t) n(t) && (12.4)\\ &\equiv\; x_n + n_n && (12.5)\end{aligned}$$

for $n = 1 \ldots N$. 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:

$$n_n \sim \text{Normal}(0, N_0/2), \tag{12.6}$$

where $N_0$ is the spectral density introduced above. [This is the definition of $N_0$.] 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 signal amplitudes $x_n$,

$$\sum_n x_n^2 \leq PT \qquad \Rightarrow \qquad \overline{x_n^2} \leq \frac{PT}{N} \tag{12.7}$$

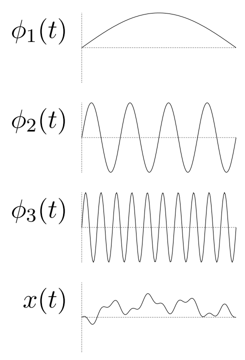


Figure 12.1. Three basis functions, and a weighted combination of them, $x(t) = \sum_{n=1}^{N} x_n \phi_n(t)$, with $x_1 = 0.4$, $x_2 = -0.2$, and $x_3 = 0.1$.

Before returning to the Gaussian channel, let us define the *bandwidth* (measured in Hertz) of the continuous channel to be:

$$W = \frac{N^{\max}}{2T}, \tag{12.8}$$

where $N^{\max}$ is the maximum number of orthonormal functions that can be produced in an interval of length $T$. This definition can be motivated 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 of $R$ 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.

It is conventional to measure the rate-compensated 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 $N_0$:

$$E_b/N_0 = \frac{\overline{x_n^2}}{2\sigma^2 R}. \tag{12.9}$$

$E_b/N_0$ is dimensionless, but it is usually reported in the units of decibels; the value given is $10 \log_{10} E_b/N_0$.

The difference in $E_b/N_0$ is one of the measures used to compare coding schemes for Gaussian channels.

## 12.2 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

$$P(\mathbf{n}) = \left[\det\left(\frac{\mathbf{A}}{2\pi}\right)\right]^{1/2} \exp\left(-\frac{1}{2}\mathbf{n}^{\mathsf{T}}\mathbf{A}\mathbf{n}\right), \tag{12.10}$$

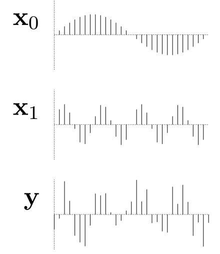


Figure 12.2. Two pulses $\mathbf{x}_0$ and $\mathbf{x}_1$, represented as 31-dimensional vectors, and a noisy version of one of them, $\mathbf{y}$.

where $\mathbf{A}$ is the inverse of the variance-covariance matrix of the noise, a symmetric and positive-definite matrix. (If $\mathbf{A}$ is a multiple of the identity matrix, $\mathbf{I}/\sigma^2$, then the noise is 'white'. For more general $\mathbf{A}$, the noise is 'coloured'.) The probability of the received vector $\mathbf{y}$ given that the source signal was $s$ (either zero or one) is then

$$P(\mathbf{y}|s) = \left[\det\left(\frac{\mathbf{A}}{2\pi}\right)\right]^{1/2} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x}_s)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_s)\right). \tag{12.11}$$

The optimal detector is based on the posterior probability ratio:

$$\begin{aligned}
\frac{P(s=1|\mathbf{y})}{P(s=0|\mathbf{y})} &= \frac{P(\mathbf{y}|s=1)}{P(\mathbf{y}|s=0)}\frac{P(s=1)}{P(s=0)} & (12.12)\\
&= \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{x}_1)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_1) + \frac{1}{2}(\mathbf{y}-\mathbf{x}_0)^{\mathsf{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}_0) + \ln\frac{P(s=1)}{P(s=0)}\right) & \\
&= \exp\left(\mathbf{y}^{\mathsf{T}}\mathbf{A}(\mathbf{x}_1-\mathbf{x}_0) + \theta\right), & (12.13)
\end{aligned}$$

where $\theta$ is a constant independent of the received vector $\mathbf{y}$,

$$\theta = -\frac{1}{2}\mathbf{x}_1^{\mathsf{T}}\mathbf{A}\mathbf{x}_1 + \frac{1}{2}\mathbf{x}_0^{\mathsf{T}}\mathbf{A}\mathbf{x}_0 + \ln\frac{P(s=1)}{P(s=0)}. \tag{12.14}$$

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*:

$$a(\mathbf{y}) \equiv \mathbf{y}^{\mathsf{T}}\mathbf{A}(\mathbf{x}_1-\mathbf{x}_0) + \theta \tag{12.15}$$

with the decisions

$$\begin{array}{lcl} a(\mathbf{y}) > 0 & \rightarrow & \text{guess } s=1 \\ a(\mathbf{y}) < 0 & \rightarrow & \text{guess } s=0 \\ a(\mathbf{y}) = 0 & \rightarrow & \text{guess either.} \end{array} \tag{12.16}$$

Notice that $a(\mathbf{y})$ is a linear function of the received vector,

$$a(\mathbf{y}) = \mathbf{w}^{\mathsf{T}}\mathbf{y} + \theta, \tag{12.17}$$

where $\mathbf{w} \equiv \mathbf{A}(\mathbf{x}_1 - \mathbf{x}_0)$.

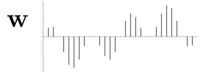


Figure 12.3. The weight vector $\mathbf{w} \propto \mathbf{x}_1 - \mathbf{x}_0$ that is used to discriminate between $\mathbf{x}_0$ and $\mathbf{x}_1$.

## 12.3 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 $d_1d_2d_3\ldots d_N$ by setting $x = d_1d_2d_3\ldots d_N000\ldots000$. The amount of error-free information conveyed in just a single transmission could be made 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 – see 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

$$v(x) = x^2 \tag{12.18}$$

such that the 'average power' $\overline{x^2}$ of the input is constrained. We 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.

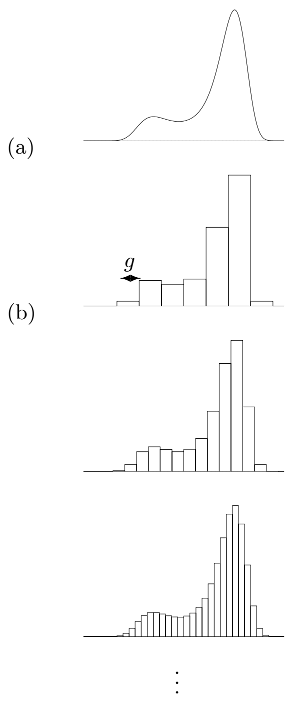


Figure 12.4. (a) A probability density $P(x)$. Question: can we define the 'entropy' of this density? (b) We could evaluate the entropies of a sequence of probability distributions with decreasing grain-size $g$, but these entropies tend to $\int P(x) \log \frac{1}{P(x)g}\,\mathrm{d}x$, which is not independent of $g$: the entropy goes up by one bit for every halving of $g$. $\int P(x) \log \frac{1}{P(x)}\,\mathrm{d}x$ is an illegal integral.

### *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 $[x]^{-1}$).

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,

$$I(X;Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}. \tag{12.19}$$

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 and replace the sum by an integral:

$$\begin{aligned} I(X;Y) &= \int dx\, dy\, P(x,y) \log \frac{P(x,y)}{P(x)P(y)} && (12.20) \\ &= \int dx\, dy\, P(x)P(y|x) \log \frac{P(y|x)}{P(y)}. && (12.21) \end{aligned}$$

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 still measure the maximum error free communication rate of this real channel, as it did for the discrete channel?

**Exercise 12.1:**$^{D3}$ 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 12.2:**$^{B2}$ Show that the mutual information $I(X;Y)$, in the case of this optimized distribution, is

$$C = \frac{1}{2} \log \left( 1 + \frac{v}{\sigma^2} \right). \tag{12.22}$$

This is an important result. We see that the capacity of the Gaussian channel is a function of the *signal to noise ratio* $v/\sigma^2$.

## *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:

$$\begin{aligned} P(x|y) &\propto P(y|x)P(x) && (12.23) \\ &\propto \exp(-(y-x)^2/2\sigma^2) \exp(-x^2/2v) && (12.24) \\ &= \text{Normal}\left( x; \frac{v}{v+\sigma^2}\, y\, , \left( \frac{1}{v} + \frac{1}{\sigma^2} \right)^{-1} \right). && (12.25) \end{aligned}$$

[The step from (12.24) to (12.25) is made by completing the square in the exponent.] 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, $x = 0$:

$$\frac{v}{v+\sigma^2}\, y = \frac{1/\sigma^2}{1/v + 1/\sigma^2}\, y + \frac{1/v}{1/v + 1/\sigma^2}\, 0. \tag{12.26}$$

The weights $1/\sigma^2$ and $1/v$ are the *precisions* of the two Gaussians that we multiplied together in equation (12.24): 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 dual 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 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

$$V(r, N) = \tfrac{\pi^{N/2}}{\Gamma(N/2+1)} r^N. \tag{12.27}$$

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 $\mathbf{x}$:

$$M \leq \left( \frac{\sqrt{N(v+\sigma^2)}}{\sqrt{N\sigma^2}} \right)^N \tag{12.28}$$

Thus the capacity is bounded by:

$$C = \frac{1}{N} \log M \leq \frac{1}{2} \log \left( 1 + \frac{v}{\sigma^2} \right). \tag{12.29}$$

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:

$$C = W \log\left(1 + \frac{P}{N_0 W}\right) \text{ bits per second.} \tag{12.30}$$

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 use of that power? Introducing $W_0 = P/N_0$, i.e., the bandwidth for which the signal to noise ratio is 1, figure 12.5 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 higher-power, narrow-bandwidth transmitters.

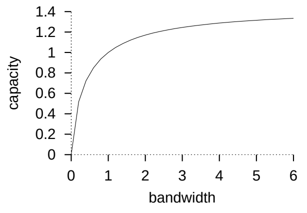


Figure 12.5. Capacity versus bandwidth for a real channel: $C/W_0 = W/W_0 \log(1 + W_0/W)$ as a function of $W/W_0$.

## 12.4 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 are 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. Repetition codes are an example of a bad code family. (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 $S = 2^K$ codewords $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(2^K)}\}$, each of length $N$: $\mathbf{x}^{(s)} \in \mathcal{A}_X^N$. The signal to be encoded, $s$, which comes from an alphabet of size $2^K$, is encoded as $\mathbf{x}^{(s)}$.

**A linear** $(N, K)$ **block code** is a block code in which the codewords $\{\mathbf{x}^{(s)}\}$ make up 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 $\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}$ modulo 2.

The codewords $\{\mathbf{t}\}$ can be defined as the set of vectors satisfying $\mathbf{Ht} = 0 \bmod 2$, where $\mathbf{H}$ is the *parity check matrix* of the code.

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 $\mathbf{G}^{\mathsf{T}}\mathbf{s} \bmod 2$.

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1000 \\ 0100 \\ 0010 \\ 0001 \\ 1110 \\ 0111 \\ 1011 \end{bmatrix}$$

Coding theory was born with the work of Hamming, who invented a family of practical 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 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 resulting transmitted bit stream is the convolution of the source stream with a 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 54.

## *Are linear codes 'good'?*

One might ask, is the reason that the Shannon limit is not achieved in practice 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 14), 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 $\mathbf{G}^{\mathsf{T}}\mathbf{s} + \mathbf{n} = \mathbf{r}$) is in fact NP-complete (Berlekamp *et al.* 1978). NP-complete problems are computational problems

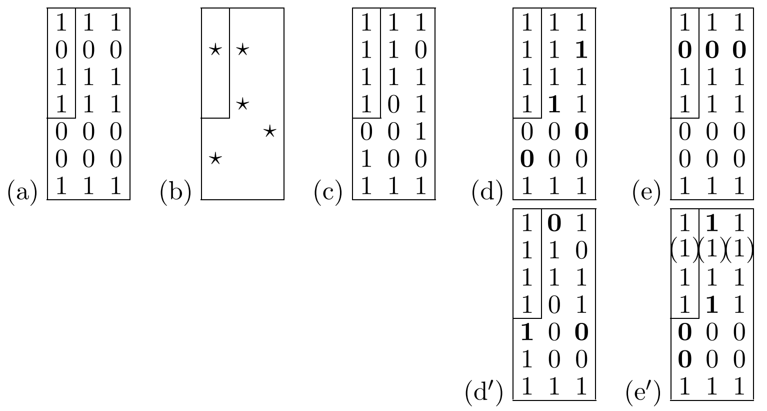


Figure 12.6. A product code. (a) A string 1011 encoded using a concatenated code consisting of two Hamming codes, $H(3,1)$ and $H(7,4)$. (b) a noise pattern that flips 5 bits. (c) The received vector. (d) After decoding using the horizontal (3,1) decoder, and (e) after subsequently using the vertical (7,4) decoder. The decoded vector matches the original.
(d′, e′) After decoding in the other order, three errors still remain.

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 for which there is a fast decoding algorithm.

## *Concatenation*

One trick for building codes with practical decoders is the idea of concatenation.

$$\mathcal{C}' \to \underbrace{\mathcal{C} \to Q \to \mathcal{D}}_{Q'} \to \mathcal{D}'$$

An encoder-channel-decoder system $\mathcal{C} \to Q \to \mathcal{D}$ 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.

As an example, figure 12.6 shows a product code in which we encode first with the repetition code $\mathrm{R}_3$ (also known as the Hamming code $H(3,1)$) 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 $\mathrm{R}_3$.

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 12.6(c-e) shows what happens if we receive the codeword of figure 12.6(a) 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 12.6(d′,e′) 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.

### *Interleaving*

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 B.5) 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.

Solutions on p.225

**Exercise 12.3:**$^{B2}$ 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 if one used interleaving and treated 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).

$$
\mathbf{H} = \begin{bmatrix}
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} \\
0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 12.8. Example of a low-density parity-check matrix with weight per column equal to 3. The parity-check 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 $N \simeq 10000$.

## 12.5 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, and codes based on them*

**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 was 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 – 15 – for its convolutional code and a larger Reed-Solomon code was developed by the Jet Propulsion Laboratory. The details of this code are unpublished outside JPL, and the decoding is only possible using a room full of 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 two constituent codes were 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 *message-passing* algorithm called the *sum-product algorithm* or *probability propagation*.

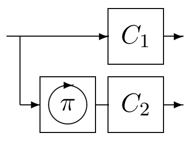


Figure 12.7. The encoder of a turbo code. Each box $C_1$, $C_2$, contains a convolutional code. The source bits are reordered using a permutation $\pi$ before they are fed to $C_2$. The transmitted codeword is obtained by

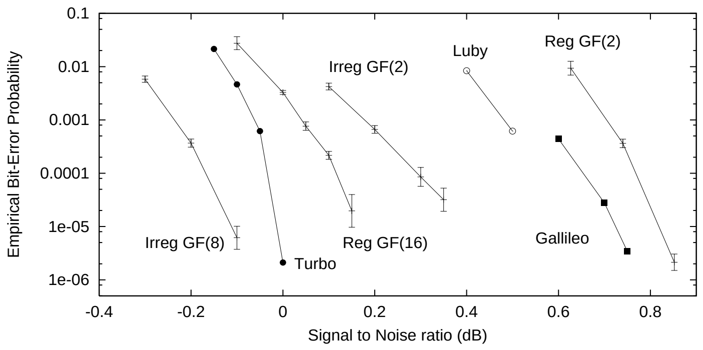


Figure 12.9. State-of-the-art codes. The figure shows the performance of various codes with rate 1/4 over the Gaussian channel. From left to right: Irregular LDPCC over $GF(8)$, blocklength 48000 bits (Davey 1999); JPL turbo code (JPL 1996) blocklength 65536; Regular LDPCC over $GF(16)$, blocklength 24448 bits (Davey and MacKay 1998a); Irregular binary LDPCC, blocklength 16000 bits (Davey 1999); Luby *et al.* (1998) irregular binary LDPCC, blocklength 64000 bits; JPL code for Galileo (in 1992, this was the best known code of rate 1/4); Regular binary LDPCC: blocklength 40000 bits (MacKay 1999). The Shannon limit is at about $-0.79$ dB.

*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 12.5). In a 'regular' low-density parity-check code, the non-zero entries are arranged at random in the matrix subject to 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. [The number of non-zero entries in each column exceeds one, so this parity check matrix is in 'non-systematic' form.]

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 *message-passing* 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 49 and 53.

Only the Galileo code and turbo codes outperform the original regular, binary Gallager codes. The best known Gallager codes, which are irregular, outperform the Galileo code and turbo codes too (Davey and MacKay 1998a; Richardson *et al.* 2001). The performance of the above codes are compared for Gaussian channels in figure 12.9.

## 12.6 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. For a non-random code, encoding 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.

## 12.7 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 pattern. The likely errors affecting the film involve dirt and scratches, which produce large numbers of 1s and 0s respectively. We therefore want none of the codewords to look like all-1s or all-0s, 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.

## 12.8 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 the receiver's time, $u$, is an imperfectly known function $u(t)$ of the transmitter's time $t$, then the capacity of this channel for communication must be reduced. The theory of such channels is incomplete, compared with the synchronized channels we have discussed thus far. Not even the capacity of channels with synchronization errors is known (Levenshtein 1966; Ferreira *et al.* 1997); codes for reliable communication over channels with synchronisation errors are an active research area (Davey and MacKay 2001).

## 12.9 Exercises

*The Gaussian channel*

Exercise 12.4:$^{B2}$ Consider a Gaussian channel with a real input $x$, and signal to noise ratio $v/\sigma^2$.

(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

$$y \to y' = \begin{cases} 1 & y > 0 \\ 0 & y \leq 0 \end{cases}, \tag{12.31}$$

what is the capacity $C''$ of the resulting channel?

(d) Plot the three capacities above as a function of $v/\sigma^2$ from 0.1 to 2. [You'll need to do a numerical integral to evaluate $C'$.]

Exercise 12.5:$^{B3}$ For large integers $K$ and $N$, what fraction of all binary error-correcting codes of length $N$ and rate $R = K/N$ are *linear* codes? [The answer will depend on whether you choose to define the code to be an *ordered* list of $2^K$ codewords, that is, a mapping from $s \in \{1, 2, \ldots, 2^K\}$ to $\mathbf{x}^{(s)}$, or to define the code to be an unordered list, so that two codes consisting of the same codewords are identical. The latter definition is preferred. A code is a set of codewords. How the encoder operates is not part of the definition of the code.]

*Erasure channels*

Exercise 12.6:$^{B5}$ Design a code for the binary erasure channel, and a decoding algorithm, and evaluate their probability of error.

Exercise 12.7:$^{B5}$ Design a code for the $q$-ary erasure channel, whose input $x$ is drawn from $0, 1, 2, 3, \ldots, (q-1)$, and whose output $y$ is equal to $x$ with probability $(1-f)$ and equal to ? otherwise. [This erasure channel is a good model for packets transmitted over the internet, which are either received reliably or are lost.]

*Further reading*

Exercise 12.8:$^{C2}$ 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? Some information is provided in the solution section. See `http://www.acnc.com/raid2.html` and `http://www.dfountain.com/` for more.

[Some people say RAID stands for 'redundant array of inexpensive discs', but I think that's silly – RAID would still be a good idea even if the discs were expensive!]

# *Solutions to Chapter 12's exercises*

Solution to exercise 12.1 (p.214): Introduce a Lagrange multiplier $\lambda$ for the power constraint and another, $\mu$, for the constraint of normalization of $P(x)$.

$$\begin{aligned} F &= I(X;Y) - \lambda \textstyle\int dx P(x) x^2 - \mu \textstyle\int dx P(x) && (12.32) \\ &= \int dx\, P(x) \left[ \int dy\, P(y|x) \log \frac{P(y|x)}{P(y)} - \lambda x^2 - \mu \right] && (12.33) \end{aligned}$$

We make the functional derivative with respect to $P(x^*)$.

$$\begin{aligned} \frac{\delta F}{\delta P(x^*)} &= \int dy\, P(y|x^*) \log \frac{P(y|x^*)}{P(y)} - \lambda x^{*2} - \mu \\ &\quad - \int dx\, P(x) \int dy\, P(y|x) \frac{1}{P(y)} \frac{\delta P(y)}{\delta P(x^*)}. && (12.34) \end{aligned}$$

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 $\mu$ 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:

$$\int dy P(y|x) \log \frac{P(y|x)}{P(y)} - \lambda x^2 - \mu' = 0 \qquad (12.35)$$

$$\Rightarrow \int dy\, \frac{\exp(-(y-x)^2/2\sigma^2)}{\sqrt{2\pi\sigma^2}} \log\left[P(y)\sigma\right] = -\lambda x^2 - \mu' - \frac{1}{2} \qquad (12.36)$$

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 12.2 (p.214): 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:

$$\begin{aligned} I(X;Y) &= \int \mathrm{d}x\, \mathrm{d}y\, P(x) P(y|x) \ln P(y|x) - \int \mathrm{d}y\, P(y) \ln P(y) && (12.37) \\ &= \frac{1}{2} \ln \frac{1}{\sigma^2} - \frac{1}{2} \ln \frac{1}{v + \sigma^2} && (12.38) \\ &= \frac{1}{2} \ln \left(1 + \frac{v}{\sigma^2}\right). && (12.39) \end{aligned}$$

Solution to exercise 12.3 (p.219): 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

$$C = 1 - \left(\frac{1}{N} H_2(b) + b\right) = 1 - 0.207 = 0.793. \tag{12.40}$$

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 12.4 (p.222):

(a) Putting together the results of exercises 12.1 and 12.2, we deduce that a Gaussian channel with real input $x$, and signal to noise ratio $v/\sigma^2$ has capacity

$$C = \frac{1}{2} \ln\left(1 + \frac{v}{\sigma^2}\right). \tag{12.41}$$

(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,

$$C'' = \int_{-\infty}^{\infty} \mathrm{d}y\, N(y;0) \log N(y;0) - \int_{-\infty}^{\infty} \mathrm{d}y\, P(y) \log P(y) \tag{12.42}$$

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 (12.41), 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 $\Phi$ defined on page 187. The capacity is

$$C'' = 1 - H_2(f), \text{ where } f = \Phi(\sqrt{v}/\sigma). \tag{12.43}$$

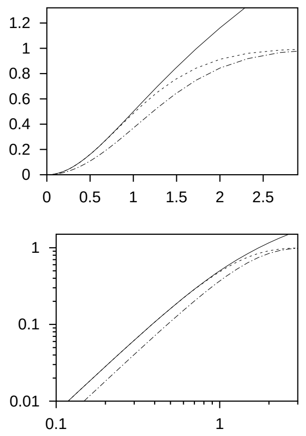


Figure 12.10. Capacities (from top to bottom in each graph) $C$, $C'$, and $C''$, versus the signal to noise ratio ($\sqrt{v}/\sigma$). The lower graph is a log-log plot.

Solution to exercise 12.6 (p.223): The design of good codes for erasure channels is an active research area (Spielman 1996; Byers *et al.* 1998). Have fun!

Solution to exercise 12.8 (p.223): 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 12.9: 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 12.9 (p.226): The (7,4) 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. [c.f. exercise 14.12 (p.256) with $q = 2$: there are no binary MDS codes. This deficit is discussed further in section 14.9]

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. A better code would be the digital fountain (Byers *et al.* 1998), which I hope to describe in a future edition of this book.

# Part III

# Further Topics in Information Theory

# *About Chapter 13*

In chapters 1–12, 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.

*While the contents of this chapter are nonexaminable, I would like you to think about the topic of efficient information retrieval, since this is one of the problems that brains seem to solve effortlessly, and content-addressable memory is one of the topics we will study in the neural networks section of this course.*

# 13

# Hash Codes: Codes for Efficient Information Retrieval *

## 13.1 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.

| | |
|---|---|
| string length | $N \simeq 200$ |
| number of strings | $S \simeq 2^{23}$ |
| number of possible strings | $2^N \simeq 2^{200}$ |

Figure 13.1. Cast of characters

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 that, 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 memory full of phone numbers to find the required 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 $2^N \log_2 S$, $\log_2 S$ being 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 if 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, since on average five million pairwise comparisons will be made.

Exercise 13.1:$^{B2}$ 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 $S + N$ binary comparisons. (Note that you don't have to compare the whole string of length $N$; show that you need on average two binary comparisons 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 improves on the look-up table and the raw list by using an *alphabetically ordered list*.

**Alphabetical 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 $\mathbf{x} \rightarrow s$ from $N$ bits to $\log_2 S$ bits. 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 compression 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 that 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.

| | |
|---|---|
| string length | $N \simeq 200$ |
| number of strings | $S \simeq 2^{23}$ |
| size of hash function | $M \simeq 30$ bits |
| size of hash table | $2^M \simeq 2^{30}$ |

Figure 13.2. Revised cast of characters

## 13.2 Hash codes

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*, which 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 $\mathbf{h}$. [Regardless 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 $\leq 65536$, the string is hashed twice in this way, with the initial running total being set to 0 and 1 respectively (figure 13.2). The result is a 16-bit hash.

Having picked a hash function $\mathbf{h}(\mathbf{x})$, we implement an 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

```
unsigned char Rand8[256];            /* This array contains a random
                                        permutation from 0..255 to 0..255  */
int Hash(char *x) {                  /* x is a pointer to the first char;  */
     int h;                          /*    *x is the first character       */
     unsigned char h1, h2;

     if (*x == 0) return 0;          /* Special handling of empty string   */
     h1 = *x; h2 = *x + 1;           /* Initialize two hashes              */
     x++;                            /* Proceed to the next character      */
     while (*x) {
         h1 = Rand8[h1 ^ *x];        /* Exclusive-or with the two hashes   */
         h2 = Rand8[h2 ^ *x];        /*    and put through the randomizer  */
         x++;
     }                               /* End of string is reached when *x=0 */
     h = ((int)(h1)<<8) | (int) h2 ; /* Shift h1 left 8 bits and add h2    */
     return h ;                      /* Hah is concatenation of h1 and h2  */
}
```

Algorithm 13.1. C code implementing the variable string exclusive-or method to create a hash h in the range $0 \ldots 65535$ from a string x. Author: Thomas Niemann.

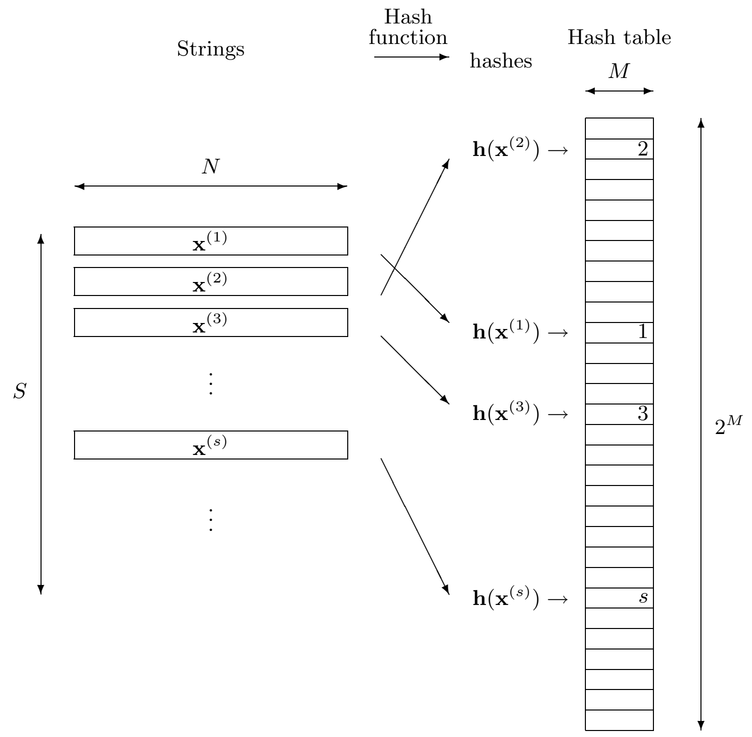

Strings
Hash function
hashes
Hash table
$M$
$N$
$S$
$\mathbf{x}^{(1)}$
$\mathbf{x}^{(2)}$
$\mathbf{x}^{(3)}$
$\mathbf{x}^{(s)}$
$\mathbf{h}(\mathbf{x}^{(2)}) \rightarrow$
$\mathbf{h}(\mathbf{x}^{(1)}) \rightarrow$
$\mathbf{h}(\mathbf{x}^{(3)}) \rightarrow$
$\mathbf{h}(\mathbf{x}^{(s)}) \rightarrow$
2
1
3
$s$
$2^M$

Figure 13.3. Use of hash functions for information retrieval. For each string $\mathbf{x}^{(s)}$, the hash $\mathbf{h} = \mathbf{h}(\mathbf{x}^{(s)})$ is computed, and the value of $s$ is written into the $\mathbf{h}$th row of the hash table. Blank rows in the hash table contain the value zero.

0 and $S$. This table is initially set to zero throughout. Each memory $\mathbf{x}^{(s)}$ 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* between $\mathbf{x}^{(s)}$ 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 13.2:$^{B2}$ 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 original source strings are random, and 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.

## 13.3 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 $\mathbf{x}$.

## 13.4 Planning for collisions: a birthday problem

Exercise 13.3:$^{A2}$ 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 $S$?

[Notice the similarity of this problem to exercise 10.20 (p.188).]

## 13.5 Other roles for hash codes

### *Checking arithmetic*

If you wish to check an addition that was done by hand, you may find useful the method of *casting out nines*. In casting out nines, one finds the sum, modulo nine, of all the *digits* of the numbers to be summed and compares it with the sum, modulo nine, of the digits of the putative answer. [With a little practice, these sums can be computed much more rapidly than the calculation proper.]

Example 13.4: In the calculation

```
   189
 +1254
  +238
 -----
  1681
```

the sum, modulo nine, of the digits in `189+1254+238` is `7`, and the sum, modulo nine, of `1+6+8+1` is `7`. The calculation thus passes the casting-out-nines test.

Casting out nines gives a simple example of a hash function. For any addition expression of the form $a+b+c+\ldots$, where $a, b, c, \ldots$ are decimal numbers we define $h \in \{0,1,2,3,4,5,6,7,8\}$ by

$$h(a+b+c+\ldots) = \text{ sum modulo nine of all digits in } a, b, c \ ; \tag{13.1}$$

then it is nice property of decimal arithmetic that if

$$a+b+c+\ldots = m+n+o+\ldots \tag{13.2}$$

then the hashes $h(a+b+c+\ldots)$ and $h(m+n+o+\ldots)$ are equal.

Exercise 13.5:$^{B1}$ What evidence does a correct casting-out-nines match give in favour of the hypothesis that the addition has been done correctly?

*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 the two files are on separate machines, 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. [And even if we did transfer one of the files, we would still like a way to confirm whether it has been received without modifications!]

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 $M$-bit hash function, and the two hashes match, then Bob can deduce that the two files are almost surely the same.

Example 13.6: 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 $2^{-M}$. 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. (Such a cyclic redundancy check is a set of 32 parity check bits similar to the 3 parity check bits of the (7,4) Hamming code.)

To have a false-negative rate smaller than one in a billion, $M = 32$ bits is plenty, if the errors are produced by noise.

Exercise 13.7:$^{B2}$ 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 too?

### *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. 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. *Linear hash functions give no security against forgers.*

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 file he received. Such a hash function – easy to compute, but hard to invert – 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 to confirm that two files do not differ is `MD5`, which produces a 128 bit hash. The details of how it works are quite complicated, involving convoluted exclusive-or-ing and if-ing and and-ing of bits with each other.[1]

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 further tiny modification to it such that its hash matches the original hash of Alice's file. This would take some time – on average, 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.

---

Fiona has to hash $2^M$ files to cheat. $2^{32}$ file modifications is not very many, so a 32-bit hash function is not large enough for forgery prevention.

---

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. Everyone can confirm that her bet is consistent with the previously publicised hash. [This method of secret publication was used by Newton and Hooke when they wished to establish priority for scientific ideas without revealing them. Hooke's hash function was alphabetization as illustrated by the conversion of *UT TENSIO, SIC VIS* into the anagram

[1] `http://www.freesoft.org/CIE/RFC/1321/3.htm`

`CEIIINOSSSTTUV`.] 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. For example, if she'd like to cheat by placing two bets for the price of one, she could make a large number $N_1$ of versions of bet one (differing from each other in minor details only), and a large number $N_2$ of versions of bet two, and hash them all. If there's a collision between bets of different types, then she can submit the common hash and thus buy herself the option of placing either bet.

Example 13.8: 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 10.20 (p.188). 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

$$N_1 N_2 2^{-M}, \tag{13.3}$$

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.

---

Alice has to hash $2^{M/2}$ files to cheat. [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.

## 13.6 Further reading

I highly recommend reading the story of Doug McIlroy's `spell` program, as told in section 13.8 of *Programming Pearls* (Bentley 2000). This astonishing piece of software makes use of a 64-kilobyte data structure to store the spellings of all the words of 75,000-word dictionary.

## 13.7 Problems

*Information theory and the real world (questions relating to hash functions)*

Exercise 13.9:$^{A1}$ 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 13.10:$^{A2}$ 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 13.11:$^{B2}$ **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 often involves 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 $3 \times 10^9$ 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

$$\underline{\texttt{GCCCCC}}\texttt{CACCCCT}\underline{\texttt{GCCCCC}} \tag{13.4}$$

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)?

# *Solutions to Chapter 13's exercises*

Solution to exercise 13.1 (p.230): 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, so the expected number of comparisons is 2 (exercise 2.23, p.41).

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 $2S/2$ binary comparisons; and comparing the correct strings takes $N$ binary comparisons, giving a total expectation of $S+N$ 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 13.2 (p.233): The likelihood ratio for the two hypotheses, $\mathcal{H}_0$: $\mathbf{x}^{(s)} = \mathbf{x}$, and $\mathcal{H}_1$: $\mathbf{x}^{(s)} \neq \mathbf{x}$, contributed by the datum 'the first bits of $\mathbf{x}^{(s)}$ and $\mathbf{x}$ are equal' is

$$\frac{P(\text{Datum}|\mathcal{H}_0)}{P(\text{Datum}|\mathcal{H}_1)} = \frac{1}{1/2} = 2. \tag{13.5}$$

If the first $r$ bits all match, the likelihood ratio is $2^r$ to one. On finding that 30 bits match, the odds are a billion to one in favour of $\mathcal{H}_0$, assuming we start from even odds. [For a complete answer, we should compute the prior probability of $\mathcal{H}_0$ and $\mathcal{H}_1$ given the prior information, that the hash entry $s$ has been found in the table at $\mathbf{h}(\mathbf{x})$. This gives further evidence in favour of $\mathcal{H}_0$.]

Solution to exercise 13.3 (p.234): Let the hash function have an output alphabet of size $A = 2^M$. If $M$ were equal to $\log_2 S$ then we would have exactly enough bits for each entry to have its own unique hash. 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

$$S(S-1)/(2A). \tag{13.6}$$

If we would like this to be smaller than 1, then we need $A > S(S-1)/2$ so

$$M > 2\log_2 S. \tag{13.7}$$

We need *twice as many* bits as the number of bits, $\log_2 S$, that would be 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 $A > S/f$ [since the probability that one particular name is collided-with is $f \simeq S/A$] so

$$M > \log_2 S + \log[1/f], \tag{13.8}$$

which means for $f \simeq 0.01$ that we need an extra 7 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 $A$ needs to be of order $S$ only.

Solution to exercise 13.5 (p.235): The posterior probability ratio for the two hypotheses, $\mathcal{H}_+ =$ 'calculation correct' and $\mathcal{H}_- =$ 'calculation incorrect' is the product of the prior probability ratio $P(\mathcal{H}_+)/P(\mathcal{H}_-)$ and the likelihood ratio, $P(\text{match}|\mathcal{H}_+)/P(\text{match}|\mathcal{H}_-)$. This second factor is the answer to the question. The numerator $P(\text{match}|\mathcal{H}_+)$ is equal to 1. The denominator's value depends on our model of errors. If we know that the human calculator is prone to errors involving multiplication of the answer by 10, or to transposition of adjacent digits, neither of which affects the hash value, then $P(\text{match}|\mathcal{H}_-)$ could be equal to 1 also, so that the correct match gives no evidence in favour of $\mathcal{H}_+$. But if we assume that errors are 'random from the point of view of the hash function' then the probability of a false positive is $P(\text{match}|\mathcal{H}_-) = 1/9$, and the correct match gives evidence 9:1 in favour of $\mathcal{H}_+$.

Solution to exercise 13.7 (p.235): 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 one-in-a-billion. To do error *correction* requires far more check bits, the number depending on the expected types of corruption, and on the file size. For example, if just eight random bits in a megabyte file are corrupted, it would take about $\log_2 \binom{2^{23}}{8} \simeq 23 \times 8 \simeq 180$ bits to specify which are the corrupted bits, and the number of parity bits used by a successful error-correcting code would have to be at least this number, by Shannon's noisy-channel coding theorem.

Solution to exercise 13.10 (p.237): 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 $L \geq 16/\log_{10} 37 \simeq 10$. Because of the redundancy and repetition of humanity's writings, it is possible that $L \simeq 10$ is an overestimate.

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 string.

As for a new melody, if we focus on the sequence of notes, ignoring duration and stress, and allow leaps of up to an octave at each note, then the number of choices per note is 23. The pitch of the first note is arbitrary. The number of melodies of length $r$ notes in this rather ugly ensemble of Schönbergian tunes is $23^{r-1}$; for example, there are 250,000 of length $r = 5$. Restricting the permitted intervals will reduce this figure [If we restrict the permitted intervals to repetitions and tones or semitones, the reduction is particularly severe; is this why the melody of 'Ode to joy' sounds so boring?]; including duration and stress will increase it again. The number of recorded compositions is probably not quite in the millions. If you learn 100 new melodies per week for every week of your life then you will have learned 250,000 melodies at age 50. Based on empirical experience of playing the game '`guess that tune`', it seems to me that whereas many four-note sequences are shared in common between melodies, the number of collisions between five-note sequences is rather smaller – most famous five-note sequences are unique.

Solution to exercise 13.11 (p.238): (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`'; so, to be precise, 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

$$(1 - (1/4)^L)^N \simeq \exp(-N(1/4)^L), \tag{13.9}$$

which is close to one only if

$$N4^{-L} \ll 1, \tag{13.10}$$

that is,

$$L > \log N / \log 4. \tag{13.11}$$

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?

- 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.

- 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 is proportional to sequence length and 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.

# 14

# *Binary Codes* *

We've established Shannon's noisy channel coding theorem for a general channel with any input and output alphabets. A great deal of attention in coding theory focuses on channels with binary inputs, the first implicit choice being the binary symmetric channel.

The optimal decoder for a code, given a binary symmetric channel, finds the codeword that is closest to the received vector, closest in *Hamming distance*. The Hamming distance between two binary vectors is the number of coordinates in which the two vectors differ. Decoding errors will occur if the noise takes us from the transmitted codeword $\mathbf{t}$ to a received vector $\mathbf{r}$ that is closer to some other codeword. The *distances* between codewords are thus relevant to the probability of a decoding error.

Example:
The Hamming distance between `00001111` and `11001101` is 3.

## 14.1 Distance properties of a code

**The *distance* of a code** is the smallest separation between two of its codewords.

Example 14.1**:** The (7,4) Hamming code (p.13) has distance $d = 3$. All pairs of its codewords differ in at least 3 bits. The minimum number of errors it can correct is $t = 1$; in general a code with distance $d$ is $\lfloor (d-1)/2 \rfloor$-error-correcting.

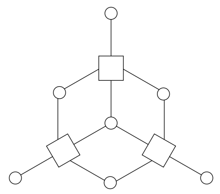
A more precise term for distance is the *minimum distance* of the code. The distance of a code is often denoted by $d$ or $d_{\min}$.

**The *weight enumerator function* of a code** , $A(w)$, summarises all the distances between its codewords. $A(w)$ is defined to be the number of codewords in the code that have weight $w$.

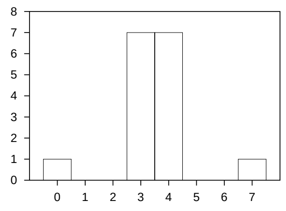
The weight enumerator function is also known as the *distance distribution* of the code.

Example 14.2**:** The weight enumerator functions of the (7,4) Hamming code and the dodecahedron code are shown in figure 14.1.

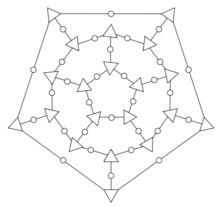
(a)



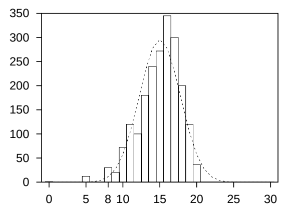
| $w$ | $A(w)$ |
|---|---|
| 0 | 1 |
| 3 | 7 |
| 4 | 7 |
| 7 | 1 |
| Total | 16 |



(b)



| $w$ | $A(w)$ |
|---|---|
| 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 |



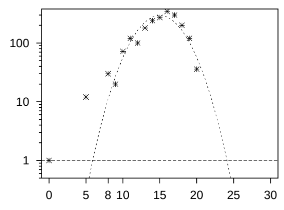


Figure 14.1. (a) The graph of the (7,4) Hamming code, and the weight enumerator function. (b) The graph defining the (30,11) dodecahedron code (the circles are the 30 transmitted bits and the triangles are the 20 parity checks, one of which is redundant) and the weight enumerator function (solid lines). The dotted lines show the average weight enumerator function of all random linear codes with the same size of generator matrix (dotted lines), which will be computed shortly. The lower figure shows the same functions on a log scale.

## 14.2 Obsession with distance

Since the minimum number of errors that a code can *guarantee* to correct, $t$, is related to its distance $d$ by $t = \lfloor (d-1)/2 \rfloor$, many coding theorists focus on the distance of a code, searching for codes of a given size that have the biggest possible distance. Much of practical coding theory has focused on decoders that give the optimal decoding for all error patterns of weight up to the half-distance $t$ of their codes.

$d = 2t + 1$ if $d$ is odd, and
$d = 2t + 2$ if $d$ is even.

**A bounded-distance decoder** is a decoder that returns the closest codeword to a received binary vector $\mathbf{r}$ if the distance from $\mathbf{r}$ to that codeword is less than or equal to $t$; otherwise it returns a failure message.

The rationale for not trying to decode when more than $t$ errors have occurred might be 'we can't *guarantee* that we can correct more than $t$ errors, so we won't bother trying – who would be interested in a decoder that corrects some error patterns of weight greater than $t$, but not others?' This defeatist attitude is an example of *worst-case-ism*, a widespread mental ailment which this book is intended to cure.

The fact is that bounded-distance decoders cannot get to the Shannon limit of the binary symmetric channel; only a decoder that often corrects more than $t$ errors can do this. The state of the art in error-correcting codes have decoders that work way beyond the minimum distance of the code.

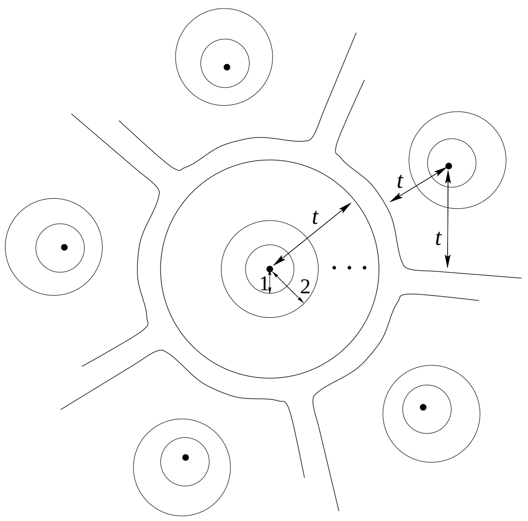


Figure 14.2. Schematic picture of part of Hamming space perfectly filled by $t$-spheres centred on the codewords of a perfect code.

*Definitions of good and bad distance properties*

Given a family of codes of increasing blocklength $N$, and with rates approaching a limit $R > 0$, we may be able to put that family in one of the following categories, which have some similarities to the categories of 'good' and 'bad' codes defined earlier (p.216):

**A sequence of codes has 'good' distance** if $d/N$ tends to a constant greater than zero.

**A sequence of codes has 'bad' distance** if $d/N$ tends to zero.

**A sequence of codes has 'very bad' distance** if $d$ tends to a constant.

While having large distance is no bad thing, we'll see, later on, why an emphasis on distance can be unhealthy.

## 14.3 Perfect codes

A $t$-sphere (or a sphere of radius $t$) in Hamming space centred on a point $\mathbf{x}$, is the set of points whose Hamming distance from $\mathbf{x}$ is less than or equal to $t$.

The (7,4) Hamming code has the beautiful property that if we place 1-spheres about each of its 16 codewords, those spheres perfectly fill Hamming space without overlapping. As we saw in chapter 1, every binary vector of length 7 is within a distance of $t = 1$ of exactly one codeword of the Hamming code.

**A code is a perfect $t$-error-correcting code** if the set of $t$-spheres centred on the codewords of the code fill the Hamming space without overlapping. (See figure 14.2.)

Let's recap our cast of characters. The number of codewords is $S = 2^K$. The number of points in the entire Hamming space is $2^N$. The number of points in a Hamming sphere of radius $t$ is

$$\sum_{w=0}^{t} \binom{N}{w}. \tag{14.1}$$

For a code to be perfect with these parameters, we require $S$ times the number of points in the $t$-sphere to equal $2^N$:

$$\text{for a perfect code,}\quad 2^K \sum_{w=0}^{t} \binom{N}{w} \;=\; 2^N \tag{14.2}$$

$$\text{or, equivalently,}\quad \sum_{w=0}^{t} \binom{N}{w} \;=\; 2^{N-K}. \tag{14.3}$$

For a perfect code, the number of noise vectors in one sphere must equal the number of possible syndromes. The (7,4) Hamming code satisfies this numerological condition because

$$1 + \binom{7}{1} = 2^3.$$

*How happy we would be to use perfect codes*

If there were large numbers of perfect codes to choose from, with a wide range of blocklengths and rates, then these would be the perfect solution to Shannon's problem. We could communicate over a binary symmetric channel with noise level $f$, for example, by picking a perfect $t$-error-correcting code with blocklength $N$ and $t = f^*N$, where $f^* = f + \delta$ and $N$ and $\delta$ are chosen such that the probability that the noise flips more than $t$ bits is satisfactorily small.

However, *there are almost no perfect codes.* The only nontrivial perfect binary codes are ☆☆☆!

1. the Hamming codes, which are perfect codes with $t = 1$ and blocklength $N = 2^M - 1$, defined below; the rate of a Hamming code approaches 1 as its blocklength $N$ increases;

2. the repetition codes of odd blocklength $N$, which are perfect codes with $t = (N-1)/2$; the rate of repetition codes goes to zero as $1/N$; and

3. one remarkable 3-error-correcting code with $2^{12}$ codewords of blocklength $N = 23$ known as the binary Golay code. [A second 2-error-correcting Golay code of length-$N = 11$ over a ternary alphabet was discovered by a Finnish football-pool enthusiast called Juhani Virtakallio in 1947.]

There are no other binary perfect codes. Why this shortage of perfect codes? Is it because numerological coincidences like the one satisfied by the Golay code's parameters,

$$1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2^{11}, \tag{14.4}$$

are rare? Perhaps there are plenty of 'almost-perfect' codes for which the $t$-spheres fill *almost* the whole space?

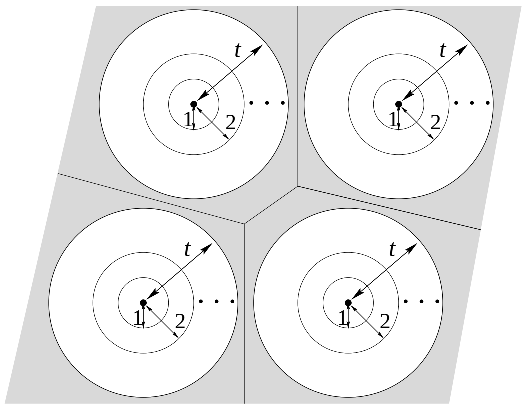


Figure 14.3. Schematic picture of Hamming space not perfectly filled by $t$-spheres centred on the codewords of a code. The grey regions show points that are at a Hamming distance of more than $t$ from any codeword. This is a misleading picture, as, for any code with large $t$ in high dimensions, the grey space between the spheres takes up almost all of Hamming space.

No. In fact, the picture of Hamming spheres centred on the codewords *almost* filling Hamming space (figure 14.3) is a misleading one: for most codes, whether they are good codes or bad codes, almost all the Hamming space is taken up by the space *between* $t$-spheres (which is shown in grey in figure 14.3).

Having established this gloomy picture, we spend a moment filling in the properties of the perfect codes mentioned above.

## *The Hamming codes*

The (7,4) Hamming code can be defined as the linear code whose $3 \times 7$ parity-check matrix contains, as its columns, all the 7 ($= 2^3 - 1$) non-zero vectors of length 3. Since these 7 vectors are all different, any single bit-flip produces a distinct syndrome, so all single-bit errors can be detected and corrected.

We can generalize this code, with $M = 3$ parity constraints, as follows. The Hamming codes are all single-error-correcting codes defined by picking a number of parity check constraints, $M$. The blocklength $N$ is $N = 2^M - 1$. The parity-check matrix contains, as its columns, all the $N$ non-zero vectors of length $M$ bits.

The first few Hamming codes have the following rates:

| Checks, $M$ | $(N,K)$ | $R = K/N$ | |
|---|---|---|---|
| 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 | |

Exercise 14.3:$^{A3}$ What is the probability of block error of the $(N, K)$ Hamming code to leading order, when the code is used for a binary symmetric channel with noise density $f$?

Exercise 14.4:$^{A3}$ What are the minimum distances of the $(15, 11)$ Hamming code and the $(31, 26)$ Hamming code?

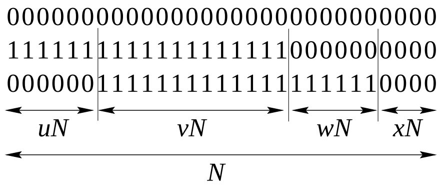


Figure 14.4. Three codewords.

Solution to exercise 14.3 (p.247): The probability of block error to leading order is $p_B = \frac{3}{N}\binom{N}{2}f^2$.

Solution to exercise 14.4 (p.247): All the Hamming codes have minimum distance $d = 3$.

## 14.4 Perfectness is unattainable – first proof *

We will show in several ways that useful perfect codes do not exist (here, 'useful' means 'having large blocklength $N$, and rate close neither to 0 nor 1').

Shannon proved that, given a binary symmetric channel with any noise level $f$, there exist codes with large blocklength $N$ and rate as close as you like to $C(f) = 1 - H_2(f)$ that enable communication with arbitrarily small error probability. For large $N$, the number of errors per block will typically by about $fN$, so these codes of Shannon are 'almost-certainly-$fN$-error-correcting' codes.

Let's pick the special case of a noisy channel with $f \in (1/3, 1/2)$. Can we find a large *perfect* code that is $fN$-error-correcting? Well, let's suppose that such a code has been found, and examine just three of its codewords. (Remember that the code ought to have rate $R \simeq 1 - H_2(f)$, so it should have an enormous number ($2^{NR}$) of codewords.) Without loss of generality, we choose one of the codewords to be the all-zero codeword and define the other two to have overlaps with it as shown in figure 14.4. The second codeword differs from the first in a fraction $u+v$ of its coordinates. The third codeword differs from the first in a fraction $v+w$, and from the second in a fraction $u+w$. A fraction $x$ of the coordinates have value zero in all three codewords. Now, if the code is $fN$-error-correcting, its minimum distance must be greater than $2fN$, so

$$u + v > 2f, \quad v + w > 2f, \quad \text{and} \quad u + w > 2f. \tag{14.5}$$

So if $f > 1/3$, we can deduce $u+v+w > 1$, so that $x < 0$, which is impossible. Such a code cannot exist.

We conclude that, whereas Shannon proved there are plenty of codes for communicating over a channel with $f > 1/3$, *there are no perfect codes that can do this.*

We now study a more general argument that shows that there are no large perfect codes for general rates (other than 0 and 1). We do this by finding the typical distance of a random code.

## 14.5 Weight enumerator function of random linear codes *

Imagine making a code by picking the binary entries in the $M \times N$ parity check matrix $\mathbf{H}$ at random. What weight enumerator function should we expect?

$$\left.\overbrace{\begin{bmatrix}1\,0\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,0\,0\,1\,0\,1\,1\,0\\0\,0\,1\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,0\,0\\1\,0\,1\,1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0\,1\,0\,0\\0\,0\,0\,0\,1\,0\,1\,1\,1\,1\,0\,0\,1\,0\,1\,1\,0\,1\,0\,0\,1\,0\,0\,0\\0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1\,0\,0\,0\,0\,0\,1\,0\,0\\1\,1\,0\,0\,1\,0\,0\,0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,1\,0\,1\,1\,1\,0\\1\,0\,1\,1\,1\,1\,1\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\,1\,1\,0\\1\,1\,0\,0\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,0\,1\,0\,1\,0\\1\,0\,0\,0\,1\,1\,1\,0\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,1\,1\,1\,1\,0\,1\\0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,0\,1\,1\,0\,1\,0\,1\,0\\0\,1\,0\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1\,0\\1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,0\,0\,0\,1\,1\end{bmatrix}}^{N}\right\}M$$

The weight enumerator of one particular code is

$$A(w)_{\mathbf{H}} = \sum_{\mathbf{x}:|\mathbf{x}|=w} \delta\left[\mathbf{H}\mathbf{x} = 0\right], \tag{14.6}$$

where the indicator function $\delta[\mathbf{H}\mathbf{x} = 0]$ equals one if $\mathbf{H}\mathbf{x} = 0$ and zero otherwise.

We can find the expected value of $A(w)$,

$$\begin{aligned}\langle A(w)\rangle &= \sum_{\mathbf{H}} P(\mathbf{H}) A(w)_{\mathbf{H}} &(14.7)\\ &= \sum_{\mathbf{x}:|\mathbf{x}|=w} \sum_{\mathbf{H}} P(\mathbf{H})\, \delta\left[\mathbf{H}\mathbf{x} = 0\right], &(14.8)\end{aligned}$$

by evaluating the probability that a particular word of weight $w > 0$ is a codeword of the code (averaging over all binary linear codes in our ensemble). By symmetry, this probability depends only on the weight $w$ of the word, not on the details of the word. The probability that the entire syndrome $\mathbf{H}\mathbf{x}$ is zero can be found by multiplying together the probabilities of each bit's being zero, for all $M$ bits in the syndrome. Each bit $z_m$ of the syndrome is a sum (mod 2) of $w$ random bits, so the probability that $z_m = 0$ is $1/2$. The probability that $\mathbf{H}\mathbf{x} = 0$ is thus

$$\sum_{\mathbf{H}} P(\mathbf{H}) \delta\left[\mathbf{H}\mathbf{x} = 0\right] = (1/2)^M = 2^{-M}, \tag{14.9}$$

independent of $w$.

The expected number of words of weight $w$ is given by summing, over all words of weight $w$, the probability that each word is a codeword. The number of words of weight $w$ is $\binom{N}{w}$, so

$$\langle A(w)\rangle = \binom{N}{w} 2^{-M} \text{ for any } w > 0. \tag{14.10}$$

For large $N$, we can use $\log \binom{N}{w} \simeq N H_2(w/N)$ and $R \simeq 1 - M/N$ to write

$$\begin{aligned}\log_2 \langle A(w)\rangle &\simeq N H_2(w/N) - M &(14.11)\\ &\simeq N[H_2(w/N) - (1-R)] \text{ for any } w > 0. &(14.12)\end{aligned}$$

As a concrete example, figure 14.5 shows the expected weight enumerator function of a rate-1/3 random linear code with $N = 540$ and $M = 360$.

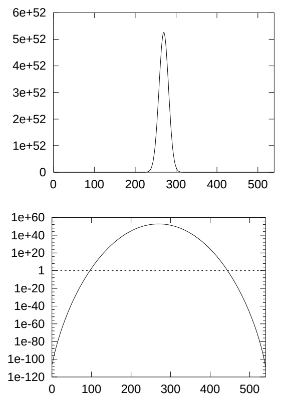


Figure 14.5. The expected weight enumerator function $\langle A(w)\rangle$ of a random linear code with $N = 540$ and $M = 360$. Lower figure shows $\langle A(w)\rangle$ on a logarithmic scale.

### *Gilbert distance*

For weights $w$ such that $H_2(w/N) < (1-R)$, the expectation of $A(w)$ is smaller than 1; for weights such that $H_2(w/N) > (1-R)$, the expectation is greater than 1. We thus expect, for large $N$, that the minimum distance of a random code will be close to the distance $d_{\mathrm{GV}}$ defined by

$$H_2(d_{\mathrm{GV}}/N) = (1-R). \tag{14.13}$$

Definition: This distance, $d_{\rm GV} \equiv NH_2^{-1}(1-R)$, is known as the Gilbert-Varshamov distance for rate $R$ and blocklength $N$.

The Gilbert-Varshamov conjecture, widely believed, asserts that (for large $N$) it is not possible to create binary codes with minimum distance significantly greater than $d_{\rm GV}$.

### *Why sphere-packing is a bad perspective, and an obsession with distance is inappropriate*

Definition: The Gilbert-Varshamov rate $R_{\rm GV}$ is the minimum rate at which you can reliably communicate with a bounded-distance decoder (as defined on p. 244), assuming that the Gilbert-Varshamov conjecture is true.

If one uses a bounded-distance decoder, the maximum tolerable noise level will flip a fraction $f_{\rm bd} = \frac{1}{2}d_{\min}/N$ of the bits, i.e., assuming $d_{\min}$ is equal to the Gilbert distance $d_{\rm GV}$ (14.13), we have:

$$H_2(f_{\rm bd}/2) = (1 - R_{\rm GV}). \tag{14.14}$$

$$R_{\rm GV} = 1 - H_2(f_{\rm bd}/2). \tag{14.15}$$

Now, here's the crunch: what did Shannon say is achievable? He said the maximum possible rate of communication is the capacity,

$$C = 1 - H_2(f). \tag{14.16}$$

So for a given rate $R$, the maximum tolerable noise level, according to Shannon, is given by

$$H_2(f) = (1 - R). \tag{14.17}$$

The conclusion: imagine a good code of rate $R$ has been chosen; equations (14.14) and (14.17) define the maximum noise levels tolerable by a bounded-distance decoder, $f_{\rm bd}$, and by Shannon's decoder, $f$.

$$f_{\rm bd} = f/2 \tag{14.18}$$

Those who use bounded-distance decoders can only ever cope with *half* the noise-level that Shannon proved is tolerable!

How does this relate to perfect codes? A code is perfect if there are $t$-spheres around its codewords which fill Hamming space without overlapping. But when a typical random linear code is used to communicate over a binary symmetric channel near to the Shannon limit, the typical number of bits flipped is $fN$, and the minimum distance between codewords is also $fN$, or a little bigger, if we are a little below the Shannon limit. So the $fN$-spheres around the codewords overlap with each other sufficiently that each sphere almost contains the centre of its nearest neighbour. The reason why this overlap is not disastrous is because, in high dimensions, the volume associated with the overlap, shown shaded in figure 14.7, is a tiny fraction of either sphere, so the probability of landing in it is extremely small.

The moral of the story is that worst-case-ism can be bad for you, halving your ability to tolerate noise. You have to be able to decode way beyond the minimum distance of a code to get to the Shannon limit!

Nevertheless, the minimum distance of a code is of interest in practice, because, under some conditions, the minimum distance dominates the errors made by a code.

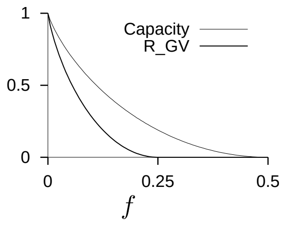


Figure 14.6. Contrast between Shannon's channel capacity $C$ and the Gilbert rate $R_{\rm GV}$ – the minimum communication rate achievable using a bounded-distance decoder, as a function of noise level $f$. For any given rate, $R$, the minimum tolerable noise level for Shannon is twice as big as the minimum tolerable noise level for a 'worst-case-ist' who uses a bounded-distance decoder.

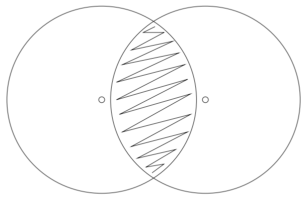


Figure 14.7. Two overlapping spheres whose radius is almost as big as the distance between their centres.

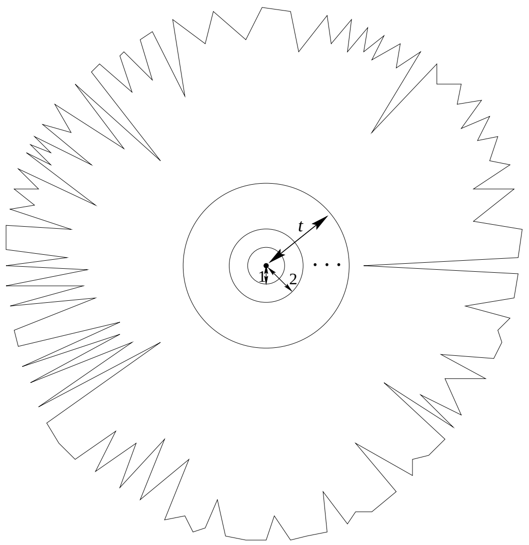


Figure 14.8. Berlekamp's schematic picture of Hamming space in the vicinity of a codeword. The solid line shows the boundary of the set of all points for which this codeword is the closest. The $t$-sphere around the codeword takes up a small fraction of this space.

## 14.6 Berlekamp's bats

A blind bat lives in a cave. It flies about the centre of the cave, which corresponds to one codeword, with its typical distance from the centre controlled by a friskiness parameter $f$. (The displacement of the bat from the centre corresponds to the noise vector.) The boundaries of the cave are made up of stalactites that point in towards the centre of the cave. Each stalactite is analagous to the boundary between the home codeword and another codeword. The stalactite is like one half of the shaded region in figure 14.7, but reshaped to convey the idea that it is a region of very small volume.

Decoding errors correspond to the bat's intended trajectory passing inside a stalactite. Collisions with stalactites at various distances from the centre are possible.

If the friskiness is very small, collisions will be rare, and when they do occur, they will usually involve the stalactites whose tips are closest to the centre point. Similarly, under low-noise conditions, decoding errors will be rare, and they will typically involve low-weight codewords. Under low-noise conditions, the minimum distance of a code is relevant to the (very small) probability of error.

If the friskiness is higher, the bat may often make excursions beyond the safe distance $t$ where the longest stalactites start, but it will collide most frequently with more distant stalactites, owing to their greater number. There's only a tiny number of stalactites at the minimum distance, so they are relatively unlikely to cause the errors. Similarly, errors in a real error-correcting code depend on the properties of the weight enumerator function.

At very high friskiness, the bat is always a long way from the centre of the cave, and almost all its collisions involve contact with distant stalactites.

## 14.7 Concatenation of Hamming codes*

It is instructive to play some more with the concatenation of Hamming codes, a concept we first visited in figure 12.6, because we will get insights into the notion of good codes and the relevance or otherwise of the minimum distance of a code.

We can create a concatenated code for a binary symmetric channel with noise density $f$ by encoding with several Hamming codes in succession.

The table recaps the key properties of the Hamming codes, indexed by number of constraints, $M$. All the Hamming codes have minimum distance $d = 3$ and can correct one error in $N$.

| | |
|---|---|
| $N = 2^M - 1$ | block length |
| $K = N - M$ | number of source bits |
| $p_B = \frac{3}{N}\binom{N}{2} f^2$ | probability of block error to leading order |

If we make a product code by concatenating a sequence of $C$ Hamming codes with increasing $M$, we can choose those parameters $\{M_c\}_{c=1}^{C}$ in such a way that the rate of the product code

$$R_C = \prod_{c=1}^{C} \frac{N_c - M_c}{N_c} \qquad (14.19)$$

tends to a non-zero limit as $C$ increases. For example, if we set $M_1 = 2$, $M_2 = 3$, $M_3 = 4$, etc., then the asymptotic rate is 0.093.

The block length $N$ is a rapidly growing function of $C$, so these codes are somewhat impractical.

A further weakness of these codes is their minimum distance is not very good. Every one of the constituent Hamming codes has minimum distance 3, so the minimum distance of the $C$th product is $3^C$. The blocklength $N$ grows faster with $C$ than $3^C$, so the ratio $d/N$ tends to zero as $C$ increases. In contrast, for typical random codes, the ratio $d/N$ tends to a constant such that $H_2(d/N) = 1 - R$. Concatenated Hamming codes have 'bad' distance.

Nevertheless, it turns out that this simple sequence of codes yields good codes for some channels – but not very good codes (see section 12.4 to recall the definitions of the terms 'good' and 'very good'). Rather than prove this result, we will simply explore it numerically.

Figure 14.11 shows the bit error probability $p_b$ of the concatenated codes assuming that the constituent codes are decoded in sequence, as described in section 12.4. [This one-code-at-a-time decoding is suboptimal, as we saw there.] The channel assumed in the figure is the binary symmetric channel with $f = 0.0588$. This is the highest noise level that can be tolerated using this code.

The take-home message from this story is *distance isn't everything*. The minimum distance of a code, although widely worshipped by coding theorists, is not of fundamental importance if our aim is to fulfill Shannon's mission of achieving reliable communication over noisy channels.

Exercise 14.5:$^{B3}$ Prove that there exist families of codes with 'bad' distance that are 'very good' codes.

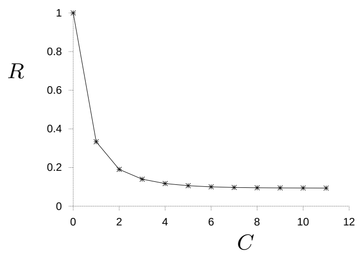


Figure 14.9. The rate $R$ of the concatenated Hamming code as a function of the number of concatenations, $C$.

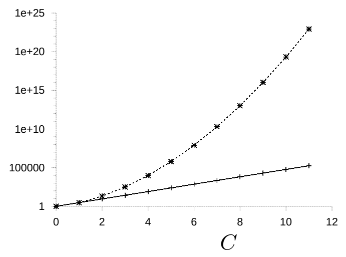


Figure 14.10. The blocklength $N_C$ and minimum distance $d_C$ of the concatenated Hamming code as a function of the number of concatenations $C$.

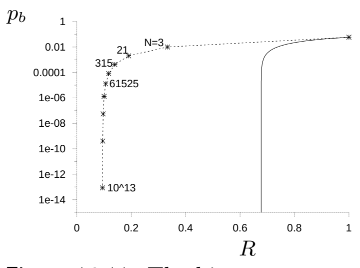


Figure 14.11. The bit error probabilities versus the rates $R$ of the concatenated Hamming codes, for the binary symmetric channel with $f = 0.0588$. The solid line shows the Shannon limit for this channel.
The bit error probability drops to zero while the rate tends to 0.093, so the concatenated Hamming codes are a 'good' code family.

## 14.8 Dual codes*

A concept that has some importance in coding theory, though we will have no immediate use for it in this book, is the idea of the *dual* of a linear error-correcting code.

An $(N, K)$ linear error-correcting code can be thought of as a set of $2^K$ codewords generated by adding together all combinations of $K$ independent basis codewords. The generator matrix of the code consists of those $K$ basis codewords, conventionally written as row vectors. For example, the (7,4) Hamming code's generator matrix (from equation (1.11)) is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{14.20}$$

and its sixteen codewords were displayed in table 1.10. The codewords of this code are linear combinations of the four vectors [1 0 0 0 1 0 1], [0 1 0 0 1 1 0], [0 0 1 0 1 1 1], and [0 0 0 1 0 1 1],

An $(N, K)$ code may also be described in terms of an $M \times N$ parity check matrix (where $M = N - K$) as the set of vectors $\{\mathbf{t}\}$ that satisfy

$$\mathbf{Ht} = 0. \tag{14.21}$$

One way of thinking of this equation is that each row of $\mathbf{H}$ specifies a vector to which $\mathbf{t}$ must be orthogonal if it is a codeword. Thus the generator matrix specifies $K$ vectors from which all codewords can be built, and the parity check matrix specifies a set of $M$ vectors to which all codewords are orthogonal.

The dual of a code is obtained by exchanging the generator matrix and the parity check matrix.

Definition. The set of *all* vectors of length $N$ that are orthogonal to all codewords in a code, $\mathcal{C}$, is called the dual of the code, $\mathcal{C}^\perp$.

If $\mathbf{t}$ is orthogonal to $\mathbf{h}_1$ and $\mathbf{h}_2$, then it is also orthogonal to $\mathbf{h}_3 \equiv \mathbf{h}_1 + \mathbf{h}_2$; so all codewords are orthogonal to any linear combination of the $M$ rows of $\mathbf{H}$. So the set of all linear combinations of the rows of the parity check matrix is the dual code.

For our Hamming (7,4) code, the parity check matrix is (from equation (1.13)):

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{14.22}$$

The dual of the (7,4) Hamming code $\mathcal{H}_{(7,4)}$ is the code shown in figure 14.12.

| | | | |
|---|---|---|---|
| 0000000 | 0101101 | 1001110 | 1100011 |
| 0010111 | 0111010 | 1011001 | 1110100 |

Figure 14.12. The eight codewords of the dual of the (7,4) Hamming code. [Compare with figure 1.10.]

A possibly unexpected property of this pair of codes is that the dual, $\mathcal{H}^\perp_{(7,4)}$, is contained within the code $\mathcal{H}_{(7,4)}$ itself: every word in the dual code is a codeword of the original (7,4) Hamming code. This relationship can be written using set notation:

$$\mathcal{H}^{\perp}_{(7,4)} \subset \mathcal{H}_{(7,4)}. \tag{14.23}$$

The possibility that the set of dual vectors can overlap the set of codeword vectors is counterintuitive if we think of the vectors as real vectors – how can a vector be orthogonal to itself? When we work in modulo-two arithmetic, many non-zero vectors are indeed orthogonal to themselves!

Exercise 14.6:$^{B1}$ Give a simple rule that distinguishes whether a binary vector is orthogonal to itself, as is each of the three vectors [1 1 1 0 1 0 0], [0 1 1 1 0 1 0], and [1 0 1 1 0 0 1].

## *Some more duals*

In general, if a code has a systematic generator matrix,

$$\mathbf{G} = [\mathbf{I}_K | \mathbf{P}^{\mathsf{T}}], \tag{14.24}$$

where $\mathbf{P}$ is a $K \times M$ matrix, then its parity check matrix is

$$\mathbf{H} = [\mathbf{P} | \mathbf{I}_M]. \tag{14.25}$$

Example 14.7: The repetition code $\mathrm{R}_3$ has generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}; \tag{14.26}$$

its parity check matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{14.27}$$

The two codewords are [1 1 1] and [0 0 0].

The dual code has generator matrix

$$\mathbf{G}^{\perp} = \mathbf{H} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{14.28}$$

or equivalently, modifying $\mathbf{G}^{\perp}$ into systematic form by row additions,

$$\mathbf{G}^{\perp} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{14.29}$$

We call this dual code the *simple parity code* $\mathrm{P}_3$; it is the code with one parity check bit, which is equal to the sum of the two source bits. The dual code's four codewords are [1 1 0], [1 0 1], [0 0 0], and [0 1 1].

In this case, the only vector common to the code and the dual is the all-zero codeword.

### *Goodness of duals*

If a sequence of codes is 'good', are their duals good too? Examples can be constructed of all cases: good codes with good duals (random linear codes); bad codes with bad duals; and good codes with bad duals. Nevertheless it is interesting the note the importance of the last category: many state-of-the-art codes have the property that their duals are bad. The classic example is the low-density parity-check code, whose dual is a low-density generator-matrix code.

Exercise 14.8:$^{B3}$ Show that low-density generator-matrix codes are bad. A family of low-density generator-matrix codes is defined by two parameters $j, k$, which are the column weight and row weight of all rows and columns respectively of $\mathbf{G}$. These weights are fixed, independent of $N$; for example, $(j, k) = (3, 6)$. [Hint: show that the code has low-weight codewords.]

Exercise 14.9:$^{D5}$ Show that low-density parity-check codes are good, and have good distance. (For solutions, see Gallager (1963) and MacKay (1999).)

### *Self-dual codes*

The (7,4) Hamming code had the property that the dual was contained in the code itself. It is intriguing, though not necessarily useful, to look at codes that are *self-dual*. A code $\mathcal{C}$ is self-dual if the dual of the code is identical to the code.

$$\mathcal{C}^{\perp} = \mathcal{C}. \tag{14.30}$$

Some properties of self-dual codes are fairly obvious:

1. If a code is self-dual, then its generator matrix is also a parity-check matrix for the code.
2. Self-dual codes have rate 1/2, i.e., $M = K = N/2$.
3. All codewords have even weight.

Exercise 14.10:$^{B2}$ What property must the matrix $\mathbf{P}$ satisfy, if the code with generator matrix $\mathbf{G} = [\mathbf{I}_K|\mathbf{P}^{\mathsf{T}}]$ is self-dual?

Solution to exercise 14.10 (p.255): The self-dual code has two equivalent parity check matrices, $\mathbf{H}_1 = \mathbf{G} = [\mathbf{I}_K|\mathbf{P}^{\mathsf{T}}]$ and $\mathbf{H}_2 = [\mathbf{P}|\mathbf{I}_K]$; these must be equivalent to each other through row additions, that is, there is a matrix $\mathbf{U}$ such that $\mathbf{U}\mathbf{H}_2 = \mathbf{H}_1$, so

$$[\mathbf{U}\mathbf{P}|\mathbf{U}\mathbf{I}_K] = [\mathbf{I}_K|\mathbf{P}^{\mathsf{T}}]\,. \tag{14.31}$$

From the right hand sides of this equation, we have $\mathbf{U} = \mathbf{P}^{\mathsf{T}}$, so the left hand sides become:

$$\mathbf{P}^{\mathsf{T}}\mathbf{P} = \mathbf{I}_K. \tag{14.32}$$

Thus if a a code with generator matrix $\mathbf{G} = [\mathbf{I}_K|\mathbf{P}^{\mathsf{T}}]$ is self-dual then $\mathbf{P}$ is an *orthogonal* matrix, modulo 2, and *vice versa*.

*Examples of self-dual codes*

1. The repetition code $\mathrm{R}_2$ is a simple example of a self-dual code.

$$\mathbf{G} = \mathbf{H} = \left[ \begin{array}{cc} 1 & 1 \end{array} \right] \tag{14.33}$$

2. The smallest non-trivial self-dual code is the following (8,4) code.

$$\mathbf{G} = \left[ \, \mathbf{I}_4 \, \middle| \, \mathbf{P}^{\mathsf{T}} \, \right] = \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array} \right] \tag{14.34}$$

Exercise 14.11:$^{B2}$ Find the relationship of the above (8,4) code to the (7,4) Hamming code.

*Duals and graphs*

Let a code is represented by a graph in which there are nodes of two types, parity check constraints, and equality constraints, joined by edges which represent the bits of the code (not all of which need be transmitted).

The dual code's graph is obtained by replacing all parity check nodes by equality nodes and vice versa. This type of graph is called a normal graph by Forney (2001).

## 14.9 Generalizing perfectness to other channels

Having given up on the search for perfect codes for the binary symmetric channel, we could console ourselves by changing channel. We could call a code 'a perfect $u$-error-correcting code for the binary erasure channel' if it can restore any $u$ erased bits, and never more. Rather than using the word perfect, however, the conventional term for such a code is a 'maximum distance separable code'.

As we already noted in exercise 12.9 (p.226), the (7,4) Hamming code is *not* a maximum distance separable code. It can recover *some* sets of 3 erased bits, but not all. If any 3 bits corresponding to a codeword of weight 3 are erased, then one bit of information is unrecoverable. This is why the (7,4) code is a poor choice for a RAID system.

A tiny example of a maximum distance separable code is the simple parity check code $P_3$ whose parity check matrix is $\mathbf{H} = [1\,1\,1]$. This code has 4 codewords, all of which have even parity. All codewords are separated by a distance of 2. Any single erased bit can be restored by setting it to the parity of the other two bits. The repetition codes are also maximum distance separable codes.

Exercise 14.12:$^{B5}$ Can you make an $(N, K)$ code, with $M = N - K$ parity symbols, for a $q$-ary erasure channel, such that the decoder can recover the codeword when *any* $M$ symbols are erased in a block of $N$? [Example: for the channel with $q = 4$ symbols there is an $(N, K) = (5, 2)$ code which can correct any $M = 3$ erasures.]

For the $q$-ary erasure channel with $q > 2$, there are large numbers of MDS codes, of which the Reed-Solomon codes are the most famous and most widely used. As long as the field size $q$ is bigger than the blocklength $N$, MDS block codes of any rate can be found.

## 14.10 Summary

Shannon's codes for the binary symmetric channel can almost always correct $fN$ errors, but they are not $fN$-error-correcting codes.
Reasons why the distance of a code has little relevance:

1. The Shannon limit shows you that you must be able to cope with a noise level twice as big as the maximum noise level for a bounded-distance decoder.

2. When the binary symmetric channel has $f > 1/4$, no code with a bounded-distance decoder can communicate at all; but Shannon says good codes exist for such channels.

3. Concatenation shows that we can get good performance even if the distance is bad.

The whole weight enumerator function is relevant to the question of whether a code is a good code.

Further discussion of the relationship between good codes and distance properties can be found in exercise 27.6 (p.364).

### Exercises

Exercise 14.13:$^{B2}$ Let $A(w)$ be the average weight enumerator function of a rate-1/3 random linear code with $N = 540$ and $M = 360$. Estimate, from first principles, the value of $A(w)$ at $w = 1$.

Exercise 14.14:$^{A3}$ Investigate the possibility of achieving the Shannon limit with linear block codes, using the following counting argument. Assume a linear code of large blocklength $N$ and rate $R = K/N$. The code's parity check matrix $\mathbf{H}$ has $M = N - K$ rows. Assume that the code's optimal decoder, which solves the syndrome decoding problem $\mathbf{Hn} = \mathbf{z}$, allows reliable communication over a binary symmetric channel with flip probability $f$.

How many 'typical' noise vectors $\mathbf{n}$ are there?

Roughly how many distinct syndromes $\mathbf{z}$ are there?

Since $\mathbf{n}$ is reliably deduced from $\mathbf{z}$ by the optimal decoder, the number of syndromes must be greater than or equal to the number of typical noise vectors. What does this tell you about the largest possible value of rate $R$ for a given $f$?

Exercise 14.15: Todd Ebert's 'hat puzzle', as reported in the New York Times.

Three players enter a room and a red or blue hat is placed on each person's head. The colour of each hat is determined by a coin toss, with the outcome of one coin toss having no effect on the others. Each person can see the other players' hats but not his own.

No communication of any sort is allowed, except for an initial strategy session before the group enters the room. Once they have had a chance to look at the other hats, the players must simultaneously guess their own hat's colour or pass. The group shares a $3 million prize if at least one player guesses correctly and no players guess incorrectly.

The same game can be played with any number of players. The general problem is to find a strategy for the group that maximizes its chances of winning the prize. Find the best strategies for groups of size **three** and **seven**.

[Hint: when you've done **three** and **seven**, you might be able to solve **fifteen**.]

If you already know the hat puzzle, you could try the 'Scottish version' of the rules in which the prize is only awarded to the group if they *all* guess correctly.

In the 'Reformed Scottish version', all the players must guess correctly, and there are *two* rounds of guessing. Those players who guess during round one leave the room. The remaining players must guess in round two. What strategy should the team adopt to maximize their chance of winning?

## Solutions

Solution to exercise 14.6 (p.254): A binary vector is perpendicular to itself if it has even weight, i.e., an even number of 1s.

Solution to exercise 14.11 (p.256): The (8,4) and (7,4) codes are intimately related. The (8,4) code, whose parity check matrix is

$$\mathbf{H} = \left[\ \mathbf{P} \mid \mathbf{I}_4\ \right] = \left[\begin{array}{cccc|cccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \tag{14.35}$$

is obtained by (a) appending an extra parity check bit which is the parity of all seven bits of the (7,4) Hamming code; and (b) reordering the first four bits (which were permuted in equation (14.34), relative to our earlier definition, so as to make **G** look pretty).

Solution to exercise 14.12 (p.256): If an $(N, K)$ code, with $M = N - K$ parity symbols, has the property that the decoder can recover the codeword when *any* $M$ symbols are erased in a block of $N$, then the code is said to be *maximum distance separable* (MDS).

No MDS binary codes exist, apart from the repetition codes and simple parity codes. For $q > 2$, some MDS codes can be found.

As a simple example, here is a (9,2) code for the 8-ary erasure channel. The code is defined in terms of the multiplication and addition rules of $GF(8)$, which are given in appendix B.6. The elements of the input alphabet are

$\{0, 1, A, B, C, D, E, F\}$ and the generator matrix of the code is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & A & B & C & D & E & F \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{14.36}$$

The resulting 64 codewords are:

```
000000000   011111111   0AAAAAAAA   0BBBBBBBB   0CCCCCCCC   0DDDDDDDD   0EEEEEEEE   0FFFFFFFF
101ABCDEF   110BADCFE   1AB01EFCD   1BA10FEDC   1CDEF01AB   1DCFE10BA   1EFCDAB01   1FEDCBA10
A0ACEB1FD   A1BDFA0EC   AA0EC1BDF   AB1FD0ACE   ACE0AFDB1   ADF1BECA0   AECA0DF1B   AFDB1CE0A
B0BEDFC1A   B1AFCED0B   BA1CFDEB0   BB0DECFA1   BCFA1B0DE   BDEB0A1CF   BED0B1AFC   BFC1A0BED
C0CBFEAD1   C1DAEFBC0   CAE1DC0FB   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
```

Solution to exercise 14.13 (p.257): A code has a word of weight 1 if an entire column of the parity check matrix is zero. There is a chance of $2^{-M} = 2^{-360}$ that all entries in a given column are zero. There are $M = 360$ columns. So the expected value at $w = 1$ is

$$A(1) = M2^{-M} = 360 \times 2^{-360} \simeq 10^{-111}. \tag{14.37}$$

Solution to exercise 14.14 (p.257): The number of 'typical' noise vectors $\mathbf{n}$ is roughly $2^{NH_2(f)}$. The number of distinct syndromes $\mathbf{z}$ is $2^M$. So reliable communication implies

$$M \geq NH_2(f), \tag{14.38}$$

or, in terms of the rate $R = 1 - M/N$,

$$R \leq 1 - H_2(f), \tag{14.39}$$

a bound which agrees precisely with the capacity of the channel.

Solution to exercise 14.15 (p.257):

In the three-player case, it is possible for the group to win three quarters of the time.

Three quarters of the time, two of the players will have hats of the same colour and the third player's hat will be the opposite colour. The group can win every time this happens by using the following strategy: Once the game starts, each player looks at the other two players' hats. If the two hats are *different* colours, he passes. If they are the *same* colour, the player guesses his own hat is the *opposite* colour.

This way, every time the hat colours are distributed two and one, one player will guess correctly and the others will pass, and the group will win the game. When all the hats are the same colour, however, *all three* players will guess incorrectly and the group will lose.

When any particular player guesses a colour, it is true that there is only a 50:50 chance that their guess is right. The reason that the group wins 75% of the time is that their strategy ensures that when players are guessing wrong, a great many are guessing wrong.

For larger numbers of players, the aim is to ensure that most of the time no one is wrong and occasionally everyone is wrong at once. In the game with 7 players, there is a strategy for which the group wins 7 out of every 8 times they play. In the game with 15 players, the group can win 15 out of 16 times.

If the number of players, $N$, is $2^r - 1$, the optimal strategy can be defined using a Hamming code of length $N$. Each player is identified with a number

$n \in 1 \ldots N$. The two colours are mapped onto `0` and `1`. Any state of their hats can be viewed as a received vector out of a binary channel. A random binary vector of length $N$ is either a codeword of the Hamming code, with probability $1/(N+1)$, or it differs in exactly one bit from a codeword. Each player looks at all the other bits and considers whether his bit can be set to a colour such that the state is a codeword (which can be deduced using the decoder of the Hamming code). If it can, then the player guesses that his hat is the *other* colour. If the state is actually a codeword, all players will guess and will guess wrong. If the state is a non-codeword, only one player will guess, and his guess will be correct.

# 15

# 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 270.

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*

Exercise 15.1:$^{B2}$ 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 assigned 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 15.2:$^{B2}$ 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 15.3:$^{A2}$ 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 15.4:$^{B2}$ Find a probability sequence $\mathbf{p} = (p_1, p_2, \ldots)$ such that $H(\mathbf{p}) = \infty$.

Exercise 15.5:$^{B2}$ 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 $T_{N\beta}$ (equation 5.29) where $N = 8$ and $\beta = 0.1$.

Exercise 15.6:$^{B2}$
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

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2/3 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1/3 & 0 \end{bmatrix}. \tag{15.1}$$

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 $\epsilon$. Be as explicit as possible. In particular, *do not* invoke Shannon's noisy-channel coding theorem.

Exercise 15.7:$^{B2}$ 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 15.8:$^{A2}$ Find the capacity and optimal input distribution for the three-input, three-output channel whose transition probabilities are:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 1/3 & 2/3 \end{bmatrix} \tag{15.2}$$

Exercise 15.9: 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 15.10: The ten-digit number on the cover of a book known as the ISBN (e.g., 0521642981) 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?

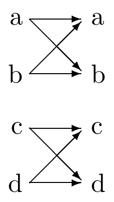
If the tenth digit were defined to be

$$x_{10} = \left( \sum_{n=1}^{9} n x_n \right) \bmod 10,$$

why would the code not work so well? (Discuss the detection of both modifications of single digits and transpositions of digits.)

**Exercise 15.11:** 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 $H_2(f) = f \log_2 \frac{1}{f} + (1-f) \log_2 \frac{1}{(1-f)}$.

Remember $\frac{d}{dp} H_2(p) = \log_2 \frac{1-p}{p}$.

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 15.12:** 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)?

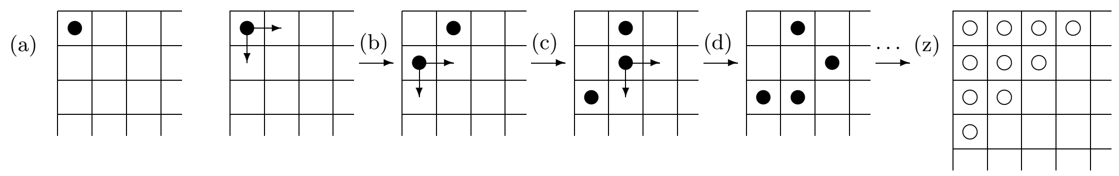


Figure 15.1. The `southeast` puzzle. Thanks to Tadashi for this one.

## *Lyapunov functions*

The `southeast` puzzle is played on a semi-infinite chess board, starting at its northwest (top left) corner. There are three rules:

1. In the starting position, one piece is placed in the northwest-most square (figure 15.1(a)).
2. It is not permitted for more than one piece to be on any given square.
3. At each step, you remove one piece from the board, and replace it with two pieces, one in the square immediately to the East, and one in the the square immediately to the South, as illustrated in figure 15.1(b). Every such step increases the number of pieces on the board by one.

After move (b) has been made, either piece may be selected for the next move. Figure 15.1(c) shows the outcome of moving the lower piece. At the next move, either the lowest piece or the middle piece of the three may be selected; the uppermost piece may not be selected, since that would violate rule 2. At move (d) we have selected the middle piece. Now any of the pieces may be moved, except for the leftmost piece.

Now, here is the puzzle:

Solutions on p.270

Exercise 15.13:$^{A4}$ Is it possible to obtain a position in which all the ten squares closest to the northwest corner, marked in figure 15.1(z), are empty?

## *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 15.14:$^{C3}$ 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 15.2a). The signals $x^{(A)}$ and $x^{(B)}$ are strongly correlated, so their joint information content is only a little greater than the marginal information content of either of them. For example, consider a weather collator who wishes to receive a string of reports saying whether it is raining in Derby ($x^{(A)}$) and whether it is raining in Nottingham ($x^{(B)}$). The joint probability of $x^{(A)}$ and $x^{(B)}$ might be

$$P(x^{(A)}, x^{(B)}): \quad \begin{array}{cc|cc} & & \multicolumn{2}{c}{x^{(A)}} \\ & & 0 & 1 \\ \hline x^{(B)} & 0 & 0.49 & 0.01 \\ & 1 & 0.01 & 0.49 \end{array} \tag{15.3}$$

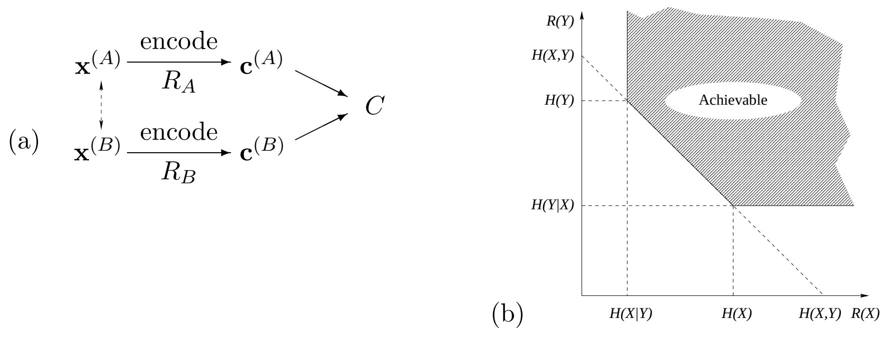


Figure 15.2. Communication of correlated information. (a) The communication situation: $x^{(A)}$ and $x^{(B)}$ are correlated sources (the correlation is represented by the dotted arrow). Strings of values of each variable are encoded using codes of rate $R_A$ and $R_B$ into transmissions $\mathbf{t}^{(A)}$ and $\mathbf{t}^{(B)}$, which are communicated over a noise free channel to a receiver $C$. (b) The achievable rate region. [Apologies for the AB-XY switch.] The interesting idea is that both strings can be conveyed without error even though $R_A < H(X^{(A)})$ and $R_B < H(X^{(B)})$.

The central receiver would like to know $N$ successive values of $x^{(A)}$ and $x^{(B)}$ exactly, but, being a penny-pincher, he is interested in the possibility of avoiding transmitting $N$ bits from source $A$ and $N$ bits from source $B$. 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 15.2. 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 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, $\mathbf{t}^{(B)} = \mathbf{x}^{(B)}$, and the other is encoded.

Exercise 15.15:$^{C3}$ Multiple access channels. Consider a channel with two sets of inputs and one output – for example, a shared telephone line (figure 15.3a). 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

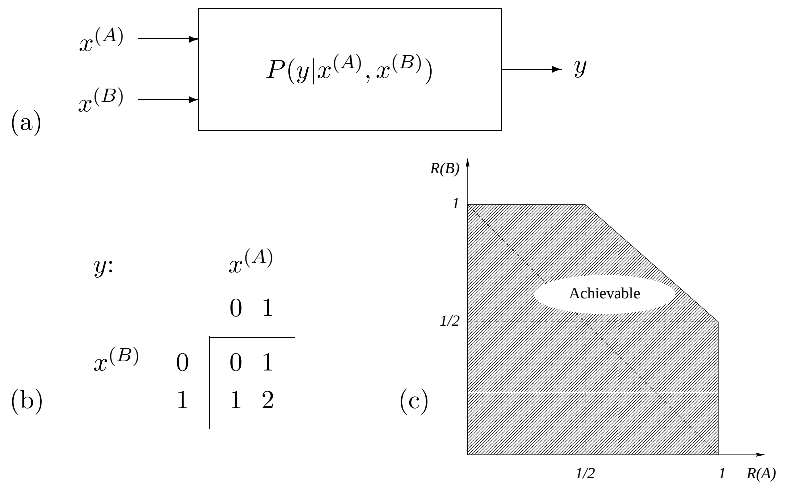


Figure 15.3. Multiple access channels. (a) A general multiple access channel with two transmitters and one receiver. (b) A binary multiple access channel with output equal to the sum of two inputs. (c) The achievable region.

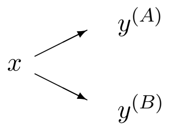


Figure 15.4. The broadcast channel. $x$ is the channel input; $y^{(A)}$ and $y^{(B)}$ are the outputs.

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 15.3. 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 15.16:$^{C3}$ Broadcast channels. A broadcast channel consists of a single transmitter and two or more receivers. The properties of the channel are defined by a conditional distribution $Q(y^{(A)}, y^{(B)}|x)$. (We'll assume the channel is memoryless.) The task is to add an encoder and two decoders to enable reliable communication of a common message at rate $R_0$ to both receivers, an individual message at rate $R_A$ to receiver $A$, and an individual message at rate $R_B$ to receiver $B$. The *capacity* region of the broadcast channel is the convex hull of the set of achievable rate triplets $(R_0, R_A, R_B)$.

A simple benchmark for such a channel is given by time-division signaling. If the capacities of the two channels, considered separately, are $C^{(A)}$ and $C^{(B)}$, then by devoting a fraction $\phi_A$ of the transmission time to channel $A$ and $\phi_B = 1 - \phi_A$ to channel B, we can achieve $(R_0, R_A, R_B) = (0, \phi_A C^{(A)}, \phi_B C^{(B)})$.

We can do better than this, however. As an analogy, imagine speaking simultaneously to an American and a Golgafrinchan telephone sanitizer; you are fluent in American and in Golgafrinchan, but, needless to say, neither of your two receivers understands the other's language. If each receiver can distinguish whether a word is in their own language or not, then an extra binary file can be conveyed to both recipients by using its bits to decide whether the next transmitted word should be from the American source text or from the Golgafrinchan source text. Each recipient can concatenate the words that they understand in order to receive their personal message, and can also recover the binary string.

As a special case of a broadcast channel, consider the case where the inputs and outputs are binary, and there are symmetric flip probabilities $f_A$ and $f_B$ associated with the two halves of the channel. We'll assume that $A$ has the better half-channel, i.e., $f_A < f_B < 1/2$. A closely related channel is a 'degraded' broadcast channel, in which the conditional probabilities are such that the random variables have the structure of a Markov chain,

$$x \rightarrow y^{(A)} \rightarrow y^{(B)}, \tag{15.4}$$

that is $y^{(B)}$ is a further degraded version of $y^{(A)}$. In this special case, it turns out that whatever information is getting through to receiver $B$ can also be recovered by receiver $A$. So there is no point distinguishing between $R_0$ and $R_B$: our task is to find the capacity region for the rate pair $(R_0, R_A)$, where $R_0$ is the rate of information reaching both $A$ and $B$, and $R_A$ is the rate of the extra information reaching $A$.

Exercise 15.17:$^{C3}$ Variable-rate error-correcting codes for channels with unknown noise level. In real life, channels may sometimes not be well characterized before the encoder is installed. As a model of this situation, we could imagine that a channel could be a binary symmetric channel with noise level $f_A$, or it might have a higher noise level $f_B$. Let the two capacities be $C_A$ and $C_B$.

Those who like to live dangerously might install a system designed for noise level $f_A$ with rate $R_A \simeq C_A$; in the event that the noise level turns out to be $f_B$, our experience of Shannon's theories would lead us to expect that there would be a catastrophic failure to communicate information reliably (solid line in figure 15.5).

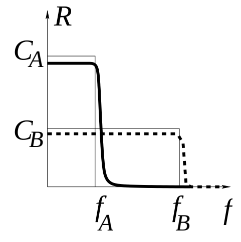


Figure 15.5. Rate of reliable communication $R$, as a function of noise level $f$, for Shannonesque codes designed to operate at noise levels $f_A$ (solid line) and $f_B$ (dashed line).

A conservative approach would design the encoding system for the worst case scenario, installing a code with rate $R_B \simeq C_B$. In the event that the lower noise level, $f_A$, holds true, then the managers would have a feeling of regret because of the wasted capacity difference $C_A - R_B$ (dashed line in figure 15.5).

Is it possible to create a system that not only transmits reliably at some rate $R_0$ whatever the noise level, but also communicates some extra, 'lower-priority' bits if the noise level is low, as shown in figure 15.6?

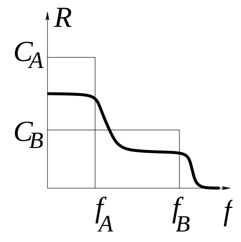
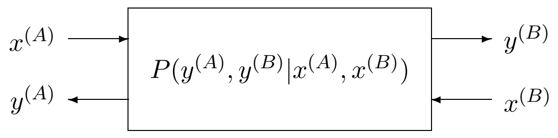


Figure 15.6. Rate of reliable communication $R$, as a function of noise level $f$, for a desired *variable-rate* code that communicates the high-priority bits reliably at all noise levels between $f_A$ and $f_B$, and communicates the low-priority bits also if the noise level is $f_A$ or below.

This problem is mathematically equivalent to the previous problem, the degraded broadcast channel. The lower rate of communication was there called $R_0$, and the rate at which the low-priority bits are communicated if the noise level is low was called $R_A$.

Exercise 15.18:$^{C3}$ Multiterminal information networks are both important practically and intriguing theoretically. Consider the following example of a two-way binary channel (figure 15.7a,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 $A$ 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

(a)


(b)

$$y^{(A)}: \quad \begin{array}{cc|cc} & & \multicolumn{2}{c}{x^{(A)}} \\ & & 0 & 1 \\ \hline x^{(B)} & 0 & 0 & 0 \\ & 1 & 1 & 0 \end{array} \qquad y^{(B)}: \quad \begin{array}{cc|cc} & & \multicolumn{2}{c}{x^{(A)}} \\ & & 0 & 1 \\ \hline x^{(B)} & 0 & 0 & 1 \\ & 1 & 0 & 0 \end{array}$$

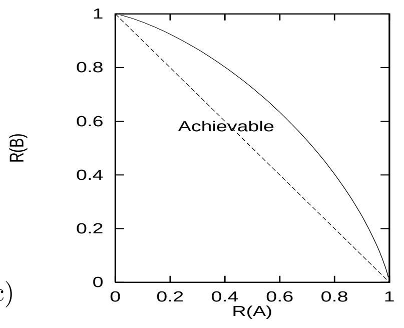
(c)


Figure 15.7. (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.

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 15.7. There exist codes that can achieve rates up to the boundary shown. There may exist better codes too.

## *Gambling oddities*

Exercise 15.19:$^{C2}$ (From Cover and Thomas (1991).) 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,

$$\sum_i \frac{1}{o_i} = 1, \tag{15.5}$$

and assuming that the probability of horse $i$'s winning is $p_i$, work out the optimal betting strategy if your aim is *Cover's aim*, namely, to maximize the *expected value of* $\log m(T)$. Show that the optimal strategy sets $\mathbf{b}$ equal to $\mathbf{p}$, independent of the bookies' odds $\mathbf{o}$. Show that when this strategy is used, the money is expected to grow exponentially as:

$$2^{nW(\mathbf{b},\mathbf{p})} \tag{15.6}$$

where $W = \sum_i p_i \log b_i o_i$.

If you only bet once, is the optimal strategy any different?

Do you think this optimal strategy makes sense? Do you really think that it's 'optimal' investment strategy, in common language, to ignore the bookies' odds? What can you conclude about 'Cover's aim'?

# *Solutions to Chapter 15's exercises*

Solution to exercise 15.1 (p.261):

(a) $\lceil \log_2 166751 \rceil = 18$ bits.

(b) $1.67 \times 10^{-3}$

Solution to exercise 15.2 (p.261):

(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 15.3 (p.261): The optimal symbol code (i.e., questioning strategy) has expected length $3\frac{11}{36}$.

Solution to exercise 15.8 (p.262): By symmetry, the optimal input distribution for the channel

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-f & f \\ 0 & f & 1-f \end{bmatrix} \tag{15.7}$$

has the form $((1-p), p/2, p/2)$. The optimal input distribution is given by

$$p^* = \frac{1}{1 + 2^{H_2(f)-1}}. \tag{15.8}$$

In the case $f = 1/3$, $p^* = 0.514$ and the capacity is $C = 1.041$ bits.

Solution to exercise 15.5 (p.261): $|T| = 2716$.

Solution to exercise 15.9 (p.262): $C(Q) = 5$ bits.
Hint for last part: a solution exists that involves a simple (8,5) code.

Solution to exercise 15.11 (p.263): The optimal input distribution is $(1/6, 1/6, 1/3, 1/3)$, and the capacity is $\log_2 3$ bits.

Solution to exercise 15.13 (p.264): When you played with this problem, I hope that you noticed its similarity to the construction of a binary symbol code. Starting from the empty string, we build a binary tree by replacing splitting a leaf into two.

An important idea in source codes was the Kraft equality. Every codeword has an implicit probability $2^{-l}$, where $l$ is the depth of the codeword in the binary tree; and for a complete binary code, the sum of these implicit probabilities is 1. Whenever we split a codeword in two and create two new codewords whose length is increased by one, the two new codewords have implicit probability equal to half that of the old codeword.

Similarly, in `southeast`, we can associate a 'weight' with each piece on the board. If we assign a weight of 1 to any piece sitting on the top left square; a weight of 1/2 to any piece on a square whose distance from the top left is one; a weight of 1/4 to any piece whose distance from the top left is two; and so forth, with 'distance' being the city-block distance; then every legal move in `southeast` leaves unchanged the total weight of all pieces on the board.

The starting weight is 1, so now we have a powerful tool: a conserved function of the state. Is it possible to find a position in which the ten highest-weight squares are vacant, and the total weight is 1? What is the total weight if *all* the other squares on the board are occupied (figure 15.8)? The total weight would be $\sum_{l=4}^{\infty}(l+1)2^{-l}$. Which is equal to 3/4. So it is impossible to empty all ten of those squares.

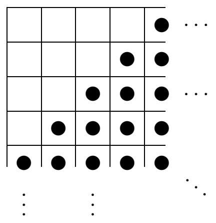


Figure 15.8. A possible position for the `southeast` puzzle?

Functions of the state of a dynamical system like the total weight, which we used here, are known as Lyapunov functions. Lyapunov functions come in two flavours: the function may be a function of state whose value is known to stay constant; or it may be a function of state that is bounded below, and whose value always decreases or stays constant.

# 16

# *Communication over Constrained Noiseless Channels* *

In this chapter we study the task of communicating efficiently over a constrained noiseless channel – a constrained channel over which not all strings from the input alphabet may be transmitted.

## 16.1 Three examples of constrained binary channels

A constrained channel can be defined by rules that define which strings are permitted.

Example 16.1: In Channel A every 1 must be followed by at least one 0.

**Channel A**: the substring 11 is forbidden.

A valid string for this channel is

$$\texttt{00100101001010100010.} \tag{16.1}$$

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 generate another.

Example 16.2: 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.

**Channel B**: 101 and 010 are forbidden.

A valid string for this channel is

$$\texttt{00111001110011000011.} \tag{16.2}$$

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 are 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 16.3: Channel C has the rule that the largest permitted runlength is two, that is, each symbol can be repeated at most once.

**Channel C**: 111 and 000 are forbidden.

A valid string for this channel is

$$\texttt{10010011011001101001.} \tag{16.3}$$

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 `1`s and a string of three `1`s, which are represented by oriented magnetizations of duration $2\tau$ and $3\tau$ respectively, where $\tau$ is the (poorly known) time taken for one bit to pass by; to avoid the possibility of confusion, and the resulting loss of synchronization of sender and receiver, we forbid the string of three `1`s.

All three of these channels are examples of *runlength-limited channels*. The rules constrain the minimum and maximum numbers of successive `1`s and `0`s.

| Channel | Runlength of `1`s | | Runlength of `0`s | |
|---|---|---|---|---|
| | min. | max. | min. | max. |
| unconstrained | 1 | $\infty$ | 1 | $\infty$ |
| A | 1 | 1 | 1 | $\infty$ |
| B | 2 | $\infty$ | 2 | $\infty$ |
| C | 1 | 2 | 1 | 2 |

In channel A, runs of `0`s may be of any length but runs of `1`s are restricted to length one. 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? [To be fair, we haven't defined the 'capacity' of such channels yet; please understand 'capacity' as meaning how many bits can be reliably conveyed per channel-use.]

### *Exploring a constrained channel*

Let us concentrate for a moment on channel A, in which runs of `0`s may be of any length but runs of `1`s are restricted to length one. 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 $C_1$ (16.4)

| $s$ | $t$ |
|---|---|
| `0` | `00` |
| `1` | `10` |

This is a rate-1/2 code, and it respects the constraints of channel A, 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 zero. We can achieve a smaller average transmitted length using a code that omits the redundant zeroes in $C_1$.

CODE $C_2$ (16.5)

| $s$ | $t$ |
|---|---|
| `0` | `0` |
| `1` | `10` |

$C_2$ 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

$$L = \frac{1}{2}1 + \frac{1}{2}2 = \frac{3}{2}, \tag{16.6}$$

so the average communication rate is

$$R = 2/3, \tag{16.7}$$

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, starting from code $C_2$, 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 $f$. [Such a stream could be obtained from an arbitrary binary file by passing the source file into the decoder of an arithmetic code that is optimal for compressing binary strings of density $f$.] The information rate $R$ achieved is the entropy of the source, $H_2(f)$, divided by the mean transmitted length,

$$L(f) = (1 - f) + 2f = 1 + f. \tag{16.8}$$

Thus

$$R(f) = \frac{H_2(f)}{L(f)} = \frac{H_2(f)}{1 + f}. \tag{16.9}$$

The original code $C_2$, without preprocessor, corresponds to $f = 1/2$. What happens if we perturb $f$ a little towards smaller $f$, setting

$$f = \frac{1}{2} + \delta, \tag{16.10}$$

for small negative $\delta$? In the vicinity of $f = 1/2$, the denominator $L(f)$ varies linearly with $\delta$. In contrast, the numerator $H_2(f)$ only has a second-order dependence on $\delta$.

**Exercise 16.4:** Find, to order $\delta^2$, the Taylor expansion of $H_2(f)$ as a function of $\delta$.

To first order, $R(f)$ increases linearly with decreasing $\delta$. It must be possible to increase $R$ by decreasing $f$. Figure 16.1 shows these functions; $R(f)$ does indeed increase as $f$ decreases and has a maximum of about 0.69 bits per channel use at $f \simeq 0.38$.

By this argument we have shown that the capacity of channel A is at least $\max_f R(f) = 0.69$.

**Exercise 16.5:** 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 $f = 0.38$?

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.

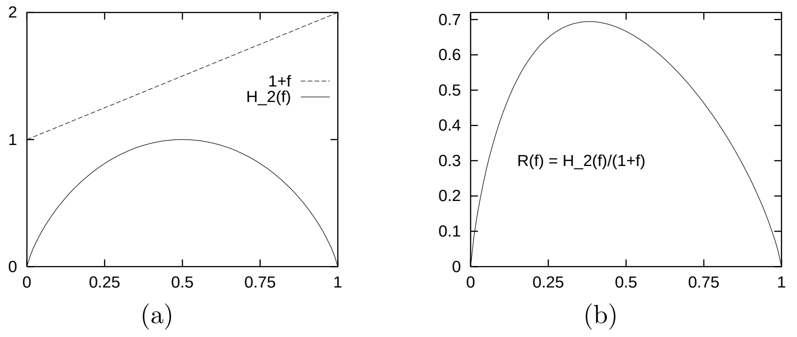


Figure 16.1. (a) The information content per source symbol and mean transmitted length per source symbol as a function of the source density. (b) The information content per transmitted symbol, in bits, as a function of $f$.

## 16.2 The capacity of a constrained noiseless channel

We defined the capacity of a noisy channel in terms of the mutual information between its input and its output, then we proved 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

$$C = \lim_{N\to\infty} \frac{1}{N} \log S(N). \qquad (16.11)$$

In the case of the constrained noiseless channel, we can adopt this identity as our definition of the channel's capacity. However, the name $s$, which, when we were making codes for noisy channels (section 10.6), ran over messages $s = 1 \ldots S$ is about to take on a new role, labelling the states of our channel, so in this chapter we will denote the number of distinguishable messages of length $N$ by $M_N$, and define the capacity to be:

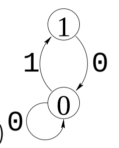
$$C = \lim_{N\to\infty} \frac{1}{N} \log M_N. \qquad (16.12)$$

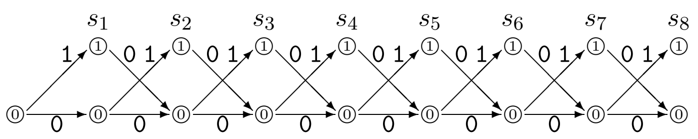
Knowing the capacity of a channel doesn't tell us how practically to achieve that rate of communication, so once we have figured out the capacity of a channel we will have to return the task of making a code for that channel.

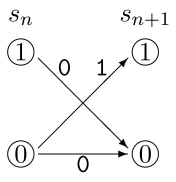
## 16.3 Counting the number of possible messages

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 16.2(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 16.2(b)). The state of the transmitter at time $n$ is called $s_n$. The set of possible state

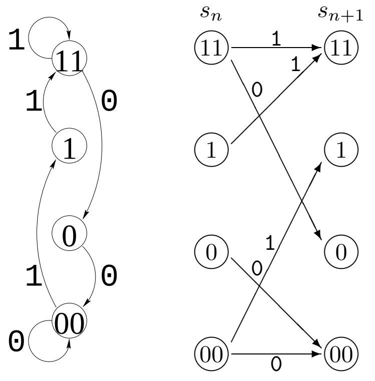
(a)


(c)


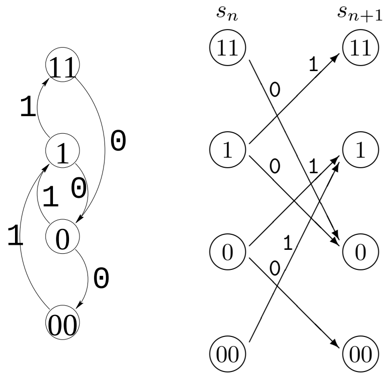
(b)


(d) $$\mathbf{A} = \begin{array}{cc|cc} & & \multicolumn{2}{c}{\text{(from)}} \\ & & 1 & 0 \\ \hline \text{(to)} & 1 & 0 & 1 \\ & 0 & 1 & 1 \end{array}$$

Figure 16.2. (a) State diagram for channel A. (b) Trellis section. (c) Trellis. (d) Connection matrix.



B $$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$



C $$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 16.3. State diagrams, trellis sections and connection matrices for channels B and C.

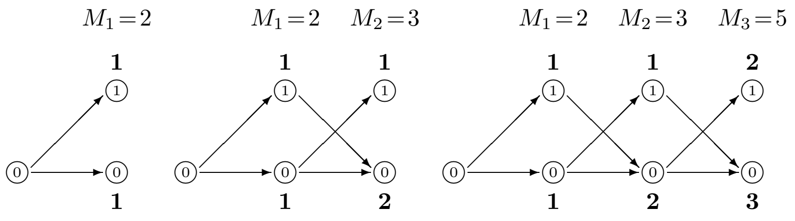


Figure 16.4. Counting number of paths in the trellis of channel A. Global properties of graphs can be computed by a local message-passing algorithm. The counts next to the nodes are accumulated by passing from left to right across the trellises.

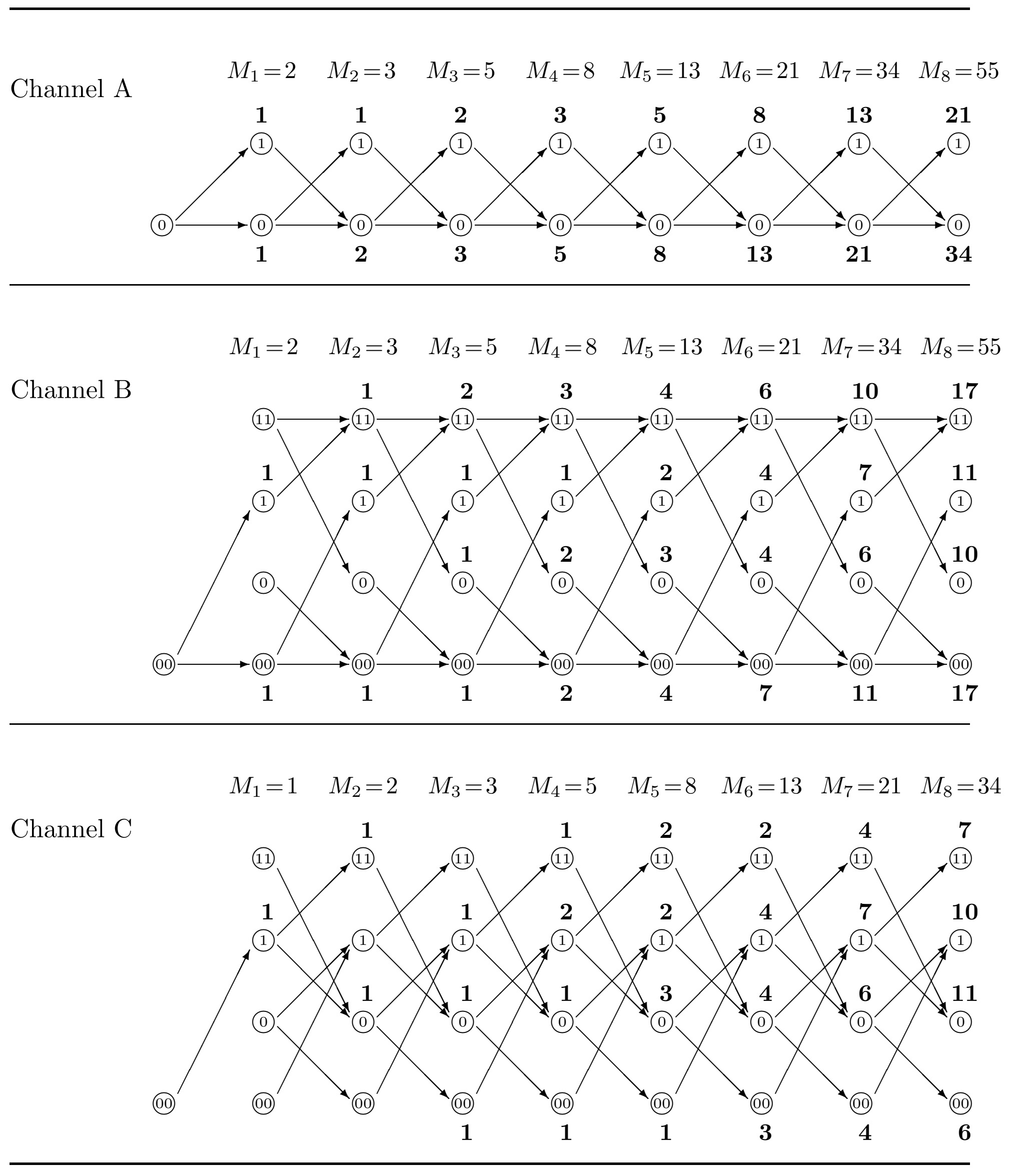

Channel A
$M_1=2$ $M_2=3$ $M_3=5$ $M_4=8$ $M_5=13$ $M_6=21$ $M_7=34$ $M_8=55$
1 1 2 3 5 8 13 21
1 2 3 5 8 13 21 34
Channel B
$M_1=2$ $M_2=3$ $M_3=5$ $M_4=8$ $M_5=13$ $M_6=21$ $M_7=34$ $M_8=55$
1 2 3 4 6 10 17
1 1 1 1 2 4 7 11
1 2 3 4 6 10
1 1 1 2 4 7 11 17
Channel C
$M_1=1$ $M_2=2$ $M_3=3$ $M_4=5$ $M_5=8$ $M_6=13$ $M_7=21$ $M_8=34$
1 1 2 2 4 7
1 1 2 2 4 7 10
1 1 1 3 4 6 11
1 1 1 3 4 6

Figure 16.5. 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.

| $n$ | $M_n$ | $M_n/M_{n-1}$ | $\log_2 M_n$ | $\frac{1}{n}\log_2 M_n$ |
|---|---|---|---|---|
| 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 16.6. Counting number of paths in the trellis of channel A.

sequences can be represented by a *trellis* as shown in figure 16.2(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 16.2(d)).

So, let's count! Figure 16.4 shows the first few steps of this counting process, and figure 16.5(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 $n$, $M_n$, is shown along the top. We recognize $M_n$ as the Fibonacci series.

Exercise 16.6: Show that the ratio of successive terms in the Fibonacci series tends to the golden ratio,

$$\gamma \equiv \frac{1+\sqrt{5}}{2} = 1.618. \tag{16.13}$$

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

$$C = \lim \frac{1}{N} \log_2 \left[\text{constant} \cdot \gamma^N\right] = \log_2 \gamma = \log_2 1.618 = 0.694. \tag{16.14}$$

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:

$$\mathbf{c}^{(n+1)} = \mathbf{A}\mathbf{c}^{(n)}. \tag{16.15}$$

So:

$$\mathbf{c}^{(N)} = \mathbf{A}^N \mathbf{c}^{(0)}, \tag{16.16}$$

where $\mathbf{c}^{(0)}$ is the state count before any symbols are transmitted. In figure 16.5 we assumed $\mathbf{c}^{(0)} = [0, 1]^{\mathsf{T}}$, i.e., that either of the two symbols is permitted at 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 $\mathbf{A}$.

$$\mathbf{c}^{(N)} \to \text{constant} \cdot \lambda_1^N \mathbf{e}_{\mathsf{R}}^{(0)} \tag{16.17}$$

Here, $\lambda_1$ 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, $\lambda_1$. Then

$$C = \log_2 \lambda_1. \tag{16.18}$$

## 16.4 Back to our model channels

Comparing figure 16.5(a) and figure 16.5(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 the same (see section B.7). And indeed the channels are intimately related.

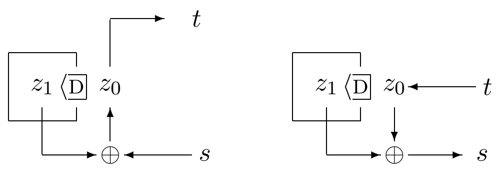


Figure 16.7. An accumulator and a differentiator.

*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:

$$\begin{array}{rcl} t_1 &=& s_1 \\ t_n &=& t_{n-1} + s_n \bmod 2 \quad \text{for } n \geq 2 \end{array}, \tag{16.19}$$

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*,

$$\begin{array}{rcl} s_1 &=& t_1 \\ s_n &=& t_n - t_{n-1} \bmod 2 \quad \text{for } n \geq 2. \end{array} \tag{16.20}$$

Because $+$ and $-$ are equivalent in modulo 2 arithmetic, the differentiator is also a blurrer, convolving the source stream with the filter $(1, 1)$.

Channel C is also intimately related to channels A and B.

**Exercise 16.7:** What is the relationship of channel C to channels A and B?

## 16.5 Practical communication over constrained channels

OK, how to do it in practice? Since all three channels are equivalent, we can concentrate on channel A.

### Fixed-length solutions

We start with explicitly-enumerated codes. The code in the margin achieves a rate of $3/5 = 0.6$.

| $s$ | $c(s)$ |
|---|---|
| 1 | 00000 |
| 2 | 10000 |
| 3 | 01000 |
| 4 | 00100 |
| 5 | 00010 |
| 6 | 10100 |
| 7 | 01010 |
| 8 | 10010 |

Exercise 16.8:$^{B1}$ Similarly, enumerate all strings of length 8 that end in the zero state. (There are 34 of them.) Hence show that we can map 5 bits (32 source strings) to 8 transmitted bits and achieve rate $5/8 = 0.625$.

What rate can be achieved by mapping an integer number of source bits to $N = 16$ transmitted bits?

### Optimal variable-length solution

The optimal way to convey information over the constrained channel is to find the optimal transition probabilities for all points in the trellis, $Q_{s'|s}$, and make transitions with these probabilities.

When discussing channel A, we already discussed the idea that a sparse source with density $f = 0.38$, driving code $C_2$, would achieve capacity. And we know how to make sparsifiers (chapter 7). We design an arithmetic code that is optimal for compressing a sparse source; then its associated decoder gives an optimal mapping from dense (i.e., random binary) strings to sparse strings.

The task of finding the optimal probabilities is left as an exercise.

Exercise 16.9:$^{C3}$ Show that the optimal transition probabilities $\mathbf{Q}$ can be found as follows.

Find the principal right- and left-eigenvectors of $\mathbf{A}$, that is the solutions of $\mathbf{A}\mathbf{e}^{(R)} = \lambda\mathbf{e}^{(R)}$ and $\mathbf{e}^{(L)^{\mathsf{T}}}\mathbf{A} = \lambda\mathbf{e}^{(L)^{\mathsf{T}}}$ with largest eigenvalue $\lambda$. Then construct a matrix $\mathbf{Q}$ whose invariant distribution is proportional to $e_i^{(R)}e_i^{(L)}$, namely

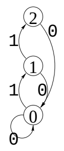
$$Q_{s'|s} = \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}}. \tag{16.21}$$

[Hint: exercise 4.3 (p.74) might give helpful cross-fertilization here.]

Exercise 16.10: Show that when sequences are generated using the optimal transition probability matrix (16.21), the entropy of the resulting sequence is 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.]

In practice, we would probably use finite-precision approximations to the optimal variable-length solution. One might dislike variable-length solutions because of the resulting unpredictability of the actual encoded length in any particular case. Perhaps in some applications we would like a guarantee that the encoded length of a source file of size $N$ bits will be less than a given length such as $N/(C + \epsilon)$. For some constrained channels we can make a simple modification to our variable-length encoding and offer such a guarantee, as follows (MacKay 2000). We find two codes, two mappings of binary strings to variable-length encodings, having the property that for any source string $\mathbf{x}$, if the encoding of $\mathbf{x}$ under the first code is shorter than average, then the encoding of $\mathbf{x}$ under the second code is longer than average, and vice versa.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

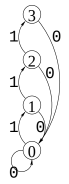


Figure 16.8. State diagrams and connection matrices for channels with maximum runlengths for 1s equal to 2 and 3.

Then to transmit a string **x** we encode it with both codes and send whichever encoding has the shortest length, prepended by a suitably encoded single bit to convey which of the two codes is being used.

Exercise 16.11**:** How many valid sequences of length 8 ending with a 0 are there for the run-length-limited channels shown in figure 16.8?

What are the capacities of these channels?

Using a computer, find the matrix **Q** for generating a random path through the trellises of the channel A, and the two run-length-limited channels shown in figure 16.8.

Exercise 16.12**:** 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 **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 generating a 1 given a preceding 0, 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 channel in which the string 1111111111 is forbidden, i.e., the maximum runlength of 1s is nine.

## 16.6 Variable symbol durations*

We can add a further frill to the task of communicating over constrained channels by assuming that the symbols we send have different *durations*, and that our aim is to communicate at the maximum possible rate per unit time. Such channels can come in two flavours: unconstrained, and constrained.

### *Unconstrained channels with variable symbol durations*

We already encountered an unconstrained noiseless channel with variable symbol durations in exercise 8.6 (p.151). Solve that problem, and you've done this topic. The task is to determine the optimal probabilities of using the symbols as a function of their durations.

Notice the analogy between this task and the task of designing a symbol code (chapter 5). When we make an optimal binary symbol code for a source with unequal probabilities $p_i$, the optimal message lengths are $l_i^* = \log_2 \frac{1}{p_i}$, so

$$p_i = 2^{-l_i^*}. \tag{16.22}$$

When we have a channel whose symbols have durations $l_i$ (in some units of time), the optimal probability with which those symbols should be used is

$$p_i^* = 2^{-\beta l_i}, \tag{16.23}$$

where $\beta$ is the capacity of the channel in bits per unit time.

*Constrained channels with variable symbol durations*

Once you have grasped the preceding topics in this chapter, you should be able to figure out how to define and find the capacity of these, the trickiest constrained channels.

Exercise 16.13:$^{C3}$ A classic example of a constrained channel with variable symbol durations is the 'Morse' channel, whose symbols are

| | |
|---|---|
| the dot | `d`, |
| the dash | `D`, |
| the short space (used between letters in morse code) | `s`, and |
| the long space (used between words) | `S`; |

the constraints are that spaces may only be followed by dots and dashes.

Find the capacity of this channel assuming (a) that all four symbols have equal durations; or (b) that the symbol durations are 2, 4, 3 and 6 time units respectively.

# *Solutions to Chapter 16's exercises*

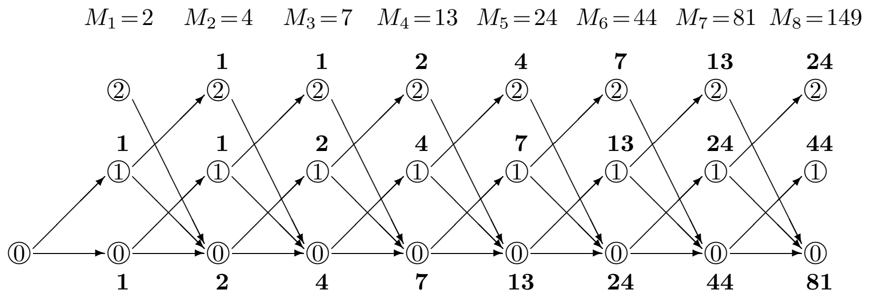
Solution to exercise 16.5 (p.274): A file transmitted by $C_2$ contains, on average, one third 1s and two thirds 0s.

If $f = 0.38$, the fraction of 1s is $f/(1+f) = (\gamma - 1.0)/(2\gamma - 1.0) = 0.2764$.

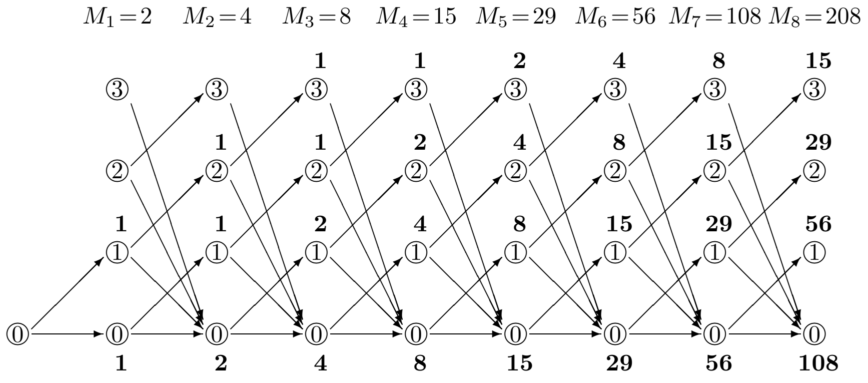
Solution to exercise 16.7 (p.279): A valid string for channel C can be obtained from a valid string for channel A by first inverting it [$1 \to 0$; $0 \to 1$], then passing it through an accumulator. These operations are invertible, so any valid string for C can also be mapped onto a valid string for 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 16.5(c)) then the two channels are exactly equivalent only if we assume that channel A's first character must be a zero.

Solution to exercise 16.8 (p.280): With $N = 16$ transmitted bits, the largest integer number of source bits that can encoded is 10, so the maximum rate of a fixed length code is 0.625.

Solution to exercise 16.11 (p.281): 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 B.7) for the matrices $\mathbf{Q}$.



$M_1 = 2$ $M_2 = 4$ $M_3 = 7$ $M_4 = 13$ $M_5 = 24$ $M_6 = 44$ $M_7 = 81$ $M_8 = 149$
1 1 2 4 7 13 24
1 1 2 4 7 13 24 44
1 2 4 7 13 24 44 81


$M_1=2$ $M_2=4$ $M_3=8$ $M_4=15$ $M_5=29$ $M_6=56$ $M_7=108$ $M_8=208$
1 1 2 4 8 15
1 1 2 4 8 15 29
1 1 2 4 8 15 29 56
1 2 4 8 15 29 56 108

Solution to exercise 16.10 (p.280): Let the invariant distribution be

$$P(s) = \alpha e_s^{(L)} e_s^{(R)}, \tag{16.24}$$

where $\alpha$ is a normalization constant. The entropy of $S_t$ given $S_{t-1}$, assuming $S_{t-1}$ comes from the invariant distribution, is

Here, as in chapter 5, $S_t$ denotes the ensemble whose random variable is the state $s_t$.

$$H(S_t|S_{t-1}) = -\sum_{s,s'} P(s)P(s'|s)\log P(s'|s) \tag{16.25}$$

$$= -\sum_{s,s'} \alpha e_s^{(L)} e_s^{(R)} \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}} \log \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}} \tag{16.26}$$

$$= -\sum_{s,s'} \alpha\, e_s^{(R)} \frac{e_{s'}^{(L)} A_{s's}}{\lambda} \left[\log e_{s'}^{(L)} + \log A_{s's} - \log\lambda - \log e_s^{(L)}\right]. \tag{16.27}$$

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

$$H(S_t|S_{t-1}) = \log\lambda + -\frac{\alpha}{\lambda}\sum_{s'}\left(\sum_s A_{s's} e_s^{(R)}\right) e_{s'}^{(L)} \log e_{s'}^{(L)} + \frac{\alpha}{\lambda}\sum_s\left(\sum_{s'} e_{s'}^{(L)} A_{s's}\right) e_s^{(R)} \log e_s^{(L)} \tag{16.28}$$

$$= \log\lambda - \frac{\alpha}{\lambda}\sum_{s'} \lambda e_{s'}^{(R)} e_{s'}^{(L)} \log e_{s'}^{(L)} + \frac{\alpha}{\lambda}\sum_s \lambda e_s^{(L)} e_s^{(R)} \log e_s^{(L)} \tag{16.29}$$

$$= \log\lambda. \tag{16.30}$$

Solution to exercise 16.12 (p.281): 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' state a fraction $2^{-L}$ of the time.

We can reuse the solution for the variable-length channel in exercise 8.6 (p.151). The capacity is the value of $\beta$ such that the equation

$$Z(\beta) = \sum_{l=1}^{L+1} 2^{-\beta l} = 1 \tag{16.31}$$

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:

$$\left[\sum_{n=0}^{N} ar^n = \frac{a(r^{N+1}-1)}{r-1}\right]$$

$$Z(\beta) = 2^{-\beta} \frac{\left(2^{-\beta}\right)^{L+1} - 1}{2^{-\beta} - 1}. \tag{16.32}$$

We anticipate that $\beta$ should be a little less than 1 in order for $Z(\beta)$ to equal 1. Rearranging and solving approximately for $\beta$, using $\ln(1+x) \simeq x$,

$$\begin{aligned} Z(\beta) &= 1 & (16.33)\\ \Rightarrow \beta &\simeq 1 - 2^{-(L+2)}/\ln 2 & (16.34) \end{aligned}$$

We evaluated the true capacities for $L = 2$ and $L = 3$ in an earlier exercise. The following table compares the approximate capacity $\beta$ with the true capacity for a selection of values of $L$.

| $L$ | $\beta$ | 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),

$$\begin{aligned} H(\text{the next } N \text{ chars}) &\simeq H_2(f) + [(N-1)(1-f) + (N-2)f]\, C \\ &= H_2(f) + NC - fC \simeq H_2(f) + N - f. \end{aligned} \tag{16.35}$$

Differentiating and setting to zero to find the optimal $f$, we obtain:

$$\log_2 \frac{1-f}{f} \simeq 1 \;\Rightarrow \frac{1-f}{f} \simeq 2 \;\Rightarrow f \simeq 1/3. \tag{16.36}$$

The probability of emitting a `1` thus decreases from about 0.5 to about 1/3 as the number of emitted `1`s increases.

Here is the optimal matrix:

$$\begin{bmatrix} 0 & .3334 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .4287 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .4669 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .4841 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .4923 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .4963 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4983 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4993 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4998 \\ 1 & .6666 & .5713 & .5331 & .5159 & .5077 & .5037 & .5017 & .5007 & .5002 \end{bmatrix} \tag{16.37}$$

Our rough theory works.

# 17

# *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.)

## 17.1 Language models

Language models are often described starting from simple models which neglect correlations between letters, or correlations between words, and progressing to more complex models that include correlations.

We will follow this same route, starting with a discussion of language modelling under the assumption that successive symbols in the source stream – here 'symbols' may denote either letters or words – come independently from a fixed distribution. This sounds such a simple model that we might feel impatient to move on to more complex models, in which the probability of a word depends on the context, but the monogram model is a surprisingly rich starting point.

### *Models for monogram statistics*

The monogram modelling problem is this: we are given a bent $I$-sided die, each face bearing a word; for simplicity we might call the words $1, 2, \ldots, i, \ldots, I$. We roll the die $F$ times. From the string of outcomes $w_1 w_2 w_3 \ldots w_F$, where $w_f \in \{1, 2, \ldots, I\}$, we wish to infer the parameter vector $\mathbf{p} = (p_1, p_2, \ldots, p_i, \ldots, p_I)$, which defines the probability of face $i$ coming up when the die is rolled. The Bayesian solution to this problem is to write down the posterior probability of $\mathbf{p}$,

$$P(\mathbf{p}|w_1w_2w_3\ldots w_F) = \frac{P(w_1w_2w_3\ldots w_F|\mathbf{p})P(\mathbf{p})}{P(w_1w_2w_3\ldots w_F)}. \tag{17.1}$$

The likelihood function is:

$$P(w_1w_2w_3\ldots w_F|\mathbf{p}) = \prod_{f=1}^{F} p_{w_f} = \prod_{i=1}^{I} p_i^{F_i}, \tag{17.2}$$

where we have introduced $F_i$, the number of times that word $i$ occured.

So now, two difficulties remain:

1. What should the prior $P(\mathbf{p})$ be?
2. What if the number of words $I$ is not known, or is potentially infinite?

For dice with a small number of faces $I$, the choice of prior may not be very important, but if $I$ is large, and there are a lot of rarely-occuring words, then the choice of prior becomes a more interesting issue.

We'll start by assuming $I$ is known.

Plan: uniform prior (Laplace prior) - special case of Dirichlet distribution. Review Dirichlet.

Then mention known property of real data – it often follows a Zipf law. Give evidence in the form of word count graphs. Show Zipf plots for Dirichlet distribution.

At some point change to infinite I case, and review Dirchlet process. Show Zipf plots for Dirichlet process.

How to make a model that has Zipf-like predictions? One idea is to have the language be made from a Dirichlet process at a microscopic level (eg letters), and then define words by letting one character be the space. Then - show plots - show theory - credit Mandelbrot - the Zipf law results.

Then go on to hierarchical Dirichlet language models.

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.

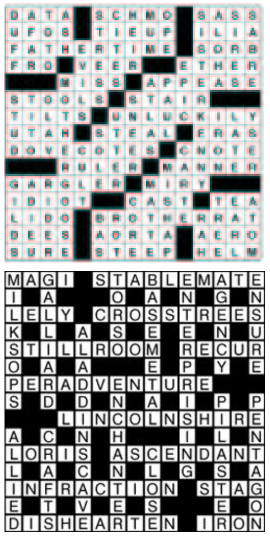


Figure 17.1. Crosswords of types A (American) and B (British)

## 17.2 Crosswords

There are two archetypal crossword formats. In a 'type A' (or American) 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' (or British) 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.

Type A crosswords are harder to *create* than type B because of the constraint that no single characters are permitted. Type B crosswords are generally harder to *solve* because there are fewer constraints per character.

*Why are crosswords possible?*

If a language has no redundancy, then any letters written on a grid form a valid crossword. Crosswords, when read horizontally or vertically are written in the 'language' of 'word-English', the language consisting of strings of words from a dictionary separated by spaces.

Exercise 17.1:$^{B2}$ Estimate the capacity of word-English, in bits per character. [Hint: think of word-English as defining a constrained channel (chapter 16) and see exercise 8.6 (p.151).]

The fact that many crosswords can be made leads to a lower bound on the entropy of word-English.

For simplicity, we now model word-English by Wenglish, the language introduced in section 5.1 which consists of $W$ words all of length $L$. The entropy of such a language, per character, including inter-word spaces, is:

$$H_W \equiv \frac{\log_2 W}{L+1} \tag{17.3}$$

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 our simple model of Wenglish. We assume that Wenglish 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 $H_0 = \log_2 A = 4.7$ bits. If instead we use chapter 2's distbn then 4.2.

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 Wenglish 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

$$|T| = 2^{f_1 S H_0}. \tag{17.4}$$

The probability that one word of length $L$ is validly filled-in is

$$\beta = \frac{W}{2^{L H_0}}, \tag{17.5}$$

and the probability that the whole crossword, made of $f_w S$ words, is validly filled-in by a single typical in-filling is

$$\beta^{f_w S}, \tag{17.6}$$

So the log of the number of valid crosswords of size $S$ is estimated to be

$$\begin{aligned}\log \beta^{f_w S}|T| &= S\left[(f_l - f_w L)H_0 + f_w \log W\right] \log \beta^{f_w S}|T| \qquad (17.7)\\ &= S\left[(f_l - f_w L)H_0 + f_w (L+1) H_w\right] \qquad (17.8)\end{aligned}$$

which is an increasing function of $S$ only if

$$(f_1 - f_w L)H_0 + f_w(L+1)H_w > 0. \qquad (17.9)$$

So arbitrarily many crosswords can be made only if there's enough words in the Wenglish dictionary that

$$H_W > \frac{(f_w L - f_1)}{f_w(L+1)} H_0. \qquad (17.10)$$

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.

In a language with fewer, longer words, the possibility of making crosswords vanishes.

Thanks to Jack Wolf.

# 18

# *Cryptography and cryptanalysis: codes for information concealment* *

Suppose Alice wants to send Bob a message, and wants to send it securely: she wants to be sure that an eavesdropper, Eve, cannot read the message. If the message is to pass through a channel that Eve can listen in on, the message must first be *encrypted*. The art and science of keeping messages secure is *cryptography*; the art and science of breaking ciphertext (that is, inferring the plaintext) is *cryptanalysis*; cryptography and cryptanalysis together make up *cryptology*. This chapter will say almost nothing about cryptology, touching only on the connections with information theory. For an in-depth read about cryptography, Schneier's (1996) book is highly recommended. It is readable, clear, and entertaining. In this chapter, I have followed Schneier's terminology and stolen many of his explanations.

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.

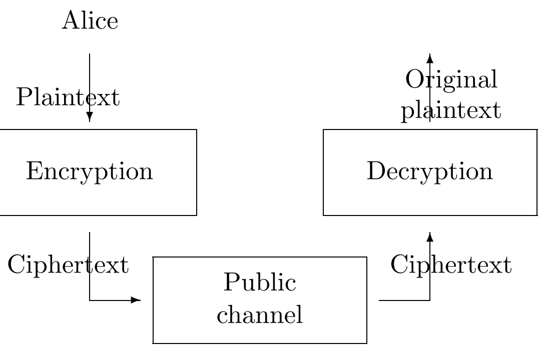


Figure 18.1. Encryption and Decryption

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.'

## 18.1 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.

# 19

# *Units of information content* *

The information content of an outcome, $x$, whose probability is $P(x)$, is defined to be

$$h(x) = \log \frac{1}{P(x)}. \tag{19.1}$$

The entropy of an ensemble is an average information content,

$$H(X) = \sum_x P(x) \log \frac{1}{P(x)}. \tag{19.2}$$

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,

$$\text{`log evidence for } \mathcal{H}_i\text{'} = \log P(D|\mathcal{H}_i), \tag{19.3}$$

or, in the case where just two hypotheses are being compared, we evaluate the 'log odds',

$$\log \frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)}, \tag{19.4}$$

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 hypothesis, then they are surprising to that hypothesis; if some other hypothesis is not so surprised by the data, then that hypothesis becomes more probable. 'Information content', 'surprise value', and log likelihood or log evidence are the same thing.

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.

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

| 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 19.1. Units of measurement of information content

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 Banburismus was played were called bans, after that town.

## 19.1 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 $8 \times 10^{12}$. 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 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 resulting correlations between the outputs of such pairs of machines provided a dribble of information-content from which Turing and his co-workers extracted their daily 129 decibans.

I now discuss this task of detecting that two messages came from machines with a common state sequence.

[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.

The hypotheses are the null hypothesis, $\mathcal{H}_0$, which states that the machines are in different states, and that the two plain messages are unrelated; and the 'match' hypothesis, $\mathcal{H}_1$, which says that the machines are in the same state, and that 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 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

$$\mathbf{x} = x_1x_2x_3\ldots = c_1(u_1)c_2(u_2)c_3(u_3)\ldots \tag{19.5}$$

and

$$\mathbf{y} = y_1y_2y_3\ldots = c'_1(v_1)c'_2(v_2)c'_3(v_3)\ldots, \tag{19.6}$$

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.

$$P(\mathbf{x},\mathbf{y}|\mathcal{H}_0) = \left(\frac{1}{A}\right)^{2T} \quad \text{for all } \mathbf{x},\mathbf{y} \text{ of length } T. \tag{19.7}$$

What about $\mathcal{H}_1$? This hypothesis asserts that a *single* time-varying permutation $c_t$ underlies both

$$\mathbf{x} = x_1x_2x_3\ldots = c_1(u_1)c_2(u_2)c_3(u_3)\ldots \tag{19.8}$$

and

$$\mathbf{y} = y_1y_2y_3\ldots = c_1(v_1)c_2(v_2)c_3(v_3)\ldots\,. \tag{19.9}$$

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 probability $P(\mathbf{x},\mathbf{y}|\mathcal{H}_1)$ would be the same as $P(\mathbf{x},\mathbf{y}|\mathcal{H}_0)$, and 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.

```
u  THEXCODEXBREAKERSXWEREXLOOKINGXFORXINSTANCESXWHEREX
v  TRIGRAMSXFORXTWOXORXMOREXMESSAGESXDIFFEREDXONLYXINX
   *.......*.........................*.............*
```

Figure 19.2. Two aligned pieces of English plaintext, $\mathbf{u}$ and $\mathbf{v}$, 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.

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

$$\sum_{u_t,v_t} P(u_t)P(v_t)\delta(u_t = v_t) = \sum_i p_i^2 \equiv m. \tag{19.10}$$

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,

$$P(x_t, y_t|\mathcal{H}_1) = \begin{cases} \frac{m}{A} & \text{if } x_t = y_t \\ \frac{(1-m)}{A(A-1)} & \text{for } x_t \neq y_t \end{cases} \tag{19.11}$$

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

$$\log \frac{P(\mathbf{x},\mathbf{y}|\mathcal{H}_1)}{P(\mathbf{x},\mathbf{y}|\mathcal{H}_0)} = M \log \frac{m/A}{1/A^2} + N \log \frac{(1-m)/A(A-1)}{1/A^2} \tag{19.12}$$

$$= M \log mA + N \log \frac{(1-m)A}{A-1} \tag{19.13}$$

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 (19.13), 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. Typically, roughly 400 characters need to be inspected in order to have a weight of evidence greater than a hundred to one (2 bans) in favour of one hypothesis or the other.

Not only do two English plaintexts have more matches than two random strings. Because consecutive characters in English are not independent, the bigram and trigram statistics of English are non-uniform and the matches tend to occur in bursts of consecutive matches. [The same observations also apply to German, the plaintext language used in the Enigma messages.] 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. Positive results were passed on to automated and human-powered codebreakers. According to Good, the longest false-positive that arose in this work was a string of 8 consecutive matches.

# 20

# *Why have sex? Information acquisition and evolution*

Exercise 20.1:$^{C5}$ **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. The bits of the teaching signal are highly redundant, because, throughout 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 20.2:$^{C3}$ 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 20.3:$^{C2}$ While the genome acquires information through natural selection at a rate of a few bits per generation (exercise 20.1), 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.

Solution to exercise 20.1 (p.298): 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 $8 \times 10^{14}$ 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 – $6 \times 10^9$ 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 $7 \times 10^8$ 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[1] 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 $4 \times 10^7$ or $4 \times 10^9$ bits of information from evolution.

Solution to exercise 20.2 (p.298): See section B.8 for some actual estimates.

> At what rate, in bits per generation, can the blind watchmaker cram information into a species by natural selection? And what is the maximum mutation rate that a species can withstand? We study a simple model of a reproducing population of $N$ individuals with a genome of size $G$ bits: fitness is a strictly additive trait subjected to directional selection; variation is produced by mutation or by recombination and truncation selection selects the $N$ fittest children at each generation to be the parents of the next. We find striking differences between populations that have recombination and populations that do not. If variation is produced by mutation alone, then the entire population gains up to roughly 1 bit per generation. If variation is created by recombination, the population can gain $O(\sqrt{G})$ bits per generation. Furthermore, recombination raises the maximum mutation rate that can be tolerated by a factor of order $\sqrt{G}$. This model explains the prevalence of sex in evolution and shows why sex persists in species with large genomes, even when they have reached evolutionary stasis.

[1] `http://www.inference.phy.cam.ac.uk/mackay/abstracts/gene.html`

## 20.1 The Model

It has been suggested that the deleterious mutation rate in hominids is so high that sex is essential for such species to persist (Eyre-Walker and Keightley 1999). In order to quantify the benefit of sex, we consider a simple model of an evolving population of $N$ individuals having a genome of size $G$. This 'threshold selection' model has previously been used in discussions of the rate of evolution by Maynard Smith (1968). We choose a crude model because it readily yields simple but striking results.

The genotype of each individual is a vector $\mathbf{x}$ of $G$ bits, each having a good state $x_g = 1$ and a bad state $x_g = 0$. The fitness $F(\mathbf{x})$ of an individual is simply the sum of her bits:

$$F(\mathbf{x}) = \sum_{g=1}^{G} x_g. \tag{20.1}$$

The bits in the genome could be considered to correspond either to genes that have good alleles ($x_g = 1$) and bad alleles ($x_g = 0$), or to the nucleotides of a genome. We will concentrate on the latter interpretation. The essential property of fitness that we are assuming is that it is locally a roughly linear function of the genome, that is, that there are many possible changes one could make to the genome, each of which has a small effect on fitness, and that these effects combine approximately linearly.

We define the normalized fitness $f(\mathbf{x}) \equiv F(\mathbf{x})/G$.

We consider evolution by natural selection under two models of variation.

**Variation by mutation:** The model assumes discrete generations. At each generation, $t$, every individual produces two children. The children's genotypes differ from the parent's by random mutations. Natural selection selects the fittest $N$ progeny in the child population to reproduce, and a new generation starts.

[The selection of the fittest $N$ individuals at each generation is known as truncation selection.]

The simplest model of mutations is that the child's bits $\{x_g\}$ are independent. Each bit has a small probability of being flipped, which, thinking of the bits as corresponding roughly to nucleotides, is taken to be a constant $m$, independent of $x_g$. [If alternatively we thought of the bits as corresponding to genes, then we would model the probability of the discovery of a good gene, $P(x_g{=}0 \rightarrow x_g{=}1)$, as being a smaller number than the probability of a deleterious mutation in a good gene, $P(x_g{=}1 \rightarrow x_g{=}0)$.]

**Variation by recombination (or crossover, or sex):** Our organisms are haploid, not diploid. They enjoy sex by recombination. The $N$ individuals in the population are married into $M = N/2$ couples, at random, and each couple has $C$ children — with $C = 4$ children being our standard assumption, so as to have the population double and halve every generation, as before. The $C$ children's genotypes are independent given the parents'. Each child obtains its genotype $\mathbf{z}$ by random crossover of its parents' genotypes, $\mathbf{x}$ and $\mathbf{y}$. The simplest model of recombination has no linkage, so that:

$$z_g = \begin{cases} x_g & \text{with probability } 1/2 \\ y_g & \text{with probability } 1/2 \end{cases}. \tag{20.2}$$

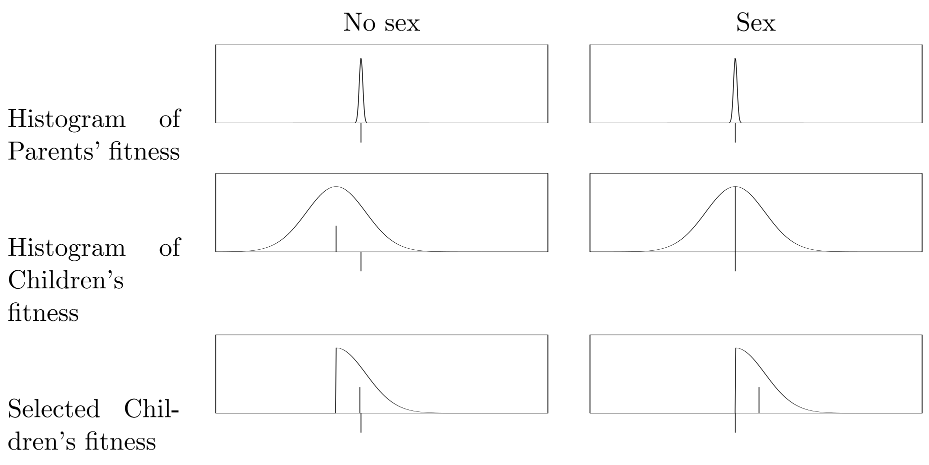


Figure 20.1. Why sex is better than parthenogenesis. If mutations are used to create variation among children, then it is unavoidable that the average fitness of the children is lower than the parents' fitness; the greater the variation, the greater the deficit. Selection bumps up the mean fitness again. In contrast, recombination produces variation without a decrease in average fitness. The typical amount of variation scales as $\sqrt{G}$, where $G$ is the genome size, so after selection, the average fitness rises by $O(\sqrt{G})$.

Once the $MC$ progeny have been born, the parents pass away, the fittest $N$ progeny are selected by natural selection, and a new generation starts.

We now study these two models of variation in detail.

## 20.2 Rate of increase of fitness

*Theory of mutations*

We assume that the genotype of an individual with normalized fitness $f = F/G$ is subjected to mutations that flip bits with probability $m$. We first show that if the average normalized fitness $f$ of the population is greater than 1/2, then the optimal mutation rate is small, and the rate of acquisition of information is of order one bit per generation.

Since it is easy to achieve a normalized fitness of $f = 1/2$ by simple mutation, we'll assume $f > 1/2$ and work in terms of the excess normalized fitness $\delta f \equiv f - 1/2$. If an individual with excess normalized fitness $\delta f$ has a child and the mutation rate $m$ is small, the probability distribution of the excess normalized fitness of the child has mean

$$\overline{\delta f}_{\text{child}} = (1 - 2m)\delta f \tag{20.3}$$

and variance

$$\frac{m(1-m)}{G} \simeq \frac{m}{G}. \tag{20.4}$$

If the population of parents has mean $\delta f(t)$ and variance $\sigma^2(t) \equiv \beta \frac{m}{G}$, then the child population, before selection, will have mean $(1 - 2m)\delta f(t)$ and variance

$(1+\beta)\frac{m}{G}$. Natural selection chooses the upper half of this distribution, so the mean fitness and variance of fitness at the next generation are given by

$$\delta f(t{+}1) = (1-2m)\delta f(t) + \alpha\sqrt{(1+\beta)}\sqrt{\frac{m}{G}}, \tag{20.5}$$

$$\sigma^2(t{+}1) = \gamma(1+\beta)\frac{m}{G}, \tag{20.6}$$

where $\alpha$ is the mean deviation from the mean, measured in standard deviations, and $\gamma$ is the factor by which the child distribution's variance is reduced by selection. The numbers $\alpha$ and $\gamma$ are of order 1. For the case of a Gaussian distribution, $\alpha = \sqrt{2/\pi} \simeq 0.8$ and $\gamma = (1-2/\pi) \simeq 0.36$. If we assume that the variance is in dynamic equilibrium, i.e., $\sigma^2(t{+}1) \simeq \sigma^2(t)$, then

$$\gamma(1+\beta) = \beta, \text{ so } (1+\beta) = \frac{1}{1-\gamma}, \tag{20.7}$$

and the factor $\alpha\sqrt{(1+\beta)}$ in equation (20.5) is equal to 1, if we take the results for the Gaussian distribution, an approximation that becomes poorest when the discreteness of fitness becomes important, i.e., for small $m$. The rate of increase of normalized fitness is thus:

$$\frac{df}{dt} \simeq -2m\,\delta f + \sqrt{\frac{m}{G}}, \tag{20.8}$$

which, assuming $G(\delta f)^2 \gg 1$, is maximized for

$$m_{\rm opt} = \frac{1}{16G(\delta f)^2}, \tag{20.9}$$

at which point,

$$\left(\frac{df}{dt}\right)_{\rm opt} = \frac{1}{8G(\delta f)}, \tag{20.10}$$

which means that the rate of increase of fitness $F = fG$ is

$$\frac{dF}{dt} = \frac{1}{8(\delta f)} \text{ per generation.} \tag{20.11}$$

For a population with low fitness ($\delta f < 0.125$), the rate of increase of fitness may exceed 1 unit per generation. Indeed, if $\delta f \lesssim 1/\sqrt{G}$, the rate of increase, if $m = 1/2$, is of order $\sqrt{G}$; this initial spurt can only last of order $\sqrt{G}$ generations. For $\delta f > 0.125$, the rate of increase of fitness is smaller than one per generation. As the fitness approaches $G$, the optimal mutation rate tends to $m = 1/(4G)$, so that an average of 1/4 bits are flipped per genotype, and the rate of increase of fitness is also equal to 1/4; information is gained at a rate of about 0.5 bits per generation. It takes about $2G$ generations for the genotypes of all individuals in the population to attain perfection.

For fixed $m$, the fitness is given by

$$\delta f(t) = \frac{1}{2\sqrt{mG}}(1 - ce^{-2mt}), \tag{20.12}$$

subject to the constraint $\delta f(t) \leq 1/2$, where $c$ is a constant of integration, equal to 1 if $f(0) = 1/2$. If the mean number of bits flipped per genotype,

$mG$, exceeds 1, then the fitness $F$ approaches an equilibrium value $F_{\rm eqm} = (1/2 + 1/(2\sqrt{mG}))G$.

If $m$ is tuned to the optimal fitness–dependent value, $m_{\rm opt}$ (20.9) then the fitness is given, assuming $\delta f(0) = 0$, by

$$\delta f(t) = \frac{t^{1/2}}{2\sqrt{G}}, \tag{20.13}$$

which hits $\delta f = 1/2$ at $t = G$.

This theory is somewhat inaccurate in that the true probability distribution of fitness is non–Gaussian, asymmetrical, and quantized to integer values. All the same, the predictions of the theory are not grossly at variance with the results of simulations in section 20.2.

### *Theory of sex*

The analysis of the sexual population becomes tractable with two approximations: first, we assume that the genepool mixes sufficiently rapidly that correlations between genes can be neglected; second, we assume *homogeneity*, i.e., that the fraction $f_g$ of bits $g$ that are in the good state is the same, $f(t)$, for all $g$.

Given these assumptions, if two parents of fitness $F = fG$ mate, the probability distribution of their children's fitness has mean equal to the parents' fitness, $F$; the variation produced by sex does not reduce the average fitness. The standard deviation of the fitness of the children scales as $\sqrt{Gf(1-f)}$. Since, after selection, the increase in fitness increases is proportional to this standard deviation, *the fitness increase per generation scales as* $\sqrt{G}$. As shown in appendix 20.7, the mean fitness $\bar{F} = fG$ evolves in accordance with the differential equation:

$$\frac{df}{dt} \simeq \frac{\eta}{\sqrt{G}} \sqrt{f(t)(1-f(t))}, \tag{20.14}$$

where $\eta \equiv \sqrt{2/(\pi+2)}$. The solution of this equation is

$$f(t) = \frac{1}{2}\left[1 + \sin\left(\frac{\eta}{\sqrt{G}}(t+c)\right)\right], \quad \text{for } t+c \in \left(-\tfrac{\pi}{2}\sqrt{G}/\eta, \tfrac{\pi}{2}\sqrt{G}/\eta\right), \tag{20.15}$$

### *Simulations*

Figure 20.2(a) shows the fitness of a population of $N = 1000$ individuals with a genome size of $G = 1000$ starting from a random initial state with normalized fitness 0.5. It also shows the theoretical curve $f(t)G$ using $f(t)$ derived for the infinite homogeneous population, equation (20.33), which fits remarkably well.

In contrast, figures 20.2(b1,2) show the evolving fitness when variation is produced by mutation at rates $0.25/G$ and $6/G$ respectively. Note the difference in the horizontal scales.

### *Small populations*

For small enough $N$, we expect that, whilst the average fitness of the population increases, some unlucky bits may become frozen into the bad state.

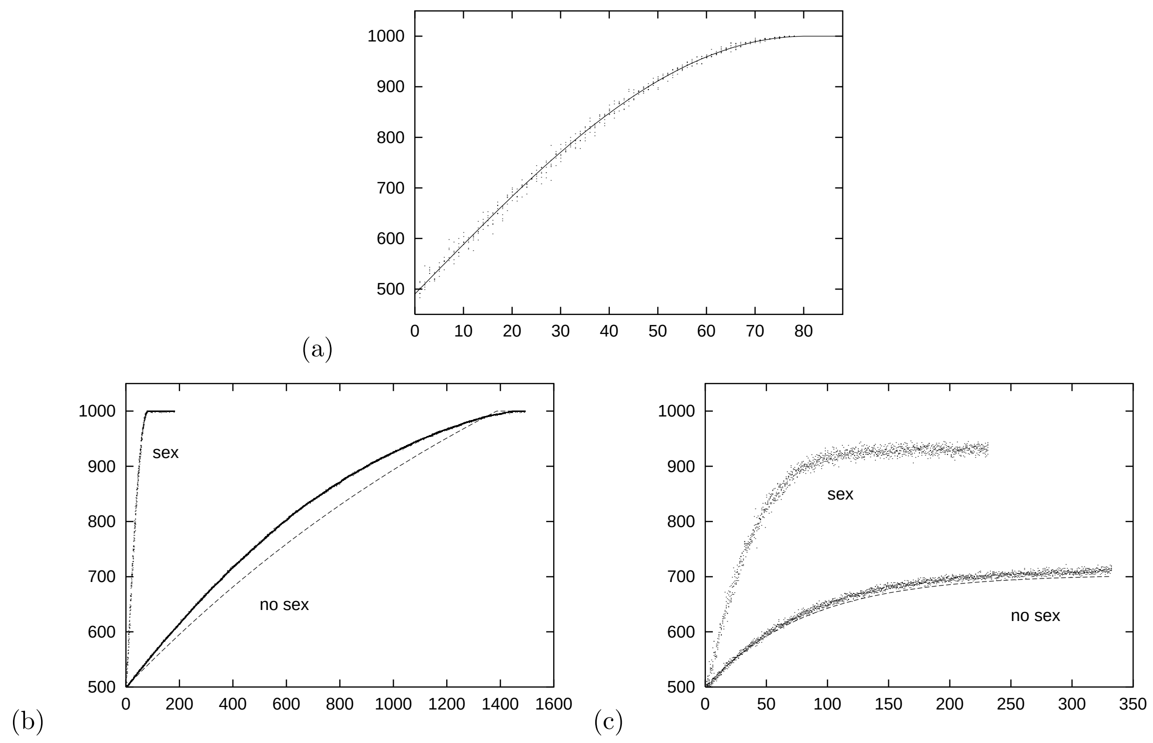


Figure 20.2. Fitness as a function of time. The genome size is $G = 1000$. The dots show the fitness of six randomly selected individuals from the birth population at each generation. The initial population of $N = 1000$ had random generated genomes with $f(0) = 0.5$ (exactly). (a) Variation produced by sex alone. Line shows theoretical curve (20.33) for infinite homogeneous population.

(b) Variation produced by mutation, with and without sex, when the mutation rate is $mG = 0.25$ bits per genome. The dashed line shows the curve equation (20.12).

(c) Variation produced by mutation, with and without sex, when the mutation rate is $mG = 6$ bits per genome.

(These bad genes are sometimes known as hitchhikers.) The homogeneity assumption breaks down. Eventually, all individuals have identical genotypes that are mainly 1–bits, but have some 0–bits too. The smaller the population, the greater the number of frozen 0–bits is expected to be.

We find experimentally that the theory based on assuming homogeneity only fits poorly if the population size $N$ is smaller than $\sim \sqrt{G}$. If $N$ is significantly smaller than $\sqrt{G}$, information cannot possibly be acquired at a rate as big as $\sqrt{G}$, since the information content of the blind watchmaker's decisions cannot be any greater than $2N$ bits per generation, this being the number of bits required to specify which of the $2N$ children get to reproduce. Baum *et al.* (1995), analyzing a similar model, show that the population size $N$ should be about $\sqrt{G}(\log G)^2$ to make hitchhikers unlikely to arise.

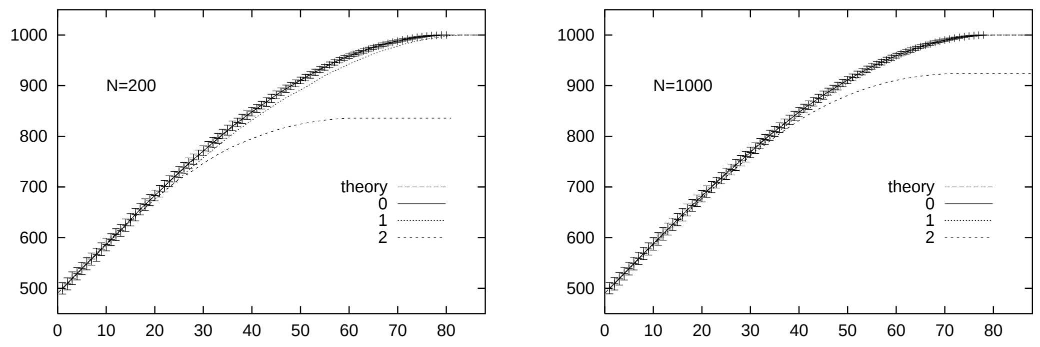


Figure 20.3. Dependence of results on crossover mechanism. Each graph is the mean fitness of the birth population. For method 0, the standard deviation is also indicated by error bars.

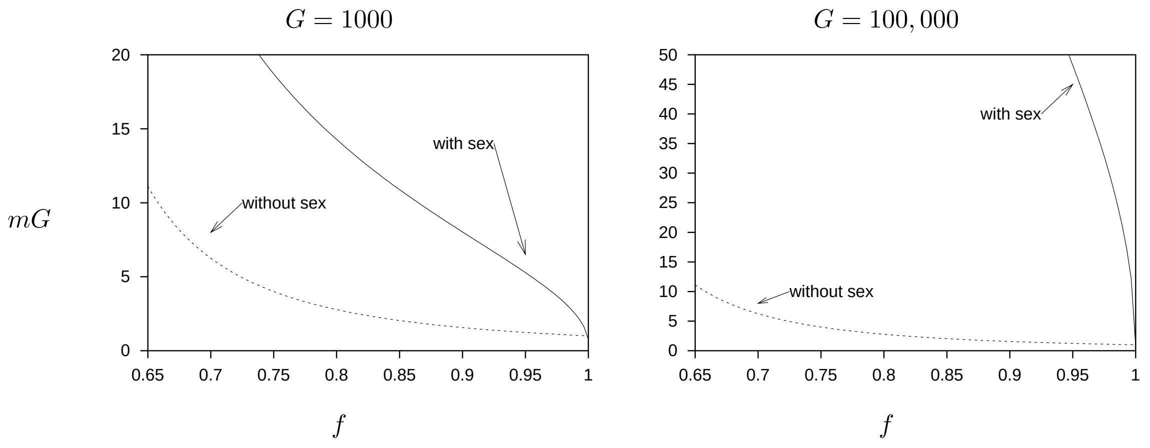


Figure 20.4. Maximal tolerable mutation rate, shown as number of errors per genome ($mG$), versus normalized fitness $f = F/G$. Independent of genome size, a parthenogenetic species can only tolerate of order 1 error per genome per generation; a species that uses recombination can tolerate far greater mutation rates.

## 20.3 The maximal tolerable mutation rate

*This section needs checking over, to confirm the details of the factors of $\eta$, etc.*

What if we combine the two models of variation? What is the maximum mutation rate that can be tolerated by a species that has sex?

The rate of increase of fitness is given by

$$\frac{df}{dt} \simeq -2m\,\delta f + \eta\sqrt{2}\sqrt{\frac{m + f(1-f)/2}{G}}, \tag{20.16}$$

which is positive if

$$2m\,\delta f < \eta\sqrt{2}\sqrt{\frac{m + f(1-f)/2}{G}}. \tag{20.17}$$

Replacing $\delta f$ by its largest value, 1/2, and omitting the $m$ on the right hand side, the rate of increase of fitness is positive, for a given $f$, if the mutation rate satisfies

$$m < \eta\sqrt{\frac{f(1-f)}{G}}. \tag{20.18}$$

Let us compare this rate with the result in the absence of sex, which, from equation (20.8), is that the maximum tolerable mutation rate is

$$m < \frac{1}{G}\frac{1}{(2\,\delta f)^2}. \tag{20.19}$$

The tolerable mutation rate with sex is of order $\sqrt{G}$ times greater than that without sex!

A parthenogenetic (non–sexual) species could try to wriggle out of this bound on its mutuation rate by increasing its litter sizes. But if mutation flips on average $mG$ bits, the probability that no bits are flipped in one genome is roughly $e^{-mG}$, so a mother needs to have roughly $e^{mG}$ offspring in order to have a good chance of having one child with the same fitness as her. The litter size of a non–sexual species thus has to be exponential in $mG$, if $mG$ is bigger than 1, if the species is to persist.

So the maximum tolerable mutation rate is pinned close to $1/G$, for a non–sexual species, whereas it is a larger number of order $1/\sqrt{G}$, for a species with recombination.[2]

[2]Turning these results around, we can predict the largest possible genome size for a given fixed mutation rate, $m$. For a parthenogenetic species, the largest genome size is of order $1/m$, and for a sexual species, $1/m^2$. Taking the figure $m = 10^{-8}$ as the mutation rate per nucleotide per generation (Eyre-Walker and Keightley 1999), and allowing for a maximum brood size of 20,000 (that is, $mG \simeq 10$), we predict that all species with more than $10^9$ coding nucleotides make at least occasional use of recombination. If the brood size is 12, then this number falls to $2.5 \times 10^8$. This crude calculation used mutation–rate figures from humans and apes, so let's do it again for prokaryots. If the mutation rate is $m = 10^{-7}$ (need a reference) per base pair per generation for a prokaryot, and $mG \leq 1$, the maximum genome size is about $10^7$ nucleotides. What then, of the Bdelloid rotifers, the fresh–water invertebrates that appear to be entirely parthenogenetic? If the Bdelloid genome is similar in size to *C. Elegans* — $6 \times 10^7$ coding basepairs, let's say — then this creature falls below the rough threshold, $2.5 \times 10^8$, derived above. My guess is that the organism evolved rapidly using sex, until it fitted its niche perfectly; thereafter, its genome size was small enough that it had no need for sex to keep itself healthy. I wait to learn the genome size.

*Dependence on the model of crossover*

The original crossover method (method 0) set each child bit at random either to the corresponding bit of parent 1 or parent 2. We have also simulated two other models of crossover, both of which model the one–dimensional local topology of a chromosome. In method 1, there is a crossover probability $\chi$ before each bit. [The special case $\chi = 0.5$ gives method 0.] In method 2, crossovers only occur at hotspots located every $d$ bits along the chromosome; the probability of crossover at a hotspot is here set to 0.5.

Figure 20.3 shows results for the three methods for various population sizes. For method 1, $\chi = 0.05$, and for method 2, $d = 10$, so these two methods have the same mean number of crossovers. We find that, for large populations, the results for methods 1 and 2 are similar to those for method 0. However, the population size below which freezing of bad genes is seen is larger for method 1 and larger still for method 2.

## 20.4 Relationship between fitness increase and information acquisition

For this simple model it is possible to relate increasing fitness to information acquisition.

If the bits are set at random, the fitness is roughly $F = G/2$. If evolution leads to a population in which all individuals have the maximum fitness $F = G$, then $G$ bits of information have been acquired by the species, namely for each bit $x_g$, the species has figured out which of the two states is the better.

There are several ways to define the information acquired at an intermediate fitness. The Shannon entropy of the posterior probability distribution of the target corner of the hypercube collapses from $G$ bits to close to zero in one generation, if the population is large, because one simply needs to take the majority vote for each bit to find its best value. This information is only accessible to a global observer, however.

An alternative measure of information acquired, suggested by Kimura (1961), is the amount of selection (measured in bits) required to select the optimal state from the gene pool. Let a fraction $f_g$ of the population have $x_g = 1$. Because $\log_2(1/f)$ is the information required to find a black ball in an urn containing black and white balls in the ratio $f : 1-f$, we define the information acquired to be

$$I = \sum_g \log_2 \frac{f_g}{1/2} \text{bits.} \tag{20.20}$$

If all the fractions $f_g$ are equal to $F/G$, then

$$I = G \log_2 \frac{2F}{G}, \tag{20.21}$$

which is well approximated by

$$\tilde{I} \equiv 2(F - G/2). \tag{20.22}$$

The rate of information acquisition is thus roughly two times the rate of increase of fitness in the population.

## 20.5 Discussion

These results quantify the well known argument for why species reproduce by sex with recombination, namely that recombination allows useful mutations to spread more rapidly through the species and allows deleterious mutations to be more rapidly cleared from the population (Maynard Smith 1978; Felsenstein 1985; Maynard Smith 1988; Maynard Smith and Száthmary 1995). A population that reproduces by recombination can acquire information from natural selection at a rate of order $\sqrt{G}$ times faster than a parthenogenetic population, and it can tolerate a mutation rate that is of order $\sqrt{G}$ times greater. For genomes of size $G \simeq 10^8$ coding nucleotides, this factor of $\sqrt{G}$ is substantial.

This enormous advantage conferred by sex has been noted before by Kondrashov (1988), but this meme, which Kondrashov calls 'the deterministic mutation hypothesis', does not seem to have diffused throughout the evolutionary research community, as there are still numerous papers in which the prevalence of sex is viewed as a mystery to be explained by elaborate mechanisms (Kelling and Rand 1995). Ridley (1993) dismisses Kondrashov's mechanism as being too slow–acting for sex to survive. However, in the next section we show that, if the mutation rate is high or the species is still climbing the fitness curve, then it only takes a few generations for asexual mutants to become relatively unfit and die out, in the present model. This model is compatible with the idea that parasites play a role in maintaining sex, by rapidly redefining the effective fitness function. But elaborate parasite mechanisms are not essential.

Maynard Smith and Száthmary (1995) assert that deleterious mutations are eliminated more effectively under recombination only if the effects of mutations have a synergistic rather than an antagonistic effect on fitness; they then argue that mutations in metabolic enzymes are likely to act antagonistically in unicellular organisms and synergistically in higher organisms, thus accounting for the general lack of sex in the former and its prevalence in the latter. I think this explanation is overly complicated. Simple organisms can do without sex because they have small genomes.

Of course, this conclusion depends on the fitness function that we assume, and on our model of selection. Is it reasonable to model fitness, to first order, as a *sum* of independent terms? Maynard Smith (1968) argues that it is: the more good genes you have, the higher you come in the pecking order, for example. The directional selection model has been used extensively in theoretical population genetic studies (Bulmer 1985). We might expect real fitness functions to involve interactions, in which case crossover might reduce the average fitness. However, since recombination gives the biggest advantage to species whose fitness functions are additive, we might predict that evolution will have favoured species that used a representation of the genome that corresponds to a fitness function that has only weak interactions. And even if there are interactions between genes, it seems plausible that the fitness would still involve a sum of such interacting terms, with the number of terms being some fraction of $G$. Fitness functions that are sums of interacting terms are investigated in section 20.6. Furthermore, if the fitness function were a highly nonlinear function of the genotype, it could be made more smooth and locally linear by the Baldwin effect. The Baldwin effect (Baldwin 1896; Hinton and Nowlan 1987) has been widely studied as a mechanism whereby *learning* guides evolution, and it could also act at the level of transcription and translation. Consider the evolution of a peptide sequence for a new purpose, assuming the effectiveness of the peptide is highly nonlinear function of the sequence, perhaps having a small island of good sequences surrounded by an ocean of equally bad sequences. In an organism whose transcription and translation machinery is flawless, the fitness will be an equally nonlinear function of the DNA sequence and evolution will wander around the ocean making progress towards the island only by a random walk. In contrast, an organism having the same DNA sequence, but whose DNA–to–RNA transcription or RNA–to–protein translation is 'faulty', will occasionally, by mistranslation or mistranscription, produce a working enzyme, and will do so with greater probability if its DNA sequence is close to a good sequence. One cell might produce 1000 proteins from the one mRNA sequence, of which 999 have no enzymatic effect, and one does. The one working catalyst will be enough for that cell to have an increased fitness relative to rivals whose DNA sequence is further from the island of good sequences. For this reason we conjecture that, at least early in evolution, and perhaps still now, the genetic code was not implemented perfectly but was implemented noisily, with some codons coding for a distribution of possible amino acids. This noisy code could even be switched on and off from cell to cell in an organism by having multiple aminoacyl–tRNA synthetases, some more reliable than others.

Is it reasonable to use truncation selection, which kills off all children with fitness below a threshold? (In using this selection model, we are not implying that those children were unfit to reproduce — merely that they were the losers in the competition for limited resources.) Other selection functions could be used; any monotonic selection function, concave or convex, is expected to give similar results, since the advantage of recombination is that it gives a variance in fitness proportional to $G$, without decreasing the mean fitness. The results obtained with 'ranking selection', in which pairs of random individuals are compared and the fitter is selected, are equivalent to the results derived here for threshold selection, except that the rate of evolution is smaller by a factor of $\sqrt{2}$.

The rate at which information can be acquired by natural selection was addressed by Kimura (1961); using two different arguments, he asserts that the maximum rate of information acquisition is 1 bit per generation (assuming that a generation is the time taken for the population to double). That the 'speed limit for evolution' is about one bit per generation is also 'proved' by Worden (1995), who goes beyond Kimura by showing that this result holds whether or not the species has sex. This contrasting conclusion is reached because of Worden's choice of model. Instead of defining an individual's fitness as a relative quantity, involving competition with other individuals, his model assumes that one's genotype determines one's probability of having children absolutely; competition is only embodied by an overall rescaling of the population at each generation, as in Fisher's (1930) 'fundamental theorem'.

Whilst our model assumed that the bits of the genome do not interact, ignored the fact that the information is represented redundantly, assumed that there is a direct relationship between phenotypic fitness and the genotype, and assumed that the crossover probability in recombination is high, we believe these qualitative results would still hold if more complex models of fitness and crossover were used: the relative benefit of sex will still scale as $\sqrt{G}$. Only in

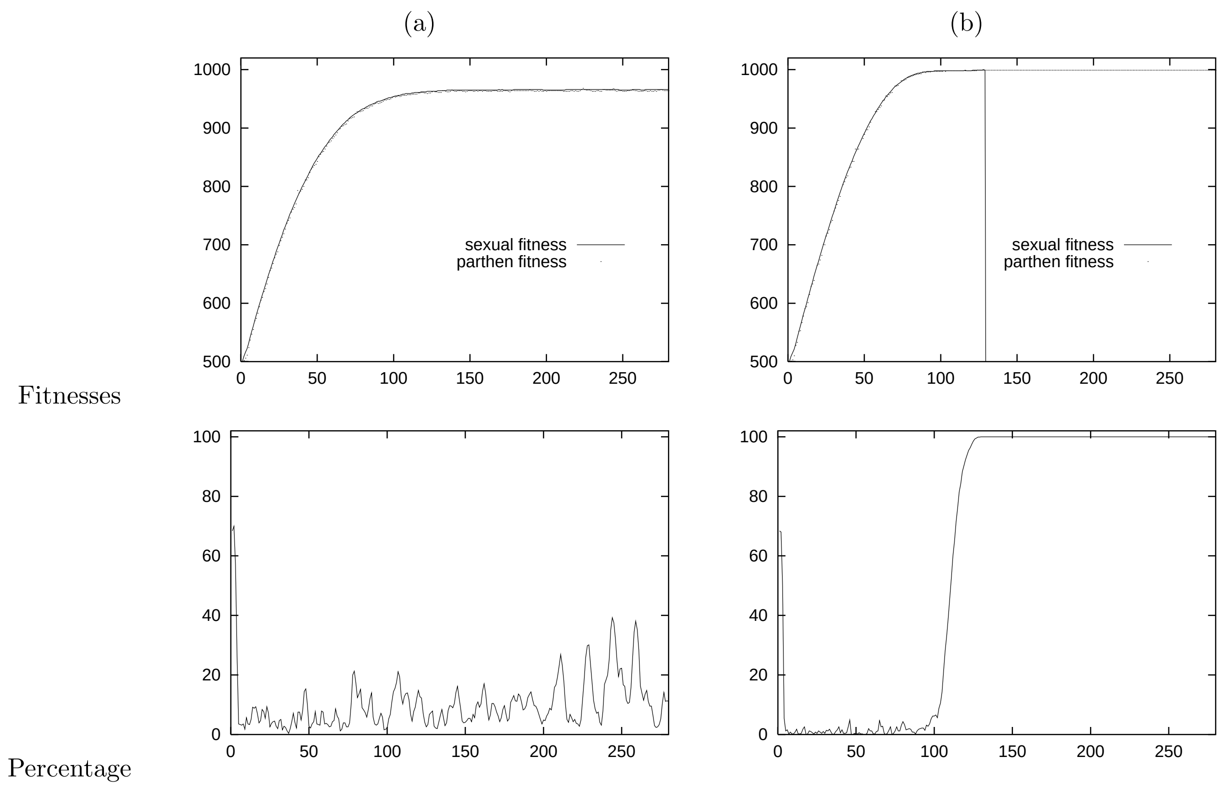


Figure 20.5. Results when there is a gene for parthenogenesis, and no interbreeding, *and single mothers produce as many children as sexual couples.* $G = 1000$, $N = 1000$. (a) $mG = 4$; (b) $mG = 1$. Vertical axis shows both fitness and percentage of the population that is parthenogenetic.

small, in–bred populations are the benefits of sex expected to be diminished.

## *Stability of a gene for sex or parthenogenesis*

I modified the model so that one of the $G$ bits in the genome determines whether an individual prefers to reproduce parthenogenetically ($x = 1$) or sexually ($x = 0$). The big advantage of parthenogenesis, from the point of view of the individual, is that one is able to pass on 100% of one's genome to one's children, instead of only 50%. The results depend on the number of children had by a single parthenogenetic mother, $K_{\rm p}$ and the number of children born by a sexual couple, $K_{\rm s}$. Both ($K_{\rm p} = 2$, $K_{\rm s} = 4$) and ($K_{\rm p} = 4$, $K_{\rm s} = 4$) are reasonable models. The former ($K_{\rm p} = 2$, $K_{\rm s} = 4$) would seem most appropriate in the case of unicellular organisms, where the cytoplasm of both parents goes into the children. The latter ($K_{\rm p} = 4$, $K_{\rm s} = 4$) is appropriate if the children are solely nurtured by one of the parents, so single mothers have just as many offspring as a sexual pair. I concentrate on the latter model, since it gives the greatest advantage to the parthenogens, who are supposedly expected to outbreed the sexual community. Because parthenogens have four children per generation, the maximum tolerable mutation rate for them is

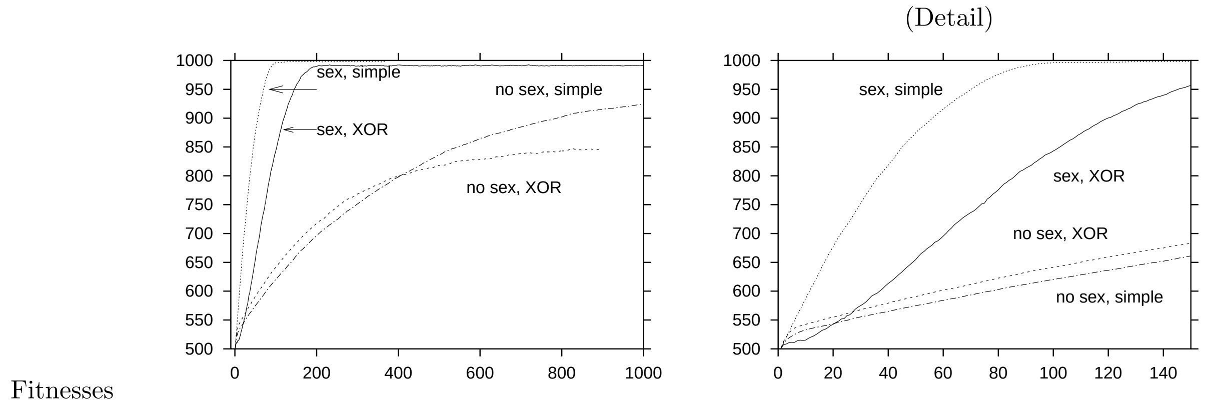


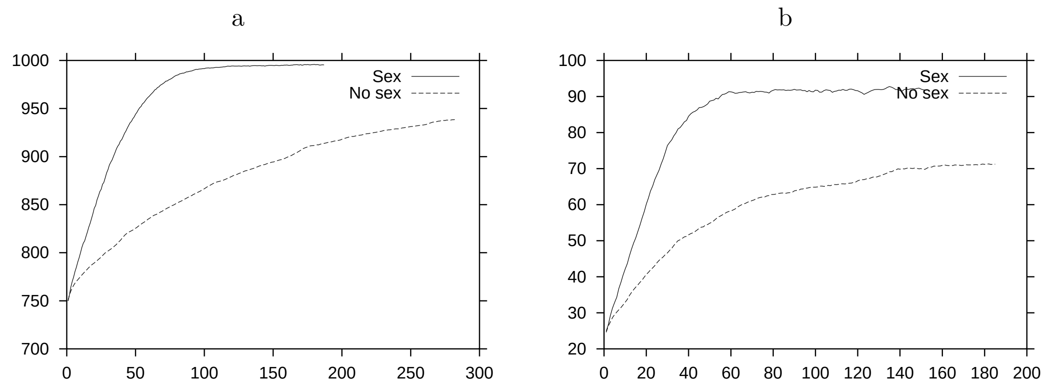
Figure 20.6. Results for a fitness function with interactions. The graphs show the fitnesses as a function of time for four separate populations with and without sex, and with an interacting and a non–interacting fitness function. These four populations did not compete with each other in these simulations. The mutation rate was $mG = 2$.

twice the expression (20.19) derived before for $K_{\mathrm{p}} = 2$. If the fitness is large, the maximum tolerable rate is $mG \simeq 2$.

Initially the genomes are set randomly with $F = G/2$, with half of the population having the gene for parthenogenesis. Figure 20.5 shows the outcome. During the 'learning' phase of evolution, in which the fitness is increasing rapidly, pockets of parthenogens appear briefly, but then disappear within a couple of generations as their sexual cousins overtake them in fitness and leave them behind. Once the population reaches its top fitness, however, the parthenogens can take over, if the mutation rate is sufficiently low ($mG = 1$). In these simulations, sex does not tend to reappear once the parthenogens have taken over, because a small sexual community, having size $N_{\mathrm{sexual}} < \sqrt{G}$, will be in–bred and will not have the advantage discussed in the rest of this paper.

In the presence of a higher mutation rate ($mG = 4$), however, the parthonegens never take over. The breadth of the sexual population's fitness is of order $\sqrt{G}$, so a mutant parthenogenetic colony arising with slightly above–average fitness will last for about $\sqrt{G}/(mG) = 1/(m\sqrt{G})$ generations before its fitness falls below that of its sexual cousins. As long as the population size is sufficiently large for some sexual individuals to survive for this time, sex will not die out.

In a sufficiently unstable environment, where the fitness function is continually changing, the parthenogens will always lag behind the sexual community. These results are consistent with the argument of Haldane and Hamilton that sex is helpful in an arms race with parasites. The parasites define an effective fitness function which changes with time, and a sexual population will always ascend the current fitness function more rapidly.



Fitnesses

Figure 20.7. (a) Results for a fitness function with 'OR' interactions. The graphs show the fitnesses as a function of time for populations with and without sex, where the fitness function is the 'or' of 1000 pairs of bits. $N = 100$, $G = 2000$.

(b) Results for 'AND', with $mG = 2$, $N = 100$, $G = 200$.

## 20.6 Interactions

What happens if we modify the fitness function such that there are interactions between bits in the genome? A simple such fitness function is

$$F_{\text{XOR}}(\mathbf{x}) = \sum_{g=1}^{G/2} x_g \text{XOR} x_{g+G/2}, \tag{20.23}$$

which can be contrasted with

$$F_{\text{noninteracting}}(\mathbf{x}) = \sum_{g=1}^{G/2} x_g, \tag{20.24}$$

which has the same statistics, that is, any value of fitness can be realised by just the same number of vectors $\mathbf{x}$ for either function.

Figure 20.6 shows the four outcomes for sexual and asexual populations evolving under these two fitness functions with $G = 2000$ and $N = 100$; the mutation rate is set to $mG = 2$. These experiments allow us to study two questions. First, if there are interactions among bits, does sex still have an advantage over sexlessness? Second, if there are two competing species, one that has a fitness function that does not have interactions, and one that does, which species will dominate?

The answer to the first question is shown by the heavy solid and dashed lines: early on in evolution, sex has a disadvantage, because crossover splits up linked bits. However, the sexual population eventually becomes fitter than the asexual one because it is better able to eliminate mutations.

The answer to the second question is that the species with the non–interacting fitness function becomes fitter faster. Thus there is an evolutionary pressure in favour of genomes that are organized to have a simple fitness function.

*Acknowledgements*

This work was stimulated by a talk given by John Maynard Smith. I thank Sanjoy Mahajan, Robert MacKay, Ken Miller, Ewan Birney, Ted Briscoe, Oscar Gaggiotti, Jonathan Swinton, Geoff Hinton and Graeme Mitchison for helpful discussions.

After completion of this work my attention was drawn to a similar paper by Alex Rogers and Adam Prügel–Bennett, and to publications on similar models of an evolving sexual population in the Genetic Algorithms community (Mühlenbein and Schlierkamp-Voosen 1993; Baum *et al.* 1995); these papers, which note that fitness increases at a rate proportional to $\sqrt{G}$, concern the relative advantages of various genetic algorithms, all of which use sex.

## 20.7 appendix

*Theory of sex*

How does $f(t{+}1)$ depend on $f(t)$? Let's first assume the two parents of a child both have exactly $f(t)G$ good bits, and, by our homogeneity assumption, that those bits are independent random subsets of the $G$ bits. The number of bits that are good in both parents is roughly $f(t)^2G$, and the number that are good in one parent only is roughly $2f(t)(1-f(t))G$, so the fitness of the child will be $f(t)^2G$ plus the sum of $2f(t)(1-f(t))G$ fair coin flips, which has a binomial distribution of mean $f(t)(1-f(t))G$ and variance $\frac{1}{2}f(t)(1-f(t))G$. The fitness of a child is thus roughly distributed as

$$F_{\text{child}} \sim \text{Normal}\left(\text{mean}{=}f(t)G, \text{variance}{=}\frac{1}{2}f(t)(1-f(t))G\right). \tag{20.25}$$

The important property of this distribution, contrasted with the distribution under mutation, is that the mean fitness is equal to the parents' fitness; the variation produced by sex does not reduce the average fitness.

We now include the parental population's variance, which we will write as $\sigma^2(t) = \beta(t)\frac{1}{2}f(t)(1-f(t))G$. The average of the two parents will have variance $\sigma^2(t)/2$, so the population of all children will have fitness, before selection, distributed as

$$F_{\text{child}} \sim \text{Normal}\left(\text{mean}{=}f(t)G, \text{variance}{=}\left(1+\frac{\beta}{2}\right)\frac{1}{2}f(t)(1-f(t))G\right). \tag{20.26}$$

Natural selection selects the children on the upper side of this distribution. The mean increase in fitness will be

$$\bar{F}(t{+}1) - \bar{F}(t) = [\alpha(1+\beta/2)^{1/2}/\sqrt{2}]\sqrt{f(t)(1-f(t))G}, \tag{20.27}$$

and the variance of the surviving children will be

$$\sigma^2(t+1) = \gamma(1+\beta/2)\frac{1}{2}f(t)(1-f(t))G, \tag{20.28}$$

where $\alpha = \sqrt{2/\pi}$ and $\gamma = (1{-}2/\pi)$. If there is dynamic equilibrium $[\sigma^2(t{+}1) = \sigma^2(t)]$ then

$$\gamma(1+\beta/2) = \beta, \text{ so } (1+\beta/2) = \frac{2}{2-\gamma}, \tag{20.29}$$

and the factor in (20.27) is

$$\alpha(1+\beta/2)^{1/2}/\sqrt{2} = \sqrt{\frac{2}{(\pi+2)}} \simeq 0.62. \tag{20.30}$$

Defining this constant to be $\eta \equiv \sqrt{2/(\pi+2)}$, we conclude that, under sex and natural selection, the mean fitness of the population increases at a rate *proportional to the square root of the size of the genome*,

$$\frac{d\bar{F}}{dt} \simeq \eta\sqrt{f(t)(1-f(t))G} \text{ bits per generation.} \tag{20.31}$$

If, recklessly, we take our homogeneity assumption to hold for all time, we can write $\bar{F} = fG$ and obtain the differential equation:

$$\frac{df}{dt} \simeq \frac{\eta}{\sqrt{G}}\sqrt{f(t)(1-f(t))}, \tag{20.32}$$

whose solution is

$$f(t) = \frac{1}{2}\left[1+\sin\left(\frac{\eta}{\sqrt{G}}(t+c)\right)\right], \quad \text{for } t+c \in \left(-\tfrac{\pi}{2}\sqrt{G}/\eta, \tfrac{\pi}{2}\sqrt{G}/\eta\right), \tag{20.33}$$

where $c$ is a constant of integration, $c = \sin^{-1}(2f(0)-1)$. So this idealized system reaches a state of eugenic perfection ($f = 1$) within a finite time: $(\pi/\eta)\sqrt{G}$ generations.

Part IV

# Probabilities and Inference

## Information Theory, Pattern Recognition and Neural Networks

Handout number 3.

Do not be disturbed by missing pagenumbers: the handouts do not include all the book's chapters.

Because the handouts are based on different drafts of the book, there may be occasional mismatches between pagereferences, etc., where cross-references occur between this and the other handouts.

Nonexaminable material is indicated by the symbol $^*$.

## Approximate roadmap for the course

| | |
|---|---|
| Lecture 1 | Introduction to Information Theory. Chapter 1. |
| Before lecture 2 | Please work on exercise 4.4 (p. 76). |
| About now | Read chapters 2 and 5 and work on exercises in chapter 2. |
| Lecture 2–3 | Information content & typicality. Chapter 5. |
| Lecture 4 | Symbol codes. Chapter 6. |
| Lecture 5 | Arithmetic codes. Chapter 7. |
| About now | Read chapter 9 and do the exercises. |
| Lecture 6 | Noisy channels. Definition of mutual information and capacity. Chapter 10. |
| Lecture 7-8 | The noisy channel coding theorem. Chapter 11. |
| Lecture 9 | Clustering. Bayesian inference. Chapter 3, 22, 24. |
| About now | Read chapter 32 (Ising models). |
| Lecture 10 | Monte Carlo methods. Chapter 30, 31. |
| Lecture 12 | Variational methods. Chapter 34. |
| Lecture 13 | Neural networks – the single neuron. Chapter 41. |
| Lecture 14 | Capacity of the single neuron. Chapter 42. |
| Lecture 15 | Learning as inference. Chapter 43. |
| Lecture 16 | The Hopfield network. Content-addressable memory. Chapter 44. |

# 21

# *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 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 chapters 23, 26, and 27.

**Approximate methods** can be subdivided into

1. **Deterministic approximations**, which include Laplace's method (chapters 28 and 29) and variational methods (chapter 34); and
2. **Monte Carlo methods** – techniques in which random numbers play an integral part – which will be discussed in chapters 30, 31, and 33.

## 21.1 Overview

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, discussed a range of simple examples of inference problems and their Bayesian solutions.

To give further motivation for the toolbox of inference methods discussed in this part, chapter 22 discusses the problem of clustering; subsequent chapters discuss the probabilistic interpretation of clustering as mixture modelling.

Chapter 23 discusses the option of dealing with probability distributions by completely enumerating all hypotheses. Chapter 24 introduces the idea of maximization methods as a way of avoiding the large cost associated with complete enumeration, and points out reasons why maximum likelihood is not good enough. Chapters 26 and 27 discuss another way of avoiding the cost of complete enumeration: marginalization. Chapter 27 discusses message-passing methods appropriate for graphical models, using the decoding of error-correcting codes as an example. This chapter is a prerequisite for the understanding of advanced error-correcting codes.

Chapter 28 discusses deterministic approximations including Laplace's method. This chapter is a prerequisite for understanding the topic of complexity control in learning algorithms, an idea that is discussed in general terms in chapter 29.

Chapter 30 discusses Monte Carlo methods. Chapter 31 gives details of state-of-the-art Monte Carlo techniques.

Chapter 32 introduces the Ising model as a test-bed for probabilistic methods. An exact message-passing 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 33 describes 'exact' Monte Carlo methods and demonstrates their application to the Ising model.

Chapter 34 discusses variational methods and their application to Ising models and to simple statistical inference problems including clustering. This chapter will help the reader understand the Hopfield network (chapter 44) and the EM algorithm, which is an important method in latent-variable modelling.

This part of the book ends with a ragbag of assorted topics. Chapter 37 discusses a simple example of decision theory. Chapter 38 discusses interesting examples of prior probability distributions that describe ignorance. Chapter 39 discusses differences between sampling theory and Bayesian methods.

## *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 do 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 5, 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.

## *About chapter 22*

Before reading the next chapter, exercise 2.26 (p. 42) and section 12.2 (inferring the input to a Gaussian channel) are recommended reading.

# 22

# *An Example Inference Task: Clustering*

Human brains are good at finding regularities in data. One way of expressing regularity is to put a set of objects into groups that are similar to each other. For example, biologists have found that most objects in the natural world fall into one of two categories: things that are brown and run away, and things that are green and don't run away. The first group they call animals, and the second, plants. We'll call this operation of grouping things together *clustering*. If the biologist further sub-divides the cluster of plants into sub-clusters, we would call this 'hierarchical clustering'; but we won't be talking about hierarchical clustering yet. In this chapter we'll just discuss ways to take a set of $N$ objects and group them into $K$ clusters.

There are several motivations for clustering. First, a good clustering has predictive power: when an early biologist encounters a new green thing he has not seen before, his internal model of plants and animals fills in predictions for attributes of the green thing: it's unlikely to jump on him and eat him; if he touches it, he might get grazed or stung; if he eats it, he might feel sick. All of these predictions, while uncertain, are useful, because they help the biologist invest his resources (for example, the time spent watching for predators) well. Thus, we perform clustering because we believe the underlying cluster labels are meaningful, will lead to a more efficient description of our data, and will help us choose better actions. This type of clustering is sometimes called 'mixture density modelling'.

Second, clusters can be a useful aid to communication because they allow lossy compression. The biologist can give directions to a friend such as 'go to the third tree on the right then take a right turn' (rather than 'go past the large green thing with red berries, then past the large green thing with thorns, then ...'). The brief category name 'tree' is helpful because it is sufficient to identify an object. Similarly, in lossy image compression, the aim is to convey in as few bits as possible a reasonable reproduction of a picture; one way to do this is to divide the image into $N$ small patches, and find a close match to each patch in an alphabet of $K$ image-templates; then we send a close fit to the image by sending the list of labels $k_1, k_2, \ldots, k_N$ of the matching templates. The task of creating a good library of image-templates is equivalent to finding a set of cluster centres. This type of clustering is sometimes called 'vector quantization'.

We can formalize a vector quantizer in terms of an *assignment rule* $\mathbf{x} \rightarrow k(\mathbf{x})$ for assigning datapoints $\mathbf{x}$ to one of $K$ codenames, and a *reconstruction rule* $k \rightarrow \mathbf{m}^{(k)}$, the aim being to choose the functions $k(\mathbf{x})$ and $\mathbf{m}^{(k)}$ so as to minimize the *expected distortion*, which might be defined to be

$$D = \sum_{\mathbf{x}} P(\mathbf{x}) \frac{1}{2} \left[ \mathbf{m}^{(k(\mathbf{x}))} - \mathbf{x} \right]^2 . \tag{22.1}$$

[The ideal objective function 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. We note in passing the similarity of the assignment rule (i.e., the encoder) of vector quantization to the *decoding* problem when decoding an error-correcting code.

A third reason for making a cluster model is that failures of the cluster model may highlight interesting objects that deserve special attention. If we have trained a vector quantizer to do a good job of compressing pictures of ocean surfaces, then maybe patches of image that are not well compressed by the vector quantizer are the patches that contain submarines! If the biologist encounters a green thing and sees it run (or slither) away, this misfit with his cluster model (which says green things don't run away) cues him to pay special attention. One can't spend all one's time being fascinated by things; the cluster model can help sift out from the multitude of objects in one's world the ones that really deserve attention.

A fourth reason for liking clustering algorithms is that they may serve as models of learning processes in neural systems. The clustering algorithm that we now discuss, the K-means algorithm, is an example of a *competitive learning* algorithm. The algorithm works by having the $K$ means compete with each other for the right to own the data points.

## 22.1 K-means clustering

The K-means algorithm is an algorithm for putting $N$ data points in an $I$-dimensional space into $K$ clusters. Each cluster is parameterized by a vector $\mathbf{m}^{(k)}$ called its mean.

About the name... As far as I know, the K in K-means clustering simply refers to the chosen number of clusters. 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.

The data points will be denoted by $\mathbf{x}^{(n)}$ where the superscript $n$ runs from 1 to the number of data points $N$. Each vector $\mathbf{x}$ is a vector with $I$ components $x_i$. We will assume that the space that $\mathbf{x}$ lives in is a real space and that we have a metric that defines distances between points, for example,

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \sum_i (x_i - y_i)^2 \tag{22.2}$$

To start the K-means algorithm, algorithm 22.1, the $K$ means $\{\mathbf{m}^{(k)}\}$ are initialized in some way, for example to random values. The algorithm is then an iterative two-step algorithm. In the *assignment step*, each data point $n$ is assigned to the nearest mean. In the *update step*, the means are adjusted to match the sample means of the data points that they are responsible for.

The K-means algorithm is demonstrated for a toy two-dimensional data set in figure 22.1, where 2 means are used. The assignments of the points to the two clusters are indicated by two point styles, and the two means are shown by the circles. The algorithm converges after three iterations, at which point

**Initialization.** Set $K$ means $\{\mathbf{m}^{(k)}\}$ to random values.

**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 $\hat{k}^{(n)}$.

$$\hat{k}^{(n)} = \text{argmin}_k\{d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\}. \tag{22.3}$$

An alternative, equivalent 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.

$$r_k^{(n)} = \begin{cases} 1 & \text{if} \quad \hat{k}^{(n)} = k \\ 0 & \text{if} \quad \hat{k}^{(n)} \neq k. \end{cases} \tag{22.4}$$

*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, $\hat{k}^{(n)}$ is set to the smallest of the winning $\{k\}$.

**Update step.** The model parameters, the means, are adjusted to match the sample means of the data points that they are responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \tag{22.5}$$

where $R^{(k)}$ is the total responsibility of mean $k$,

$$R^{(k)} = \sum_n r_k^{(n)}. \tag{22.6}$$

*What about means with no responsibilities?* — If $R^{(k)} = 0$, then we leave the mean $\mathbf{m}^{(k)}$ where it is.

**Repeat the assignment step and update step** until the assignments do not change.

Algorithm 22.1. The K-means algorithm

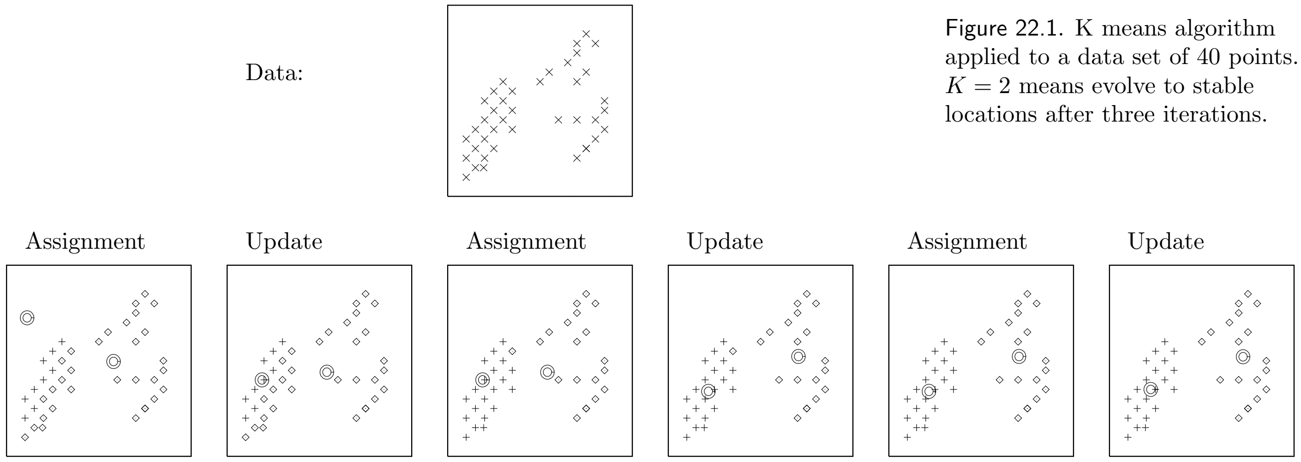


Figure 22.1. K means algorithm applied to a data set of 40 points. $K = 2$ means evolve to stable locations after three iterations.

the assignments are unchanged so the means remain unmoved when updated. The K-means algorithm always converges to a fixed point.

Exercise 22.1:$^{C4}$ See if you can prove that K-means always converges. [Hint: find a physical analogy and an associated Lyapunov function.]

A Lyapunov function is a function of the state of the algorithm that decreases whenever the state changes and that is bounded below. If a system has a Lyapunov function then its dynamics converge.

The K-means algorithm with a larger number of means, 4, is demonstrated in figure 22.2. The outcome of the algorithm depends on the initial condition. In the first case, after five iterations, a steady state is found in which the data points are fairly evenly split between the four clusters. In the second case, after six iterations, half the data points are in one cluster, and the others are shared among the other three clusters.

### *Questions about this algorithm*

The K-means algorithm has several *ad hoc* features. Why does the update step set the 'mean' to the mean of the assigned points? Where did the distance $d$ come from? What if we used a different measure of distance between $\mathbf{x}$ and $\mathbf{m}$? How can we choose the 'best' distance? [In vector quantization, the distance function is provided as part of the problem definition; but I'm assuming we are interested in data-modelling rather than vector quantization.] How do we choose $K$? Having found multiple alternative clusterings for a given $K$, how can we choose among them?

### *Cases where K means might be viewed as failing.*

Further questions arise when we look for cases where the algorithm behaves badly (compared with what the man in the street would call 'clustering'). Figure 22.3(a) shows a set of 75 data points generated from a mixture of two Gaussians. The right-hand Gaussian has less weight (only one fifth of the data points), and it is a less broad cluster. Figure 22.3(b) shows the outcome of using K-means clustering with $K = 2$ means. Four of the big cluster's data points have been assigned to the small cluster, and both means end up

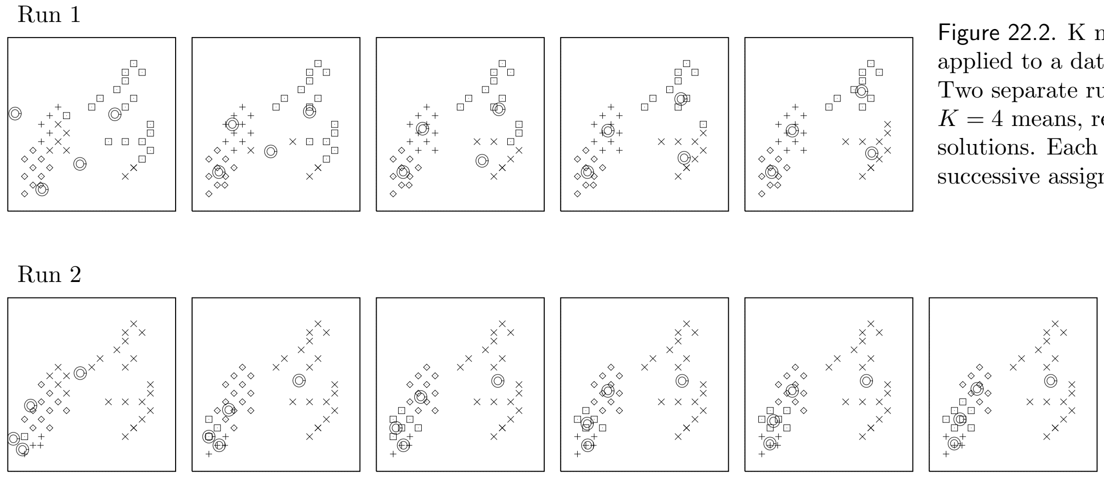


Figure 22.2. K means algorithm applied to a data set of 40 points. Two separate runs, both with $K = 4$ means, reach different solutions. Each frame shows a successive assignment step.

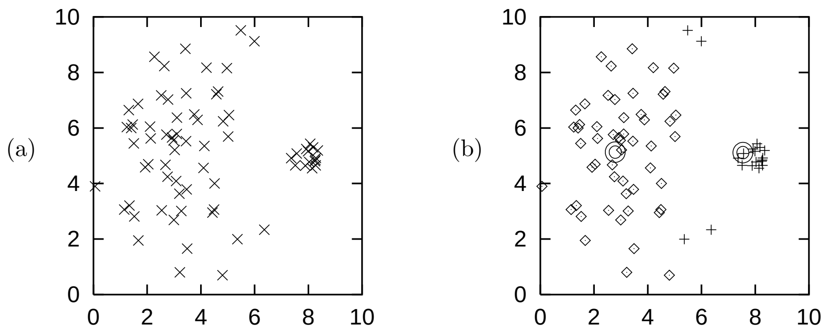


Figure 22.3. K means algorithm for a case with two dissimilar clusters. (a) Data. (b) A stable set of assignments and means.

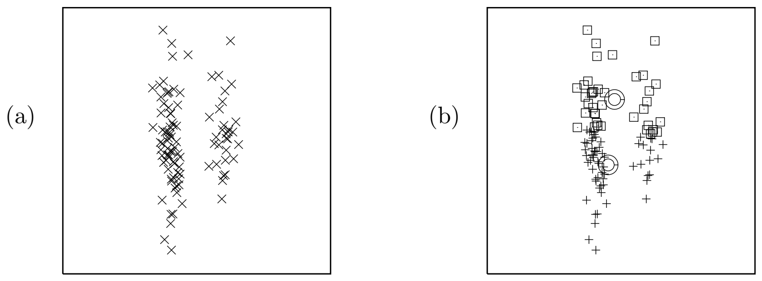


Figure 22.4. Two elongated clusters, and the stable solution found by the K-means algorithm.

displaced to the left of the true centres of the clusters. 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.

Figure 22.4 shows another case of K-means behaving badly. The data evidently falls into two elongated clusters. But the only stable state of the K-means algorithm is that shown in figure 22.4(b): the two clusters have been sliced in half at their midpoints! These two examples show that there is something wrong with the distance $d$ in the K-mean algorithm. The K-means algorithm has no way of representing the size or shape of a cluster.

A final criticism of K-means is that it is a 'hard', rather than a 'soft' algorithm: points are assigned to exactly one cluster and all points assigned to a cluster are equals in that cluster. Points located near the border between two or more clusters should, arguably, play a *partial* role in determining the locations of all the clusters that they could plausibly be assigned to. But in the K-means algorithm, each borderline point is dumped in one cluster, and has an equal vote with all the other points in that cluster, and no vote in any other clusters.

## 22.2 Soft K-means clustering

These criticisms of K-means motivate the 'soft K-means algorithm', algorithm 22.2. The algorithm has one parameter, $\beta$, which we could term the *stiffness*.

**Assignment step.** Each data point $\mathbf{x}^{(n)}$ is given a soft 'degree of assignment' to each of the means. We call the degree to which $\mathbf{x}^{(n)}$ is assigned to cluster $k$ the *responsibility* $r_k^{(n)}$ (the responsibility of cluster $k$ for point $n$).

$$r_k^{(n)} = \frac{\exp\left(-\beta\, d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\right)}{\sum_{k'} \exp\left(-\beta\, d(\mathbf{m}^{(k')}, \mathbf{x}^{(n)})\right)}. \tag{22.7}$$

The sum of the $K$ responsibilities for the $n$th point is 1.

**Update step.** The model parameters, the means, are adjusted to match the sample means of the data points that they are responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \tag{22.8}$$

where $R^{(k)}$ is the total responsibility of mean $k$,

$$R^{(k)} = \sum_n r_k^{(n)}. \tag{22.9}$$

Algorithm 22.2. Soft K-means algorithm, version 1.

Notice the similarity of this soft K-means algorithm 22.2 to the hard K-means algorithm 22.1. The update step is identical; the only difference is that

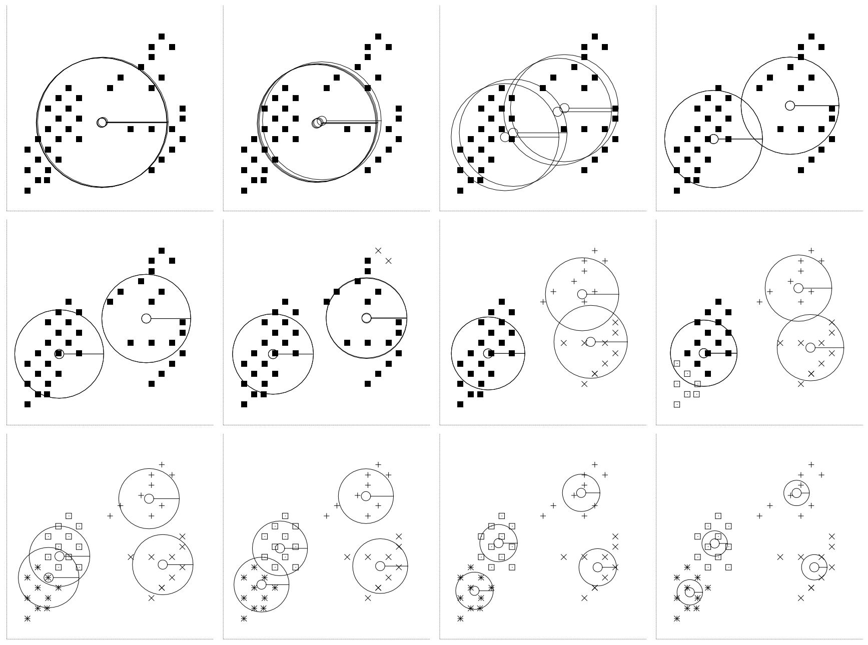


Figure 22.5. Soft K-means algorithm, version 1, applied to a data set of 40 points. $K = 4$. Implicit lengthscale parameter $\sigma = 1/\beta^{1/2}$ varied from a large to a small value. Each picture shows the state of all four means, with the implicit lengthscale shown by the radius of the four circles, after running the algorithm for several tens of iterations. At the largest lengthscale, all four means converge exactly to the data mean. Then the four means separate into two groups of two. At shorter lengthscales, each of these pairs itself bifurcates into subgroups.

the responsibilities $r_k^{(n)}$ can take on values between 0 and 1. Whereas the assignment $\hat{k}^{(n)}$ in the K-means algorithm involved a 'min' over the distances, the rule for assigning the responsibilities is a 'soft-min'.

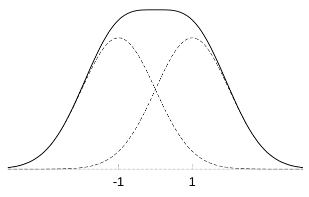
Exercise 22.2:$^{B2}$ Show that as the stiffness $\beta$ goes to $\infty$, the soft K means algorithm becomes identical to the original hard K-means algorithm, except for the way in which means with no assigned points behave. Describe what those means do instead of sitting still.

Dimensionally, the stiffness $\beta$ is an inverse-length-squared, so we can associate a lengthscale, $\sigma \equiv 1/\sqrt{\beta}$, with the value of $\beta$. The soft K-means algorithm is demonstrated in figure 24.2. The lengthscale is shown by the radius of the circles surrounding the four means. Each figure shows the final fixed point reached for a different value of the lengthscale $\sigma$, with large lengthscale at the top left and short lengthscale at the bottom right.

Solutions on p.326

Exercise 22.3:$^{B3}$ Explore the properties of the soft K-means algorithm, version 1 by assuming that the datapoints $\{\mathbf{x}\}$ come from a *single* separable two-dimensional Gaussian distribution with mean zero and variances $(\mathrm{var}(x_1), \mathrm{var}(x_2)) = (\sigma_1^2, \sigma_2^2)$, with $\sigma_1^2 > \sigma_2^2$. Set $K = 2$, assume $N$ is large, and investigate the fixed points of the algorithm as $\beta$ is varied. [Hint: assume that $\mathbf{m}^{(1)} = (m, 0)$ and $\mathbf{m}^{(2)} = (-m, 0)$.]

Exercise 22.4:$^{B3}$ Consider the soft K-means algorithm applied to a large amount of one-dimensional data that comes from a mixture of two equal-weight Gaussians with true means $\mu = \pm 1$ and standard deviation $\sigma_P$, for example $\sigma_P = 1$. Show that the hard K-means algorithm with $K = 2$ leads to a solution in which the two means are further apart than the two true



means. Discuss what happens for other values of $\beta$, and find the value of $\beta$ such that the soft algorithm puts the two means in the correct places.

*Conclusion*

At this point, we may have fixed some of the problems with the original K-means algorithm, by introducing an extra parameter $\beta$ whose value controls the algorithm's outcome. How should we set $\beta$? And what about the problem of the elongated clusters, and the clusters of unequal weight and width? Adding one stiffness parameter $\beta$ is not going to make all these problems go away.

We'll come back to these questions in a later chapter, as we develop the mixture-density-modelling view of clustering.

## 22.3 Solutions

Solution to exercise 22.1 (p.322): We can associate an 'energy' with the state of the K-means algorithm by connecting a spring between each point $\mathbf{x}^{(n)}$ and the mean that is responsible for it. The energy of one spring is proportional to its squared-length, namely $\beta d(\mathbf{x}^{(n)}, \mathbf{m}^{(k)})$ where $\beta$ is the stiffness of the spring. The total energy of all the springs is a Lyapunov function for the algorithm, because (a) the assignment step can only decrease the energy – a point only changes its allegiance if the length of its spring is reduced; (b) the update step can only decrease the energy – moving $\mathbf{m}^{(k)}$ to the mean is the way to minimize the energy of its springs; and (c) the energy is bounded below – which is the second condition for a Lyapunov function. Since the algorithm has a Lyapunov function, it converges.

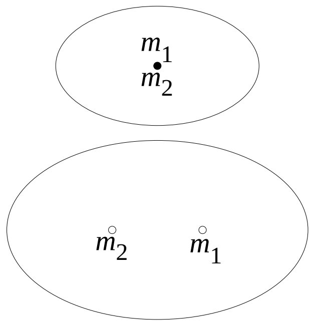


Figure 22.6. Schematic diagram of the bifurcation as the largest data variance $\sigma_1$ increases from below $1/\beta^{1/2}$ to above $1/\beta^{1/2}$. The data variance is indicated by the ellipse.

Solution to exercise 22.3 (p.325): If the means are initialized to $\mathbf{m}^{(1)} = (m, 0)$ and $\mathbf{m}^{(1)} = (-m, 0)$, the assignment step for a point at location $x_1, x_2$ gives

$$r_1(\mathbf{x}) = \frac{\exp(-\beta(x_1 - m)^2/2)}{\exp(-\beta(x_1 - m)^2/2) + \exp(-\beta(x_1 + m)^2/2)} \tag{22.10}$$

$$= \frac{1}{1 + \exp(-2\beta m x_1)}, \tag{22.11}$$

and the updated $m$ is

$$m' = \frac{\int dx_1\, P(x_1)\, x_1\, r_1(\mathbf{x})}{\int dx_1\, P(x_1)\, r_1(\mathbf{x})} \tag{22.12}$$

$$= 2 \int dx_1\, P(x_1)\, x_1 \frac{1}{1 + \exp(-2\beta m x_1)}. \tag{22.13}$$

Now, $m = 0$ is a fixed point, but the question is, is it stable or unstable? For tiny $m$ (that is, $\beta \sigma_1 m \ll 1$), we can Taylor expand

$$\frac{1}{1 + \exp(-2\beta m x_1)} \simeq \frac{1}{2}(1 + \beta m x_1) + \ldots \tag{22.14}$$

so

$$m' \simeq \int dx_1\, P(x_1)\, x_1\, (1 + \beta m x_1) \tag{22.15}$$

$$= \int dx_1\, P(x_1)\, x_1^2\, \beta m \tag{22.16}$$

$$= \sigma_1^2 \beta m. \tag{22.17}$$

For small $m$, $m$ either grows or decays exponentially under this mapping, depending on whether $\sigma_1^2\beta$ is greater than or less than 1. The fixed point $m = 0$ is *stable* if

$$\sigma_1^2 \leq 1/\beta \tag{22.18}$$

and *unstable* otherwise. [Incidentally, this derivation shows that this result is general, holding for any true probability distribution $P(x_1)$ having variance $\sigma_1^2$, not just the Gaussian.]

If $\sigma_1^2 > 1/\beta$ then there is a bifurcation and there are two stable fixed points surrounding the unstable fixed point at $m = 0$. To illustrate this bifurcation, figure 22.7 shows the outcome of running the soft K-means algorithm with $\beta = 1$ on one-dimensional data with standard deviation $\sigma_1$ for various values of $\sigma_1$. Figure 22.8 shows this pitchfork bifurcation from the other point of view, where the data's standard deviation $\sigma_1$ is fixed and the algorithm's lengthscale $\sigma = 1/\beta^{1/2}$ is varied on the horizontal axis.

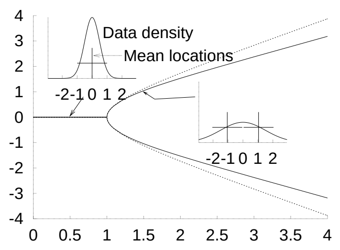


Figure 22.7. The stable mean locations as a function of $\sigma_1$, for constant $\beta$ (thick lines), and the approximation (22.23).

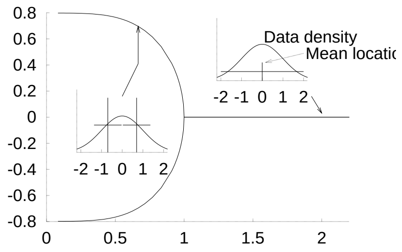


Figure 22.8. The stable mean locations as a function of $1/\beta^{1/2}$, for constant $\sigma_1$.

At (22.21) we use the fact that $P(x_1)$ is Gaussian to find the fourth moment.

Here is a cheap theory to model how the fitted parameters $\pm m$ behave beyond the bifurcation, based on continuing the series expansion. This continuation of the series is rather suspect, since the series isn't necessarily expected to converge beyond the bifurcation point, but the theory fits well anyway.

We take our analytic approach one term further in the expansion

$$\frac{1}{1+\exp(-2\beta m x_1)} \simeq \frac{1}{2}(1 + \beta m x_1 - \frac{1}{3}(\beta m x_1)^3) + \ldots \tag{22.19}$$

then we can solve for the shape of the bifurcation to leading order, which depends on the fourth moment of the distribution:

$$\begin{aligned} m' &\simeq \int dx_1\, P(x_1) x_1 (1 + \beta m x_1 - \frac{1}{3}(\beta m x_1)^3) && (22.20)\\ &= \sigma_1^2 \beta m - \frac{1}{3}(\beta m)^3 3\sigma_1^4 && (22.21)\end{aligned}$$

This map has a fixed point at $m$ such that

$$\sigma_1^2\beta(1 - (\beta m)^2\sigma_1^2) = 1, \tag{22.22}$$

i.e.,

$$m = \pm\beta^{-1/2}\frac{(\sigma_1^2\beta - 1)^{1/2}}{\sigma_1^2\beta}, \tag{22.23}$$

The thin line in figure 22.7 shows this theoretical approximation.

Exercise 22.5:$^{B2}$ Why does the pitchfork in figure 22.8 tend to the values $\sim \pm 0.8$ as $1/\beta^{1/2} \to 0$? Give an analytic expression for this asymptote.

Solution to exercise 22.5 (p.327): The asymptote is the mean of the rectified Gaussian,

$$\frac{\int_0^\infty \text{Normal}(x, 1) x \,\mathrm{d}x}{1/2} = \sqrt{2/\pi} \simeq 0.798. \tag{22.24}$$

## *Further reading*

For a vector-quantization approach to clustering see (Luttrell 1989; Luttrell 1990).

# 23

# *Exact Inference by Complete Enumeration*

We open our toolbox of methods for handling probabilities by discussing a brute-force method of handling inferences: complete enumeration of all hypotheses, and evaluation of their probabilities. This approach is an exact method, and the difficulty of carrying it out will motivate the smarter exact and approximate methods introduced in the following chapters.

## 23.1 The burglar alarm

Bayesian probability theory is sometimes called 'common sense, amplified'. When thinking about the following questions, please ask your common sense what it thinks the answers are; we will then see how Bayesian methods confirm your everyday intuition.

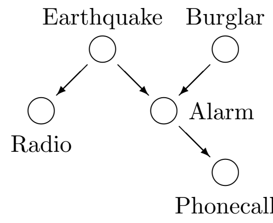


Figure 23.1. Belief network for the burglar alarm problem.

Example 23.1: 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 was 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. 'Oh', he says, feeling relieved, 'it was probably the earthquake that set off the alarm'. What is the probability that there was a burglar in his house? (After Pearl, 1988).

Let's introduce variables $b$ (a burglar is present in Fred's house today), $a$ (the alarm is ringing), $p$ (Fred receives a phonecall from the neighbour reporting the alarm), $e$ (a small earthquake took place today near Fred's house), and $r$ (the radio report of earthquake is heard by Fred). The probability of all these variables might factorize as follows:

$$P(b, e, a, p, r) = P(b)P(e)P(a \mid b, e)P(p \mid a)P(r \mid e), \tag{23.1}$$

and plausible values for the probabilities are:

1. Burglar probability:

$$P(b{=}1) = \beta, \quad P(b{=}0) = 1 - \beta, \tag{23.2}$$

e.g., $\beta = 0.001$ gives a mean burglary rate of once every three years.

2. Earthquake probability:

$$P(e{=}1) = \epsilon, \quad P(e{=}0) = 1 - \epsilon, \tag{23.3}$$

with, e.g., $\epsilon = 0.001$; our assertion that the earthquakes are independent of burglars, i.e., the prior probability of $b$ and $e$ is $P(b, e) = P(b)P(e)$, seems reasonable unless we take into account opportunistic burglars who strike immediately after earthquakes.

3. Alarm ringing probability: we assume the alarm will ring if *any* of the following three events happens: (a) a burglar enters the house, and triggers the alarm (let's assume the alarm has a reliability of $\alpha_b = 0.99$, i.e., 99% of burglars trigger the alarm); (b) an earthquake takes place, and triggers the alarm (perhaps $\alpha_e = 1\%$ of alarms are triggered by earthquakes?); or (c) some other event causes a false alarm; let's assume the false alarm rate $f$ is 0.001, so Fred has false alarms from non-earthquake causes once every three years. The probabilities of $a$ given $b$ and $e$ are then:

The dependence of $a$ on $b$ and $e$ is known as a 'noisy-or'.

$$\begin{array}{rclrcl}
P(a{=}0 \,|\, b{=}0, e{=}0) &=& (1-f), & P(a{=}1 \,|\, b{=}0, e{=}0) &=& f \\
P(a{=}0 \,|\, b{=}1, e{=}0) &=& (1-f)(1-\alpha_b), & P(a{=}1 \,|\, b{=}1, e{=}0) &=& 1-(1-f)(1-\alpha_b) \\
P(a{=}0 \,|\, b{=}0, e{=}1) &=& (1-f)(1-\alpha_e), & P(a{=}1 \,|\, b{=}0, e{=}1) &=& 1-(1-f)(1-\alpha_e) \\
P(a{=}0 \,|\, b{=}1, e{=}1) &=& (1-f)(1-\alpha_b)(1-\alpha_e), & P(a{=}1 \,|\, b{=}1, e{=}1) &=& 1-(1-f)(1-\alpha_b)(1-\alpha_e)
\end{array}$$

or, in numbers,

$$\begin{array}{rclrcl}
P(a{=}0 \,|\, b{=}0, e{=}0) &=& 0.999, & P(a{=}1 \,|\, b{=}0, e{=}0) &=& 0.001 \\
P(a{=}0 \,|\, b{=}1, e{=}0) &=& 0.00999, & P(a{=}1 \,|\, b{=}1, e{=}0) &=& 0.99001 \\
P(a{=}0 \,|\, b{=}0, e{=}1) &=& 0.98901, & P(a{=}1 \,|\, b{=}0, e{=}1) &=& 0.01099 \\
P(a{=}0 \,|\, b{=}1, e{=}1) &=& 0.0098901, & P(a{=}1 \,|\, b{=}1, e{=}1) &=& 0.9901099
\end{array}$$

We assume the neighbour would never phone if the alarm is not ringing [$P(p{=}1 \,|\, a{=}0) = 0$]; and that the radio is a trustworthy reporter too [$P(r{=}1 \,|\, e{=}0) = 0$]; we won't need to specify the probabilities $P(p{=}1 \,|\, a{=}1)$ or $P(r{=}1 \,|\, e{=}1)$ in order to answer the questions above, since the outcomes $p = 1$ and $r{=}1$ give us certainty respectively that $a{=}1$ and $e{=}1$.

We can answer the two questions about the burglar by computing the posterior probabilities of all hypotheses given the available information. Let's start by reminding ourselves that the probability that there is a burglar, before either $p$ or $r$ is observed, is $P(b{=}1) = \beta = 0.001$, and the probability that an earthquake took place is $P(e{=}1) = \epsilon = 0.001$, and these two propositions are *independent*.

First, when $p{=}1$, we know that the alarm is ringing: $a{=}1$. The posterior probability of $b$ and $e$ becomes:

$$P(b, e \,|\, a{=}1) = \frac{P(a{=}1 \,|\, b, e)P(b)P(e)}{P(a{=}1)} \tag{23.4}$$

The numerator's four possible values are

$$\begin{array}{rclcl}
P(a{=}1 \,|\, b{=}0, e{=}0) \cdot P(b{=}0) \cdot P(e{=}0) &=& 0.001 \cdot 0.999 \cdot 0.999 &=& 0.000998 \\
P(a{=}1 \,|\, b{=}1, e{=}0) \cdot P(b{=}1) \cdot P(e{=}0) &=& 0.99001 \cdot 0.001 \cdot 0.999 &=& 0.000989 \\
P(a{=}1 \,|\, b{=}0, e{=}1) \cdot P(b{=}0) \cdot P(e{=}1) &=& 0.01099 \cdot 0.999 \cdot 0.001 &=& 0.000010979 \\
P(a{=}1 \,|\, b{=}1, e{=}1) \cdot P(b{=}1) \cdot P(e{=}1) &=& 0.9901099 \cdot 0.001 \cdot 0.001 &=& 9.9 \times 10^{-7},
\end{array}$$

The normalizing constant is the sum of these four numbers, $P(a{=}1) = 0.002$, and the posterior probabilities are

$$\begin{aligned} P(b{=}0, e{=}0 \,|\, a{=}1) &= 0.4993 \\ P(b{=}1, e{=}0 \,|\, a{=}1) &= 0.4947 \\ P(b{=}0, e{=}1 \,|\, a{=}1) &= 0.0055 \\ P(b{=}1, e{=}1 \,|\, a{=}1) &= 0.0005 \end{aligned} \tag{23.5}$$

To answer the question, 'what's the probability a burglar was there?' we *marginalize* over the earthquake variable $e$:

$$\begin{aligned} P(b{=}0 \,|\, a{=}1) &= P(b{=}0, e{=}0 \,|\, a{=}1) + P(b{=}0, e{=}1 \,|\, a{=}1) = 0.505 \\ P(b{=}1 \,|\, a{=}1) &= P(b{=}1, e{=}0 \,|\, a{=}1) + P(b{=}1, e{=}1 \,|\, a{=}1) = 0.495. \end{aligned} \tag{23.6}$$

So there is nearly a 50% chance that there was a burglar present. It is important to note that the variables $b$ and $e$, which were independent *a priori*, are now *dependent*. The posterior distribution (23.5) is not a separable function of $b$ and $e$. This fact is illustrated most simply by studying the effect of learning that $e = 1$.

When we learn $e{=}1$, the posterior probability of $b$ is given by $P(b \,|\, e{=}1, a{=}1) = P(b, e{=}1 \,|\, a{=}1)/P(e{=}1 \,|\, a{=}1)$, i.e., by dividing the bottom two rows of (23.5), by their sum $P(e{=}1 \,|\, a{=}1) = 0.0060$. The posterior probability of $b$ is:

$$\begin{aligned} P(b{=}0 \,|\, e{=}1, a{=}1) &= 0.92 \\ P(b{=}1 \,|\, e{=}1, a{=}1) &= 0.08 \end{aligned} \tag{23.7}$$

There is thus now an 8% chance that a burglar was in Fred's house. It is in accordance with everyday intuition that the probability that $b{=}1$ (a possible cause of the alarm) is reduced by Fred's learning that an earthquake, an alternative explanation of the alarm, has happened.

This phenomenon, that one of the possible causes ($b = 1$) of some data (the data here being $a = 1$) becomes *less* probable when another of the causes ($e = 1$) becomes more probable, even though those two causes were independent variables *a priori*, is known as *explaining away*. Explaining away is an important feature of correct inferences, and one that any artificial intelligence should replicate.

If we believe that the neighbour and the radio service are unreliable or capricious, so that we are not certain that the alarm really is ringing or that an earthquake really has happened, the calculations become more complex, but the explaining-away effect persists; and the arrival of the earthquake report $r$ simultaneously makes it *more* probable that the alarm truly is ringing, and *less* probable that the burglar was present.

In summary, we solved the inference questions about the burglar by enumerating all four hypotheses about the variables $(b, e)$, finding their posterior probabilities, and marginalizing to obtain the required inferences about $b$.

## 23.2 Exact inference for continuous hypothesis spaces

Many of the hypothesis spaces we will consider are naturally thought of as continuous. For example, the unknown decay length $\lambda$ of section 3.1 (p.57) lives in a continuous one-dimensional space; and the unknown mean and standard deviation of a Gaussian $\mu, \sigma$ live in a continuous two-dimensional space.

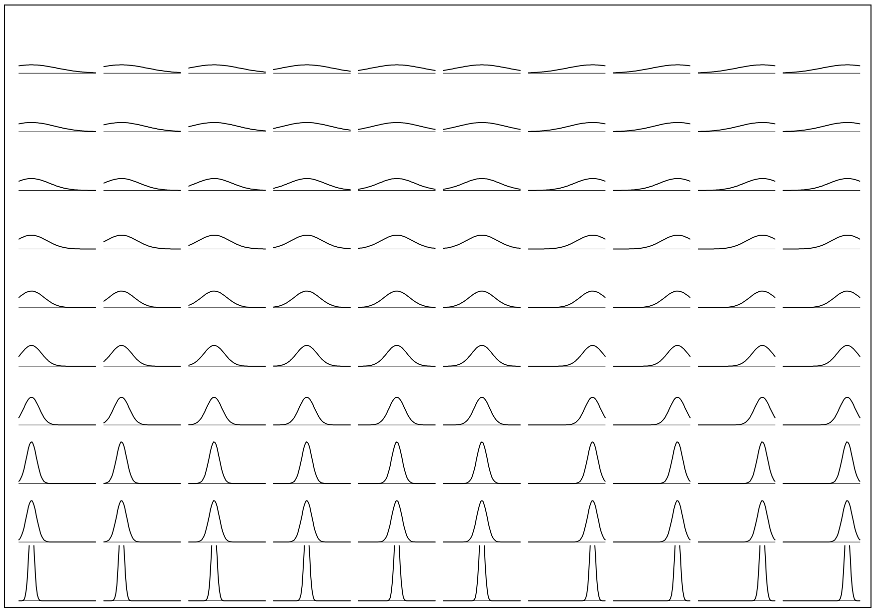


Figure 23.2. One Gaussian. Enumeration of an entire (discretized) hypothesis space.

In any practical computer implementation, such continuous spaces will necessarily be discretized, however, and so can, in principle, be enumerated – at a grid of parameter values, for example. In figure 3.2 we plotted the likelihood function for the decay length as a function of $\lambda$ by evaluating the likelihood at a finely-spaced grid of points.

Let's look at the Gaussian distribution as an example of a model with a two-dimensional hypothesis space. The one-dimensional Gaussian distribution is parameterized by a mean $\mu$ and a standard deviation $\sigma$:

$$P(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \equiv \text{Normal}(x; \mu, \sigma^2). \tag{23.8}$$

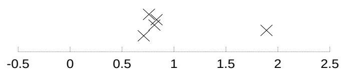


Figure 23.3. Five data points $\{x_n\}_{n=1}^{5}$. The horizontal coordinate is the value of the datum, $x_n$; the vertical coordinate has no meaning.

Figure 23.2 shows an enumeration of one hundred hypotheses about the mean and standard deviation of a one-dimensional Gaussian distribution. These hypotheses are evenly spaced in a ten by ten square grid covering ten values of $\mu$ and ten values of $\sigma$. Each hypothesis is represented by the probability density that it puts on $x$. We now examine the inference of $\mu$ and $\sigma$ given data points $x_n$, $n = 1, \ldots, N$, assumed to be drawn independently from this density.

Imagine that we acquire data, for example the five points shown in figure 23.3. We can now evaluate the posterior probability of each of the one hundred subhypotheses by evaluating the likelihood of each, that is, the value of $P(\{x_n\}_{n=1}^{5} \mid \mu, \sigma)$. The likelihood values are shown diagrammatically in figure 23.4 using the line thickness to encode the value of the likelihood. Subhypotheses with likelihood smaller than $e^{-8}$ times the maximum likelihood have been deleted.

Using a finer grid, we can represent the same information by plotting the likelihood function as a surface plot or contour plot as a function of $\mu$ and $\sigma$ (figure 23.5).

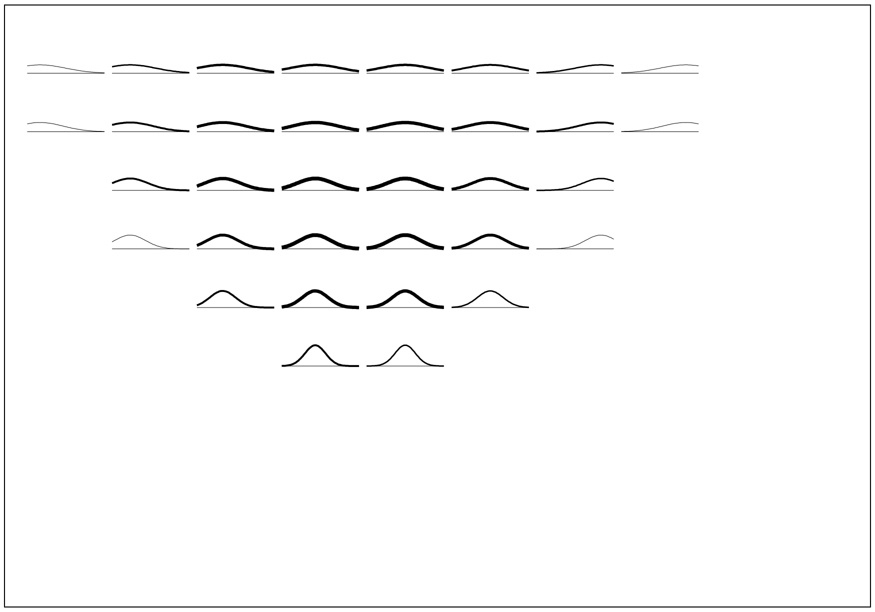


Figure 23.4. One Gaussian. Likelihood function represented by line thickness. Subhypotheses having likelihood smaller than $e^{-8}$ times the maximum likelihood are not shown.

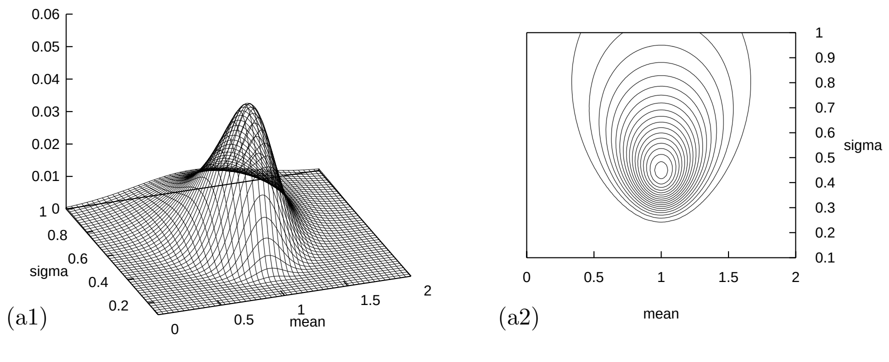


Figure 23.5. The likelihood function for the parameters of a Gaussian distribution. (a1,a2) Surface plot and contour plot of the log likelihood as a function of $\mu$ and $\sigma$. The data set of $N = 5$ points had mean $\bar{x} = 1.0$ and $S^2 = \sum(x - \bar{x})^2 = 1.0$.

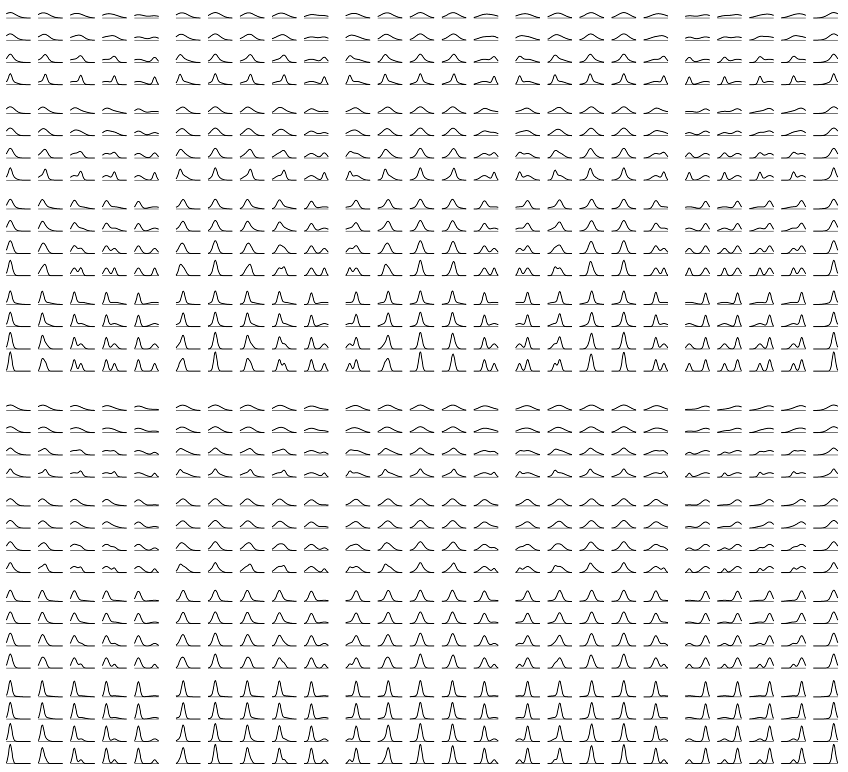


Figure 23.6. Mixture of two Gaussians. Enumeration of the entire (discretized) hypothesis space. Weight of the mixture components is $\pi_1, \pi_2 = 0.6, 0.4$ in the top half and $0.8, 0.2$ in the bottom half. Means $\mu_1$ and $\mu_2$ vary horizontally, and standard deviations $\sigma_1$ and $\sigma_2$ vary vertically.

Eyeballing the data (figure 23.3), you might agree that it seems more plausible that they come not from a single Gaussian but from a mixture of two Gaussians, defined by two means, two standard deviations, and two mixing coefficients $\pi_1$ and $\pi_2$, satisfying $\pi_1 + \pi_2 = 1$, $\pi_i \geq 0$.

$$P(x|\mu_1, \sigma_1, \pi_1, \mu_2, \sigma_2, \pi_2) = \frac{\pi_1}{\sqrt{2\pi}\sigma_1} \exp\left(-\tfrac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{\pi_2}{\sqrt{2\pi}\sigma_2} \exp\left(-\tfrac{(x-\mu_2)^2}{2\sigma_2^2}\right)$$

Let's enumerate the subhypotheses for this alternative model. The parameter space is five-dimensional, so it becomes challenging to represent it on a single page. Figure 23.6 enumerates 800 subhypotheses with different values of the five parameters $\mu_1, \mu_2, \sigma_1, \sigma_2, \pi_1$. The means are varied between five values each in the horizontal directions. The standard deviations take on four values each vertically. And $\pi_1$ takes on two values vertically. We can represent the inference about these five parameters in the light of the five datapoints as shown in figure 23.7.

If we wish to compare the one-gaussian model with the mixture-of-two model, we can find their posterior probabilities by evaluating the marginal likelihood or evidence for each model $\mathcal{H}$, $P(\{x\} \,|\, \mathcal{H})$. The evidence is given by integrating over the parameters, $\theta$; the integration can be implemented numerically by summing over the alternative enumerated values of $\theta$,

$$P(\{x\} \,|\, \mathcal{H}) = \sum_{\theta} P(\theta) P(\{x\} \,|\, \theta, \mathcal{H}), \tag{23.9}$$

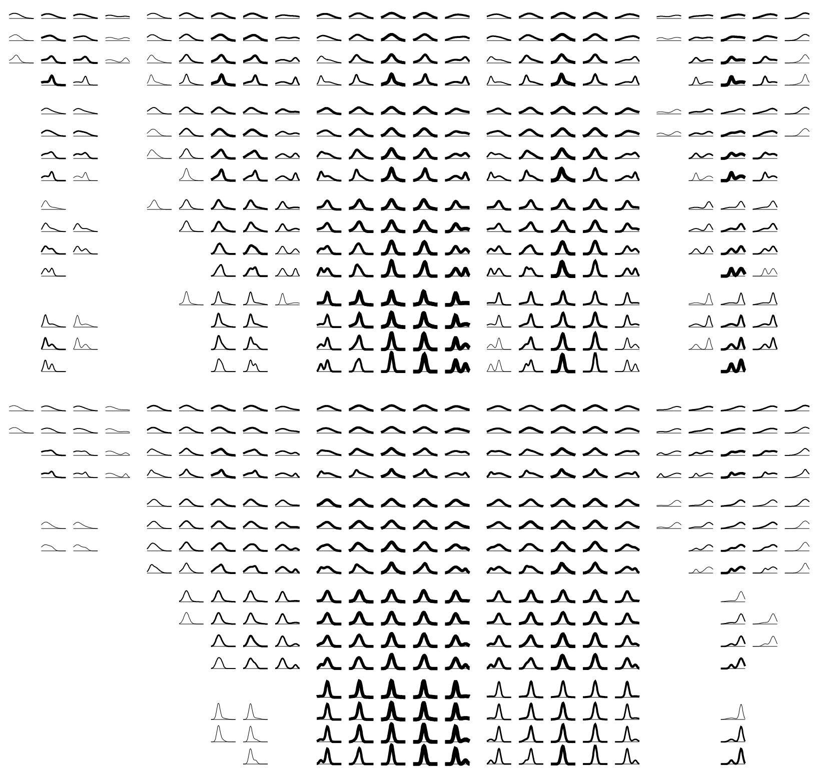


Figure 23.7. Mixture of two Gaussians. Likelihood function represented by line thickness. Subhypotheses having likelihood smaller than $e^{-8}$ times the maximum likelihood are not shown.

where $P(\theta)$ is the prior distribution over the grid of parameter values, which I take to be uniform.

For the mixture of two Gaussians this integral is a five-dimensional integral; if it is to be performed at all accurately, the grid of points will need to be much finer than the grids shown in the figures. If the uncertainty about each of $K$ parameters has been reduced by, say, a factor of ten by observing the data, then brute force integration will require a grid of at least $10^K$ points. This exponential growth of computation with model size is the reason why complete enumeration is rarely a feasible computational strategy.

# 24

# *Maximum Likelihood and Clustering*

*Under construction.*

Rather than enumerate all hypotheses – which may be exponential in number – we can save a lot of time by homing in on one good hypothesis which fits the data well. This is the philosophy behind the maximum likelihood method, which identifies the setting of the parameters $\theta$ which maximizes the likelihood, $P(\text{Data}|\theta, \mathcal{H})$.

For some models the maximum likelihood parameters can be identified instantly from the data; for more complex models, finding the maximum likelihood parameters may require an iterative algorithm.

For any model, it is usually easiest to work with the logarithm of the likelihood, rather than the likelihood, since likelihoods, being products of the probabilities of many data points, tend to be very small.

We return to the Gaussian for our first examples. Assume we have data $\{x_n\}_{n=1}^N$. The log likelihood is:

$$\begin{aligned} \log P(\{x_n\}_{n=1}^N|\mu,\sigma) &= -N\log(\sqrt{2\pi}\sigma) - \sum_n (x_n-\mu)^2/(2\sigma^2), && (24.1)\\ &= -N\log(\sqrt{2\pi}\sigma) - [N(\mu-\bar{x})^2 + S]/(2\sigma^2), && (24.2)\end{aligned}$$

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'.

Example 24.1**:** Differentiate the log likelihood with respect to $\mu$ and $\log\sigma$ and show that the maximum likelihood mean $\mu$ of a Gaussian, whose standard deviation is known to be $\sigma$, is equal to the sample mean,

$$\bar{x} \equiv \textstyle\sum_{n=1}^N x_n/N, \qquad (24.3)$$

regardless of $\sigma$.

If we Taylor-expand the log-likelihood about the maximum, we can use a quadratic approximation to estimate how far from the maximum likelihood parameter setting we can go before the likelihood falls by some standard factor, for example $e^{1/2}$, or $e^{4/2}$, so as to define error bars on the maximum likelihood parameter.

Example 24.2**:** Find the second derivative of the log likelihood with respect to $\mu$, and find the error bars on $\mu$, given the data and $\sigma$.

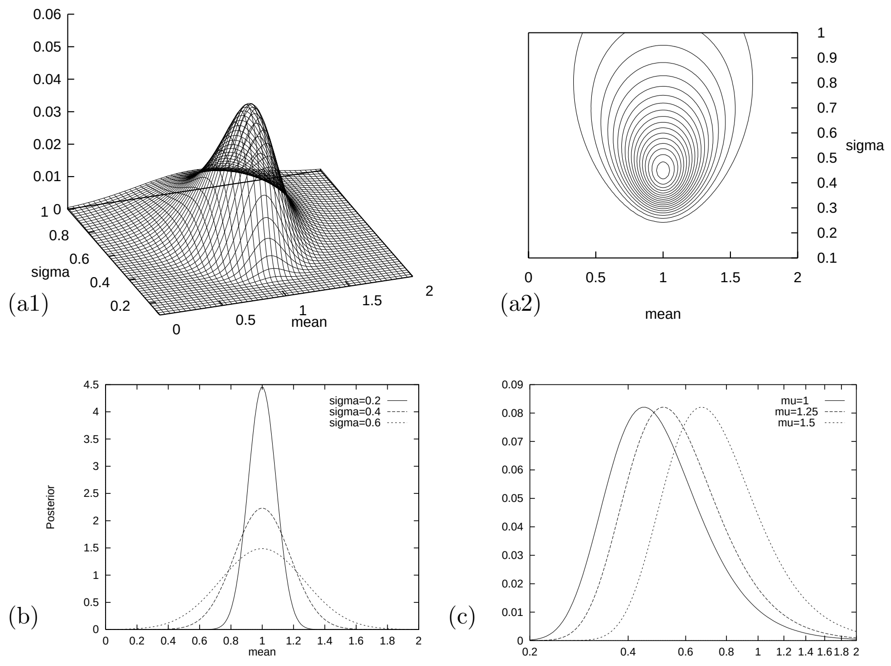


Figure 24.1. The likelihood function for the parameters of a Gaussian distribution.
(a1,a2) Surface plot and contour plot of the log likelihood as a function of $\mu$ and $\sigma$. The data set of $N = 5$ points had mean $\bar{x} = 1.0$ and $S^2 = \sum(x - \bar{x})^2 = 1.0$.
(b) The posterior probability of $\mu$ for various values of $\sigma$.
(c) The posterior probability of $\sigma$ for various fixed values of $\mu$.

Example 24.3: Find the maximum likelihood standard deviation $\sigma$ of a Gaussian, whose mean is known to be $\mu$, in the light of data $\{x_n\}_{n=1}^N$. Find the second derivative of the log likelihood with respect to $\log\sigma$, and error bars on $\log\sigma$.

Exercise 24.4:$^{B2}$ Show that the values of $\mu$ and $\log\sigma$ that jointly maximize the likelihood are: $\{\mu, \sigma\}_{\text{ML}} = \left\{\bar{x}, \sigma_N = \sqrt{S/N}\right\}$, where

$$\sigma_N \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N}}. \qquad (24.4)$$

We now derive an algorithm for fitting a mixture of Gaussians to one-dimensional data.

Exercise 24.5:$^{A2}$ A random variable $x$ is assumed to have a probability distribution that is a *mixture of two Gaussians*,

$$P(x|\mu_1, \mu_2, \sigma) = \left[\sum_{k=1}^{2} p_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma^2}\right)\right],$$

where the two Gaussians are given the labels $k = 1$ and $k = 2$; the prior probability of the class label $k$ is $\{p_1 = 1/2, p_2 = 1/2\}$; $\{\mu_k\}$ are the means of the two Gaussians; and both have standard deviation $\sigma$. For brevity, we denote these parameters by $\theta \equiv \{\{\mu_k\}, \sigma\}$.

A data set consists of $N$ points $\{x_n\}_{n=1}^N$ which are assumed to be independent samples from this distribution. Let $k_n$ denote the unknown class label of the $n$th point.

Assuming that $\{\mu_k\}$ and $\sigma$ are known, show that the posterior probability of the class label $k_n$ of the $n$th point can be written as

$$\begin{aligned} P(k_n = 1|x_n, \theta) &= \frac{1}{1+\exp[-(w_1 x + w_0)]} \\ P(k_n = 2|x_n, \theta) &= \frac{1}{1+\exp[+(w_1 x + w_0)]} \end{aligned}, \qquad (24.5)$$

and give expressions for $w_1$ and $w_0$.

Assume now that the means $\{\mu_k\}$ are *not* known, and that we wish to infer them from the data $\{x_n\}_{n=1}^N$. (The standard deviation $\sigma$ is known.) In the remainder of this question we will derive an iterative algorithm for finding values for $\{\mu_k\}$ which maximize the likelihood,

$$P(\{x_n\}_{n=1}^N|\{\mu_k\}, \sigma) = \prod_n P(x_n|\{\mu_k\}, \sigma).$$

Let $L$ denote the log of the likelihood. Show that the derivative of the log likelihood with respect to $\mu_k$ is given by

$$\frac{\partial}{\partial \mu_k} L = \sum_n p_{k|n} \frac{(x_n - \mu_k)}{\sigma^2},$$

where $p_{k|n} \equiv P(k_n = k|x_n, \theta)$ appeared above at equation (24.5).

Show, neglecting terms in $\frac{\partial}{\partial \mu_k} P(k_n = k|x_n, \theta)$, that the second derivative is approximately given by

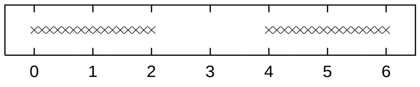
$$\frac{\partial^2}{\partial \mu_k^2} L = -\sum_n p_{k|n} \frac{1}{\sigma^2}.$$

Hence show that from an initial state $\mu_1, \mu_2$, an approximate Newton-Raphson step updates these parameters to $\mu_1', \mu_2'$, where

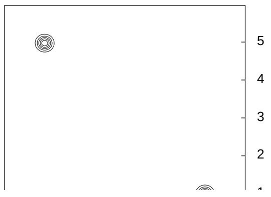
$$\mu_k' = \frac{\sum_n p_{k|n} x_n}{\sum_n p_{k|n}}.$$

[The Newton-Raphson method for maximizing $L(\mu)$ updates $\mu$ 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 $\mu_1$ and $\mu_2$ for the data set shown below. The data set consists of 32 points. Describe the peaks in your sketch and indicate their widths.



Solution to exercise 24.5 (p.338):
The likelihood function for the 32 data points is shown as a contour plot. The peaks are pretty-near centred on the points (1,5) and (5,1), and are pretty-near circular in their contours. The width of each of the peaks is a standard deviation of $\sigma/\sqrt{16} = 1/4$. The peaks are roughly Gaussian in shape.



**Assignment step.** The responsibilities are

$$r_k^{(n)} = \frac{\pi_k \frac{1}{(\sqrt{2\pi}\sigma_k)^I} \exp\left(-\frac{1}{\sigma_k^2}\, d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\right)}{\sum_{k'} \pi_k \frac{1}{(\sqrt{2\pi}\sigma_{k'})^I} \exp\left(-\frac{1}{\sigma_k^2}\, d(\mathbf{m}^{(k')}, \mathbf{x}^{(n)})\right)}. \tag{24.6}$$

where $I$ is the dimensionality.

**Update step.** Each cluster's parameters, $\mathbf{m}^{(k)}$, $\pi_k$, and $\sigma_k^2$, are adjusted to match the data points that it is responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \tag{24.7}$$

$$\sigma_k^2 = \frac{\sum_n r_k^{(n)} (\mathbf{x}^{(n)} - \mathbf{m}^{(k)})^2}{I R^{(k)}} \tag{24.8}$$

$$\pi_k = \frac{R^{(k)}}{\sum_k R^{(k)}} \tag{24.9}$$

where $R^{(k)}$ is the total responsibility of mean $k$,

$$R^{(k)} = \sum_n r_k^{(n)}, \tag{24.10}$$

and $I$ is the dimensionality of $\mathbf{x}$.

Algorithm 24.1. The soft K-means algorithm, version 2.

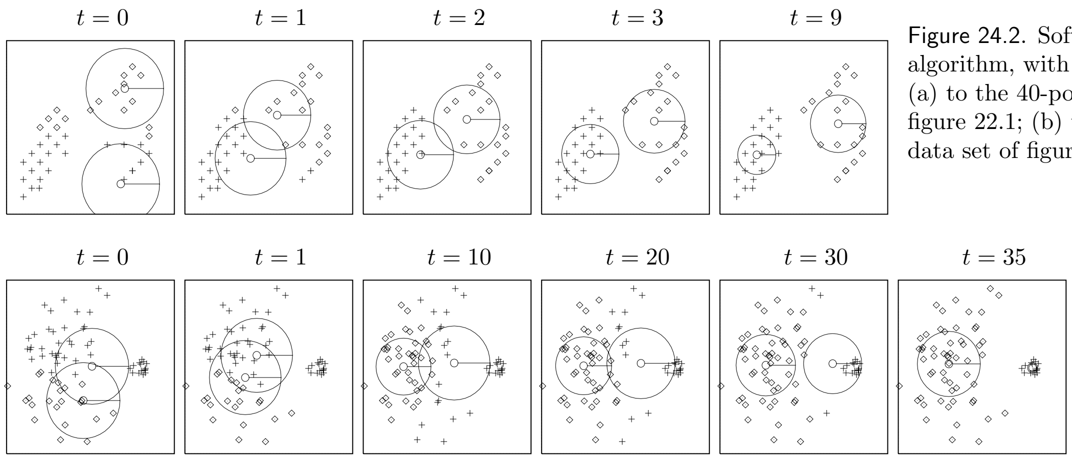


Figure 24.2. Soft K-means algorithm, with $K = 2$, applied (a) to the 40-point data set of figure 22.1; (b) to the little'n'large data set of figure 22.3.

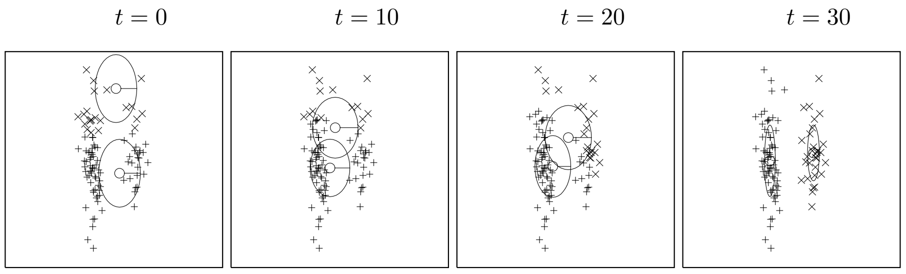


Figure 24.3. Soft K-means algorithm, version 3, applied to the data consisting of two cigar-shaped clusters. $K = 2$ (c.f. figure 22.4).

Notice that the algorithm you have derived for maximizing the likelihood is identical to the soft K-means algorithm (as in section 22.2). Now that it is clear that clustering can be viewed as mixture-density-modelling, we are now able to derive enhancements to the K-means algorithm which rectify the problems we noted earlier.

Algorithm 24.1 shows a version of the soft-K-means algorithm corresponding to a modelling assumption that each cluster is a spherical Gaussian having its own width (each cluster has its own $\beta^{(k)} = 1/\sigma_k^2$). The algorithm updates the lengthscales $\sigma_k$ for itself. The algorithm also includes cluster weight parameters $\pi_1, \pi_2, \dots \pi_K$ which also update themselves, allowing accurate modelling of data from clusters of unequal weights. This algorithm is demonstrated in figure 24.2 for two data sets that we've seen before. The second example shows that convergence can take a long time, but eventually the algorithm identifies the small cluster and the large cluster.

Soft K-means, version 2, is a maximum-likelihood algorithm for fitting a mixture of *spherical Gaussians* to data – 'spherical' meaning that the variance of the Gaussian is the same in all directions. This algorithm is still no good at modelling the cigar-shaped clusters of figure 22.4. If we wish to model the clusters by axis-aligned Gaussians with possibly-unequal variances, we replace the assignment rule (24.6) and the variance update rule (24.8) by the rules (24.11) and (24.12) displayed in algorithm 24.2.

A proof that the algorithm does indeed maximize the likelihood is deferred to a later chapter.

$$r_k^{(n)} = \frac{\pi_k \frac{1}{\prod_{i=1}^{I} \sqrt{2\pi}\sigma_i^{(k)}} \exp\left( - \sum_{i=1}^{I} (m_i^{(k)} - x_i^{(n)})^2 \Big/ 2(\sigma_i^{(k)})^2 \right)}{\sum_{k'} \text{ (numerator, with } k' \text{ in place of } k)} \tag{24.11}$$

$$\sigma_i^{2(k)} = \frac{\sum_n r_k^{(n)} (x_i^{(n)} - m_i^{(k)})^2}{R^{(k)}} \tag{24.12}$$

Algorithm 24.2. The soft K-means algorithm, version 3, which corresponds to a model of axis-aligned Gaussians.

This third version of soft K-means is demonstrated in figure 24.3 on the 'two cigars' data set of figure 22.4. After 30 iterations, the algorithm has correctly located the two clusters. Figure 24.4 shows the same algorithm applied to the little'n'large data set, where, again, the correct cluster locations are found.

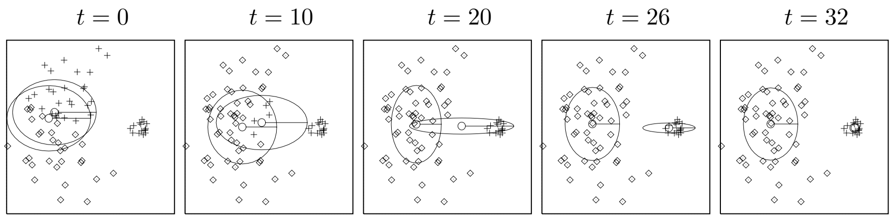


Figure 24.4. Soft K-means algorithm, version 3, applied to the little'n'large data set. $K = 2$.

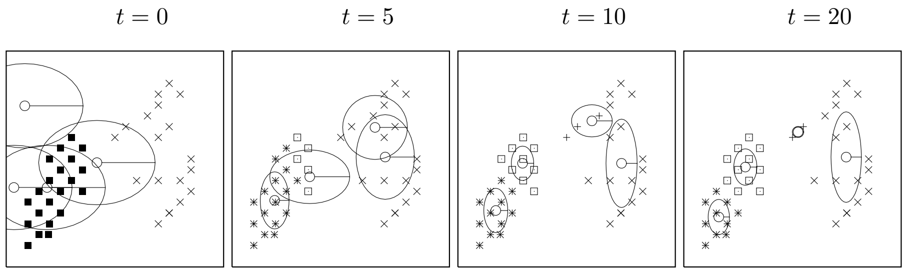


Figure 24.5. Soft K means algorithm applied to a data set of 40 points. $K = 4$. Notice that at convergence, one very small cluster has formed around two data points.

## 24.1 A fatal flaw of maximum likelihood

Finally, figure 24.5 sounds a cautionary note: when we fit $K = 4$ means to our first toy data set, we sometimes find that very small clusters form, covering just one or two data points. This is a pathological property of soft K-means clustering, versions 2 and 3.

Exercise 24.6:$^{B2}$ Investigate what happens if one mean $\mathbf{m}^{(k)}$ sits exactly on top of one data point; show that if the variance $\sigma_k^2$ is sufficiently small, then no return is possible: $\sigma_k^2$ becomes ever-smaller.

*KABOOM!*

Soft K-means can blow up. Put one of the two clusters exactly on one data point and let its variance go to zero; you can obtain an arbitrarily large likelihood! Maximum likelihood methods can break down absurdly by finding highly tuned models that fit part of the data perfectly. This phenomenon is known as overfitting. The reason we are not interested in these solutions with enormous likelihood is this: sure, these parameter-settings may have enormous posterior probability *density*, but the density is large over only a very small *volume* of parameter space. So the probability mass associated with these likelihood spikes is usually tiny.

This overfitting problem is one reason why we must say bye-bye to maximum likelihood. Another example of overfitting: Imagine we are interested in making a model of the surnames in a telephone directory; one theory says that 5% of people are called Smith, 3% Jones, 2% Davis, etc.; another theory says that 20% are called Lo and 15% are called Li; indeed we can imagine a high-dimensional continuum of such hypotheses. You tear a random page from the phone directory and pick a random name: it's `Shercliff`. In the light of this datum, what is the maximum likelihood hypothesis? Answer: the hypothesis that says that 100% of the surnames are `Shercliff`!

Other reasons for abandoning maximum likelihood:

1. Even if the likelihood does not have arbitrarily-large spikes, the maximum of the likelihood is often unrepresentative. Multiplying the likelihood by a prior and maximizing the posterior instead does not make the problem go away. The maximum of the posterior probability density is also often unrepresentative. Think back to the concept of typicality, which we encountered in chapter 5: in high dimensions, most of the probability mass is in a typical set whose properties are quite different from the points that have the maximum probability density.

2. The unrepresentative nature of maximum likelihood solutions also shows up in low-dimensional problems. As you may know from high-school statistics, the maximum likelihood estimator (24.4) for a Gaussian's standard deviation, $\sigma_N$, is a *biased* estimator, a topic that we'll take up in chapter 26.

## Exercises

Exercise 24.7:$^{C3}$ Make a version of the K-means algorithm which models the data as a mixture of $K$ arbitrary Gaussians, i.e., Gaussians that are not constrained to be axis-aligned.

Exercise 24.8:$^{B2}$ This exercise explores the idea that maximizing a probability density is a poor way to find a representative point. Consider a Gaussian distribution in a $k$-dimensional space, $P(\mathbf{w}) = (1/\sqrt{2\pi}\,\sigma_W)^k \exp(-\sum_1^k w_i^2/2\sigma_W^2)$. Show that 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 $\sigma_W$ by just 1%, and which contain equal total probability mass. Show that the maximum probability density is greater at the centre of the Gaussian with smaller $\sigma_W$ by a factor of $\sim \exp(0.01k) \simeq 20{,}000$.

In ill-posed problems, a typical posterior distribution is often 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.

### *Further reading*

The soft K-means algorithm is at the heart of the automatic classification package, 'AutoClass' (Hanson *et al.* 1991b; Hanson *et al.* 1991a).

# 25

# Useful Probability Distributions

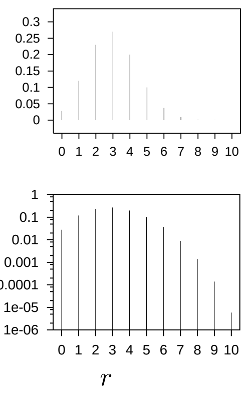


Figure 25.1. The binomial distribution $P(r|f{=}0.3, N{=}10)$, on a linear scale (top) and a logarithmic scale (bottom).

In Bayesian data modelling, there's a small collection of probability distributions that come up again and again. The purpose of this chapter is to introduce these distributions so that they won't be intimidating when encountered in combat situations.

There is no need to memorize any of them, except perhaps the Gaussian; if a distribution is important enough, it will memorize itself, and otherwise, it can easily be looked up.

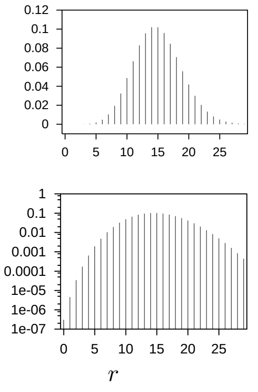


Figure 25.2. The Poisson distribution $P(r|\lambda{=}15)$, on a linear scale (top) and a logarithmic scale (bottom).

## 25.1 Distributions over integers

We already encountered the binomial distribution and the Poisson distribution on page 7.

The *binomial distribution* for an integer $r$ with parameters $f$ (the bias, $f \in [0, 1]$) and $N$ (the number of trials) is:

$$P(r|f, N) = \binom{N}{r} f^r (1-f)^{N-r} \;\; r \in \{0, 1, 2, \ldots, N\}. \tag{25.1}$$

The binomial distribution arises, for example, when we flip a bent coin, with bias $f$, $N$ times, and observe the number of heads, $r$.

The *Poisson distribution* with parameter $\lambda > 0$ is:

$$P(r|\lambda) = e^{-\lambda} \frac{\lambda^r}{r!} \;\; r \in \{0, 1, 2, \ldots\}. \tag{25.2}$$

The Poisson distribution arises, for example, when we count the number of photons $r$ that arrive in a pixel during a fixed interval, given that the mean intensity on the pixel corresponds to an average number of photons $\lambda$.

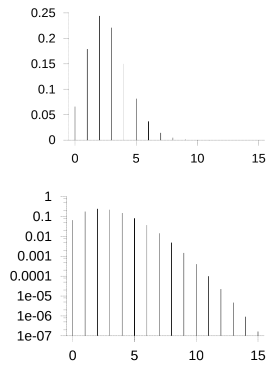


Figure 25.3. The Poisson distribution $P(r|\lambda{=}2.7)$, on a linear scale (top) and a logarithmic scale (bottom).

The *exponential distribution on integers*,

$$P(r|f) = f^r (1-f) \;\; r \in (0, 1, 2, \ldots, \infty), \tag{25.3}$$

arises in waiting problems. How long will you have to wait until a six is rolled, if a fair six-sided dice is rolled? Answer: the probability distribution of the number of rolls, $r$, is exponential over integers with parameter $f = 5/6$. The distribution may also be written

$$P(r|f) = (1-f) e^{-\lambda r} \;\; r \in (0, 1, 2, \ldots, \infty), \tag{25.4}$$

where $\lambda = \log(1/f)$.

## 25.2 Distributions over unbounded real numbers

The *Gaussian distribution* or normal distribution with mean $\mu$ and standard deviation $\sigma$ is

$$P(x|\mu,\sigma) = \frac{1}{Z}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad x \in (-\infty,\infty), \tag{25.5}$$

where

$$Z = \sqrt{2\pi\sigma^2}. \tag{25.6}$$

It is sometimes useful to work with the quantity $\tau \equiv 1/\sigma^2$, which is called the *precision* parameter of the Gaussian. The Gaussian distribution is widely used and often asserted to be a very common distribution in the real world, but I am sceptical about this assertion. Yes, *unimodal* distributions may be common; but a Gaussian is a special, rather extreme, unimodal distribution. It has very light tails: the log-probability-density decreases quadratically. The typical deviation of $x$ from $\mu$ is $\sigma$, but the respective probabilities that $x$ deviates from $\mu$ by more than $2\sigma$, $3\sigma$, $4\sigma$, and $5\sigma$, are 0.046, 0.003, $6\times10^{-5}$, and $6\times10^{-7}$. In my experience, deviations from a mean four or five times greater than the typical deviation may be rare, but they can happen more often than $6\times10^{-5}$! I therefore urge caution in the use of Gaussian distributions: if a variable that is modelled with a Gaussian actually has a heavier-tailed distribution, the rest of the model will contort itself to reduce the deviations of the outliers, like a sheet of paper being crushed by a rubber band.

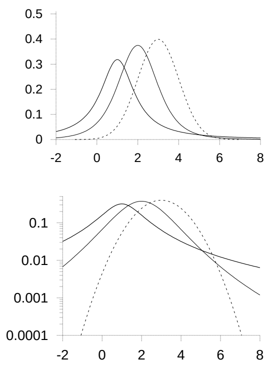


Figure 25.4. Two Student distributions, with parameters $(m,s)=(1,1)$ (heavy line) (a Cauchy distribution) and $(2,4)$ (light line), and a Gaussian distribution with mean $\mu=3$ and standard deviation $\sigma=3$, shown on linear vertical scales (top) and logarithmic vertical scales (bottom). Notice that the heavy tails of the Cauchy distribution are scarcely evident in the upper 'bell-shaped curve'.

Exercise 25.1:$^{B1}$ Pick a variable that is supposedly bell-shaped in probability distribution, gather data, and make a plot of the variable's empirical distribution. Show the distribution as a histogram on a logscale and investigate whether the tails are well-modelled by a Gaussian distribution. [One example of a variable to study is the amplitude of an audio signal.]

One distribution with heavier tails than a Gaussian is a *mixture of Gaussians*. A mixture of two Gaussians, for example, is defined by two means, two standard deviations, and two *mixing coefficients* $\pi_1$ and $\pi_2$, satisfying $\pi_1+\pi_2=1$, $\pi_i \geq 0$.

$$P(x|\mu_1,\sigma_1,\pi_1,\mu_2,\sigma_2,\pi_2) = \frac{\pi_1}{\sqrt{2\pi}\sigma_1}\exp\left(-\tfrac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{\pi_2}{\sqrt{2\pi}\sigma_2}\exp\left(-\tfrac{(x-\mu_2)^2}{2\sigma_2^2}\right)$$

If we take an appropriately weighted mixture of an infinite number of Gaussians, all having mean $\mu$, we obtain a *Student-t distribution*,

$$P(x|\mu,s,n) = \frac{1}{Z}\frac{1}{(1+(x-\mu)^2/(ns^2))^{(n+1)/2}} \tag{25.7}$$

where (CHECK)

$$Z = \sqrt{\pi n s^2}\frac{\Gamma(n/2)}{\Gamma((n+1)/2)} \tag{25.8}$$

and $n$ is called the number of degrees of freedom and $\Gamma$ is the gamma function. If $n>1$ then the Student distribution (25.7) has a mean and that mean is $\mu$. If $n>2$ the distribution also has a finite variance, $\sigma^2 = ns^2/(n-2)$. As $n\to\infty$, the Student distribution approaches the normal distribution with mean $\mu$ and standard deviation $s$. The Student distribution arises both in classical statistics (as the sampling-theoretic distribution of certain statistics) and in Bayesian inference (as the probability distribution of a variable coming from a Gaussian distribution whose standard deviation we aren't sure of).

In the special case $n = 1$, the Student distribution is called the *Cauchy distribution*.

A distribution whose tails are intermediate in heaviness between Student and Gaussian is the *biexponential distribution*,

$$P(x|\mu, s) = \frac{1}{Z} \exp\left(-\frac{|x-\mu|}{s}\right) \quad x \in (-\infty, \infty) \tag{25.9}$$

where

$$Z = 2s. \tag{25.10}$$

The *inverse-cosh distribution*

$$P(x|\beta) \propto \frac{1}{[\cosh(\beta x)]^{1/\beta}} \tag{25.11}$$

is a popular model in independent component analysis. In the limit of large $\beta$, the non-linearity becomes a step function and the probability distribution $P(x|\beta)$ becomes a biexponential distribution; In the limit $\beta \to 0$ $P(x|\beta)$ approaches a Gaussian with mean zero and variance $1/\beta$.

## *Distributions over* positive *real numbers*

The *exponential distribution*,

$$P(x|s) = \frac{1}{Z} \exp\left(-\frac{x}{s}\right) \quad x \in (0, \infty), \tag{25.12}$$

where

$$Z = s, \tag{25.13}$$

arises in waiting problems. How long will you have to wait for a bus in Poissonville, given that buses arrive independently at random with one every $s$ minutes on average? Answer: the probability distribution of your wait, $x$, is exponential with mean $s$.

The *gamma distribution* is like a Gaussian distribution, except whereas the Gaussian goes from $-\infty$ to $\infty$, gamma distributions go from 0 to $\infty$. The Gaussian distribution has two parameters $\mu$ and $\sigma$ which control the mean and width of the distribution. Similarly, the gamma distribution has two parameters. The gamma distribution is given by multiplying the one-parameter exponential distribution (25.12) by a polynomial, $x^{c-1}$. The exponent $c$ in the polynomial is the second parameter.

$$P(x|s, c) \;=\; \Gamma(x; s, c) \;=\; \frac{1}{Z}\left(\frac{x}{s}\right)^{c-1} \exp\left(-\frac{x}{s}\right), \quad 0 \le x < \infty \tag{25.14}$$

where

$$Z = \Gamma(c) s \tag{25.15}$$

This is a simple peaked distribution with mean $sc$ and variance $s^2 c$.

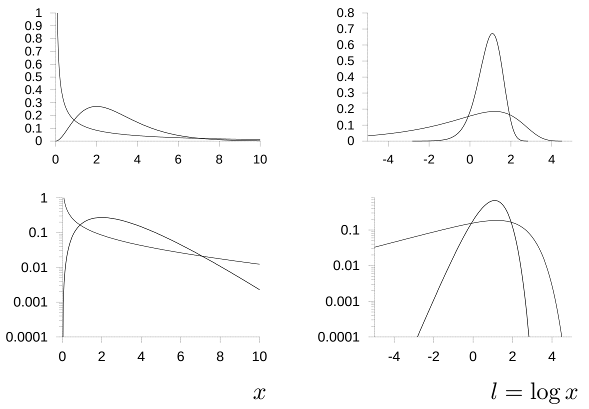


Figure 25.5. Two gamma distributions, with parameters $(s, c) = (1, 3)$ (heavy lines) and $10, 0.3$ (light lines), shown on linear vertical scales (top) and logarithmic vertical scales (bottom); and shown as a function of $x$ on the left (26.2) and $l = \log x$ on the right (25.17).

It is often natural to represent a positive real variable $x$ in terms of its logarithm $l = \log x$. The probability of density of $l$ is given by

$$P(l) = P(x(l)) \left| \frac{\partial x}{\partial l} \right| = P(x(l))x(l) \tag{25.16}$$

$$= \frac{1}{Z_l} \left( \frac{x}{s} \right)^c \exp\left( -\frac{x}{s} \right), \quad 0 \leq x < \infty \tag{25.17}$$

where

$$Z_l = \Gamma(c). \tag{25.18}$$

The gamma distribution is named after its normalizing constant – an odd convention, it seems to me!

The figures show a couple of gamma distributions as a function of $x$ and of $l$. Notice that where the original gamma distribution (26.2) may have a 'spike' at $x = 0$, the distribution over $l$ never has such a spike. The spike is an artefact of a bad choice of basis.

In the limit $sc = 1, c \to 0$, we obtain the noninformative prior for a scale parameter, the $1/x$ prior. This improper prior is called noninformative because it has no associated length scale, no characteristic value of $x$, so it prefers all values of $x$ equally. It is invariant under the reparameterization $x = mx$. If we transform the $1/x$ probability density into a density over $l = \log x$ we find the latter density is uniform.

Exercise 25.2:$^{B1}$ Imagine that we reparameterize a positive variable $x$ in terms of its cube root, $u = x^{1/3}$. If the probability density of $x$ is the improper distribution $1/x$, what is the probability density of $u$?

The gamma distribution is always a unimodal density over $l = \log x$, and, as can be seen in the figures, it is asymmetric. If $x$ has a gamma distribution, and we decide to work in terms of the inverse of $x$, $v = 1/x$, we obtain a new distribution, in which the density over $l$ is flipped left-for-right: the probability density of $v$ is called an *inverse-gamma distribution*,

$$P(v|s, c) = \frac{1}{Z_v} \left( \frac{1}{sv} \right)^{c+1} \exp\left( -\frac{1}{sv} \right), 0 \leq v < \infty \tag{25.19}$$

where (CHECK)

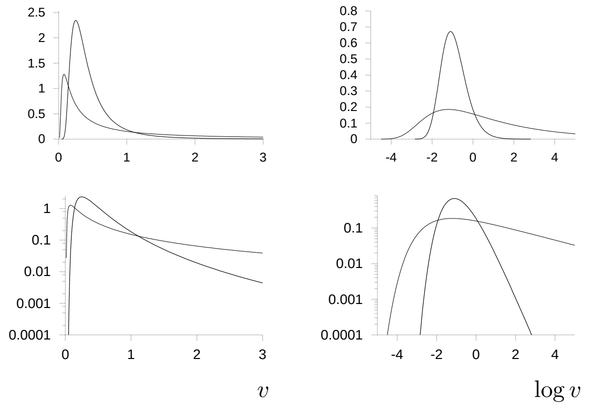


Figure 25.6. Two inverse gamma distributions, with parameters $(s, c) = (1, 3)$ (heavy lines) and $10, 0.3$ (light lines), shown on linear vertical scales (top) and logarithmic vertical scales (bottom); and shown as a function of $x$ on the left and $l = \log x$ on the right.

$$Z_v = \Gamma(c)/s. \tag{25.20}$$

Gamma and inverse gamma distributions crop up in many inference problems in which a positive quantity is inferred from data. Examples include inferring the variance of Gaussian noise from some noise samples, and inferring the rate parameter of a Poisson distribution from the count.

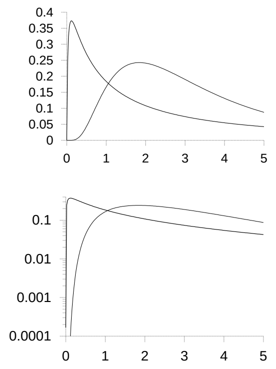


Figure 25.7. Two log-normal distributions, with parameters $(m, s) = (3, 1.8)$ (heavy line) and $(3, 0.7)$ (light line), shown on linear vertical scales (top) and logarithmic vertical scales (bottom). [Yes, they really do have the same value of the median, $m$]

### *Log-normal distribution*

Another distribution over a positive real number $x$ is the *log-normal* distribution, which is the distribution that results when $l = \log x$ has a normal distribution. We define $m$ to be the median value of $x$, and $s$ to be the standard deviation of $\log x$.

$$P(l|m, s) = \frac{1}{Z} \exp\left(-\frac{(l - \log m)^2}{2s^2}\right) \quad l \in (-\infty, \infty), \tag{25.21}$$

where

$$Z = \sqrt{2\pi s^2}, \tag{25.22}$$

implies

$$P(x|m, s) = \frac{1}{x} \exp\left(-\frac{(\log x - \log m)^2}{2s^2}\right) \quad l \in (-\infty, \infty), \tag{25.23}$$

### *Distributions over periodic variables**

A periodic variable $\theta$ is a real number $\in (0, 2\pi)$ having the property that $\theta = 0$ and $\theta = 2\pi$ are equivalent.

A distribution that plays for periodic variables the role played by the Gaussian distribution for real variables is the *Von Mises distribution*:

$$P(\theta|\mu, \beta) = \frac{1}{Z} \exp\left(\beta \cos(\theta - \mu)\right). \tag{25.24}$$

The normalizing constant is a Bessel function.

One distribution that arises from Brownian diffusion around the circle is the Gaussian distribution with wrap-around,

$$P(\theta|\mu,\sigma) = \sum_{n=-\infty}^{\infty} \text{Normal}(\theta;(\mu+2\pi n),\sigma) \;\; \theta \in (0,2\pi). \tag{25.25}$$

## 25.3 Distributions over probabilities

The *Beta distribution* is a probability density over a variable $p$ that is a probability, $p \in (0,1)$:

$$P(p|u_1,u_2) = \frac{1}{Z(u_1,u_2)} p^{u_1-1}(1-p)^{u_2-1}. \tag{25.26}$$

The parameters $u_1, u_2$ may take any positive value. The normalizing constant is the beta function. (CHECK)

$$Z(u_1,u_2) = \frac{\Gamma(u_1)\Gamma(u_2)}{\Gamma(u_1+u_2)} \tag{25.27}$$

Special cases include the uniform distribution – $u_1{=}1, u_2{=}1$; the Jeffreys prior – $u_1{=}0.5, u_2{=}0.5$; and the improper Laplace prior – $u_1{=}0, u_2{=}0$. If we transform the beta distribution to the corresponding density over the logit $l \equiv \log p/(1-p)$, we find it is always a pleasant bell-shaped density over $l$, while the density over $p$ may have singularities at $p = 0$ and $p = 1$.

### *More dimensions*

The *Dirichlet distribution* is a density over an $I$–dimensional vector $\mathbf{p}$ whose $I$ components are positive and sum to 1. The beta distribution is a special case of a Dirichlet distribution with $I = 2$. The Dirichlet distribution is parameterized by a measure $\mathbf{u}$ (a vector with all coefficients $u_i > 0$) which I will write here as $\mathbf{u} = \alpha\mathbf{m}$, where $\mathbf{m}$ is a normalized measure over the $I$ components ($\sum m_i = 1$), and $\alpha$ is positive:

$$P(\mathbf{p}|\alpha\mathbf{m}) = \frac{1}{Z(\alpha\mathbf{m})} \prod_{i=1}^{I} p_i^{\alpha m_i - 1} \delta\left(\textstyle\sum_i p_i - 1\right) \equiv \text{Dirichlet}^{(I)}(\mathbf{p}|\alpha\mathbf{m}) \tag{25.28}$$

The function $\delta(x)$ is the Dirac delta function which restricts the distribution to the simplex such that $\mathbf{p}$ is normalized, i.e., $\sum_i p_i = 1$. The normalizing constant of the Dirichlet distribution is:

$$Z(\alpha\mathbf{m}) = \prod_i \Gamma(\alpha m_i) / \Gamma(\alpha) \,. \tag{25.29}$$

The vector $\mathbf{m}$ is the mean of the probability distribution:

$$\int \text{Dirichlet}^{(I)}(\mathbf{p}|\alpha\mathbf{m}) \, \mathbf{p} \, \mathrm{d}^I\mathbf{p} = \mathbf{m} \tag{25.30}$$

The role of $\alpha$ can be characterized in two ways. First, the parameter $\alpha$ measures the sharpness of the distribution; it measures how different we expect typical samples $\mathbf{p}$ from the distribution to be from the mean $\mathbf{m}$, just as the precision $\tau = \frac{1}{\sigma^2}$ of a Gaussian measures how far samples stray from its mean.

A large value of $\alpha$ produces a distribution over $\mathbf{p}$ that is sharply peaked around $\mathbf{m}$. The effect of $\alpha$ can be visualized by drawing a typical sample from the distribution Dirichlet$^{(I)}(\mathbf{p}|\alpha\mathbf{m})$, with $\mathbf{m}$ set to the uniform vector $m_i = 1/I$, and making a Zipf plot, that is, a ranked plot of the values of the components $p_i$. It is traditional to plot both $p_i$ (vertical axis) and the rank (horizontal axis) on logarithmic scales so that power law relationships appear as straight lines. Figure 25.8 shows these plots for a single sample from ensembles with $I = 100$ and $I = 1000$ and with $\alpha$ from 0.1 to 1000. For large $\alpha$, the plot is shallow with many components having similar values. For small $\alpha$, typically one component $p_i$ receives an overwhelming share of the probability, and of the probability that remains to be shared among the other components, another component $p_{i'}$ receives a similarly large share. In the limit as $\alpha$ goes to zero, the plot tends to an increasingly steep power law.

Second, we can characterize the role of $\alpha$ in terms of the predictive distribution that results when we observe samples from $\mathbf{p}$ and obtain counts $\mathbf{F} = (F_1, F_2, \ldots F_I)$ of the possible outcomes. The value of $\alpha$ defines the number of samples from $\mathbf{p}$ that are required in order that the data dominate over the prior in predictions. See (MacKay and Peto 1995) for fun with Dirichlets.

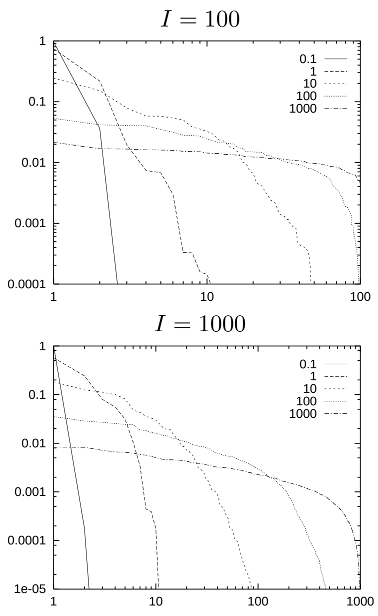


Figure 25.8. Zipf plots for random samples from Dirichlet distributions with various values of $\alpha = 0.1 \ldots 1000$. For each given $I$ and $\alpha$, one sample $\mathbf{p}$ from the Dirichlet distribution was generated. The Zipf plot shows the probabilities $p_i$, ranked by magnitude, versus their rank.

# 26

# *Exact Marginalization*

How can we avoid the exponential cost of complete enumeration of all hypotheses? Before we stoop to approximate methods, we explore two approaches to exact marginalization: first, marginalization over continuous variables (sometimes known as nuisance parameters) by doing *integrals*; and second, summation over discrete variables by message-passing.

Exact marginalization over continuous parameters is a macho activity enjoyed by those who are fluent in definite integration. This chapter uses gamma distributions; as was explained in the previous chapter, gamma distributions are a lot like Gaussian distributions, except that whereas the Gaussian goes from $-\infty$ to $\infty$, gamma distributions go from 0 to $\infty$.

## 26.1 Inferring the mean and variance of a Gaussian distribution

We discuss again the one-dimensional Gaussian distribution, parameterized by a mean $\mu$ and a standard deviation $\sigma$:

$$P(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \equiv \text{Normal}(x;\mu,\sigma^2). \tag{26.1}$$

When inferring these parameters, we must specify their prior distribution. The prior gives us the opportunity to include specific knowledge that we have about $\mu$ and $\sigma$ (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 section 23.2, we assumed a uniform prior over the range of parameters plotted. If we wish to be able to perform exact marginalizations, it may be useful to consider *conjugate priors*; these are priors whose functional form combines naturally with the likelihood such that the inferences have a convenient form.

### *Conjugate priors for $\mu$ and $\sigma$**

The conjugate prior for a mean $\mu$ is a Gaussian: we introduce two 'hyperparameters', $\mu_0$ and $\sigma_\mu$, which parameterize the prior on $\mu$, and write $P(\mu|\mu_0,\sigma_\mu) = \text{Normal}(\mu;\mu_0,\sigma_\mu)$. 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 $P(\mu) = \text{const.}$ is also an *improper* prior, that is, it is not normalizable.

The conjugate prior for a standard deviation $\sigma$ is a gamma distribution, which has two parameters $b_\beta$ and $c_\beta$. It is most convenient to define the prior density of the inverse variance $\beta = 1/\sigma^2$:

The inverse variance is sometimes called the *precision* parameter of the Gaussian.

$$P(\beta) = \Gamma(\beta; b_\beta, c_\beta) = \frac{1}{\Gamma(c_\beta)} \frac{\beta^{c_\beta - 1}}{b_\beta^{c_\beta}} \exp\left(-\frac{\beta}{b_\beta}\right), 0 \leq \beta < \infty \tag{26.2}$$

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 $\sigma$ by saying 'well, it could be 10, or it could be 1, or it could be 0.1, ...' Scale variables such as $\sigma$ 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 $\mu$ and $\sigma$. Using improper priors is viewed as distasteful in some circles, so let me excuse myself by saying it's for the sake of readability; if I included proper priors, the calculations could still be done but the key points would be obscured by the flood of extra parameters.

### *Maximum likelihood and marginalization: $\sigma_N$ and $\sigma_{N-1}$*

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 $\sigma_N$ and $\sigma_{N-1}$ buttons on their calculator. Let us recap the formulae, then derive them.

Given data $D = \{x_n\}_{n=1}^N$, an 'estimator' of $\mu$ is

$$\bar{x} \equiv \textstyle\sum_{n=1}^N x_n/N, \tag{26.3}$$

and two estimators of $\sigma$ are:

$$\sigma_N \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N}} \quad \text{and} \quad \sigma_{N-1} \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N-1}} \tag{26.4}$$

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 assumptions about the model and the data, 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 (that is, the specification of the model and all its probability distributions), and figuring out how to do the computations that implement inference in that space. The

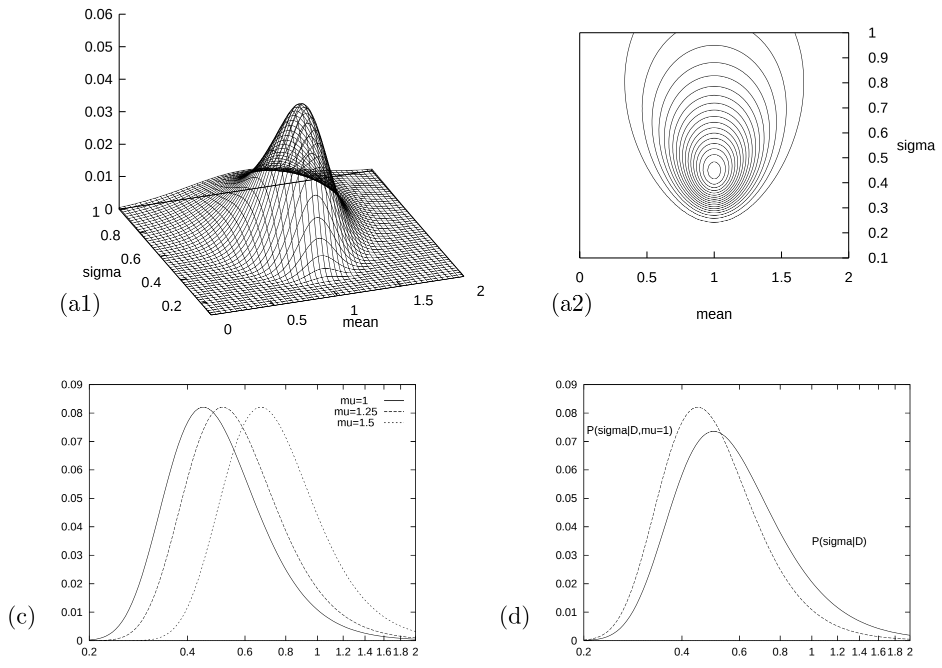


Figure 26.1. The likelihood function for the parameters of a Gaussian distribution, repeated from figure 23.5.
(a1,a2) Surface plot and contour plot of the log likelihood as a function of $\mu$ and $\sigma$. The data set of $N = 5$ points had mean $\bar{x} = 1.0$ and $S^2 = \sum(x - \bar{x})^2 = 1.0$. Notice that the maximum is skew in $\sigma$. The two estimators of standard deviation have values $\sigma_N = 0.45$ and $\sigma_{N-1} = 0.50$.
(c) The posterior probability of $\sigma$ for various fixed values of $\mu$.
(d) The posterior probability of $\sigma$, $P(\sigma|D)$, assuming a flat prior on $\mu$, obtained by projecting the probability mass in (a) onto the $\sigma$ axis. The maximum of $P(\sigma|D)$ is at $\sigma_{N-1}$. By contrast, the maximum of $P(\sigma|D, \mu = \bar{x}, \sigma)$ is at $\sigma_N$.

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 $\mu$ which, out of all the possible unbiased estimators of $\mu$, 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 $(\mu, \sigma)$. The estimator $\sigma_N$ is *biased*, however: the expectation of $\sigma_N$, given $\sigma$, averaging over many imagined experiments, is not $\sigma$.

Exercise 26.1:$^{A2}$ Give an intuitive explanation why the estimator $\sigma_N$ is biased.

This bias motivates the invention, in sampling theory, of $\sigma_{N-1}$, which can be shown to be an unbiased estimator. Or to be precise, it is $\sigma^2_{N-1}$ which is an unbiased estimator of $\sigma^2$.

We now look at some Bayesian inferences for this problem, assuming non-informative priors for $\mu$ and $\sigma$. 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 $\mu$ and $\sigma$ is proportional to the likelihood function illustrated by a contour plot in figure 26.1a. The log likelihood is:

$$\log P(\{x_n\}_{n=1}^N|\mu,\sigma) = -N\log(\sqrt{2\pi}\sigma) - \sum_n (x_n-\mu)^2/(2\sigma^2), \quad (26.5)$$

$$= -N\log(\sqrt{2\pi}\sigma) - [N(\mu-\bar{x})^2 + S]/(2\sigma^2), \quad (26.6)$$

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 $\mu$ and $\sigma$ is, using the improper priors:

$$P(\mu, \sigma | \{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu, \sigma)}{P(\{x_n\}_{n=1}^N)} \tag{26.7}$$

$$= \frac{\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N(\mu-\bar{x})^2+S}{2\sigma^2}\right) \frac{1}{\sigma_\mu} \frac{1}{\sigma}}{P(\{x_n\}_{n=1}^N)} \tag{26.8}$$

This function describes the answer to the question, 'given the data, and the noninformative priors, what might $\mu$ and $\sigma$ be?' It may be of interest to find the parameter values that maximize the posterior probability, though it should be 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.

Exercise 26.2:$^{B2}$ Differentiate the log likelihood with respect to $\mu$ and $\log\sigma$ and show that the maximum likelihood solution is: $\{\mu, \sigma\}_{\text{ML}} = \left\{\bar{x}, \sigma_N = \sqrt{S/N}\right\}$.

There is more to the posterior distribution than just its mode. As can be seen in figure 26.1a, the likelihood has a skew peak. As we increase $\sigma$, the width of the conditional distribution of $\mu$ increases (figure 24.1b). And if we fix $\mu$ to a sequence of values moving away from the sample mean $\bar{x}$, we obtain a sequence of conditional distributions over $\sigma$ whose maxima move to increasing values of $\sigma$ (figure 26.1c).

The posterior probability of $\mu$ given $\sigma$ is

$$P(\mu | \{x_n\}_{n=1}^N, \sigma) = \frac{P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu)}{P(\{x_n\}_{n=1}^N | \sigma)} \tag{26.9}$$

$$\propto \exp(-N(\mu - \bar{x})^2/(2\sigma^2)) \tag{26.10}$$

$$= \text{Normal}(\mu; \bar{x}, \sigma^2/N). \tag{26.11}$$

We note the familiar $\sigma/\sqrt{N}$ scaling of the error bars on $\mu$.

Let us now ask the question 'given the data, and the noninformative priors, what might $\sigma$ be?' This question differs from the first one we asked in that we are now not interested in $\mu$. This parameter must therefore be *marginalized* over. The posterior probability of $\sigma$ is:

$$P(\sigma | \{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N | \sigma) P(\sigma)}{P(\{x_n\}_{n=1}^N)}. \tag{26.12}$$

The data-dependent term $P(\{x_n\}_{n=1}^N | \sigma)$ appeared earlier as the normalizing constant in equation (26.9); one name for this quantity is the 'evidence', or marginal likelihood, for $\sigma$. We obtain the evidence for $\sigma$ by integrating out $\mu$; 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 $\sigma_\mu$. The Gaussian integral, $P(\{x_n\}_{n=1}^N | \sigma) = \int P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu)\, d\mu$, yields:

$$\log P(\{x_n\}_{n=1}^N | \sigma) = -N \log(\sqrt{2\pi}\sigma) - \frac{S}{2\sigma^2} + \log \frac{\sqrt{2\pi}\sigma/\sqrt{N}}{\sigma_\mu}. \tag{26.13}$$

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 $\sigma$. (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 $\sigma$, the additional volume factor $(\sigma/\sqrt{N})$ shifts the maximum from $\sigma_N$ to

$$\sigma_{N-1} = \sqrt{S/(N-1)} \tag{26.14}$$

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 $\mu$ from the data causes one dimension of noise to be gobbled up in unavoidable overfitting.

Figure 26.1d shows the posterior probability of $\sigma$, which is proportional to the marginal likelihood. This may be contrasted with the posterior probability of $\sigma$ with $\mu$ fixed to its most probable value, $\bar{x} = 1$, which is shown in figure 26.1c and d.

The final inference we might wish to make is 'given the data, what is $\mu$?'

Exercise 26.3:$^{B3}$ Marginalize over $\sigma$ and obtain the posterior marginal distribution of $\mu$, which is a Student t-distribution:

$$P(\mu|D) \propto 1/\left(N(\mu - \bar{x})^2 + S\right)^{N/2}. \tag{26.15}$$

## *Further reading*

A bible of exact marginalization is Bretthorst's (1988) book on Bayesian spectrum analysis and parameter estimation.

Solution to exercise 26.1 (p.353): 1. The data points are distributed with mean squared deviation $\sigma^2$ about the true mean. 2. The sample mean is unlikely to exactly equal the true mean. 3. The sample mean is the value of $\mu$ that minimizes the sum squared deviation of the data points from $\mu$. So the expected mean squared deviation from the sample mean is necessarily smaller than the mean squared deviation $\sigma^2$ about the true mean.

# 27

# *Exact Marginalization in Trellises*

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 23.1 (p.329). 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 (see chapter 4).

This chapter is a possible location for the first introduction of Markov chains, and/or hidden Markov models.

## 27.1 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

$$P(\mathbf{t}|\mathbf{y}) = \frac{P(\mathbf{y}|\mathbf{t})P(\mathbf{t})}{P(\mathbf{y})}. \qquad (27.1)$$

**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,

$$P(\mathbf{y}|\mathbf{t}) = \prod_{n=1}^{N} P(y_n|t_n). \tag{27.2}$$

For example, if the channel is a Gaussian channel with transmissions $\pm x$ and additive noise of standard deviation $\sigma$, then the probability density of the received signal $y_n$ in the two cases $t_n = 0, 1$ is

$$P(y_n|t_n{=}1) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - x)^2}{2\sigma^2}\right) \tag{27.3}$$

$$P(y_n|t_n{=}0) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n + x)^2}{2\sigma^2}\right). \tag{27.4}$$

From the point of view of decoding, all that matters is the *likelihood ratio* which for the case of the Gaussian channel is

$$\frac{P(y_n|t_n = 1)}{P(y_n|t_n = 0)} = \exp\left(\frac{2xy_n}{\sigma^2}\right). \tag{27.5}$$

Exercise 27.1:$^{A2}$ 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 in (27.1) is the normalizing constant

$$P(\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{y}|\mathbf{t})P(\mathbf{t}). \tag{27.6}$$

The complete solution to the codeword decoding problem is a list of all codewords and their probabilities as given by equation (27.1). 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 27.4, 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 (27.1) by *marginalizing* over the other bits.

$$P(t_n|\mathbf{y}) = \sum_{\{t_{n'}:\, n' \neq n\}} P(\mathbf{t}|\mathbf{y}) \tag{27.7}$$

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.

$$P(t_n = 1|\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t}|\mathbf{y})\delta(t_n = 1) \tag{27.8}$$

$$P(t_n = 1|\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t}|\mathbf{y})\delta(t_n = 0) \tag{27.9}$$

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.

Exercise 27.2:$^{B3}$ Prove that the probability of error of the optimal bitwise-decoder is closely related to the probability of error of the optimal codeword-decoder, by proving the following statement:

If a binary linear code has minimum distance $d_{\min}$, then, for any given channel, the codeword bit error probability of the optimal bitwise decoder, $P_b$, and the block error probability of the maximum likelihood decoder, $P_B$, are related by:

$$P_B \geq P_b \geq \frac{1}{2}\frac{d_{\min}}{N} P_B. \tag{27.10}$$

## 27.2 Codes and trellises*

In chapters 1 and 12, 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 are the parity-check bits. This means that the generator matrix of the code can be written

$$\mathbf{G}^{\mathsf{T}} = \left[ \begin{array}{c} \mathbf{I}_K \\ \mathbf{P} \end{array} \right], \tag{27.11}$$

and the parity-check matrix can be written

$$\mathbf{H} = \left[ \begin{array}{cc} \mathbf{P} & \mathbf{I}_M \end{array} \right], \tag{27.12}$$

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.

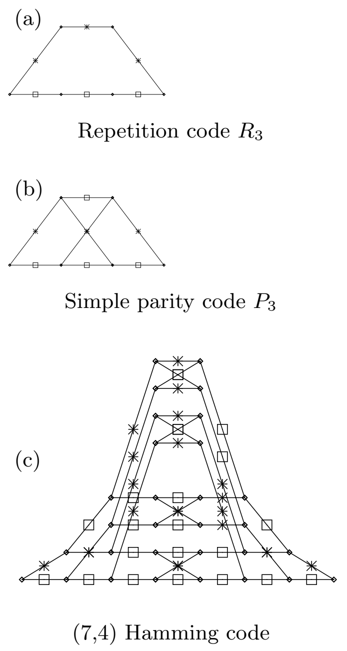


Figure 27.1. Examples of trellises. Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

## *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.

I need to reconsider this terminology: I would like to be able to talk about 'a four-state trellis' and 'the state as a function of time'; this usage conflicts with the idea that the encoder passes through an ordered sequence of 'states'.

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 $n$th 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 27.1(a,b,c) show the trellises corresponding to the repetition code $R_3$ which has $(N,K) = (3,1)$; the parity code $P_3$ with $(N,K) = (3,2)$; and the (7,4) Hamming code.

Exercise 27.3:$^{B2}$ Confirm that the sixteen codewords listed in figure **??** are generated by the trellis shown in figure 27.1(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 27.4.

Notice that for the linear trellises in figure 27.1, 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 27.4. But we now know enough to discuss the decoding problem.

| $\mathbf{t}$ | likelihood | posterior probability |
|---|---|---|
| `0000000` | 0.0275562 | 0.25 |
| `0001011` | 0.0001458 | 0.0013 |
| `0010111` | 0.0013122 | 0.012 |
| `0011100` | 0.0030618 | 0.027 |
| `0100110` | 0.0002268 | 0.0020 |
| `0101101` | 0.0000972 | 0.0009 |
| `0110001` | 0.0708588 | 0.63 |
| `0111010` | 0.0020412 | 0.018 |
| `1000101` | 0.0001458 | 0.0013 |
| `1001110` | 0.0000042 | 0.0000 |
| `1010010` | 0.0030618 | 0.027 |
| `1011001` | 0.0013122 | 0.012 |
| `1100011` | 0.0000972 | 0.0009 |
| `1101000` | 0.0002268 | 0.0020 |
| `1110100` | 0.0020412 | 0.018 |
| `1111111` | 0.0000108 | 0.0001 |

Figure 27.2. 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 | |
|---|---|---|---|---|
| | $P(y_n \mid t_n{=}1)$ | $P(y_n \mid t_n{=}0)$ | $P(t_n{=}1 \mid \mathbf{y})$ | $P(t_n{=}0 \mid \mathbf{y})$ |
| 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 27.3. Marginal posterior probabilities for the 7 bits under the posterior distribution of figure 27.2.

## 27.3 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. 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 27.4: Consider the case of a a single transmission from the Hamming (7,4) trellis shown in figure 27.1c.

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

$$\frac{P(y_1|x_1{=}1)}{P(y_1|x_1{=}0)} = \frac{0.1}{0.9}, \quad \frac{P(y_2|x_2{=}1)}{P(y_2|x_2{=}0)} = \frac{0.4}{0.6}, \quad \text{etc.} \tag{27.13}$$

How should this received signal be decoded?

(a) If we threshold the likelihoods at 0.5 to turn the signal into a binary received vector, we have $\mathbf{r} = (0, 0, 1, 0, 0, 0, 0)$, which decodes, using the decoder for the binary symmetric channel (chapter 1), into $\hat{\mathbf{t}} = (0, 0, 0, 0, 0, 0, 0)$.

This is not the optimal decoding procedure. Optimal inferences are always obtained by using Bayes' theorem.

(b) We can find the posterior probability over codewords by explicit enumeration of all sixteen codewords. This posterior distribution is shown in figure 27.2. 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 $\mathbf{t}$ = `0110001`. This is more than twice as probable as the answer found by thresholding, `0000000`.

Using the posterior probabilities shown in figure 27.2, we can also compute the posterior marginal distributions of each of the bits. The result is shown in figure 27.3. 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 27.5:$^{A2}$ 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.

[Hint: concentrate on the few codewords that have the largest probability.]

## 27.4 The min-sum algorithm*

*UNFINISHED SECTION.*

Connect this section to counting paths in the constrained channel, chapter 16, and to the message-passing chapter 4.

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.

# *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 27.4.

| Codeword | 0000000 | 0001011 | 0100110 | 1100011 | 0101101 |
|---|---|---|---|---|---|
| Span | 0000000 | 0001111 | 0111110 | 1111111 | 0111111 |

Figure 27.4. Some codewords and their spans

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.

## *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

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{27.14}$$

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:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{27.15}$$

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 27.5.

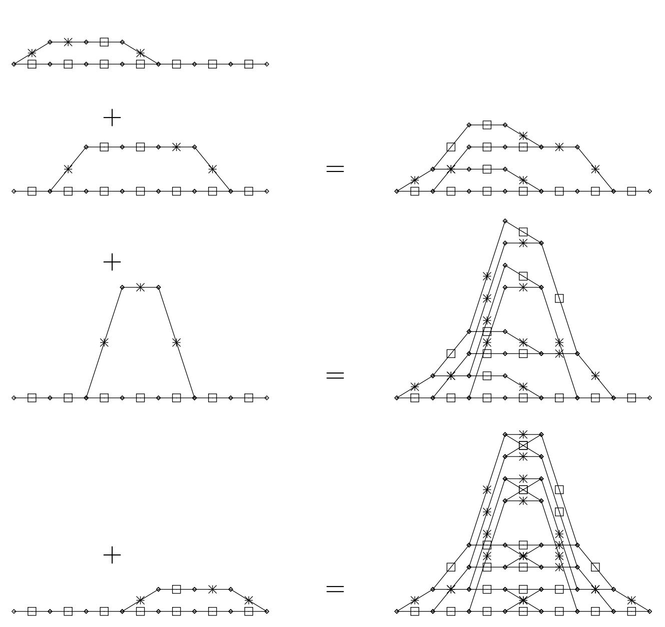


Figure 27.5. Trellises for four subcodes of the (7,4) Hamming code (left column), and the sequence of trellises that are made when constructing the trellis for the (7,4) Hamming code (right column).
Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

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...*

Another (7,4) Hamming code can be generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{27.16}$$

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 The parity-check matrix corresponding to this permutation is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{27.17}$$

The trellis obtained from the permuted matrix $\mathbf{G}$ given in equation (27.16) is shown in figure 27.6(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 27.1(c). We thus observe that *rearranging the order of the codeword bits can sometimes lead to smaller, simpler trellises.*

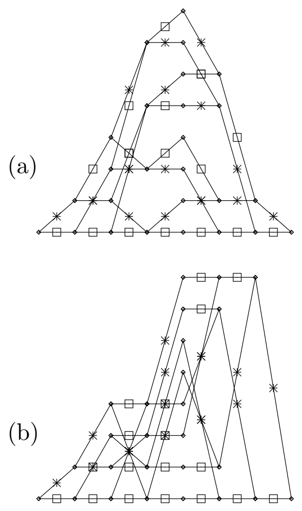


Figure 27.6. Trellises for the permuted (7,4) Hamming code generated from (a) the generator matrix by the method of figure 27.5; (b) the parity-check matrix by the method on page 363.
Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

## *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 syndrome 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 only a 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 27.6(b) shows the trellis corresponding to the parity-check matrix of equation (27.17).

## 27.5 Exercises

Exercise 27.6:$^{C3}$ Consider optimal decoders for the two decoding problems of section 27.1. Prove the following theorem, which relates the probability of error of these two decoders.

**Theorem 27.1** *If a code has minimum distance $d_{\min}$, then, for any given channel, the codeword bit error probability of the optimal bitwise decoder, $P_b$, and the block error probability of the maximum likelihood decoder, $P_B$, are related by:*

$$P_B \geq P_b \geq \frac{1}{2}\frac{d_{\min}}{N} P_B. \tag{27.18}$$

# *Solutions to Chapter 27's exercises*

Solution to exercise 27.5 (p.361): The posterior probability over codewords is:

| $\mathbf{t}$ | likelihood | posterior probability |
|---|---|---|
| 0000000 | 0.026 | 0.3006 |
| 0001011 | 0.00041 | 0.0047 |
| 0010111 | 0.0037 | 0.0423 |
| 0011100 | 0.015 | 0.1691 |
| 0100110 | 0.00041 | 0.0047 |
| 0101101 | 0.00010 | 0.0012 |
| 0110001 | 0.015 | 0.1691 |
| 0111010 | 0.0037 | 0.0423 |
| 1000101 | 0.00041 | 0.0047 |
| 1001110 | 0.00010 | 0.0012 |
| 1010010 | 0.015 | 0.1691 |
| 1011001 | 0.0037 | 0.0423 |
| 1100011 | 0.00010 | 0.0012 |
| 1101000 | 0.00041 | 0.0047 |
| 1110100 | 0.0037 | 0.0423 |
| 1111111 | 0.000058 | 0.0007 |

The most probable codeword is 0000000. The marginal posterior probabilities of all seven bits are:

| $n$ | Likelihood | | Posterior marginals | |
|---|---|---|---|---|
| | $P(y_n \mid t_n{=}1)$ | $P(y_n \mid t_n{=}0)$ | $P(t_n{=}1 \mid \mathbf{y})$ | $P(t_n{=}0 \mid \mathbf{y})$ |
| 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.

Solution to exercise 27.6 (p.364): Quick, rough proof of the theorem: Let $\mathbf{x}$ denote the difference between the reconstructed codeword and the transmitted codeword. For any given channel output $\mathbf{r}$, there is a posterior distribution over $\mathbf{x}$. This posterior distribution is positive only on vectors $\mathbf{x}$ belonging to the code; the sums that follow are over codewords $\mathbf{x}$. The block error probability is:

$$P_B = \sum_{\mathbf{x} \neq 0} P(\mathbf{x} \mid \mathbf{r}) \tag{27.19}$$

The average bit error probability, averaging over all bits in the codeword, is:

$$P_b = \sum_{\mathbf{x} \neq 0} P(\mathbf{x}|\mathbf{r}) \frac{w(\mathbf{x})}{N} \tag{27.20}$$

where $w(\mathbf{x})$ is the weight of codeword $\mathbf{x}$. Now the weights of the non-zero codewords satisfy

$$1 \geq \frac{w(\mathbf{x})}{N} \geq \frac{d_{\min}}{N}. \tag{27.21}$$

Substituting the inequalities (27.21) into the definitions (27.19,27.20), we obtain:

$$P_B \geq P_b \geq \frac{d_{\min}}{N} P_B, \tag{27.22}$$

which is a factor of two stronger, on the right, than the stated result (27.18). In making the proof watertight, I have weakened the result a little.

Careful proof of theorem 1. The first theorem relates the performance of the optimal block decoding algorithm and the optimal bitwise decoding algorithm.

We introduce another pair of decoding algorithms, called the block-guessing decoder and the bit-guessing decoder. The idea is that these two algorithms are similar to the optimal block decoder and the optimal bitwise decoder, but lend themselves more easily to analysis.

We now define these decoders. Let $\mathbf{x}$ denote the inferred codeword. For any given code:

**The Optimal Block decoder** returns the codeword $\mathbf{x}$ that maximizes the posterior probability $P(\mathbf{x}|\mathbf{r})$, which is proportional to the likelihood $P(\mathbf{r}|\mathbf{x})$.

The probability of error of this decoder is called $P_B$.

**The Optimal Bit decoder** returns for each of the $N$ bits, $x_n$, the value of $a$ that maximizes the posterior probability $P(x_n{=}a|\mathbf{r}) = \sum_{\mathbf{x}} P(\mathbf{x}|\mathbf{r})\delta(x_n{=}a)$.

The probability of error of this decoder is called $P_b$.

**The Block-guessing decoder** returns a random codeword $\mathbf{x}$ with probability distribution given by the posterior probability $P(\mathbf{x}|\mathbf{r})$.

The probability of error of this decoder is called $P_B^G$.

**The Bit-guessing decoder** returns for each of the $N$ bits, $x_n$, a random bit from the probability distribution $P(x_n{=}a|\mathbf{r})$.

The probability of error of this decoder is called $P_b^G$.

The theorem states that the optimal bit error probability $P_b$ is bounded above by $P_B$ and below by a given multiple of $P_B$ (27.18).

The left-hand inequality in (27.18) is trivially true – if a block is correct, all its constituent bits are correct; so if the optimal block decoder outperformed the optimal bit decoder, we could make a better bit decoder from the block decoder.

We prove the right-hand inequality by establishing the following two lemmas:

(a) the bit-guessing decoder is nearly as good as the optimal bit decoder:

$$P_b^G \leq 2P_b. \tag{27.23}$$

(b) the bit-guessing decoder's error probability is related to the block-guessing decoder's by

$$P_b^G \geq \frac{d_{\min}}{N} P_B^G. \tag{27.24}$$

Then since $P_B^G \geq P_B$, we have

$$P_b > \frac{1}{2} P_b^G \geq \frac{1}{2} \frac{d_{\min}}{N} P_B^G \geq \frac{1}{2} \frac{d_{\min}}{N} P_B. \tag{27.25}$$

We now prove the two lemmas.

Near-optimality of guessing: Consider first the case of a single bit, with posterior probability $\{p_0, p_1\}$. The optimal bit decoder has probability of error

$$P^{\text{optimal}} = \min(p_0, p_1). \tag{27.26}$$

The guessing decoder picks from 0 and 1. The truth is also distributed with the same probability. The probability that the guesser and the truth match is $p_0^2 + p_1^2$; the probability that they mismatch is the guessing error probability,

$$P^{\text{guess}} = 2p_0p_1 \leq 2\min(p_0, p_1) = 2P^{\text{optimal}}. \tag{27.27}$$

Since $P_b^G$ is the average of many such error probabilities, $P^{\text{guess}}$, and $P_b$ is the average of the coresponding optimal error probabilities, $P^{\text{optimal}}$, we obtain the desired relationship (27.23) between $P_b^G$ and $P_b$.

Relationship between bit error probability and block error probability: The bit-guessing and block-guessing decoders can be combined in a single system: We can draw a sample $x_n$ from the marginal distribution $P(x_n|\mathbf{r})$ by drawing a sample $(x_n, \mathbf{x})$ from the joint distribution $P(x_n, \mathbf{x}|\mathbf{r})$, then discarding the value of $\mathbf{x}$.

We can distinguish between two cases: the discarded value of $\mathbf{x}$ is the correct codeword, or not. The probability of bit error for the bit-guessing decoder can then be written as a sum of two terms:

$$\begin{aligned} P_b^G &= P(\mathbf{x}\text{ correct})P(\text{bit error}|\mathbf{x}\text{ correct}) \\ &\quad +P(\mathbf{x}\text{ incorrect})P(\text{bit error}|\mathbf{x}\text{ incorrect}) \\ &= 0 + P_B^G P(\text{bit error}|\mathbf{x}\text{ incorrect}). \end{aligned}$$

Now, whenever the guessed $\mathbf{x}$ is incorrect, the true $\mathbf{x}$ must differ from it in at least $d$ bits, so the probability of bit error in these cases is at least $d/N$. So

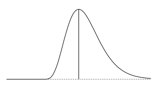
$$P_b^G \geq \frac{d}{N} P_B^G.$$

QED. □

# 28 Laplace's method

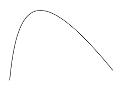
The idea behind the Laplace approximation is very simple. We assume that an unnormalized probability density $P^*(x)$, whose normalizing constant

$$Z_P \equiv \int P^*(x)\,\mathrm{d}x \tag{28.1}$$



is of interest, has a peak at a point $x_0$.

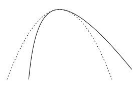
We Taylor-expand the logarithm of $P^*(x)$ around this peak:



$$\ln P^*(x) \simeq \ln P^*(x_0) - \frac{c}{2}(x-x_0)^2 + \dots, \tag{28.2}$$

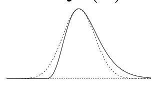
where

$$c = -\left.\frac{\partial^2}{\partial x^2}\ln P^*(x)\right|_{x=x_0}. \tag{28.3}$$



We then approximate $P^*(x)$ by an unnormalized Gaussian,

$$Q^*(x) \equiv P^*(x_0)\exp\left[-\frac{c}{2}(x-x_0)^2\right], \tag{28.4}$$



and we approximate the normalizing constant $Z_P$ by the normalizing constant of this Gaussian,

$$Z_Q = P^*(x_0)\sqrt{\frac{2\pi}{c}}. \tag{28.5}$$

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:

$$A_{ij} = -\left.\frac{\partial^2}{\partial x_i \partial x_j}\ln P^*(\mathbf{x})\right|_{\mathbf{x}=\mathbf{x}_0}, \tag{28.6}$$

so that the expansion (28.2) is generalized to

$$\ln P^*(\mathbf{x}) \simeq \ln P^*(\mathbf{x}_0) - \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^{\mathsf{T}}\mathbf{A}(\mathbf{x}-\mathbf{x}_0) + \dots, \tag{28.7}$$

then the normalizing constant can be approximated by:

$$Z_P \simeq Z_Q = P^*(x_0)\frac{1}{\sqrt{\det \frac{1}{2\pi}\mathbf{A}}} = P^*(x_0)\sqrt{\frac{(2\pi)^K}{\det \mathbf{A}}}. \tag{28.8}$$

Physicists also call this widely-used approximation the *saddle-point approximation.*

The fact that the normalizing constant of a Gaussian is given by

$$\int d^K\mathbf{x}\, \exp\left[-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}\right] = \sqrt{\frac{(2\pi)^K}{\det \mathbf{A}}} \tag{28.9}$$

can be proved by making an orthogonal transformation into the basis $\mathbf{u}$ in which $\mathbf{A}$ is transformed into a diagonal matrix. The integral then separates into a product of one-dimensional integrals, each of the form

$$\int du_i \exp\left[-\frac{1}{2}\lambda_i u_i^2\right] = \sqrt{\frac{2\pi}{\lambda_i}}. \tag{28.10}$$

The product of the eigenvalues $\lambda_i$ is the determinant of $\mathbf{A}$.

Predictions can be made using the approximation $Q$.

## 28.1 Examples

*Under construction*

In the maximum likelihood chapter we found the second derivative for a few examples.

Start with a tiny example – note that for the mean of a Gaussian with known $\sigma$ the approximation is exact. For several other models (eg interpolation) the approximation is exact.

# 29

# *Model Comparison and Occam's Razor*

*Under construction.*

## 29.1 Probability theory and Occam's razor

Bayesian probability theory provides a unifying framework for data modelling. A Bayesian data-modeller's aim is to develop probabilistic models that are well-matched to the data, and make optimal predictions using those models. The Bayesian framework has several advantages.

Probability theory forces us to make explicit all our modelling assumptions, whereupon our inferences are mechanistic. Once a model space has been defined, then, whatever question we wish to pose, the rules of probability theory give a unique answer which consistently takes into account all the given information. This is in contrast to sampling theory statistics, in which one must invent 'estimators' of quantities of interest and then choose 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.

Bayesian inference satisfies the likelihood principle: our inferences depend only on the probabilities assigned to the data that were received, not on properties of other data sets which might have occurred but did not.

Probabilistic modelling handles uncertainty in a natural manner. There is a unique prescription (marginalization) for incorporating uncertainty about parameters into our predictions of other variables.

Finally, Bayesian model comparison embodies *Occam's razor*, the principle that states a preference for simple models.

### *Motivations for Occam's razor*

If several explanations are compatible with a set of observations, Occam's razor advises us to buy the least complex explanation. This principle is often advocated for one of two reasons: the first is aesthetic ("A theory with mathematical beauty is more likely to be correct than an ugly one that fits some experimental data" (Paul Dirac)); the second reason is the past empirical success of Occam's razor. However there is a different justification for Occam's razor, namely:

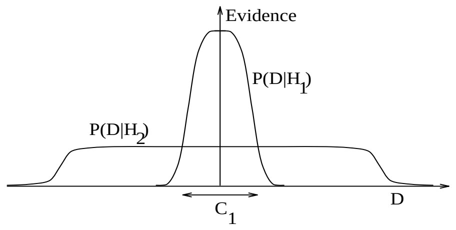


Figure 29.1. Why Bayesian inference embodies Occam's razor. This figure gives the basic intuition for why complex models can turn out to be less probable. The horizontal axis represents the space of possible data sets $D$. Bayes' theorem rewards models in proportion to how much they *predicted* the data that occurred. These predictions are quantified by a normalized probability distribution on $D$. This probability of the data given model $\mathcal{H}_i$, $P(D|\mathcal{H}_i)$, is called the evidence for $\mathcal{H}_i$.
A simple model $\mathcal{H}_1$ makes only a limited range of predictions, shown by $P(D|\mathcal{H}_1)$; a more powerful model $\mathcal{H}_2$, that has, for example, more free parameters than $\mathcal{H}_1$, is able to predict a greater variety of data sets. This means, however, that $\mathcal{H}_2$ does not predict the data sets in region $\mathcal{C}_1$ as strongly as $\mathcal{H}_1$. Suppose that equal prior probabilities have been assigned to the two models. Then, if the data set falls in region $\mathcal{C}_1$, the *less powerful* model $\mathcal{H}_1$ will be the *more probable* model.

Coherent inference automatically embodies Occam's razor, quantitatively.

### *Model comparison and Occam's razor*

We evaluate the plausibility of two alternative theories $\mathcal{H}_1$ and $\mathcal{H}_2$ in the light of data $D$ as follows: using Bayes' theorem, we relate the plausibility of model $\mathcal{H}_1$ given the data, $P(\mathcal{H}_1|D)$, to the predictions made by the model about the data, $P(D|\mathcal{H}_1)$, and the prior plausibility of $\mathcal{H}_1$, $P(\mathcal{H}_1)$. This gives the following probability ratio between theory $\mathcal{H}_1$ and theory $\mathcal{H}_2$:

$$\frac{P(\mathcal{H}_1|D)}{P(\mathcal{H}_2|D)} = \frac{P(\mathcal{H}_1)}{P(\mathcal{H}_2)} \frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)}. \tag{29.1}$$

The first ratio $(P(\mathcal{H}_1)/P(\mathcal{H}_2))$ on the right hand side measures how much our initial beliefs favoured $\mathcal{H}_1$ over $\mathcal{H}_2$. The second ratio expresses how well the observed data were predicted by $\mathcal{H}_1$, compared to $\mathcal{H}_2$.

How does this relate to Occam's razor, when $\mathcal{H}_1$ is a simpler model than $\mathcal{H}_2$? The first ratio $(P(\mathcal{H}_1)/P(\mathcal{H}_2))$ gives us the opportunity, if we wish, to insert a prior bias in favour of $\mathcal{H}_1$ on aesthetic grounds, or on the basis of experience. This would correspond to the aesthetic and empirical motivations for Occam's razor mentioned earlier. But such a prior bias is not necessary: the second ratio, the data-dependent factor, embodies Occam's razor *automatically*. Simple models tend to make precise predictions. Complex models, by their nature, are capable of making a greater variety of predictions (figure 29.1). So if $\mathcal{H}_2$ is a more complex model, it must spread its predictive probability $P(D|\mathcal{H}_2)$ more thinly over the data space than $\mathcal{H}_1$. Thus, in the case where the data are compatible with both theories, the simpler $\mathcal{H}_1$ will turn out more probable than $\mathcal{H}_2$, without our having to express any subjective dislike for complex models. Our subjective prior just needs to assign equal prior probabilities to the possibilities of simplicity and complexity. Probability theory then allows the observed data to express their opinion.

Let us turn to a simple example. Here is a sequence of numbers:

$$-1, 3, 7, 11$$

The task is to predict what the next two numbers are likely to be, and infer what the underlying process probably was, that gave rise to this sequence. A popular answer to this question is the prediction "15, 19", with the explanation "add 4 to the previous number".

What about the alternative answer "$-19.9, 1043.8$" with the underlying rule being: "get the next number from the previous number, $x$, by evaluating

$-x^3/11 + 9/11x^2 + 23/11$"? I assume that this prediction seems rather less plausible. But the second rule fits the data (-1, 3, 7, 11) just as well as the rule "add 4". So why should we find it less plausible? Let us give labels to the two general theories:

$\mathcal{H}_a -$ the sequence is an *arithmetic* progression, 'add $n$', where $n$ is an integer.

$\mathcal{H}_c -$ the sequence is generated by a *cubic* function of the form $x \rightarrow cx^3 + dx^2 + e$, where $c$, $d$ and $e$ are fractions.

One reason for finding the second explanation, $\mathcal{H}_c$, less plausible, might be that arithmetic progressions are more frequently encountered than cubic functions. This would put a bias in the prior probability ratio $P(\mathcal{H}_a)/P(\mathcal{H}_c)$ in equation (29.1). But let us give the two theories equal prior probabilities, and concentrate on what the data have to say. How well did each theory predict the data?

To obtain $P(D|\mathcal{H}_a)$ we must specify the probability distribution that each model assigns to its parameters. First, $\mathcal{H}_a$ depends on the added integer $n$, and the first number in the sequence. Let us say that these numbers could each have been anywhere between -50 and 50. Then since only the pair of values $\{n{=}4, \text{ first number}{=}-1\}$ give rise to the observed data $D =$ (-1, 3, 7, 11), the probability of the data, given $\mathcal{H}_a$, is:

$$P(D|\mathcal{H}_a) = \frac{1}{101}\frac{1}{101} = 0.00010$$

To evaluate $P(D|\mathcal{H}_c)$, we must similarly say what values the fractions $c, d$ and $e$ might take on. [I choose to represent these numbers as fractions rather than real numbers because if we used real numbers, the model would assign, relative to $\mathcal{H}_a$, an infinitessimal probability to $D$. Real parameters are the norm however, and are assumed in the rest of this chapter.] A reasonable prior might state that for each fraction the numerator could be any number between -50 and 50, and the denominator is any number between 1 and 50. As for the initial value in the sequence, let us leave its probability distribution the same as in $\mathcal{H}_a$. There are four ways of expressing the fraction $c = -1/11 = -2/22 = -3/33 = -4/44$ under this prior, and similarly there are four and two possible solutions for $d$ and $e$, respectively. So the probability of the observed data, given $\mathcal{H}_c$, is found to be:

$$\begin{aligned} P(D|\mathcal{H}_c) &= \left(\frac{1}{101}\right)\left(\frac{4}{101}\frac{1}{50}\right)\left(\frac{4}{101}\frac{1}{50}\right)\left(\frac{2}{101}\frac{1}{50}\right) \\ &= 0.0000000000025 = 2.5 \times 10^{-12} \end{aligned}$$

Thus comparing $P(D|\mathcal{H}_c)$ with $P(D|\mathcal{H}_a) = 0.00010$, even if our prior probabilities for $\mathcal{H}_a$ and $\mathcal{H}_c$ are equal, the odds, $P(D|\mathcal{H}_a) : P(D|\mathcal{H}_c)$, in favour of $\mathcal{H}_a$ over $\mathcal{H}_c$, given the sequence $D =$ (-1, 3, 7, 11), are about forty million to one.

This answer depends on several subjective assumptions; in particular, the probability assigned to the free parameters $n$, $c$, $d$, $e$ of the theories. Bayesians make no apologies for this: there is no such thing as inference or prediction without assumptions. However, the quantitative details of the prior probabilities have no effect on the qualitative Occam's razor effect; the complex theory $\mathcal{H}_c$ always suffers an 'Occam factor' because it has more parameters,

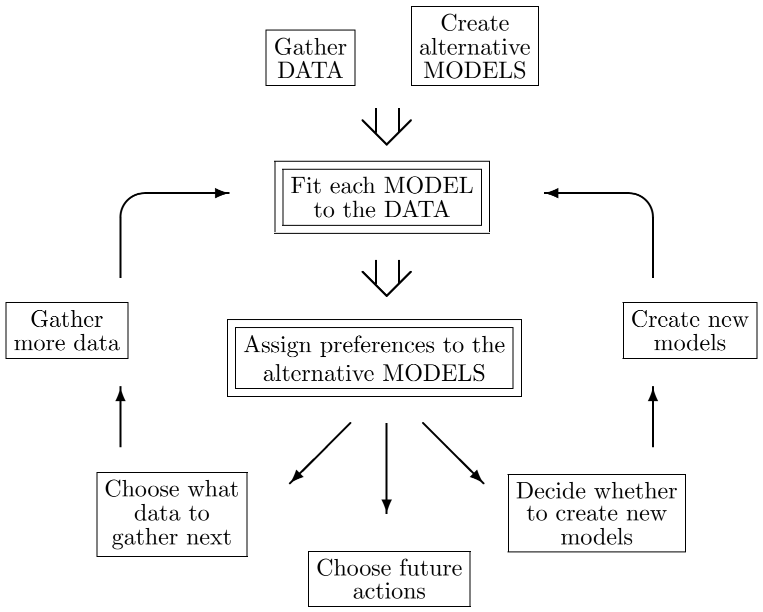

Gather DATA
Create alternative MODELS
Fit each MODEL to the DATA
Gather more data
Assign preferences to the alternative MODELS
Create new models
Choose what data to gather next
Choose future actions
Decide whether to create new models

Figure 29.2. Where Bayesian inference fits into the data modelling process.
This figure illustrates an abstraction of the part of the scientific process in which data is collected and modelled. In particular, this figure applies to pattern classification, learning, interpolation, etc.. The two double–framed boxes denote the two steps which involve *inference*. It is only in those two steps that Bayes's theorem can be used. Bayes does not tell you how to invent models, for example.
The first box, 'fitting each model to the data', is the task of inferring what the model parameters might be given the model and the data. Bayes may be used to find the most probable parameter values, and error bars on those parameters. The result of applying Bayes to this problem is often little different from the answers given by orthodox statistics.
The second inference task, model comparison in the light of the data, is where Bayes is in a class of its own. This second inference problem requires a quantitative Occam's razor to penalise over–complex models. Bayes can assign objective preferences to the alternative models in a way that automatically embodies Occam's razor.

and so can predict a greater variety of data sets (figure 29.1). This was only a small example, and there were only four data points; as we move to larger and more sophisticated problems the magnitude of the Occam factors typically increases, and the degree to which our inferences are influenced by the quantitative details of our subjective assumptions becomes smaller.

### *Bayesian methods and data analysis*

Let us now relate the discussion above to real problems in data analysis.

There are countless problems in science, statistics and technology which require that, given a limited data set, preferences be assigned to alternative models of differing complexities. For example, two alternative hypotheses accounting for planetary motion are Mr. Inquisition's geocentric model based on 'epicycles', and Mr. Copernicus's simpler model of the solar system. The epicyclic model fits data on planetary motion at least as well as the Copernican model, but does so using more parameters. Coincidentally for Mr. Inquisition, two of the extra epicyclic parameters for every planet are found to be identical to the period and radius of the sun's 'cycle around the earth'. Intuitively we find Mr. Copernicus's theory more probable.

### *The mechanism of the Bayesian razor: the evidence and the Occam factor*

Two levels of inference can often be distinguished in the process of data modelling. At the first level of inference, we assume that a particular model is true, and we fit that model to the data, i.e., we infer what values its free parameters should plausibly take, given the data. The results of this inference are often summarized by the most probable parameter values, and error bars on those parameters. This analysis is repeated for each model. The second level of inference is the task of model comparison. Here we wish to compare the models in the light of the data, and assign some sort of preference or ranking to the alternatives.

Note that both levels of *inference* are distinct from *decision theory*. The goal of inference is, given a defined hypothesis space and a particular data set, to assign probabilities to hypotheses. Decision theory typically chooses between alternative *actions*

Bayesian methods are able consistently and quantitatively to solve both the inference tasks. There is a popular myth that states that Bayesian methods only differ from orthodox statistical methods by the inclusion of subjective priors which are difficult to assign, and usually don't make much difference to the conclusions. It is true that, at the first level of inference, a Bayesian's results will often differ little from the outcome of an orthodox attack. What is not widely appreciated is how a Bayesian performs the second level of inference; this section will therefore focus on Bayesian model comparison.

Model comparison is a difficult task because it is not possible simply to choose the model that fits the data best: more complex models can always fit the data better, so the maximum likelihood model choice would lead us inevitably to implausible, over-parameterized models which generalize poorly. Occam's razor is needed.

Let us write down Bayes' theorem for the two levels of inference described above, so as to see explicitly how Bayesian model comparison works. Each model $\mathcal{H}_i$ is assumed to have a vector of parameters $\mathbf{w}$. A model is defined by a collection of probability distributions: a 'prior' distribution $P(\mathbf{w}|\mathcal{H}_i)$ which states what values the model's parameters might be expected to take; and a set of conditional distributions, one for each value of $\mathbf{w}$, defining the predictions $P(D|\mathbf{w},\mathcal{H}_i)$ that the model makes about the data $D$.

1. **Model fitting.** At the first level of inference, we assume that one model, the $i$th, say, is true, and we infer what the model's parameters $\mathbf{w}$ might be, given the data $D$. Using Bayes' theorem, the **posterior probability** of the parameters $\mathbf{w}$ is:

$$P(\mathbf{w}|D,\mathcal{H}_i) = \frac{P(D|\mathbf{w},\mathcal{H}_i)P(\mathbf{w}|\mathcal{H}_i)}{P(D|\mathcal{H}_i)}, \tag{29.2}$$

that is,

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}.$$

The normalizing constant $P(D|\mathcal{H}_i)$ is commonly ignored since it is irrelevant to the first level of inference, i.e., the inference of $\mathbf{w}$; but it becomes important in the second level of inference, and we name it the **evidence** for $\mathcal{H}_i$. It is common practice to use gradient-based methods to find the maximum of the posterior, which defines the most probable value for the parameters, $\mathbf{w}_{\text{MP}}$; it is then usual to summarize the posterior distribution by the value of $\mathbf{w}_{\text{MP}}$, and error bars or confidence intervals on these best fit parameters. Error bars can be obtained from the curvature of the posterior; evaluating the Hessian at $\mathbf{w}_{\text{MP}}$, $\mathbf{A} = -\nabla\nabla \log P(\mathbf{w}|D,\mathcal{H}_i)|_{\mathbf{w}_{\text{MP}}}$, and Taylor-expanding the log posterior probability with $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{MP}}$:

$$P(\mathbf{w}|D,\mathcal{H}_i) \simeq P(\mathbf{w}_{\text{MP}}|D,\mathcal{H}_i) \exp\left(-\tfrac{1}{2}\Delta\mathbf{w}^{\mathsf{T}}\mathbf{A}\Delta\mathbf{w}\right), \tag{29.3}$$

we see that the posterior can be locally approximated as a Gaussian with covariance matrix (equivalent to error bars) $\mathbf{A}^{-1}$. [Whether this approximation is good or not will depend on the problem we are solving. Indeed, the maximum and mean of the posterior distribution have no fundamental status in Bayesian inference—they both change under non-linear reparameterizations. Maximization of a posterior probability is only useful if an approximation like equation (29.3) gives a good summary of the distribution.]

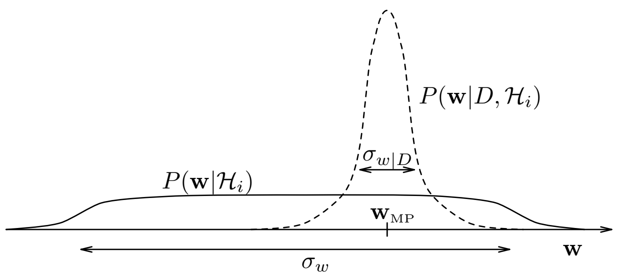


Figure 29.3. The Occam factor. This figure shows the quantities that determine the Occam factor for a hypothesis $\mathcal{H}_i$ having a single parameter $\mathbf{w}$. The prior distribution (solid line) for the parameter has width $\sigma_w$. The posterior distribution (dashed line) has a single peak at $\mathbf{w}_{\text{MP}}$ with characteristic width $\sigma_{w|D}$. The Occam factor is

$$\sigma_{w|D} P(\mathbf{w}_{\text{MP}}|\mathcal{H}_i) = \frac{\sigma_{w|D}}{\sigma_w}.$$

2. **Model comparison.** At the second level of inference, we wish to infer which model is most plausible given the data. The posterior probability of each model is:

$$P(\mathcal{H}_i|D) \propto P(D|\mathcal{H}_i)P(\mathcal{H}_i). \tag{29.4}$$

Notice that the data-dependent term $P(D|\mathcal{H}_i)$ is the evidence for $\mathcal{H}_i$, which appeared as the normalizing constant in (29.2). The second term, $P(\mathcal{H}_i)$, is the subjective prior over our hypothesis space which expresses how plausible we thought the alternative models were before the data arrived. Assuming that we choose to assign equal priors $P(\mathcal{H}_i)$ to the alternative models, *models $\mathcal{H}_i$ are ranked by evaluating the evidence.* The normalizing constant $P(D) = \sum_i P(D|\mathcal{H}_i)P(\mathcal{H}_i)$ has been omitted from equation (29.4) because in the data modelling process we may develop new models after the data have arrived, when an inadequacy of the first models is detected, for example. Inference is open ended: we continually seek more probable models to account for the data we gather.

To reiterate the key concept: to rank alternative models $\mathcal{H}_i$, a Bayesian evaluates the evidence $P(D|\mathcal{H}_i)$. This concept is very general: the evidence can be evaluated for parametric and 'non-parametric' models alike; whatever our data modelling task, a regression problem, a classification problem, or a density estimation problem, the evidence is a transportable quantity for comparing alternative models. In all these cases the evidence naturally embodies Occam's razor.

## *Evaluating the evidence*

Let us now study the evidence more closely to gain insight into how the Bayesian Occam's razor works. The evidence is the normalizing constant for equation (29.2):

$$P(D\,|\mathcal{H}_i) = \int P(D|\mathbf{w},\mathcal{H}_i)P(\mathbf{w}|\mathcal{H}_i)\,\mathrm{d}\mathbf{w}. \tag{29.5}$$

For many problems, including interpolation, it is common for the posterior $P(\mathbf{w}|D,\mathcal{H}_i) \propto P(D|\mathbf{w},\mathcal{H}_i)P(\mathbf{w}|\mathcal{H}_i)$ to have a strong peak at the most probable parameters $\mathbf{w}_{\text{MP}}$ (figure 29.3). Then, taking for simplicity the one-dimensional case, the evidence can be approximated, using Laplace's method, by the height of the peak of the integrand $P(D|\mathbf{w},\mathcal{H}_i)P(\mathbf{w}|\mathcal{H}_i)$ times its width, $\sigma_{w|D}$:

$$\begin{array}{rcl} P(D\,|\mathcal{H}_i) & \simeq & \underbrace{P(D\,|\mathbf{w}_{\text{MP}},\mathcal{H}_i)} \times \underbrace{P(\mathbf{w}_{\text{MP}}|\mathcal{H}_i)\,\sigma_{w|D}}\,. \\ \text{Evidence} & \simeq & \text{Best fit likelihood} \times \text{Occam factor} \end{array} \tag{29.6}$$

Thus the evidence is found by taking the best fit likelihood that the model can achieve and multiplying it by an 'Occam factor', which is a term with magnitude less than one that penalizes $\mathcal{H}_i$ for having the parameter $\mathbf{w}$.

### *Interpretation of the Occam factor*

The quantity $\sigma_{w|D}$ is the posterior uncertainty in $\mathbf{w}$. Suppose for simplicity that the prior $P(\mathbf{w}|\mathcal{H}_i)$ is uniform on some large interval $\sigma_w$, representing the range of values of $\mathbf{w}$ that were possible *a priori*, according to $\mathcal{H}_i$ (figure 29.3). Then $P(\mathbf{w}_{\text{MP}}|\mathcal{H}_i) = 1/\sigma_w$, and

$$\text{Occam factor} = \frac{\sigma_{w|D}}{\sigma_w},$$

i.e., *the Occam factor is equal to the ratio of the posterior accessible volume of $\mathcal{H}_i$'s parameter space to the prior accessible volume,* or the factor by which $\mathcal{H}_i$'s hypothesis space collapses when the data arrive. The model $\mathcal{H}_i$ can be viewed as consisting of a certain number of exclusive submodels, of which only one survives when the data arrive. The Occam factor is the inverse of that number. The logarithm of the Occam factor is a measure of the amount of information we gain about the model's parameters when the data arrive.

A complex model having many parameters, each of which is free to vary over a large range $\sigma_w$, will typically be penalized by a larger Occam factor than a simpler model. The Occam factor also penalizes models which have to be finely tuned to fit the data, favouring models for which the required precision of the parameters $\sigma_{w|D}$ is coarse. The magnitude of the Occam factor is thus a measure of complexity of the model; it relates to the complexity of the predictions that the model makes in data space. This depends not only on the number of parameters in the model, but also on the prior probability that the model assigns to them. Which model achieves the greatest evidence is determined by a trade-off between minimizing this natural complexity measure and minimizing the data misfit. In contrast to alternative measures of model complexity, the Occam factor for a model is straightforward to evaluate: it simply depends on the error bars on the parameters, which we already evaluated when fitting the model to the data.

Figure 29.4 displays an entire hypothesis space so as to illustrate the various probabilities in the analysis. There are three models, $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$, which have equal prior probabilities. Each model has one parameter $\mathbf{w}$ (each shown on a horizontal axis), but assigns a different prior range $\sigma_W$ to that parameter. $\mathcal{H}_3$ is the most 'flexible' or 'complex' model, assigning the broadest prior range. A one-dimensional data space is shown by the vertical axis. Each model assigns a joint probability distribution $P(D, \mathbf{w}|\mathcal{H}_i)$ to the data and the parameters, illustrated by a cloud of dots. These dots represent random samples from the full probability distribution. The total number of dots in each of the three model subspaces is the same, because we assigned equal prior probabilities to the models.

When a particular data set $D$ is received (horizontal line), we infer the posterior distribution of $\mathbf{w}$ for a model ($\mathcal{H}_3$, say) by reading out the density along that horizontal line, and normalizing. The posterior probability $P(\mathbf{w}|D, \mathcal{H}_3)$ is shown by the dotted curve at the bottom. Also shown is the prior distribution $P(\mathbf{w}|\mathcal{H}_3)$ (c.f. figure 29.3). [In the case of model $\mathcal{H}_1$ which is very poorly matched to the data, the shape of the posterior distribution will depend on

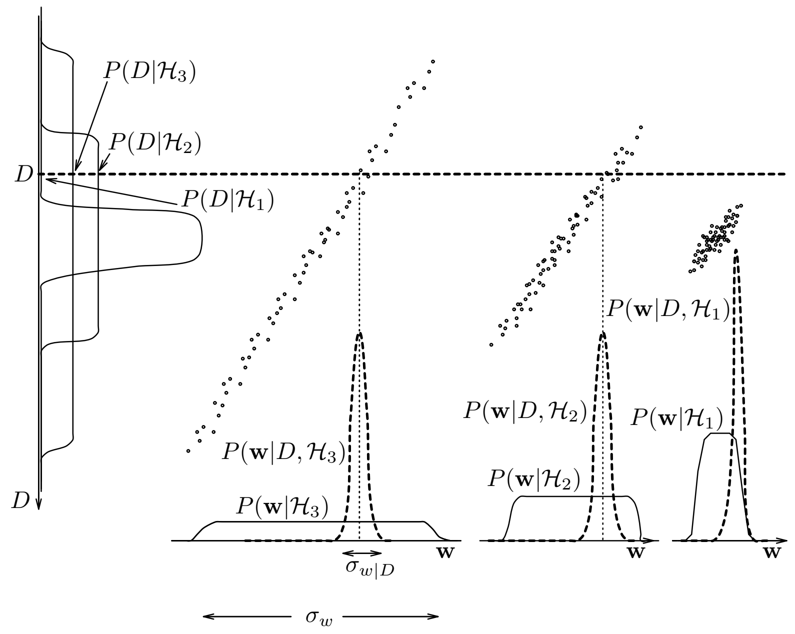

$P(D|\mathcal{H}_3)$
$P(D|\mathcal{H}_2)$
$P(D|\mathcal{H}_1)$
$D$
$D$
$P(\mathbf{w}|D,\mathcal{H}_1)$
$P(\mathbf{w}|D,\mathcal{H}_2)$
$P(\mathbf{w}|\mathcal{H}_1)$
$P(\mathbf{w}|D,\mathcal{H}_3)$
$P(\mathbf{w}|\mathcal{H}_2)$
$P(\mathbf{w}|\mathcal{H}_3)$
$\mathbf{w}$
$\sigma_{w|D}$
$\sigma_w$

Figure 29.4. A hypothesis space consisting of three exclusive models, each having one parameter $\mathbf{w}$, and a one-dimensional data set $D$. The 'data set' is a single measured value which differs from the parameter $\mathbf{w}$ by a small amount of additive noise. Typical samples from the joint distribution $P(D, w, \mathcal{H})$ are shown by dots. (NB, these are not data points.) The observed 'data set' is a single particular value for $D$ shown by the dashed horizontal line. The dashed curves below show the posterior probability of $\mathbf{w}$ for each model given this data set (c.f. figure 29.1). The evidence for the different models is obtained by marginalizing onto the $D$ axis at the left hand side (c.f. figure 29.3).

the details of the tails of the prior $P(\mathbf{w}|\mathcal{H}_1)$ and the likelihood $P(D|\mathbf{w}, \mathcal{H}_1)$; the curve shown is for the case where the prior falls off more strongly.]

We obtain figure 29.1 by marginalizing the joint distributions $P(D, \mathbf{w}|\mathcal{H}_i)$ onto the $D$ axis at the left hand side. For the data set $D$ shown by the dotted horizontal line, the evidence $P(D|\mathcal{H}_3)$ for the more flexible model $\mathcal{H}_3$ has a smaller value than the evidence for $H_2$. This is because $\mathcal{H}_3$ placed less predictive probability (fewer dots) on that line. In terms of the distributions over $\mathbf{w}$, model $\mathcal{H}_3$ has smaller evidence because the Occam factor $\sigma_{w|D}/\sigma_w$ is smaller for $\mathcal{H}_3$ than for $\mathcal{H}_2$. The simplest model $\mathcal{H}_1$ has the smallest evidence of all, because the best fit that it can achieve to the data $D$ is very poor. Given this data set, the most probable model is $\mathcal{H}_2$.

## *Occam factor for several parameters*

If the posterior is well approximated by a Gaussian, then the Occam factor is obtained from the determinant of the corresponding covariance matrix (c.f. equation (29.6)):

$$\begin{array}{rcl} P(D\,|\mathcal{H}_i) & \simeq & \underbrace{P(D\,|\mathbf{w}_{\rm MP}, H_i)} \times \underbrace{P(\mathbf{w}_{\rm MP}|\mathcal{H}_i)\det{}^{-\frac{1}{2}}(\mathbf{A}/2\pi)}, \\ \text{Evidence} & \simeq & \text{Best fit likelihood} \times \quad \text{Occam factor} \end{array} \tag{29.7}$$

where $\mathbf{A} = -\nabla\nabla \log P(\mathbf{w}|D, \mathcal{H}_i)$, the Hessian which we evaluated when we calculated the error bars on $\mathbf{w}_{\rm MP}$ (equation 29.3). As the number of data collected, $N$, increases, this Gaussian approximation is expected to become increasingly accurate.

In summary, Bayesian model comparison is a simple extension of maximum likelihood model selection: *the evidence is obtained by multiplying the best fit likelihood by the Occam factor.*

To evaluate the Occam factor we need only the Hessian $\mathbf{A}$, if the Gaussian approximation is good. Thus the Bayesian method of model comparison by evaluating the evidence is no more demanding computationally than the task of finding for each model the best fit parameters and their error bars.

## *Further reading*

Bayesian methods are introduced and contrasted with orthodox statistics in (Jaynes 1983; Gull 1988; Loredo 1990). The Bayesian Occam's razor is demonstrated on model problems in (Gull 1988; MacKay 1992a). Useful textbooks are (Box and Tiao 1973; Berger 1985).

# *About Chapter 30*

The last couple of chapters have assumed that a Gaussian approximation to the probability distribution we are interested in is adequate. What if it is not? We have already seen an example – clustering – where the likelihood function is multimodal, and has nasty unboundedly high spikes in certain locations in the parameter space; so maximizing the posterior probability and fitting a Gaussian is not always going to work. This difficulty with Laplace's method is one motivation for being interested in Monte Carlo methods. In fact, Monte Carlo methods provide a general-purpose set of tools with applications in Bayesian data modelling and many other fields.

This chapter describes a sequence of 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. [A graphical model is a probabilistic model in which dependences and independences of variables are represented by edges in a graph whose nodes are the variables.] Along the way, the terminology of Markov chain Monte Carlo methods is presented. The subsequent chapter gives a discussion of advanced 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 (1993), Gilks *et al.* (1996), and Tanner (1996).

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

$$\mathbf{u} = \mathbf{M}\mathbf{v} \tag{29.8}$$

to

$$\mathbf{u}^{\mathsf{T}} = \mathbf{v}^{\mathsf{T}}\mathbf{M}^{\mathsf{T}}. \tag{29.9}$$

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 $T(z; y)T(y; x)\pi(x)$.

# 30

# Monte Carlo methods

## 30.1 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

$$\Phi = \langle \phi(\mathbf{x}) \rangle \equiv \int d^N\mathbf{x} \; P(\mathbf{x})\phi(\mathbf{x}). \tag{30.1}$$

The probability distribution $P(\mathbf{x})$, which we 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 in include the means and variances 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 $\phi_2(\mathbf{x}) = (t(\mathbf{x}))^2$,

$$\Phi_1 \equiv \mathcal{E}[\phi_1(\mathbf{x})] \;\text{ and }\; \Phi_2 \equiv \mathcal{E}[\phi_2(\mathbf{x})], \tag{30.2}$$

then using

$$\bar{t} = \Phi_1 \;\text{ and }\; \mathrm{var}(t) = \Phi_2 - \Phi_1^2. \tag{30.3}$$

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

$$\hat{\Phi} \equiv \frac{1}{R}\sum_r \phi(\mathbf{x}^{(r)}). \tag{30.4}$$

If the vectors $\{\mathbf{x}^{(r)}\}_{r=1}^{R}$ are generated from $P(\mathbf{x})$ then the expectation of $\hat{\Phi}$ is $\Phi$. Also, as the number of samples $R$ increases, the variance of $\hat{\Phi}$ will decrease as $\frac{\sigma^2}{R}$, where $\sigma^2$ is the variance of $\phi$,

$$\sigma^2 = \int d^N\mathbf{x}\ P(\mathbf{x})(\phi(\mathbf{x}) - \Phi)^2. \tag{30.5}$$

This is one of the important properties of Monte Carlo methods.

> **The accuracy of the Monte Carlo estimate (equation (30.4)) depends only on the variance of $\phi$, not on 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 $\Phi$ 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

$$P(\mathbf{x}) = P^*(\mathbf{x})/Z. \tag{30.6}$$

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

$$Z = \int d^N\mathbf{x}\ P^*(\mathbf{x}). \tag{30.7}$$

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. Correct samples from $P$ will by definition tend to come from places in $\mathbf{x}$-space where $P(\mathbf{x})$ is big; how can we identify those places where $P(\mathbf{x})$ is big, without evaluating $P(\mathbf{x})$ *everywhere*? There are only a few high-dimensional densities from which it is easy to draw samples, for example the Gaussian distribution.

> A sample $z$ from a standard univariate Gaussian can be generated by computing
>
> $$z = \cos(2\pi u_1)\sqrt{2\ln(1/u_2)}, \tag{30.8}$$
>
> where $u_1$ and $u_2$ are uniformly distributed in $(0, 1)$.
>
> A second sample $z_2 = \sin(2\pi u_1)\sqrt{2\ln(1/u_2)}$, independent of the first, can then be obtained for free.

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

$$P^*(x) = \exp\left[0.4(x - 0.4)^2 - 0.08x^4\right],\ \ x \in (-\infty, \infty). \tag{30.9}$$

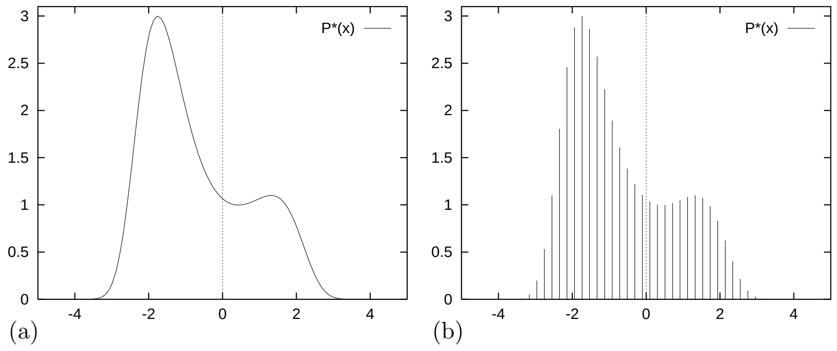


Figure 30.1. (a) The function $P^*(x) = \exp\left[0.4(x-0.4)^2 - 0.08x^4\right]$. How to draw samples from this density? (b) The function $P^*(x)$ evaluated at a discrete set of uniformly spaced points $\{x_i\}$. How to draw samples from this discrete distribution?

We can plot this function (figure 30.1a). 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 30.1b). How could we solve this problem? If we evaluate $p_i^* = P^*(x_i)$ at each point $x_i$, we can compute

$$Z = \sum_i p_i^* \tag{30.10}$$

and

$$p_i = p_i^*/Z \tag{30.11}$$

and we can then sample from the probability distribution $\{p_i\}$ using various methods based on a source of random bits (see section 7.3). 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 $Z$ (30.10). To compute $Z$ we have to visit every point in the space. In figure 30.1b 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 a trillion ($2^{40}$) states every second, and if we ran those $2^{266}$ computers for a time equal to the age of the universe ($2^{58}$ seconds), they 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.

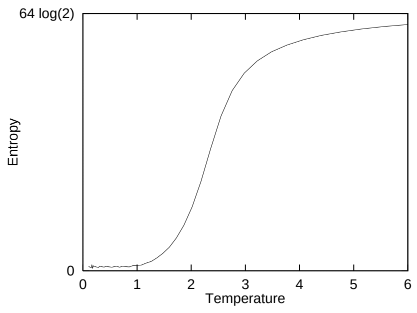
Systems with $2^{1000}$ states are two a penny.* One example is a collection of 1000 spins such as a $30 \times 30$ fragment of an Ising model whose probability distribution is proportional to

$$P^*(\mathbf{x}) = \exp\left[-\beta E(\mathbf{x})\right] \tag{30.12}$$

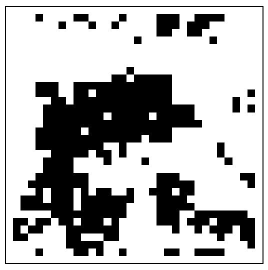
where $x_n \in \{\pm 1\}$ and

$$E(\mathbf{x}) = -\left[\frac{1}{2}\sum_{m,n} J_{mn} x_m x_n + \sum_n H_n x_n\right]. \tag{30.13}$$

* Translation for American readers: 'such systems are a dime a dozen'; incidentally, this equivalence (10 ¢= 6 p) 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.



(a)



(b)

Figure 30.2. (a) Entropy of a 64-spin Ising model as a function of temperature. (b) One state of a 1024-spin Ising model.

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; the first 'exact' samples from this distribution were created in the pioneering 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

$$Z_R = \sum_{r=1}^{R} P^*(\mathbf{x}^{(r)}), \tag{30.14}$$

and estimate $\Phi = \int d^N\mathbf{x}\ \phi(\mathbf{x})P(\mathbf{x})$ by

$$\hat{\Phi} = \sum_{r=1}^{R} \phi(\mathbf{x}^{(r)}) \frac{P^*(\mathbf{x}^{(r)})}{Z_R}. \tag{30.15}$$

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 5, 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 $P(\mathbf{x})$. 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 $\Phi$ if we make the number of samples $R$ sufficiently large that we are likely to hit the typical set at least once or twice. So, how many samples are required? Let us take the case of the Ising model again. (Strictly, the Ising model may not be a good example, since it doesn't necessarily have a typical set, as defined in chapter 5; the definition of a typical set was that all states had log probability close to the entropy, which for an Ising model would mean that the *energy* is very close to the *mean energy*; but in the vicinity of phase transitions, the variance of energy, also known as the heat capacity, may diverge, which means that the energy of a random state is not necessarily expected to be very close to the mean energy.) 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

$$R_{\min} \simeq 2^{N-H}. \tag{30.16}$$

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, which means $R_{\min}$ is of order 1. Under these conditions, uniform sampling may well be a satisfactory technique for estimating $\Phi$. 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 30.2). For this probability distribution the number of samples required simply to hit the typical set once is of order

$$R_{\min} \simeq 2^{N-N/2} = 2^{N/2}, \tag{30.17}$$

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*.

## 30.2 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 any chosen point $\mathbf{x}$, at least to within a multiplicative constant; thus we can evaluate a function $P^*(x)$ such that

$$P(x) = P^*(x)/Z. \tag{30.18}$$

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 $\phi$ is shown in figure 30.3. We call $Q$ the *sampler density.*

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 $\Phi$ by equation (30.4). 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'

$$w_r \equiv \frac{P^*(x^{(r)})}{Q^*(x^{(r)})} \tag{30.19}$$

which we use to adjust the 'importance' of each point in our estimator thus:

$$\hat{\Phi} \equiv \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r}. \tag{30.20}$$

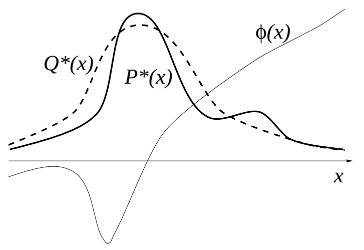


Figure 30.3. Functions involved in importance sampling. We wish to estimate the expectation of $\phi(x)$ under $P(x) \propto P^*(x)$. We can generate samples from the simpler distribution $Q(x) \propto Q^*(x)$. We can evaluate $Q^*$ and $P^*$ at any point.

Exercise 30.1:$^{B2}$ Prove that, if $Q(x)$ is non-zero for all $x$ where $P(x)$ is non-zero, the estimator $\hat{\Phi}$ converges to $\Phi$, 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 (30.20). 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 $\Phi$ 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.

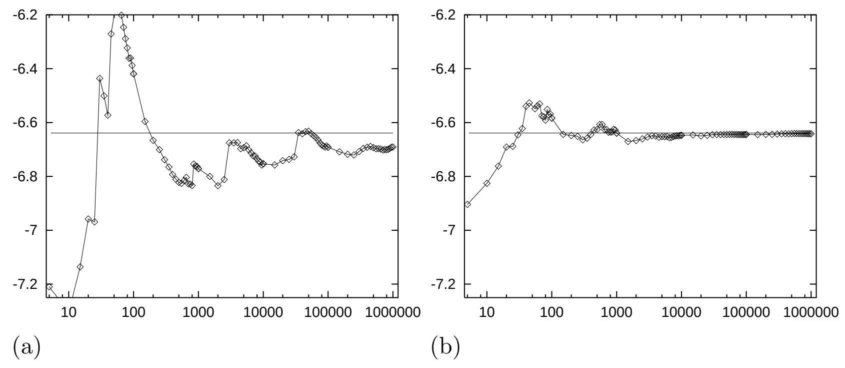
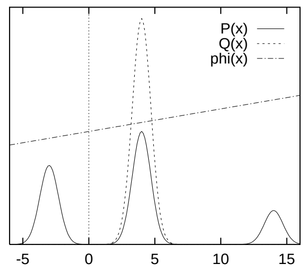


Figure 30.4. Importance sampling in action: (a) using a Gaussian sampler density; (b) using a Cauchy sampler density. Vertical axis shows the estimate $\hat{\Phi}$. The horizontal line indicates the true value of $\Phi$. Horizontal axis shows number of samples on a log scale.

### *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 30.4. 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; it converges (on the scale shown here) after about 5000 samples.

This example illustrates the fact that an importance sampler should have **heavy tails**.



Exercise 30.2:$^{A2}$ 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 $\Phi$ 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,

$$P^*(\mathbf{x}) = \begin{cases} 1 & 0 \leq \rho(\mathbf{x}) \leq R_P \\ 0 & \rho(\mathbf{x}) > R_P \end{cases} , \tag{30.21}$$

where $\rho(\mathbf{x}) \equiv (\sum_i x_i^2)^{1/2}$, and the proposal density is a Gaussian centred on the origin,

$$Q(\mathbf{x}) = \prod_i \text{Normal}(x_i; 0, \sigma^2). \tag{30.22}$$

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 8.1 (p.150), for example — that if $\rho$ is the distance from the origin of a sample from $Q$, the quantity $\rho^2$ has a roughly Gaussian distribution with mean and standard deviation:

$$\rho^2 \sim N\sigma^2 \pm \sqrt{2N}\sigma^2. \tag{30.23}$$

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 $\sigma$ 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

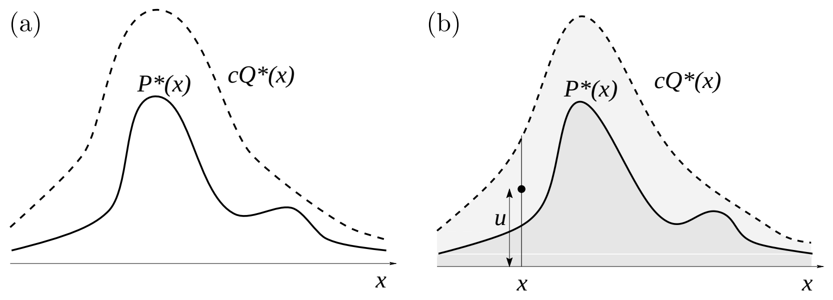


Figure 30.5. Rejection sampling. (a) The functions involved in rejection sampling. We desire samples from $P(x) \propto P^*(x)$. We are able to draw samples from $Q(x) \propto Q^*(x)$, and we know a value $c$ such that $cQ^*(x) > P^*(x)$ for all $x$. (b) A point $(x, u)$ is generated at random in the lightly shaded area under the curve $cQ^*(x)$. If this point also lies below $P^*(x)$ then it is accepted.

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

$$\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2} \pm \frac{\sqrt{2N}}{2}\right). \tag{30.24}$$

Thus the weights $w_r = P^*/Q$ will typically have values in the range

$$(2\pi\sigma^2)^{N/2} \exp\left(\frac{N}{2} \pm \frac{\sqrt{2N}}{2}\right). \tag{30.25}$$

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 (30.25). The largest weight and the median weight will typically be in the ratio:

$$\frac{w_r^{\max}}{w_r^{\text{med}}} = \exp\left(\sqrt{2N}\right). \tag{30.26}$$

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 $\exp(\sqrt{N})$ (unless $Q$ is a near-perfect approximation to $P$).

## 30.3 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 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

$$c\,Q^*(x) > P^*(x), \text{ for all } x. \tag{30.27}$$

A schematic picture of the two functions is shown in figure 30.5a.

We generate two random numbers. The first, $x$, is generated from the proposal density $Q(x)$. We then evaluate $c\,Q^*(x)$ and generate a uniformly distributed random variable $u$ from the interval $[0, c\,Q^*(x)]$. These two random numbers can be viewed as selecting a point in the two-dimensional plane as shown in figure 30.5b.

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 $c\,Q^*(x)$ as shown in figure 30.5b. 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 $c\,Q$ to exceed $P$ everywhere, $c$ will necessarily have to be large and the frequency of rejection will be large.

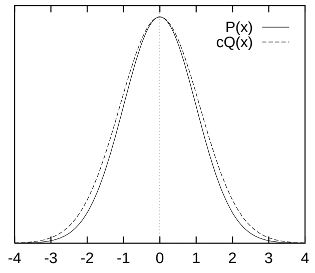


Figure 30.6. A Gaussian $P(x)$ and a slightly broader Gaussian $Q(x)$ scaled up by a factor $c$ such that $c\,Q(x) \geq P(x)$.

### *Rejection sampling in many dimensions*

In a high-dimensional problem it is very likely that the requirement that $c\,Q^*$ 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 30.6). Imagine generating samples from one with standard deviation $\sigma_Q$ and using rejection sampling to obtain samples from the other whose standard deviation is $\sigma_P$. Let us assume that these two standard deviations are close in value — say, $\sigma_Q$ is one per cent larger than $\sigma_P$. [$\sigma_Q$ must be larger than $\sigma_P$ because if this is not the case, there is no $c$ such that $c\,Q$ 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 $c\,Q$ to exceed $P$ we need to set

$$c = \frac{(2\pi\sigma_Q^2)^{N/2}}{(2\pi\sigma_P^2)^{N/2}} = \exp\left(N \log \frac{\sigma_Q}{\sigma_P}\right). \tag{30.28}$$

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 $c\,Q(\mathbf{x})$, 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 $P(\mathbf{x})$.

## 30.4 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 30.7; this figure shows the density $Q(x';x^{(t)})$ for two different states $x^{(1)}$ and $x^{(2)}$.

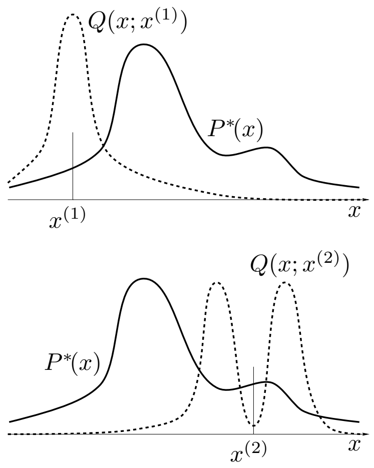


Figure 30.7. Metropolis method in one dimension. The proposal distribution $Q(x';x)$ is here shown as having a shape that changes as $x$ changes, though this is not typical of the proposal densities used in practice.

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

$$a = \frac{P^*(x')}{P^*(x^{(t)})}\frac{Q(x^{(t)};x')}{Q(x';x^{(t)})}. \qquad (30.29)$$

If $a \geq 1$ then the new state is accepted.
Otherwise, the new state is accepted with probability $a$.

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 (30.29) 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.]

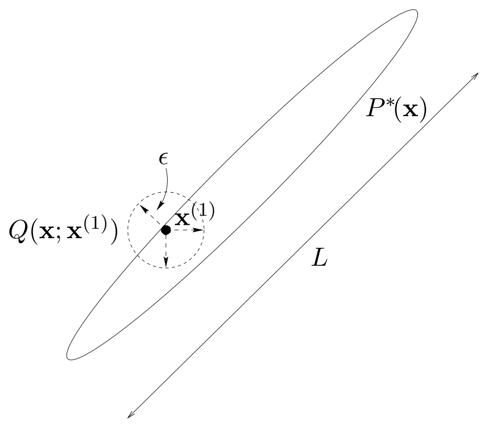


Figure 30.8. Metropolis method in two dimensions, showing a traditional proposal density that has a sufficiently small step size $\epsilon$ that the acceptance frequency will be about 0.5.

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 $\epsilon$ that is short relative to the longest length scale $L$ of the probable region (figure 30.8). 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 $\epsilon$ is large, movement around the state space will only occur when such a transition to a low-probability state 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 30.3:$^{A1}$ 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 $\epsilon$, 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 $\epsilon$ must be run for at least

$$T \simeq (L/\epsilon)^2 \tag{30.30}$$

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 30.9 shows a simulation of a Metropolis algorithm for generating samples from the distribution:

$$P(x) = \begin{cases} 1/21 & x \in \{0, 1, 2, \ldots, 20\} \\ 0 & \text{otherwise} \end{cases} . \tag{30.31}$$

The proposal distribution is

$$Q(x'; x) = \begin{cases} 1/2 & x' = x \pm 1 \\ 0 & \text{otherwise} \end{cases} . \tag{30.32}$$

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 30.9a. 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 (30.30) 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. Methods for reducing random walk behaviour are discussed in the following chapter.

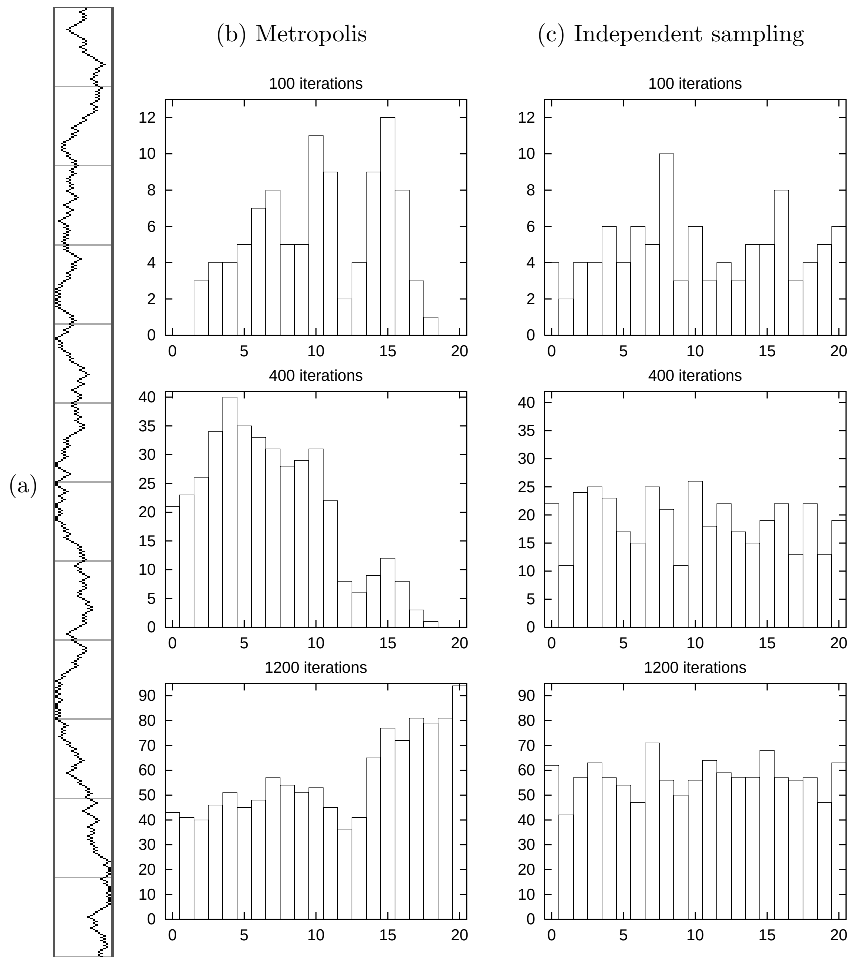


Figure 30.9. Metropolis method for a toy problem. (a) The state sequence for $t = 1, \ldots, 600$. Horizontal direction = states from 0 to 20; vertical direction = time from 1 to 600; the cross bars mark time intervals of duration 50. (b) Histogram of occupancy of the states after 100, 400 and 1200 iterations. (c) For comparison, histograms resulting when successive points are drawn *independently* from the target distribution.

*Metropolis method in high dimensions*

The rule of thumb (30.30), which gives 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 $\epsilon$. 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 deviation $\sigma_n$ in direction $n$. Let $\sigma^{\max}$ and $\sigma^{\min}$ be the largest and smallest of these standard deviations. Let us assume that $\epsilon$ 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 $\epsilon$. 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 $\epsilon$ be? The bigger it is, the smaller this number $T$ becomes, but if $\epsilon$ is too big — bigger than $\sigma^{\min}$ — then the acceptance rate will fall sharply. It seems plausible that the optimal $\epsilon$ must be similar to $\sigma^{\min}$. Strictly, this may not be true; in special cases where the second smallest $\sigma_n$ is significantly greater than $\sigma^{\min}$, the optimal $\epsilon$ may be closer to that second smallest $\sigma_n$. 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 $N$. Our computer will give answers in 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 in the next chapter.

## 30.5 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. Gibbs sampling 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*

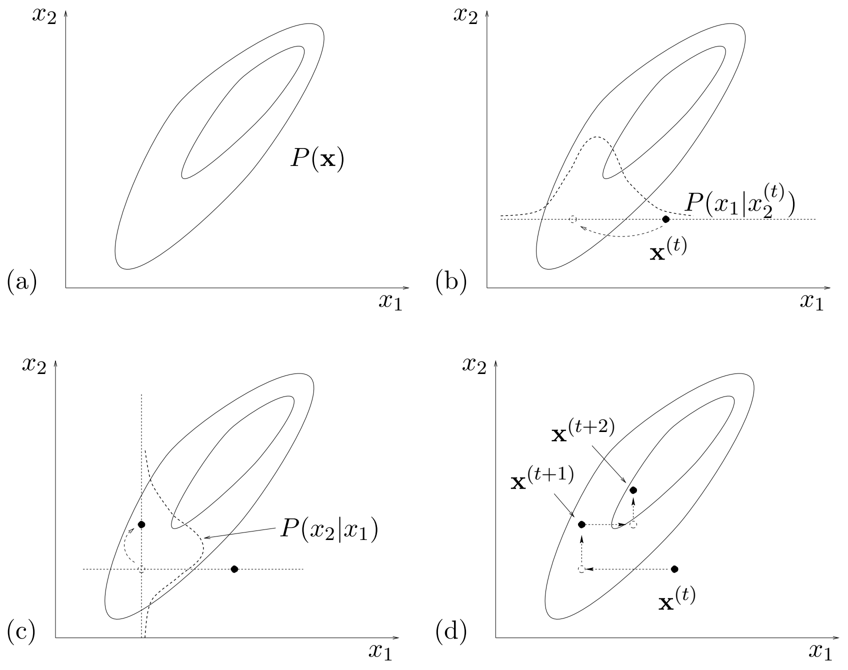


Figure 30.10. Gibbs sampling. (a) The joint density $P(\mathbf{x})$ from which samples are required. (b) Starting from a state $\mathbf{x}^{(t)}$, $x_1$ is sampled from the conditional density $P(x_1|x_2^{(t)})$. (c) A sample is then made from the conditional density $P(x_2|x_1)$. (d) A couple of iterations of Gibbs sampling.

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 30.10. 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:

$$x_1^{(t+1)} \sim P(x_1|x_2^{(t)}, x_3^{(t)}, \ldots, x_K^{(t)}) \tag{30.33}$$
$$x_2^{(t+1)} \sim P(x_2|x_1^{(t+1)}, x_3^{(t)}, \ldots, x_K^{(t)}) \tag{30.34}$$
$$x_3^{(t+1)} \sim P(x_3|x_1^{(t+1)}, x_2^{(t+1)}, \ldots, x_K^{(t)}), \text{ etc.} \tag{30.35}$$

## *Convergence of Gibbs sampling to the target density*

Exercise 30.4:$^{A2}$ 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 30.5:$^{A2}$ 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 $\mathbf{n} = (n_1, \ldots, n_n, \ldots, n_N)$,

$$P(\mathbf{n}|\mathbf{f}, \mathbf{z}) = \frac{1}{Z} \prod_{n=1}^{N} f_n^{n_n}(1-f_n)^{(1-n_n)} \delta(\mathbf{H}\mathbf{n} = \mathbf{z}), \tag{30.36}$$

where $f_n$ is the normalized likelihood for the $n$th transmitted bit and $\mathbf{z}$ is the observed syndrome. The factor $\delta(\mathbf{H}\mathbf{n} = \mathbf{z})$ 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 $\epsilon$, then it will take at least about $(L/\epsilon)^2$ iterations to generate an independent sample from the target density. Figure 31.3, p.414, 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]

## 30.6 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

$$p^{(t+1)}(\mathbf{x}') = \int d^N\mathbf{x} \; T(\mathbf{x}'; \mathbf{x}) p^{(t)}(\mathbf{x}). \tag{30.37}$$

Example 30.6 An example of a Markov chain is given by the Metropolis demonstration of section 30.4 (figure 30.9), for which the transition probability is

[1]`http://www.mrc-bsu.cam.ac.uk/bugs/`

$$\mathbf{T} = \left[\begin{array}{ccccccccccccccccccccc}
1/2 & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & \cdot & 1/2 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1/2 & 1/2
\end{array}\right]$$

and the initial distribution was

$$p^{(0)}(x) = \left[\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ 1\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \right]. \tag{30.38}$$

The probability distribution $p^{(t)}(x)$ of the state at the $t$th iteration is shown for $t = 0, 1, 2, 3, 5, 10, 100, 200, 400$ in figure 30.11; an equivalent sequence of distributions is shown in figure 30.12 for the chain that begins in initial state $x_0 = 17$. Both chains converge to the target density, the uniform density, as $t \to \infty$.

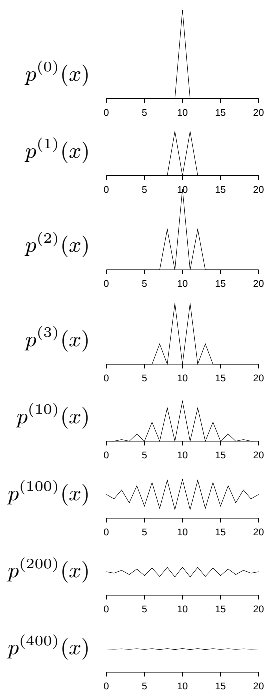


Figure 30.11. The probability distribution of the state of the Metropolis chain example.

### *Required properties*

When designing a Markov chain Monte Carlo method, 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

$$\pi(\mathbf{x}') = \int d^N\mathbf{x}\ T(\mathbf{x}'; \mathbf{x})\pi(\mathbf{x}). \tag{30.39}$$

2. The chain must also be *ergodic*, that is,

$$p^{(t)}(\mathbf{x}) \to \pi(\mathbf{x}) \text{ as } t \to \infty, \text{ for any } p^{(0)}(\mathbf{x}). \tag{30.40}$$

   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.

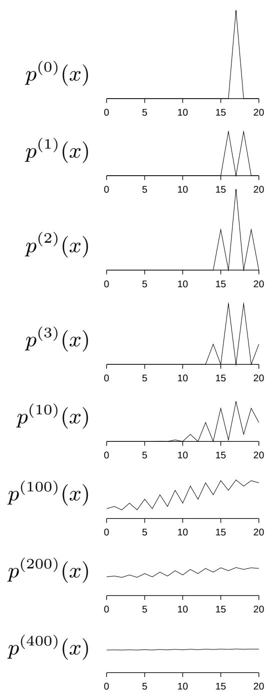


Figure 30.12. The probability distribution of the state of the Metropolis chain example for initial condition $x_0 = 17$.

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, for example, has eigenvalues equal to $+1$ and $-1$.

## *Methods of construction of Markov chains*

It is often convenient to construct $T$ by *mixing* or *concatenating* simple *base transitions* $B$ all of which satisfy

$$P(\mathbf{x}') = \int d^N\mathbf{x} \;\; B(\mathbf{x}';\mathbf{x})P(\mathbf{x}), \tag{30.41}$$

for the desired density $P(\mathbf{x})$, i.e., they all have the desired density as an invariant distribution. These base transitions need not individually be 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.,

$$T(\mathbf{x}', \mathbf{x}) = \sum_b \pi_b B_b(\mathbf{x}', \mathbf{x}), \tag{30.42}$$

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 $B_2$.

$$T(\mathbf{x}', \mathbf{x}) = \int d^N\mathbf{x}'' \, B_2(\mathbf{x}', \mathbf{x}'')B_1(\mathbf{x}'', \mathbf{x}) \tag{30.43}$$

## *Detailed balance*

Many useful transition probabilities satisfy the *detailed balance* property:

$$T(\mathbf{x}_a; \mathbf{x}_b)P(\mathbf{x}_b) = T(\mathbf{x}_b; \mathbf{x}_a)P(\mathbf{x}_a), \text{ for all } \mathbf{x}_b \text{ and } \mathbf{x}_a. \tag{30.44}$$

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 30.7:$^{A2}$ 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 30.8:$^{B2}$ 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 (30.43) satisfies detailed balance.

## 30.7 Slice sampling

Slice sampling (Neal 1997; Neal 2002) is a Markov chain Monte Carlo method which has similarities to rejection sampling, Gibbs sampling and the Metropolis method. It can be applied wherever the Metropolis method can be applied, that is, to 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 the one-dimensional conditional distributions be easy to sample from, 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, then giving a pictorial description that includes the details that make the method valid.

### *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 – like the uniform distribution under the curve $P^*(x)$ produced by rejection sampling (section 30.3).

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 $u' \sim$ Uniform$(0, P^*(x))$
3: create a horizontal interval $(x_l, x_r)$ enclosing $x$
4: loop {
```

```
5:        draw $x' \sim$ Uniform$(x_l, x_r)$
6:        evaluate $P^*(x')$
7:        if $P^*(x') > u'$ break out of loop 4-9
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.

### *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 30.13.

```
3a: draw $r \sim$ Uniform$(0, 1)$
3b: $x_l$ := $x - rw$
3c: $x_r$ := $x + (1-r)w$
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 $(x' > x)$ { $x_r$ := $x'$ }
8b: else { $x_l$ := $x'$ }
```

### *Properties of slice sampling*

Like a standard Metropolis method, slice sampling gets around by a random walk, but whereas with the Metropolis method, the choice of the typical step size is critical to the rate of progress, in slice sampling the step size is self-tuning. If the initial interval size $w$ is too small by a factor $f$ compared with the width of the probable region then the stepping-out procedure expands the interval size. The cost of this stepping out is only linear in $f$, whereas in the Metropolis method, the computer-time scales as the square of $f$, if the step size is too small.

If the chosen value of $w$ is too large by a factor $F$ then the algorithm spends a time proportional to the logarithm of $F$ shrinking the interval down to the right size. In contrast, the Metropolis algorithm responds to a too-large step size by rejecting almost all proposals, so the rate of progress is exponentially bad in $F$. There are no rejections in slice sampling. The probability of staying in exactly the same place is very small.

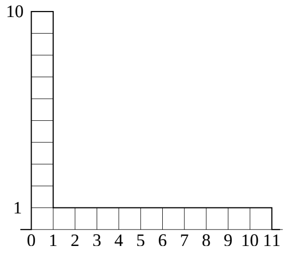


Figure 30.14. $P^*(x)$

Exercise 30.9:$^{B2}$ Investigate the properties of slice sampling applied to the density over a real variable $x$ shown in figure 30.14. How long does it take typically for slice sampling to get from an $x$ in the peak region $x \in (0, 1)$ to an $x$ in the tail region $x \in (1, 11)$, and vice versa? Confirm that the

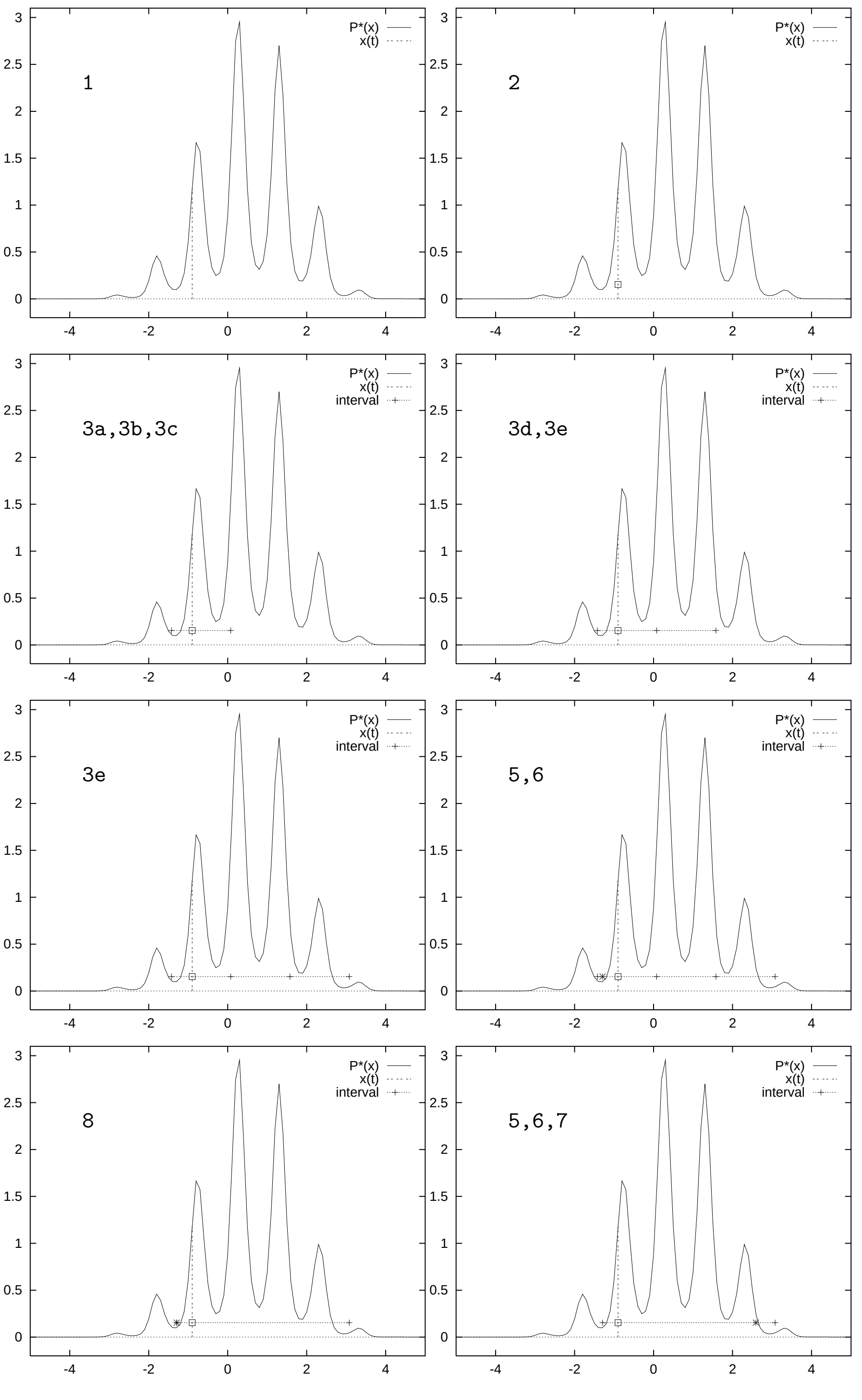


Figure 30.13. Slice sampling. Each panel is labelled by the steps of the algorithm that are executed in it. At step `1`, $P^*(x)$ is evaluated at the current point $x$. At step `2`, a vertical coordinate is selected giving the point $(x, u')$ shown by the box; At steps `3a-c`, an interval of size $w$ containing $(x, u')$ is created at random. At step `3d`, $P^*$ is evaluated at the left end of the interval and is found to be smaller than $u'$, so no stepping out to the left is needed. At step `3e`, $P^*$ is evaluated at the right end of the interval and is found to be larger than $u'$, so a step to the right of size $w$ is made. When step `3e` is repeated, $P^*$ is found to be larger than $u'$, so another step to the right is made. At step `5` a point is drawn from the interval; it is located close to the left end of the interval, shown by a $\times$. Step `6` establishes that this point is above $P^*$ and step `8` shrinks the interval to the rejected point in such a way that the original point $x$ is still in the interval. When step `5` is repeated, the new coordinate $x'$ (which is to the right hand side of the interval) gives a value of $P^*$ greater than $u'$, so this point $x'$ is the outcome at step `7`.

probabilities of these transitions do yield an asymptotic probability density that is correct.

## *Computer-friendly slice sampling* *

The real variables of a probabilistic model will always be represented in a computer using a finite number of bits. In the following implementation of slice sampling due to Skilling (Skilling and MacKay 2001), the stepping-out, randomization, and shrinking operations, described above in terms of floating-point operations, are replaced by binary and integer operations.

We assume that the variable $x$ that is being slice-sampled is represented by a $b$-bit integer $X$ taking on one of $B = 2^b$ values, $0, 1, 2, \ldots B-1$, many or all of which correspond to valid values of $x$. Using an integer grid eliminates any errors in detailed balance that might thus ensue from variable-precision rounding of floating-point numbers. The mapping from $X$ to $x$ need not be linear; if it is nonlinear, we assume that the function $P^*(x)$ is replaced by an appropriately transformed function – for example, $P^{**}(X) \propto P^*(x)|dx/dX|$, if the mapping from $X$ to $x$ is continuous.

We assume the following operators on $b$-bit integers are available:

| | |
|---|---|
| $X + N$ | arithmetic sum, modulo $B$, of $X$ and $N$. |
| $X - N$ | difference, modulo $B$, of $X$ and $N$. |
| $X \oplus N$ | bitwise exclusive-or of $X$ and $N$. |
| $N := \texttt{randbits}(l)$ | sets $N$ to a random $l$-bit integer. |

A slice-sampling procedure for integers is then as follows:

```
Given: a current point $X$ and a height $Y = P^*(X) \times \text{Uniform}(0,1) \leq P^*(X)$

1:  $U := \texttt{randbits}(b)$                  Define a random translation $U$ of the binary coor-
                                                 dinate system.
2:  set $l$ to a value $l \leq b$                Set initial $l$-bit sampling range
3:  do {
4:      $N := \texttt{randbits}(l)$              Define a random move within the current interval of
                                                 width $2^l$.
5:      $X' := ((X - U) \oplus N) + U$           Randomize the lowest $l$ bits of $X$ (in the translated
                                                 coordinate system).
6:      $l := l - 1$                             If $X'$ is not acceptable, decrease $l$ and try again
7:  } until ($X' = X$) or ($P^*(X) \geq Y$)      with a smaller perturbation of $X$; termination at or
                                                 before $l = 0$ is assured.
```

The translation $U$ is introduced to avoid permanent sharp edges, where for example the adjacent binary integers `0111111111` and `1000000000` would otherwise be permanently in different sectors, making it difficult for $X$ to move from one to the other.

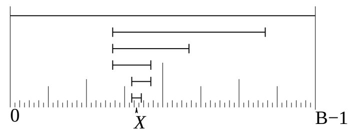


Figure 30.15. The sequence of intervals from which the new candidate points are drawn.

The sequence of intervals from which the new candidate points are drawn is illustrated in figure 30.15. First, a point is drawn from the entire interval, shown by the top horizontal line. At each subsequent draw, the interval is halved in such a way as to contain the previous point $X$.

If preliminary stepping-out from the initial range is required, step 2 above can be replaced by the following similar procedure:

```
2a: set $l$ to a value $l < b$          ($l$ sets the initial width)
2b: do {
2c:     $N := \texttt{randbits}(l)$
2d:     $X' := ((X - U) \oplus N) + U$
2e:     $l := l + 1$
2f: } until ($l = b$) or ($P^*(X) < Y$)
```

These shrinking and stepping out methods shrink and expand by a factor of two per evaluation. A variant is to shrink or expand by more than one bit each time, setting $l := l \pm \Delta l$ with $\Delta l > 1$. Taking $\Delta l$ at each step from any pre-assigned distribution (which may include $\Delta l = 0$) allows extra flexibility.

A feature of using the integer representation is that, with a suitably extended number of bits, the single integer $X$ can represent two or more real parameters – for example, by mapping $X$ to $(x_1, x_2, x_3)$ through a space-filling curve such as a Peano curve. Thus multi-dimensional slice sampling can be performed using the same software as for one dimension.

## 30.8 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 next chapter, which describes the hybrid Monte Carlo method, overrelaxation, and simulated annealing.

## 30.9 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 (1993)) and annealed importance sampling (Neal 1998).

2. 'Thermodynamic integration' during simulated annealing, the 'acceptance ratio' method, and 'umbrella sampling' (reviewed by Neal (1993)).

3. 'Reversible jump Markov chain Monte Carlo' (Green 1995).

One way of dealing with $Z$, however, may be to find a solution to one's task that does not require that $Z$ be evaluated. In Bayesian data modelling one might be able to 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 s 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 30.10:$^{B2}$ 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 $f < 1/2$.

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 to aim 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 (30.33–30.35). For big problems it may be more efficient to sample *groups* of variables jointly, that is to use several proposal distributions:

$$\begin{aligned} x_1^{(t+1)},\ldots,x_a^{(t+1)} &\sim P(x_1\ldots x_a|x_{a+1}^{(t)},\ldots,x_K^{(t)}) && (30.45)\\ x_{a+1}^{(t+1)},\ldots,x_b^{(t+1)} &\sim P(x_{a+1},\ldots,x_b|x_1^{(t+1)},\ldots,x_a^{(t+1)},x_{b+1}^{(t)},\ldots,x_K^{(t)}), \quad \text{etc..} \end{aligned}$$

*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

$$\sigma^2 = \int d^N\mathbf{x}\; P(\mathbf{x})(\phi(\mathbf{x}) - \Phi)^2. \tag{30.46}$$

We have now discussed a variety of methods for generating samples from $P(\mathbf{x})$. How many independent samples $R$ should we aim for?

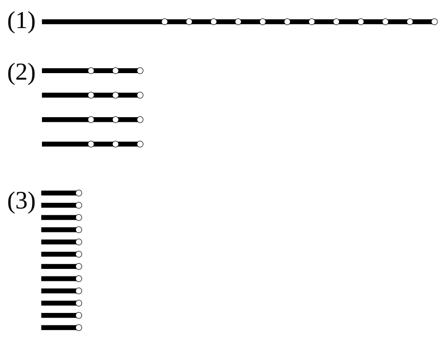


Figure 30.16. Three possible Markov Chain Monte Carlo strategies for obtaining twelve samples in a fixed amount of computer time. Time is represented by horizontal lines; samples by white circles. (1) A single run consisting of one long 'burn in' period followed by a sampling period. (2) Four medium-length runs with different initial conditions and a medium-length burn in period. (3) Twelve short runs.

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 $\Phi$ is the expected cost of the repairs. The quantity $\sigma^2$ is the variance of the cost — $\sigma$ measures by how much we should expect the actual cost to differ from the expectation $\Phi$.

Now, how accurately would a manager like to know $\Phi$? I would suggest there is little point in knowing $\Phi$ to a precision finer than about $\sigma/3$. After all, the true cost is likely to differ by $\pm\sigma$ from $\Phi$. If we obtain $R = 12$ independent samples from $P(\mathbf{x})$, we can estimate $\Phi$ to a precision of $\sigma/\sqrt{12}$ — which is smaller than $\sigma/3$. So twelve samples suffice.

## *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 30.16).

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 is popular with Markov chain Monte

Carlo experts (Gilks *et al.* 1996) 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.

## 30.10 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

$$\int \phi(\mathbf{x}) P(\mathbf{x}) \simeq \frac{1}{R} \sum_r \phi(\mathbf{x}^{(r)}). \qquad (30.47)$$

- 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 and Gibbs sampling algorithms, although widely used, perform poorly because they explore the space by a slow random walk. The next chapter will discuss methods for speeding up Markov chain Monte Carlo simulations.

- 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.

## 30.11 Exercises

Exercise 30.11:$^{B2}$ A study of importance sampling. We already established in section 30.2 that importance sampling is likely to be useless in high-dimensional problems. This exercise explores a further cautionary tale, showing that importance sampling can fail even in one dimension, with friendly Gaussian distributions.

Imagine that we want to know the expectation of a function $\phi(x)$ under a distribution $P(x)$,

$$\Phi = \int dx\, P(x) \phi(x), \qquad (30.48)$$

and that this expectation is estimated by importance sampling with a distribution $Q(x)$. Alternatively, perhaps we wish to estimate the normalizing constant $Z$ in $P(x) = P^*(x)/Z$ using

$$Z = \int dx\, P^*(x) = \int dx\, Q(x) \frac{P^*(x)}{Q(x)} = \left\langle \frac{P^*(x)}{Q(x)} \right\rangle_{x \sim Q}. \qquad (30.49)$$

Now, let $P(x)$ and $Q(x)$ be Gaussian distributions with mean zero and standard deviations $\sigma_p$ and $\sigma_q$. Each point $x$ drawn from $Q$ will have an associated weight $P^*(x)/Q(x)$. What is the variance of the weights? [Assume that $P^* = P$, so $P$ is actually normalized, and $Z = 1$, though we can pretend that we didn't know that.] What happens to the variance of the weights as $\sigma_q^2 \rightarrow \frac{1}{2}\sigma_p^2$?

Check your theory by simulating this importance-sampling problem on a computer.

Exercise 30.12:$^{A2}$ Consider the Metropolis algorithm for the one-dimensional toy problem of section 30.4. Whenever the current state is one of the end states, the proposal density given in equation (30.32) 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 30.13:$^{B2}$ 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 $\epsilon$. 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.

# *Solutions to Chapter 30's exercises*

Solution to exercise 30.1 (p.385): We wish to show that

$$\hat{\Phi} \equiv \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r} \tag{30.50}$$

converges to the expectation of $\Phi$ under $P$. We consider the numerator and the denominator separately. First, the denominator. Consider a single importance weight.

$$w_r \equiv \frac{P^*(x^{(r)})}{Q^*(x^{(r)})} \tag{30.51}$$

What is its expectation, averaged under the distribution $Q = Q^*/Z_Q$ of the point $x^{(r)}$?

$$\langle w_r \rangle = \int dx Q(x) \frac{P^*(x)}{Q^*(x)} = \int dx \frac{1}{Z_Q} P^*(x) = \frac{Z_P}{Z_Q}. \tag{30.52}$$

So the expectation of the denominator is

$$\left\langle \sum_r w_r \right\rangle = R \frac{Z_P}{Z_Q}. \tag{30.53}$$

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

$$\langle w_r \phi(x) \rangle = \int dx Q(x) \frac{P^*(x)}{Q^*(x)} \phi(x) = \int dx \frac{1}{Z_Q} P^*(x) \phi(x) = \frac{Z_P}{Z_Q} \Phi, \tag{30.54}$$

where $\Phi$ is the expectation of $\phi$ under $P$. So the numerator, divided by $R$, converges to $\frac{Z_P}{Z_Q}\Phi$ with increasing $R$. Thus $\hat{\Phi}$ converged to $\Phi$.

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 30.2 (p.386): When the true density $P$ is multimodal, it is unwise to use importance sampling with a sampler density fitted to one mode, because on the rare occasions that point is produced that lands in one of the other modes, the weight associated with that point will be enormous. The estimates will have enormous variance, but this enormous variance may not be evident to the user.

Solution to exercise 30.5 (p.394): 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.

However, clever modifications of Gibbs sampling, using several annealing parameters, have been developed by Neal (2001), who has demonstrated that Monte Carlo decoding of certain codes, while inefficient, is not impossible.

Solution to exercise 30.10 (p.403): Each Metropolis proposal will take the energy of the state up or down by some amount. The total change in energy when $B$ proposals are concatenated will be the end-point of a random walk with $B$ steps in it. This walk might have mean zero, or it might have a tendency to drift upwards (if most moves increase the energy and only a few decrease it). In general the latter will hold, if the acceptance rate $f$ is small: the mean change in energy from any one move will be some $\Delta E > 0$ and so the acceptance probability for the concatenation of $B$ moves will be of order $1/(1 + exp(-B\Delta E))$, which scales roughly as $f^B$. The mean-square-distance moved will be of order $f^B B\epsilon^2$, where $\epsilon$ is the typical step size. In contrast, the mean-square-distance moved when the moves are considered individually will be of order $fB\epsilon^2$.

Solution to exercise 30.11 (p.405): The weights are $w = P(x)/Q(x)$ and $x$ is drawn from $Q$. The mean weight is

$$\int \mathrm{d}x\, Q(x)\,[P(x)/Q(x)] = \int \mathrm{d}x\, P(x) = 1, \tag{30.55}$$

assuming the integral converges. The variance is

$$\begin{aligned} \mathrm{var}(w) &= \int \mathrm{d}x\, Q(x) \left[\frac{P(x)}{Q(x)} - 1\right]^2 & (30.56)\\ &= \int \mathrm{d}x\, \frac{P(x)^2}{Q(x)} - 2P(x) + Q(x) & (30.57)\\ &= \left[\int \mathrm{d}x\, \frac{Z_Q}{Z_P^2} \exp\left(-\frac{x^2}{2}\left(\frac{2}{\sigma_p^2} - \frac{1}{\sigma_q^2}\right)\right)\right] - 1, & (30.58)\end{aligned}$$

where $Z_Q/Z_P^2 = \sigma_q/(\sqrt{2\pi}\sigma_p^2)$. The integral in (30.58) is finite only if the coefficient of $x^2$ in the exponent is positive, i.e., if

$$\sigma_q^2 > \frac{1}{2}\sigma_p^2. \tag{30.59}$$

If this condition is satisfied, the variance is

$$\mathrm{var}(w) = \frac{\sigma_q}{\sqrt{2\pi}\sigma_p^2}\sqrt{2\pi}\left(\frac{2}{\sigma_p^2} - \frac{1}{\sigma_q^2}\right)^{-1/2} - 1 \;=\; \frac{\sigma_q^2}{\sigma_p\left(2\sigma_q^2 - \sigma_p^2\right)^{1/2}} - 1. \tag{30.60}$$

As $\sigma_q$ approaches the critical value – about $0.7\sigma_p$ – the variance becomes infinite. Notice that the *empirical* standard deviation of the $R$ weights can look quite small and well-behaved (say, at $\sigma_q \simeq= 0.3$) when the true standard deviation is infinite.

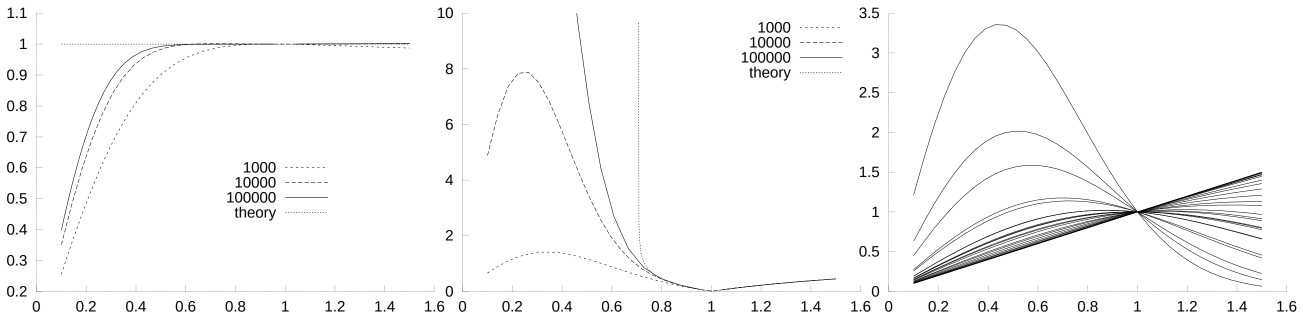


Figure 30.17. Importance sampling in one dimension. For $R = 1000$, $10^4$, and $10^5$, the normalizing constant of a Gaussian distribution (known in fact to be 1!) was estimated using importance sampling with a sampler density of standard deviation $\sigma_q$ (horizontal axis). The same random number seed was used for all runs. The three plots show (a) the estimated normalizing constant; (b) the *empirical* standard deviation of the $R$ weights; (c) 30 of the weights.

Solution to exercise 30.12 (p.406): 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

$$P'(x) = \begin{cases} 1/20 & x \in \{1, 2, \ldots, 19\} \\ 1/40 & x \in \{0, 20\} \\ 0 & \text{otherwise} \end{cases} . \tag{30.61}$$

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 30.13 (p.406): 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.

# 31

# *Efficient Monte Carlo methods* *

This chapter discusses several methods for reducing random walk behaviour in Metropolis methods.

## 31.1 Hybrid Monte Carlo

The hybrid Monte Carlo method reviewed in Neal (1993) 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

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{31.1}$$

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

$$H(\mathbf{x}, \mathbf{p}) = E(\mathbf{x}) + K(\mathbf{p}), \tag{31.2}$$

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

$$P_H(\mathbf{x}, \mathbf{p}) = \frac{1}{Z_H} \exp[-H(\mathbf{x}, \mathbf{p})] = \frac{1}{Z_H} \exp[-E(\mathbf{x})] \exp[-K(\mathbf{p})]. \tag{31.3}$$

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 $P(\mathbf{x})$.

```
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 31.1. Octave source code for the hybrid Monte Carlo method.

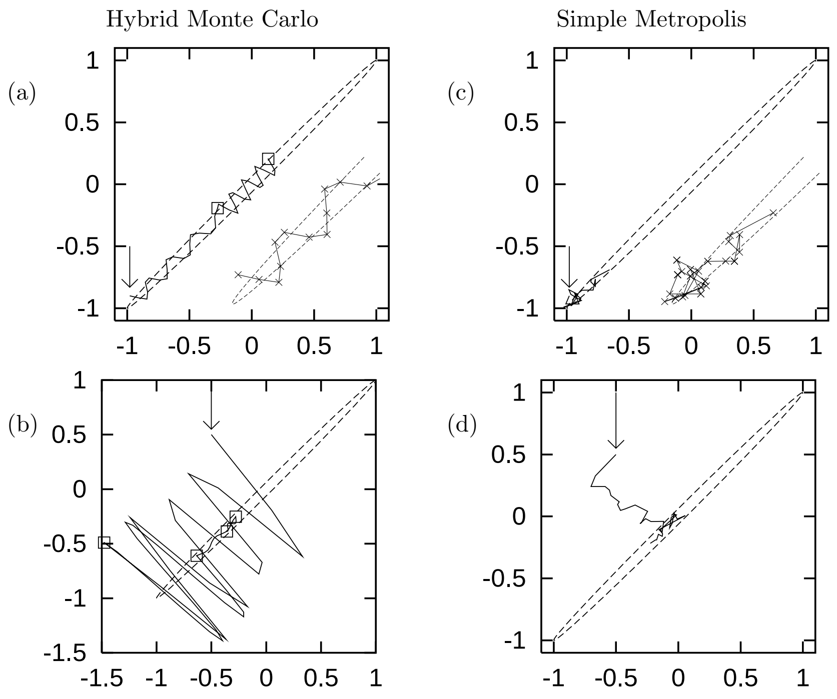


Figure 31.2. (a,b) Hybrid Monte Carlo used to generate samples from a bivariate Gaussian with correlation $\rho = 0.998$. (c,d) A simple random-walk Metropolis method for comparison.

## 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

$$\dot{\mathbf{x}} = \mathbf{p} \tag{31.4}$$

$$\dot{\mathbf{p}} = -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}}. \tag{31.5}$$

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 (30.29) 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 31.1 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 31.2 shows this algorithm generating samples from a bivariate Gaussian whose energy function is $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}$ with

$$\mathbf{A} = \begin{bmatrix} 250.25 & -249.75 \\ -249.75 & 250.25 \end{bmatrix}. \tag{31.6}$$

In figure 31.2(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 31.2(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 31.2(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.

## 31.2 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 $P(x, y) = \exp(-x^2y^2)/Z$.

### *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 31.3. 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.

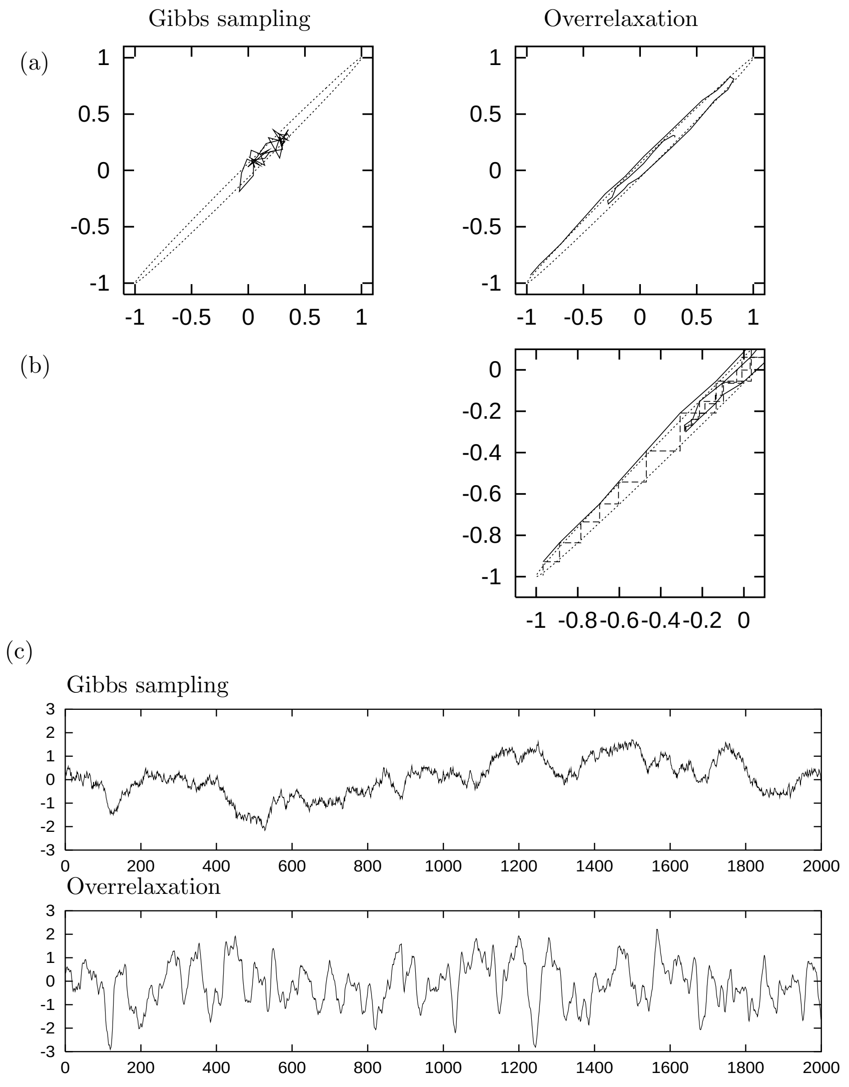


Figure 31.3. Overrelaxation contrasted with Gibbs sampling for a bivariate Gaussian with correlation $\rho = 0.998$. (a) The state sequence for 40 iterations, each iteration involving one update of both variables. The overrelaxation method had $\alpha = -0.98$. (This excessively large value is chosen to make it easy to see how the overrelaxation method reduces random walk behaviour.) The dotted line shows the contour $\mathbf{x}^{\mathsf{T}}\Sigma^{-1}\mathbf{x} = 1$. (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).)

If the conditional distribution of $x_i$ is $\text{Normal}(\mu, \sigma^2)$ and the current value of $x_i$ is $x_i^{(t)}$, then Adler's method sets $x_i$ to

$$x_i^{(t+1)} = \mu + \alpha(x_i^{(t)} - \mu) + (1 - \alpha^2)^{1/2}\sigma\nu, \tag{31.7}$$

where $\nu \sim \text{Normal}(0, 1)$ and $\alpha$ is a parameter between $-1$ and 1, usually set to a negative value. (If $\alpha$ is positive, then the method is called under-relaxation.)

Exercise 31.1:$^{A2}$ Show that this individual transition leaves invariant the conditional distribution $x_i \sim \text{Normal}(\mu, \sigma^2)$.

A single iteration of Adler's overrelaxation, like one of Gibbs sampling, updates each variable in turn as indicated in equation (31.7). 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 correlated, then they will (on a short timescale) evolve in a directed manner instead of by random walk, as shown in figure 31.3. This may significantly reduce the time required to obtain effectively independent samples.

Figure 31.3 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 $\kappa$ 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 $\alpha$ 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.

## 31.3 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

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{31.8}$$

where $E(\mathbf{x})$ can be evaluated. In the simplest simulated annealing method, we instead sample from the distribution

$$P_T(\mathbf{x}) = \tfrac{1}{Z(T)} e^{-\frac{E(\mathbf{x})}{T}} \tag{31.9}$$

and decrease $T$ gradually to 1.

Often the energy function can be separated into two terms,

$$E(\mathbf{x}) = E_0(\mathbf{x}) + E_1(\mathbf{x}), \tag{31.10}$$

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

$$P'_T(\mathbf{x}) = \tfrac{1}{Z'(T)} e^{-E_0(\mathbf{x}) - \frac{E_1(\mathbf{x})}{T}} \tag{31.11}$$

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. 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.

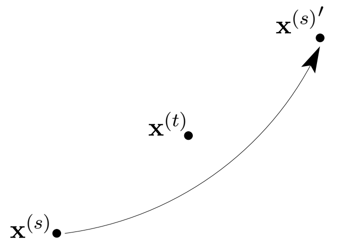
## 31.4 Skilling's leapfrog method*

A fourth method for reducing random walk behaviour was introduced by Skilling (unpublished, 2002); it has a similar spirit to the overrelaxation method, but works in more dimensions. This method is applicable to sampling from a distribution over a continuous state space, and the sole requirement is that the energy $E(\mathbf{x})$ should be easy to evaluate. The gradient is not used. This leapfrog method is not intended to be used on its own but, rather, in sequence with other Monte Carlo operators.

Instead of moving just one state vector $\mathbf{x}$ around the state space, as was the case for all the Monte Carlo methods discussed thus far, Skilling's leapfrog method simultaneously maintains a *set* of $S$ state vectors $\{\mathbf{x}^{(s)}\}$, where $S$ might be six or twelve. The aim is that all $S$ of these vectors will represent independent samples from the same distribution $P(\mathbf{x})$.

Skilling's leapfrog makes a proposal for the new state $\mathbf{x}^{(s)}$, which is accepted or rejected in accordance with the Metropolis method, by leapfrogging the current state $\mathbf{x}^{(s)}$ over another state vector $\mathbf{x}^{(t)}$:

$$\mathbf{x}^{(s)\prime} = 2\mathbf{x}^{(t)} - \mathbf{x}^{(s)}. \tag{31.12}$$



All the other state vectors are left where they are, so the acceptance probability depends only on the change in energy of $\mathbf{x}^{(s)}$.

Which vector, $t$, is the partner for the leapfrog event can be chosen in various ways. The vanilla method is to select the partner at random from the other vectors. It might be better to choose $t$ by selecting one of the nearest neighbours $\mathbf{x}^{(s)}$ – nearest by any chosen distance function – as long as one then uses an acceptance rule that ensures detailed balance by checking whether point $t$ is still among the nearest neighbours of the new point, $\mathbf{x}^{(s)\prime}$.

### *Why the leapfrog is a good idea*

Imagine that the target density $P(\mathbf{x})$ has strong correlations – for example, the density might be a needle-like Gaussian with width $\epsilon$ and length $L\epsilon$, where $L \gg 1$. As we have emphasised, motion around such a density by standard methods proceeds by a slow random walk.

Imagine now that our set of $S$ points is lurking initially in a location that is probable under the density, but in an inappropriately small ball of size $\epsilon$. Now, under Skilling's leapfrog method, a typical first move will take the point a little outside the current ball, perhaps doubling its distance from the centre of the ball. After all the points have had a chance to move, the ball will have increased in size; if all the moves are accepted, the ball will be bigger by a factor of two or so in all dimensions. The rejection of some moves will mean that the ball containing the points will probably have elongated in the needle's long direction by a factor of, say, two. After another cycle through the points, the ball will have grown in the long direction by another factor of two. So the typical distance travelled in the long dimension grows *exponentially* with the number of iterations.

Now, maybe a factor of two growth per iteration is on the optimistic side; but even if the ball only grows by a factor of, let's say, 1.1 per iteration, the growth is nevertheless exponential. It will only take a number of iterations proportional to $\log L/\log(1.1)$ for the long dimension to be explored.

Solutions on p.419

Exercise 31.2:$^{B2}$ Discuss how the effectiveness of Skilling's method scales with dimensionality, using a correlated $N$-dimensional Gaussian distribution as an example. Find an expression for the rejection probability, assuming the Markov chain is at equilibrium. Also discuss how it scales with the strength of correlation among the Gaussian variables. [Hint: Skilling's method is invariant under affine transformations, so the rejection probability at equilibrium can be found by looking at the case of a *separable* Gaussian.]

This method has some similarity to the "adaptive direction sampling" method of Gilks *et al.* (1994) but the leapfrog method is simpler and can be applied to a greater variety of distributions.

## 31.5 Multi-state methods*

In a multi-state method, multiple parameter vectors $\mathbf{x}$ are maintained; they evolve individually under moves such as Metropolis and Gibbs; there are also interactions among the vectors. The intention is that eventually all the vectors $\mathbf{x}$ should be samples from $P(\mathbf{x})$, or that information associated with them should allow us to approximate expectations under $P(\mathbf{x})$, as in importance sampling.

### *Genetic methods*

There is a good reason for wanting to use a population and have sex. Inference computational task is to hunt for the the place where the probability is big. This is difficult. As the chapter on sex and evolution (chapter 20) shows, if the problem can be decomposed at all, it's more efficient to have many individuals and crossover and selection. The following gives a particular way of making crossover and selection in a principled way.

I'll use $R$ to denote the number of vectors in the population. We aim to have $P(\{\mathbf{x}\}_1^R) = \prod P^*(\mathbf{x}^{(r)})$. We allow moves of the form $\mathbf{x}, \mathbf{y} \rightarrow \mathbf{x}', \mathbf{y}'$ and accept or reject using Metropolis. (This is not how many people do Genetic algorithm, but it is a good idea)

Birth or death rule: Skilling's method couple deaths and births to the annealing process. Reduce temperature such that just one of $R$ is likely to die.

### *Particle filters*

See next edition of the book!

## 31.6 Methods that do not necessarily help

It is common practice to use many initial conditions for a particular Markov chain (figure 30.16). If you are worried about sampling well from a complicated density $P(\mathbf{x})$, *can* you ensure the states produced by the simulations are well distributed about the typical set of $P(\mathbf{x})$ by ensuring that the initial points are 'well distributed about the whole state space'?

The answer is, unfortunately, no. In hierarchical Bayesian models, for example, a large number of parameters $\{x_n\}$ may be coupled together via a another parameter $\beta$ (known as a hyperparameter). For example, the quantities $\{x_n\}$ might be independent noise signals, and $\beta$ might be the inverse-variance of the noise source. The joint distribution of $\beta$ and $\{x_n\}$ might be

$$\begin{aligned} P(\beta, \{x_n\}) &= P(\beta) \prod_{n=1}^{N} P(x_n|\beta) \\ &= P(\beta) \prod_{n=1}^{N} \frac{1}{Z(\beta)} e^{-\beta x_n^2/2}, \end{aligned}$$

where $Z(\beta) = \sqrt{2\pi/\beta}$ and $P(\beta)$ is a broad distribution describing our ignorance about the noise level. For simplicity, let's leave out all the other variables – data and such – that might be involved in a realistic problem! Let's imagine that we want to sample effectively from $P(\beta, \{x_n\})$ by Gibbs sampling – alternately sampling $\beta$ from the conditional distribution $P(\beta|x_n)$ then sampling all the $x_n$ from their conditional distributions $P(x_n|\beta)$. [The resulting marginal distribution of $\beta$ should asymptotically be the broad distribution $P(\beta)$.]

If $N$ is large then the conditional distribution of $\beta$ given any particular setting of $\{x_n\}$ will be tightly concentrated on a particular most probable value of $\beta$, with width proportional to $1/\sqrt{N}$. Progress up and down the $\beta$-axis will therefore take place by a slow random walk with steps of size $\propto 1/\sqrt{N}$.

So, to the initialization strategy. Can we finesse our slow convergence problem by using initial conditions located 'all over the state space'? Sadly, no. If we distribute the points $\{x_n\}$ widely, what we are actually doing is favouring an initial value of the noise level $1/\beta$ that is *large*. The random walk of the parameter $\beta$ will thus tend, after the first drawing of $\beta$ from $P(\beta|x_n)$, always to start off from one end of the $\beta$-axis.

## Solutions

Solution to exercise 31.2 (p.417): Consider the spherical Gaussian distribution where all components have mean zero and variance 1. In one dimension, the $n$th, if $x_n^{(1)}$ leapfrogs over $x_n^{(2)}$, we obtain the proposed coordinate

$$(x_n^{(1)})' = 2x_n^{(2)} - x_n^{(1)}. \tag{31.13}$$

Assuming that $x_n^{(1)}$ and $x_n^{(2)}$ are Gaussian random variables from Normal(0,1), $(x_n^{(1)})'$ is Gaussian from Normal(0,$\sigma^2$), where $\sigma^2 = 2^2 + (-1)^2 = 5$. The change in energy contributed by this one dimension will be

$$\frac{1}{2}\left[(2x_n^{(2)} - x_n^{(1)})^2 - (x_n^{(1)})^2\right] = 2(x_n^{(2)})^2. \tag{31.14}$$

So the typical change in energy is $2\langle (x_n^{(2)})^2 \rangle = 2$. This positive change is bad news. In $N$ dimensions, the typical change in energy when a leapfrog move is made, at equilibrium, is thus $+2N$. The probability of acceptance of the move scales as

$$e^{-2N}. \tag{31.15}$$

This implies that Skilling's method, as described, will not be effective in very high-dimensional problems – at least, it's not effective once convergence has occured. Nevertheless it has the impressive advantage that its convergence properties are independent of the strength of correlations between the variables – a property that not even the hybrid Monte Carlo and overrelaxation methods offer.

# About Chapter 32

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 27 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 16): 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 B.3 helpful. I also recommend the book by Reif (1965).

# 32

# *Ising Models*

An Ising model is an array of spins (e.g., atoms that can take states $\pm 1$) 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 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 coupling between neighbouring spins is $J$. We define $J_{mn} = J$ if $m$ and $n$ are neighbours and $J_{mn} = 0$ otherwise. The energy of a state $\mathbf{x}$ is

$$E(\mathbf{x}; J, H) = -\left[\frac{1}{2}\sum_{m,n} J_{mn} x_m x_n + \sum_n H x_n\right], \tag{32.1}$$

where $H$ is the applied field. If $J > 0$ then the model is ferromagnetic, and if $J < 0$ it is antiferromagnetic. Note that we've included the factor of $1/2$ because each pair is counted twice in the first sum, once as $(m, n)$ and once as $(n, m)$. At equilibrium at temperature $T$, the probability that the state is $\mathbf{x}$ is

$$P(\mathbf{x}|\beta, J, H) = \frac{1}{Z(\beta, J, H)} \exp\left[-\beta E(\mathbf{x}; J, H)\right] \tag{32.2}$$

where $\beta = 1/k_{\mathrm{B}}T$, $k_{\mathrm{B}}$ is Boltzmann's constant, and

$$Z(\beta, J, H) \equiv \sum_{\mathbf{x}} \exp\left[-\beta E(\mathbf{x}; J, H)\right]. \tag{32.3}$$

## *Relevance of Ising models*

Ising models are relevant for three reasons.

Ising models are important first as models of magnetic systems that have a phase transition. 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 not only about magnetic phase transitions but also about phase transitions in many other systems.

Second, if we generalize the energy function to

$$E(\mathbf{x}; \mathbf{J}, \mathbf{h}) = -\left[\frac{1}{2}\sum_{m,n} J_{mn} x_m x_n + \sum_n h_n x_n\right], \tag{32.4}$$

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 this chapter 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 computations. Consider for example the heat capacity of a system, which is defined to be

$$C \equiv \frac{\partial}{\partial T}\bar{E}, \tag{32.5}$$

where

$$\bar{E} = \frac{1}{Z}\sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x}))\, E(\mathbf{x}). \tag{32.6}$$

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,

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})). \tag{32.7}$$

The mean energy is obtained by differentiation with respect to $\beta$:

$$\frac{\partial \log Z}{\partial \beta} = \frac{1}{Z}\sum_{\mathbf{x}} -E(\mathbf{x}) \exp(-\beta E(\mathbf{x})) = -\bar{E}. \tag{32.8}$$

A further differentiation spits out the variance of the energy:

$$\frac{\partial^2 \log Z}{\partial \beta^2} = \frac{1}{Z}\sum_{\mathbf{x}} E(\mathbf{x})^2 \exp(-\beta E(\mathbf{x})) - \bar{E}^2 = \langle E^2 \rangle - \bar{E}^2 = \mathrm{var}(E). \tag{32.9}$$

But the heat capacity is also the derivative of $\bar{E}$ with respect to temperature:

$$\frac{\partial \bar{E}}{\partial T} = -\frac{\partial}{\partial T}\frac{\partial \log Z}{\partial \beta} = -\frac{\partial^2 \log Z}{\partial \beta^2}\frac{\partial \beta}{\partial T} = -\mathrm{var}(E)(-1/k_{\mathrm{B}}T^2). \tag{32.10}$$

So for any system at temperature $T$,

$$C = \frac{\mathrm{var}(E)}{k_{\mathrm{B}}T^2} = k_{\mathrm{B}}\beta^2 \mathrm{var}(E) \tag{32.11}$$

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 words 'essentially zero' above. The heat capacity is not quite zero at high temperature, it just tends to zero. And it tends to zero as $\frac{\text{var}(E)}{k_B T^2}$, with the quantity 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 very 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

$$C^{(\beta)} \equiv \frac{\partial \bar{E}}{\partial \beta} = \text{var}(E). \tag{32.12}$$

So heat capacity and fluctuations would be identical quantities, were it not for this slip by Kelvin, Carnot et al.

Exercise 32.1:$^{B2}$ [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 $k_B = 1$.]

The entropy of a system whose states are $\mathbf{x}$, at temperature $T = 1/\beta$, is

$$S = \sum p(\mathbf{x}) \left[\log 1/p(\mathbf{x})\right]. \tag{32.13}$$

where

$$p(\mathbf{x}) = \frac{1}{Z(\beta)} \exp\left[-\beta E(\mathbf{x})\right] \tag{32.14}$$

(a) Show that

$$S = \log Z(\beta) + \beta \bar{E}(\beta) \tag{32.15}$$

where $\bar{E}(\beta)$ is the mean energy of the system.

(b) Show that

$$S = -\frac{\partial F}{\partial T}, \tag{32.16}$$

where the free energy $F = -kT \log Z$ and $kT = 1/\beta$.

## 32.1 Ising Models — Monte Carlo simulation

In this section we 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,

$$P(+1|h_n) = \frac{1}{1 + \exp(-2\beta b_n)}, \tag{32.17}$$

where $\beta = 1/k_B T$ and $b_n$ is the local field

$$b_n = \sum_{m:(m,n)\in\mathcal{N}} J x_m + H. \tag{32.18}$$

[The factor of 2 appears in equation (32.17) 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 the next spin to update is selected at random. After sufficiently many iterations, this procedure converges to the equilibrium distribution (32.2). An alternative to the Gibbs sampling formula (32.17) is the Metropolis algorithm, in which we consider the change in energy that results from flipping the chosen spin from its current state $x_n$,

$$\Delta E = 2x_n b_n, \tag{32.19}$$

and adopt this change in configuration with probability

$$P(\text{accept}; \Delta E, \beta) = \begin{cases} 1 & \Delta E \leq 0 \\ \exp(-\beta \Delta E) & \Delta E > 0 \end{cases}. \tag{32.20}$$

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.

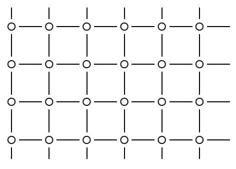


Figure 32.1. Rectangular Ising model

### *Rectangular geometry*

I first simulated an Ising model with the rectangular geometry shown in figure 32.1, 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 $m$,

$$m = \frac{1}{N} \sum_n x_n. \tag{32.21}$$

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 at each temperature, $I$, be a few hundred times the number of spins $N$, and to discard the first 1/3 of those iterations. With $N$=100, I found I needed more than 100,000 iterations to reach equilibrium at any given temperature.

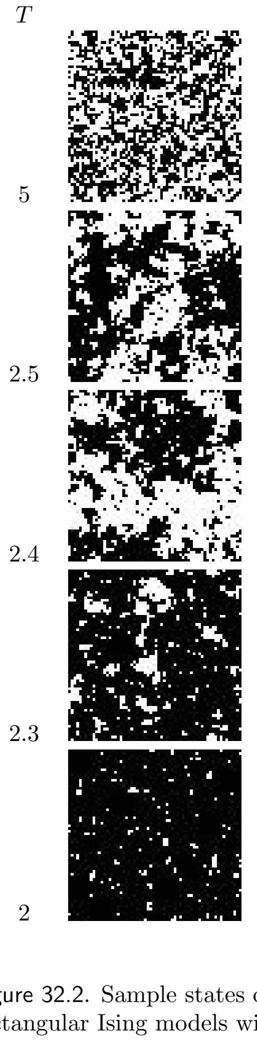


Figure 32.2. Sample states of rectangular Ising models with $J = 1$ at a sequence of temperatures $T$.

### *Results for small $N$ with $J = 1$.*

I simulated an $l \times l$ grid for $l = 4, 5, \ldots, 10, 40, 64$. 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

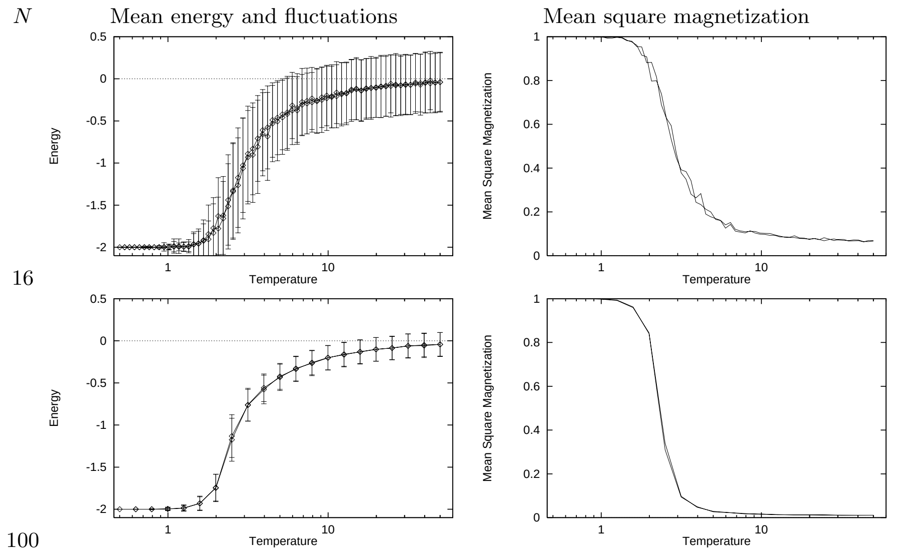


Figure 32.3. Monte Carlo simulations of rectangular Ising models with $J = 1$. Mean energy and fluctuations in energy as a function of temperature (left). Mean square magnetization as a function of temperature (right). In the top row, $N = 16$, and the bottom, $N = 100$. For even larger $N$, see later figures.

either ground state is $-2$. At high temperatures, the spins are independent, all states are equally probable, and the energy is expected to fluctuate around a mean of 0 with a standard deviation proportional to $1/\sqrt{N}$.

Let's look at some results. In all figures temperature $T$ is shown with $k_B = 1$. The basic picture emerges with as few as 16 spins (figure 32.3, top): the energy rises monotonically. As we increase the number of spins to 100 (figure 32.3, 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 32.4 shows details of the graphs for $N = 100$ and $N = 4096$. Figure 32.2 shows a sequence of typical states from the simulation of $N = 4096$ spins at a sequence of decreasing temperatures.

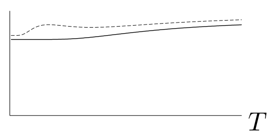


Figure 32.5. Schematic diagram to explain the meaning of a Schottky anomaly. The curve shows the heat capacity of two gases as a function of temperature. The lower curve shows a normal gas whose heat capacity is an increasing function of temperature. The upper curve has a small peak in the heat capacity, 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 or increasing functions of temperature. In contrast, systems with a finite number of levels produced small blips in the heat capacity graph (figure 32.5).

Let us refresh our memory of the simplest such system, a two-level system with states $x = 0$ (energy 0) and $x = 1$ (energy $\epsilon$). The mean energy is

$$E(\beta) = \epsilon \frac{\exp(-\beta\epsilon)}{1 + \exp(-\beta\epsilon)} = \epsilon \frac{1}{1 + \exp(\beta\epsilon)} \tag{32.22}$$

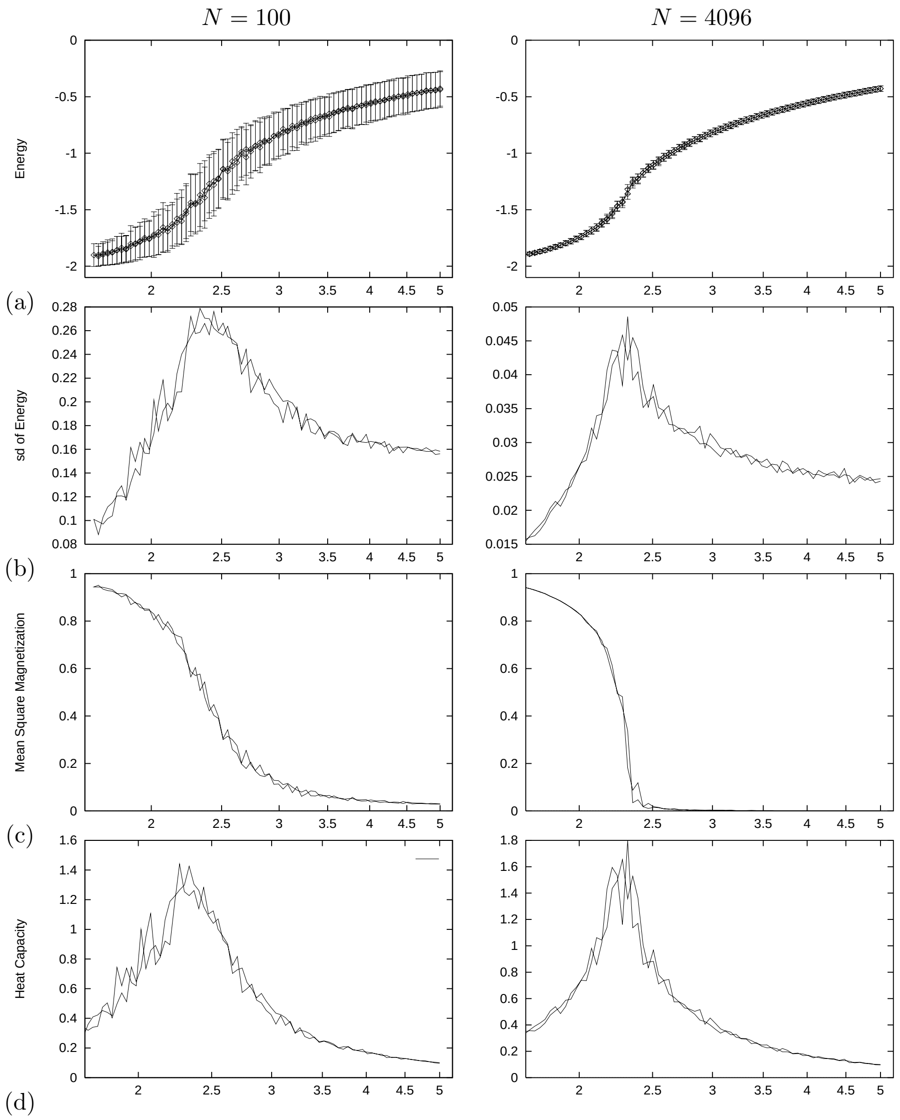


Figure 32.4. Detail of Monte Carlo simulations of rectangular Ising models with $J = 1$. (a) Mean energy and fluctuations in energy as a function of temperature. (b) Fluctuations in energy (standard deviation). (c) Mean square magnetization. (d) Heat capacity.

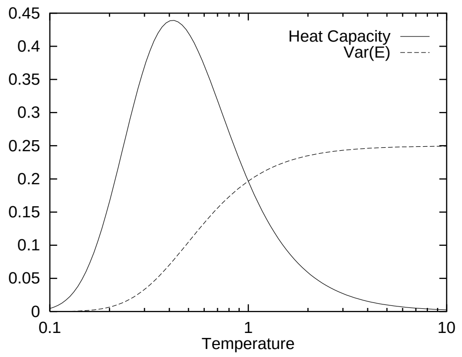


Figure 32.6. 'Schottky anomaly' — Heat capacity and fluctuations in energy as a function of temperature for a two-level system with separation $\epsilon = 1$ and $k_B = 1$.

and the derivative with respect to $\beta$ is

$$dE/d\beta = -\epsilon^2 \frac{\exp(\beta\epsilon)}{[1 + \exp(\beta\epsilon)]^2}. \tag{32.23}$$

So the heat capacity is

$$C = dE/dT = -\frac{dE}{d\beta}\frac{1}{k_B T^2} = \frac{\epsilon^2}{k_B T^2}\frac{\exp(\beta\epsilon)}{[1 + \exp(\beta\epsilon)]^2} \tag{32.24}$$

and the fluctuations in energy are given by $\text{var}(E) = C k_B T^2 = -dE/d\beta$, which was evaluated in (32.23). The heat capacity and fluctuations are plotted in figure 32.6. 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 proportional to the number of independent spins. Thus it is a peak in the *fluctuations* which is interesting, rather than a peak in the heat capacity. The Ising model has such a peak in its fluctuations, as can clearly be seen in figure 32.4.

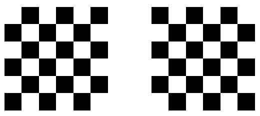


Figure 32.7. The two ground states of a rectangular Ising model with $J = -1$.

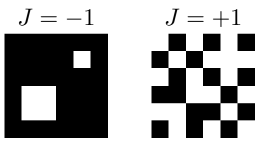


Figure 32.8. Two states of rectangular Ising models with $J = \pm 1$ that have identical energy.

## *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 (figure 32.7), 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 $J = -1$ (figure 32.8), 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,

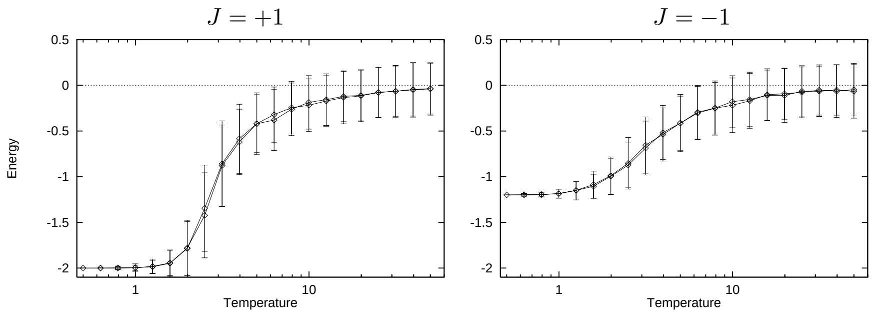


Figure 32.9. Monte Carlo simulations of rectangular Ising models with $J = \pm 1$ and $N = 25$. Mean energy and fluctuations in energy as a function of temperature. (a) $J = 1$. (b) $J = -1$.

and the energy of those ground states will not be as low as $-2$ per spin. So we 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 32.9 compares the energies of the ferromagnetic and antiferromagnetic models with $N = 25$. Here, the difference is striking.

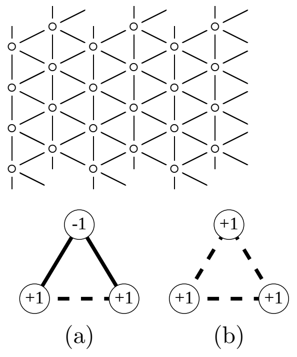


Figure 32.10. In an antiferromagnetic triangular Ising model, any three neighbouring spins are frustrated. Of the eight possible configurations of three spins, six have energy $-|J|$ (a), and two have energy $3|J|$ (b). Solid lines show 'happy' couplings which contribute $-|J|$ to the energy; dashed 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 32.10. 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. Figure 32.12 shows 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.

## 32.2 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

$$Z(\beta, \mathbf{J}, \mathbf{b}) \equiv \sum_{\mathbf{x}} \exp\left[-\beta E(\mathbf{x}; \mathbf{J}, \mathbf{b})\right] \tag{32.25}$$

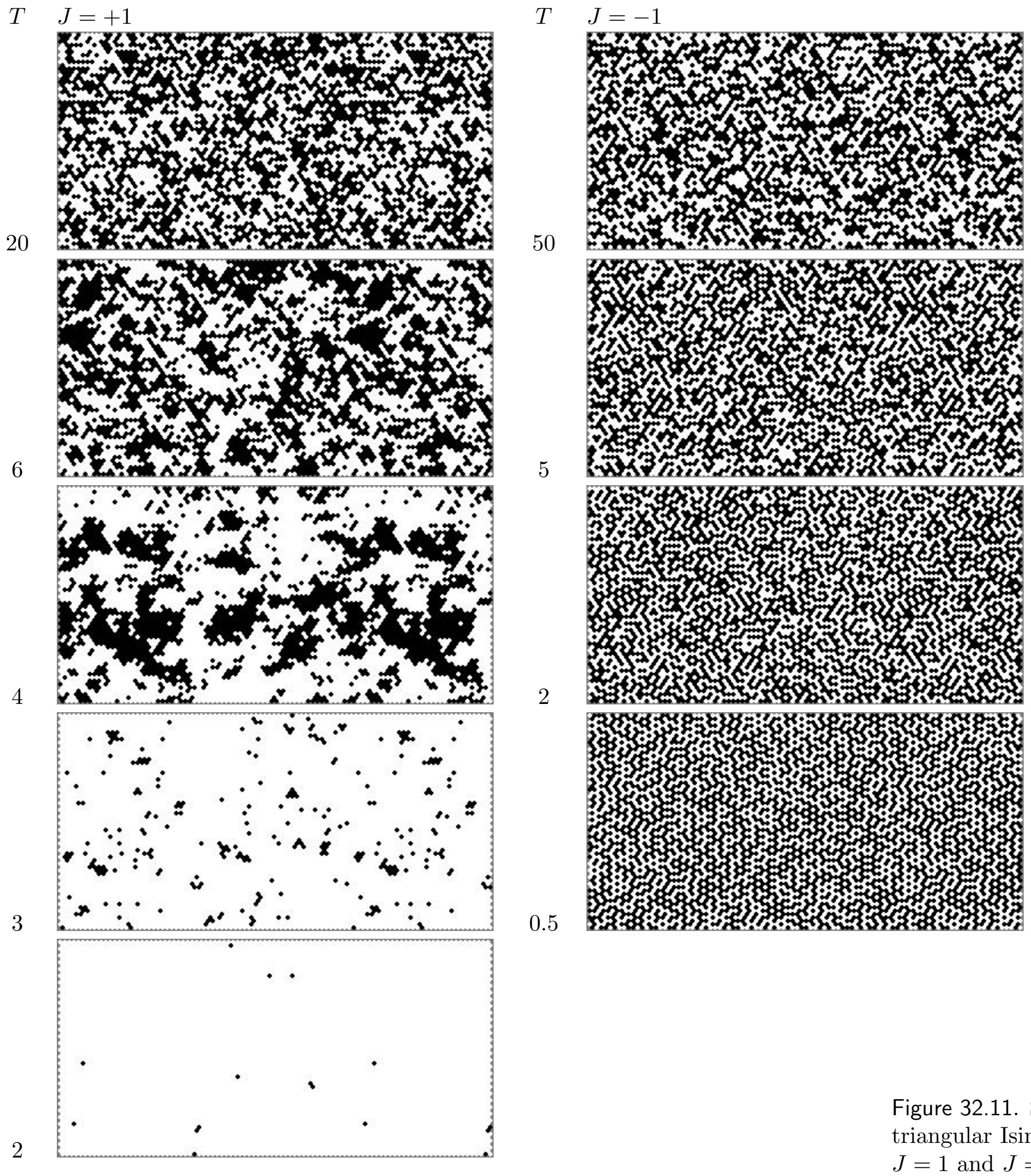


Figure 32.11. Sample states of triangular Ising models with $J = 1$ and $J = -1$.

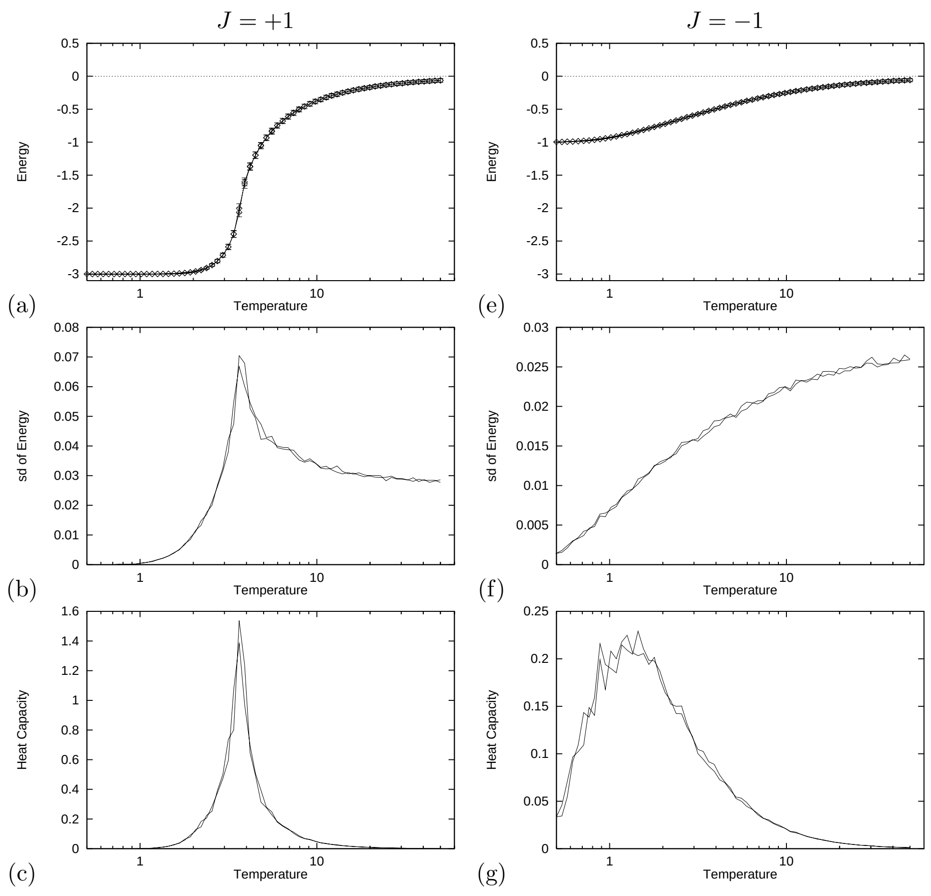


Figure 32.12. Monte Carlo simulations of triangular Ising models with $J = \pm 1$ and $N = 4096$. (a-c) $J = 1$. (e-g) $J = -1$. (a,e) Mean energy and fluctuations in energy as a function of temperature. (b,f) Fluctuations in energy (standard deviation). (c,g) Heat capacity.

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 27. 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 involves a summation over all the states at the boundary. This operation is 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 $r$th bit of $s_c$ indicates whether the spin in row $r$, column $c$ is up or down. The partition function is

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})) \tag{32.26}$$

$$= \sum_{s_1}\sum_{s_2}\cdots\sum_{s_C} \exp\left(-\beta \sum_{c=1}^{C} \mathcal{E}(s_c, s_{c+1})\right) \tag{32.27}$$

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:

$$\mathcal{E}(s_c, s_{c+1}) = \frac{1}{2} \sum_{\substack{(m,n)\in\mathcal{N}: \\ m\in c, n\in c+1}} J\, x_m x_n + \frac{1}{4} \sum_{\substack{(m,n)\in\mathcal{N}: \\ m\in c, n\in c}} J\, x_m x_n + \frac{1}{4} \sum_{\substack{(m,n)\in\mathcal{N}: \\ m\in c+1, n\in c+1}} J\, x_m x_n. \tag{32.28}$$

This definition of the energy 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 1/4 are needed because vertical links are counted four times. Let us define

$$M_{ss'} = \exp\left(-\beta \mathcal{E}(s, s')\right). \tag{32.29}$$

Then continuing from equation (32.27),

$$Z = \sum_{s_1}\sum_{s_2}\cdots\sum_{s_C} \left[\prod_{c=1}^{C} M_{s_c, s_{c+1}}\right] \tag{32.30}$$

$$= \text{Trace}\left[\mathbf{M}^C\right] \tag{32.31}$$

$$= \sum_a \mu_a^C, \tag{32.32}$$

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 $\mu_{\max}$:

$$Z \to \mu_{\max}^C. \tag{32.33}$$

So the free energy per spin in the limit of an infinite thin strip is given by:

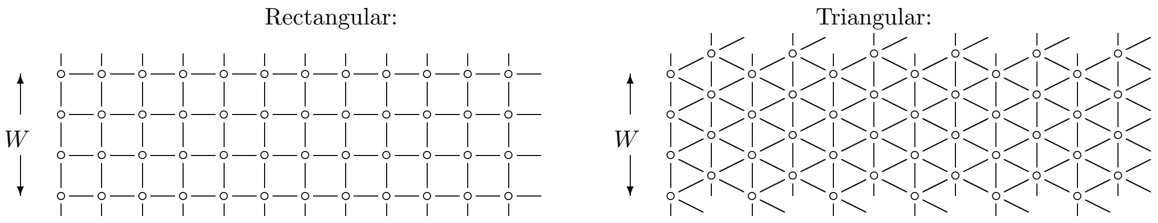
$$f = -kT \log Z/(WC) = -kTC \log \mu_{\max}/(WC) = -kT \log \mu_{\max}/W. \quad (32.34)$$

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 $\beta$ 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 (32.32) — 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.

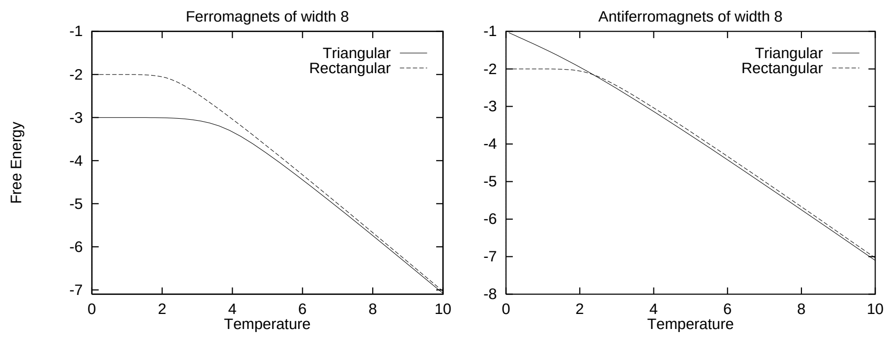


Figure 32.13. Free energy per spin of long-thin-strip Ising models. Note the non-zero gradient at $T = 0$ in the case of the triangular antiferromagnet.

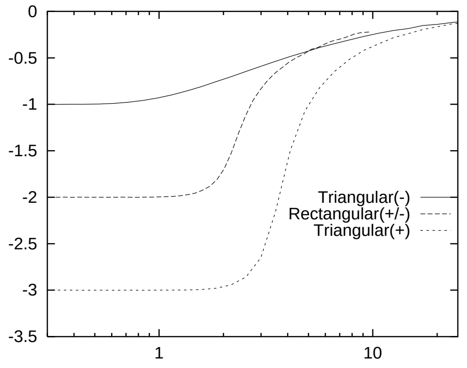


Figure 32.14. Mean energy versus temperature of long thin strip Ising models with width 8. Compare with figure 32.3.

The mean energy as a function of temperature is plotted in figure 32.14. It is evaluated using the identity $\langle E\rangle = -\partial \log Z/\partial\beta$.

Figure 32.16 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 32.17 shows the fluctuations in energy as a function of

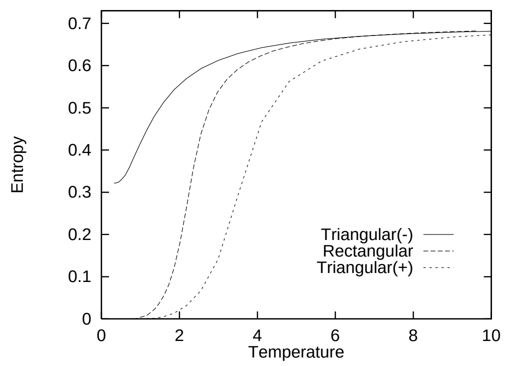


Figure 32.15. Entropies of width 8 Ising systems as a function of temperature, obtained by differentiating the free energy curves in figure 32.13. The rectangular ferromagnet and antiferromagnet have identical thermal properties. For the triangular systems, (+) denotes the ferromagnet and (−) the antiferromagnet.

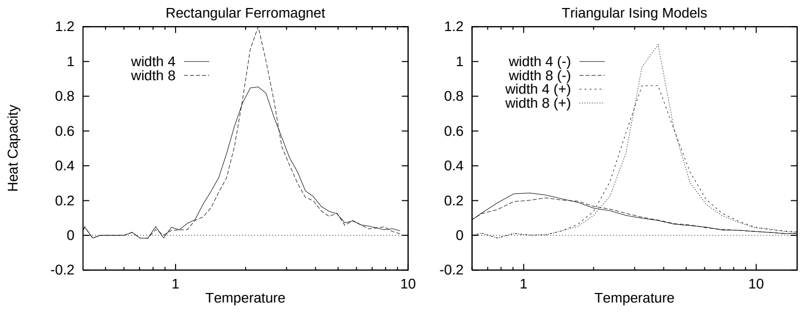


Figure 32.16. Heat Capacities of (a) Rectangular model; (b) Triangular models with different widths, (+) and (−) denoting ferromagnet and antiferromagnet. Compare with figure 32.12.

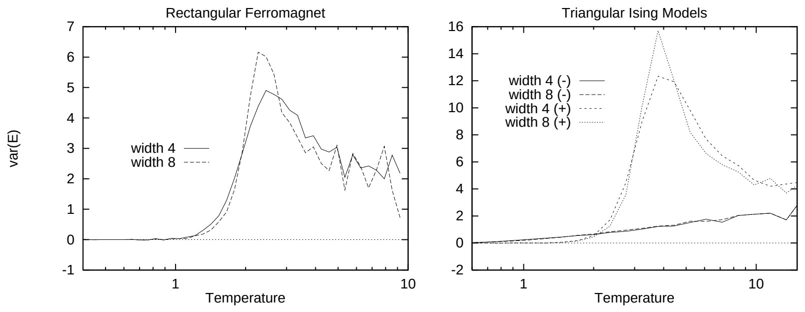


Figure 32.17. Energy variances, per spin, of (a) Rectangular model; (b) Triangular models with different widths, (+) and (−) denoting ferromagnet and antiferromagnet. Compare with figure 32.12.

temperature. All of these figures should show smooth graphs; the roughness of the curves is presumably due to inaccurate eigenvalue evaluation. The nature 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 in figure 32.13 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 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.

## 32.3 Problems

Exercise 32.2:$^{B4}$ (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 32.3:$^{A3}$ 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 $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \ldots, \mathbf{x}_{(S)}$. Can you think of a way of setting $\mathbf{J}$ such that the chosen states are low energy states? You are allowed to adjust all the $\{J_{mn}\}$ to whatever values you wish.

# Solutions to Chapter 32's exercises

Solution to exercise 32.1 (p.423): ...

Solution to exercise 32.3 (p.435): This is the problem of creating a system whose stable states are a desired set of memories. See later chapters for some ideas.

# 33

# *Exact Monte Carlo Sampling* *

## 33.1 The problem with Monte Carlo methods

For high-dimensional problems, the most widely used random sampling methods are Markov chain Monte Carlo methods like the Metropolis method, Gibbs sampling, and slice sampling.

The problem with all these methods is this: yes, a given algorithm can be guaranteed to produce samples from the target density $P(\mathbf{x})$ asymptotically, 'once the chain has converged to the equilibrium distribution'. But if one runs the chain for too short a time $T$, then the samples will come from some other distribution $P^{(T)}(\mathbf{x})$. For how long must the Markov chain be run before it has 'converged'? As was mentioned in chapter 30, this question is usually very hard to answer. However, the pioneering work of Propp and Wilson (1996) allows one, for certain chains, to answer this very question; furthermore Propp and Wilson show how to obtain 'exact' samples from the target density.

Their *exact sampling method* (also known as 'perfect simulation' or 'coupling from the past') depends on three ideas.

### *Coalescence of coupled Markov chains*

First, 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. Figure 33.1(a-i) shows twenty-one Markov chains identical to those described in section 30.4 (page 389), each having a different initial condition but driven by a single random number generator; the chains coalesce after about 80 steps. Figure 33.1(a-ii) shows the same Markov chains with a different random number seed; in this case, coalescence does not occur until 400 steps have elapsed (not shown). Figure 33.1(b) shows similar Markov chains, each of which has identical properties to those in section 30.4 and figure 33.1(a); but in figure 33.1(b), the proposed move at each step, 'left' or 'right', is obtained in the same way by all the chains at any timestep, independent of the current state. This coupling of the chains changes the statistics of coalescence. Because two neighbouring paths only merge when a rejection occurs, and rejections only occur at the walls (for this particular Markov chain), coalescence will always occur when the chains are all in one of the two end-states.

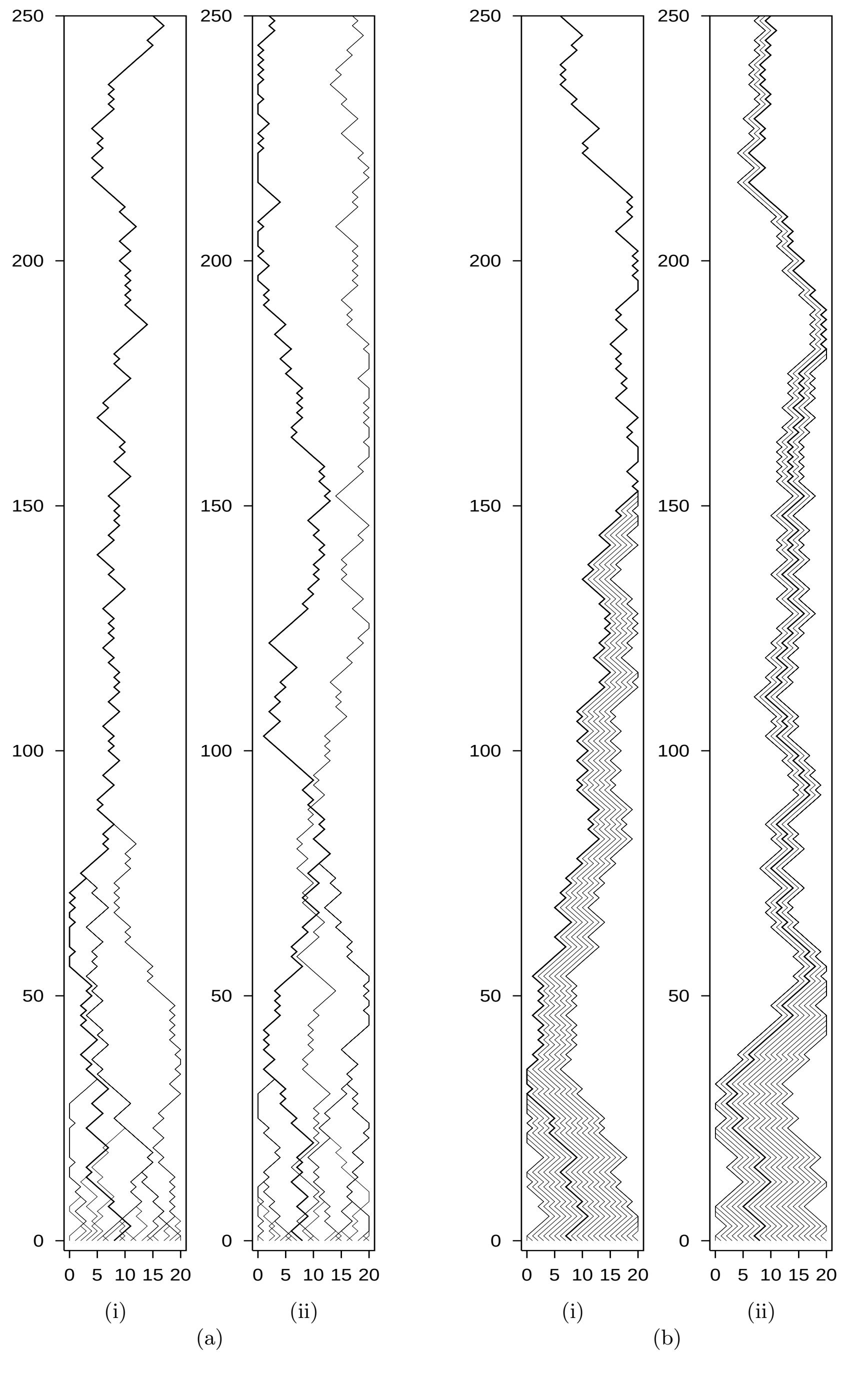


Figure 33.1. Coalescence, the first idea behind the exact sampling method. Different coalescence properties are shown depending on the way each state uses the random numbers it is supplied with. In the first and third panels shown, coalescence has occurred within 250 steps (a): Two runs of a Metropolis simulator in which the random bits that determine the proposed step depend on the current state; a different random number seed was used in each case. (b): In this simulator the random proposal ('left' or 'right') is the same for all states. In each panel, one of the paths, the one starting at location $x = 8$, has been highlighted.

## *Coupling from the past*

How can we use the 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 33.1(b), 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 $T_0$ 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 $T_0$ further into the past, *reusing the same random numbers.* The simulation is repeated at a sequence of ever more distant times $T_0$, with a doubling of $T_0$ from one run to the next being a convenient choice. When coalescence occurs at a time before 'the present', we can record $x(0)$ a an *exact sample* from the equilibrium distribution of the Markov chain. Figure 33.2 shows two exact samples produced in this way. In the leftmost panel of figure 33.2(a), we start twenty-one chains in all possible initial conditions at $T_0 = -50$ and run them forward in time. Coalescence does not occur. We restart the simulation from all possible initial conditions at $T_0 = -100$, and reset the random number generator in such a way that the random numbers generated at each time $t$ (in particular, from $t = -50$ to $t = 0$) will be identical to what they were in the first run. Notice that the trajectories produced from $t = -50$ to $t = 0$ by these runs that started from $T_0 = -100$ are identical to a *subset* of the trajectories in the first simulation with $T_0 = -50$. Coalescence still does not occur, so we double $T_0$ again to $T_0 = -200$. This time, all the trajectories coalesce and we obtain an exact sample, shown by the arrow. If we pick an earlier time such as $T_0 = -500$, all the trajectories must still end in the same point at $t = 0$, since all trajectories must pass through some state at $t = -200$, and all those states lead to the same final point. Figure 33.2(b) shows an exact sample produced in the same way with the Markov chain of figure 33.1(b).

This method, called *coupling from the past*, is important because it allows us to obtain exact samples from the equilibrium distribution; but, as described here, it is of little practical use, since we are obliged to simulate chains starting in *all* initial states. In the examples shown, there are only twenty-one states, but in any realistic sampling problem there will be an utterly enormous number of states – think of the $2^{1000}$ states of a system of 1000 binary spins, for example. The whole point of introducing Monte Carlo methods was to try to avoid having to visit all the states of such a system!

## *Monotonicity*

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 those trajectories. This property holds, for example, in the chain of figure 33.1(b), which has the property that *two trajectories never cross.* So if we simply track the two

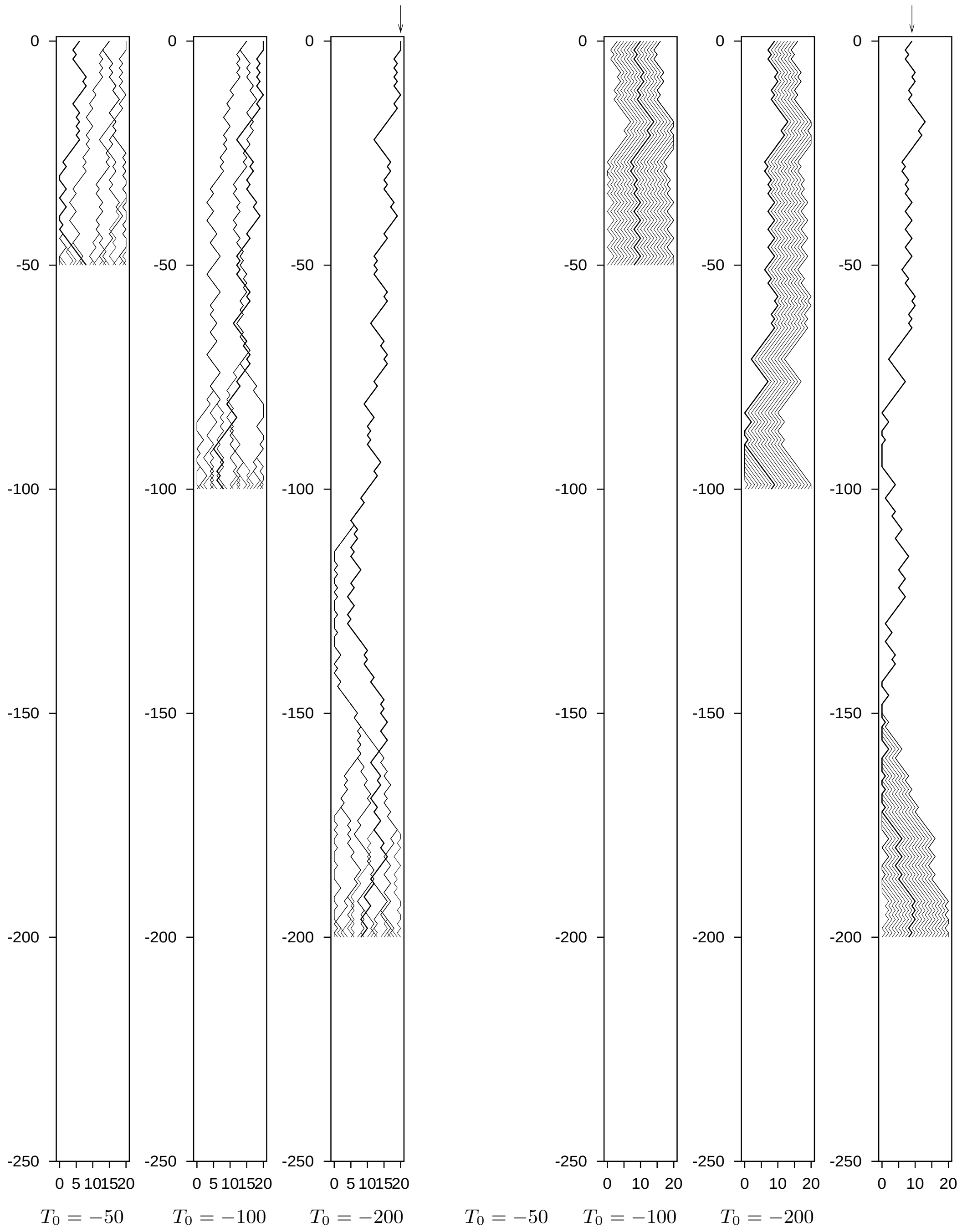


Figure 33.2.
'Coupling from the past', the second idea behind the exact sampling method.

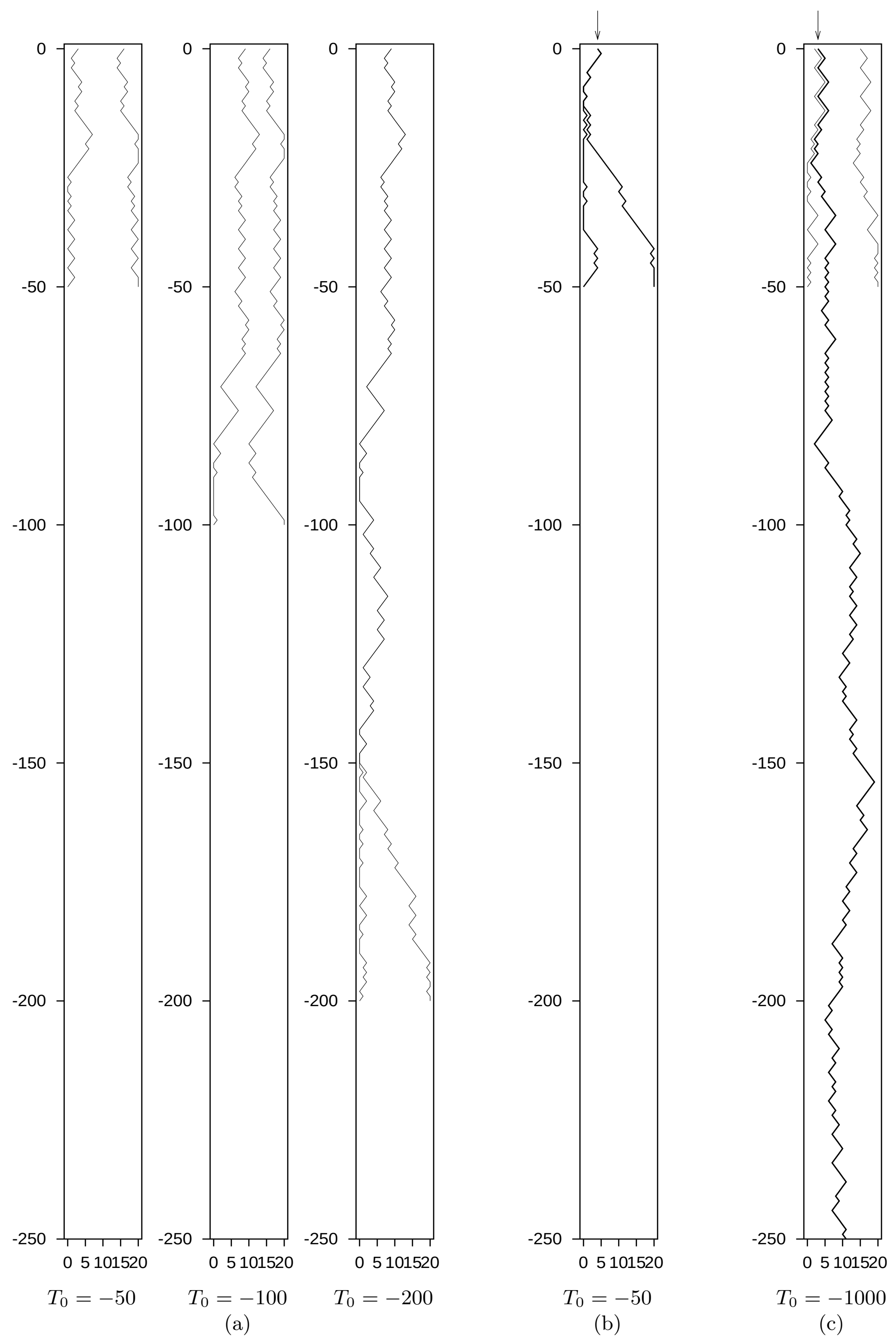


Figure 33.3. (a) Ordering of states, the third idea behind the exact sampling method. The trajectories shown here are the left-most and right-most trajectories of figure 33.2(b). In order to establish what the state at time zero is, we only need to run simulations from $T_0 = -50$, $T_0 = -100$, and $T_0 = -200$, after which point coalescence occurs.
(b,c) Two exact samples from the target density, generated by this method, and different random number seeds. The initial times required were $T_0 = -50$ and $T_0 = -1000$, respectively.

trajectories starting from the leftmost and rightmost states, we will know that coalescence of *all* trajectories has occurred when *those two* trajectories coalesce. Figure 33.3(b) illustrates this idea by showing only the left-most and right-most trajectories of figure 33.2(b). Figure 33.3(c,d) shows two more exact samples from the same equilibrium distribution generated by running the 'coupling from the past' method starting from the two end-states alone. In (c), two runs coalesced starting from $T_0 = -50$; in (d), it was necessary to try times up to $T_0 = -1000$ to achieve coalescence.

### *Exact sampling from interesting distributions*

In the toy problem we studied, the states could be put in a one-dimensional order such that no two trajectories crossed. The states of many interesting state spaces can also be put into a *partial order* and coupled Markov chains can be found that respect this partial order. For such systems, we can show that coalescence has occurred merely by verifying that coalescence has occurred for all the histories whose initial states were 'maximal' and 'minimal' states of the state space.

As an example, consider the Gibbs sampling method (or 'heat bath algorithm', or 'Glauber dynamics') applied to a ferromagnetic Ising spin system, with the partial ordering of states being defined thus: state $\mathbf{x}$ is 'greater than or equal to' state $\mathbf{y}$ if $x_i \geq y_i$ for all spins $i$. The maximal and minimal states are the the all-up and all-down states. Propp and Wilson (1996) show that exact samples can be generated for this system, although the time to find exact samples is large if the Ising model is below its critical temperature, since the Gibbs sampling method itself is slowly-mixing under these conditions. Propp and Wilson have improved on this method for the Ising model by using a Markov chain called the single-bond heat bath algorithm to sample from a related model called the random cluster model; they show that exact samples from the random cluster model can be obtained rapidly and can be converted into exact samples from the Ising model. Their ground-breaking paper includes an exact sample from a 16-million-spin Ising model at its critical temperature. A sample for a smaller Ising model is shown in figure 33.4.

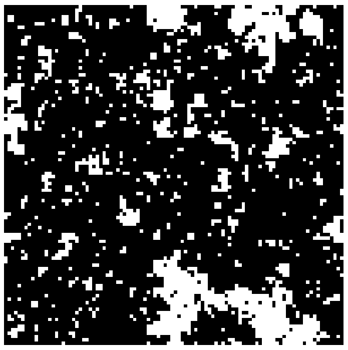


Figure 33.4. An exact sample from the Ising model at its critical temperature, produced by D. B. Wilson.

### *A generalization of the exact sampling method for 'non-attractive' distributions*

The method of Propp and Wilson for the Ising model, sketched above, can only be applied to probability distributions that are, as they call it, 'attractive'. Rather than define this term, let's say what it means, for practical purposes: the method can be applied to spin systems in which all the couplings are positive (e.g., the ferromagnet), and to a few special spin systems with negative couplings (e.g., as we already observed in chapter 32, the rectangular ferromagnet and antiferromagnet are equivalent); but it cannot be applied to general spin systems in which some couplings are negative, because in such systems the trajectories followed by the all-up and all-down states are not guaranteed to be upper and lower bounds for the set of all trajectories. Fortunately, however, we do not need to be so strict. Radford Neal has pointed out that it is possible to re-express the Propp and Wilson algorithm in a way that generalizes to the case of spin systems with negative couplings. The idea is still that we will keep track of an 'upper bound' and 'lower bound' on the set of all trajectories, and detect when these two bounds are equal, so as to find exact samples. But the upper and lower bounds will not themselves be actual trajectories, and the bounds will not necessarily be tight bounds. This is called the *summary state* version of the exact sampling method.

Instead of simulating two trajectories, each of which moves in a state space $\{-1,+1\}^N$, we simulate a single trajectory-envelope in an augmented state space $\{-1,+1,?\}^N$, where the symbol ? denotes 'either $-1$ or $+1$'. We call the state of this augmented system the 'summary state'. An example summary state of a six-spin system is ++-?+?. This summary state is shorthand for the set of states

++-+++, ++-++-, ++--++, ++--+- .

The update rule at each step of the Markov chain takes a single spin, enumerates all possible states of the neighbouring spins that are compatible with the current summary state, and, for each of these local scenarios, computes the new value (+ or -) of the spin using Gibbs sampling. If all these new values agree, then the new value of the updated spin in the summary state is set to the unanimous value (+ or -). Otherwise, the new value of the spin in the summary state is (?). The initial condition, at time $T_0$, is given by setting all the spins in the summary state to '?'.

In the case of a spin system with positive couplings, this summary state simulation will be identical to the simulation of the uppermost state and lowermost states, in the style of Propp and Wilson, with coalescence occuring when all the ? symbols have disappeared. The summary state method can be applied to general spin systems with any couplings. The only shortcoming of this method is that the envelope may describe an unnecessarily large set of states, so there is no guarantee that the summary state algorithm will converge; the time for coalescence to be *detected* may be considerably larger than the actual time taken for the underlying Markov chain to coalesce.

The summary state scheme has been applied to exact sampling in belief networks by Harvey and Neal (2000), and to the triangular antiferromagnetic Ising model by Childs *et al.* (2001).

For further reading, impressive pictures of exact samples from other distributions, and generalizations of the exact sampling method, browse the perfectly-random sampling website.[1]

## 33.2 Exercises

Solutions on p.444

Exercise 33.1:$^{B2}$ 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.

Exercise 33.2:$^{B2}$ Imagine that Fred ignores the requirement that the random bits used at some time $t$, in every run from increasingly distant times $T_0$, must be identical, and makes a coupled-Markov-chain simulator that uses fresh random numbers every time $T_0$ is changed. Describe what happens

[1] http://dimacs.rutgers.edu/∼dbwilson/exact/

if Fred applies his method to the Markov chain that is intended to sample from the uniform distribution over the states 0, 1, and 2, using the Metropolis method, driven by a random bit source as in figure 33.1(b).

## 33.3 Other uses for coupling

The idea of coupling together Markov chains by having them share a random number generator has other applications beyond exact sampling. Pinto and Neal (2001) have shown that the accuracy of estimates obtained from a Markov chain Monte Carlo simulation (the second problem discussed in section 30.1, p.380), using the estimator

$$\hat{\Phi}_P \equiv \frac{1}{T}\sum_t \phi(\mathbf{x}^{(t)}), \tag{33.1}$$

can be improved by coupling the chain of interest, which converges to $P$, to a second chain, which generates samples from a second, simpler distribution, $Q$. The coupling must be set up in such a way that the states of the two chains are strongly correlated. The idea is that we first estimate the expecations of a function of interest, $\phi$, under $P$ and under $Q$ in the normal way (33.1) and compare the estimate under $Q$, $\hat{\Phi}_Q$, with the true value of the expectation under $Q$, $\Phi_Q$ which we assume can be evaluated exactly. If $\hat{\Phi}_Q$ is an overestimate then it is likely that $\hat{\Phi}_P$ will be an overestimate too. The difference $(\hat{\Phi}_Q - \Phi_Q)$ can thus be used to correct $\hat{\Phi}_P$. For details of the correction method, see Pinto and Neal's paper.

### Solutions

Solution to exercise 33.1 (p.443): It is perhaps surprising that there is no direct relationship between the equilibration time and the time to coalescence. 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 33.1, except that no coalescence ever occurs. Thus, while the equilibration times of methods (a) and (b) are both one, the coalescence times are respectively one and infinity.

# 34

# 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.

## 34.1 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's inequality, which we repeat here.

Gibbs's inequality first appeared in equation (1.24); see also exercise 2.22 (p.41).

**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

$$D_{\rm KL}(Q||P) = \sum_x Q(x) \log \frac{Q(x)}{P(x)}. \tag{34.1}$$

The relative entropy satisfies $D_{\rm KL}(Q||P) \geq 0$ (Gibbs' inequality) with equality only if $Q{=}P$. In general $D_{\rm KL}(Q||P) \neq D_{\rm KL}(P||Q)$. In this chapter we will replace the log by ln, and measure the divergence in nats.

### *Probability distributions in statistical physics*

In statistical physics one often encounters probability distributions of the form

$$P(\mathbf{x}|\beta, \mathbf{J}) = \frac{1}{Z(\beta, \mathbf{J})} \exp\left[-\beta E(\mathbf{x}; \mathbf{J})\right], \tag{34.2}$$

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

$$E(\mathbf{x}; \mathbf{J}) = -\frac{1}{2} \sum_{m,n} J_{mn} x_m x_n - \sum_n h_n x_n. \tag{34.3}$$

The partition function is

$$Z(\beta, \mathbf{J}) \equiv \sum_{\mathbf{x}} \exp\left[-\beta E(\mathbf{x}; \mathbf{J})\right] \tag{34.4}$$

The probability distribution of equation (34.2) 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 30, 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 $\theta$. 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*

$$\beta \tilde{F}(\theta) = \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \ln \frac{Q(\mathbf{x}; \theta)}{\exp\left[-\beta E(\mathbf{x}; \mathbf{J})\right]}. \tag{34.5}$$

This expression can be manipulated into a couple of interesting forms: first,

$$\begin{aligned} \beta \tilde{F}(\theta) &= \beta \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) E(\mathbf{x}; \mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \ln \frac{1}{Q(\mathbf{x}; \theta)} & (34.6) \\ &\equiv \beta \left\langle E(\mathbf{x}; \mathbf{J}) \right\rangle_Q - S_Q, & (34.7) \end{aligned}$$

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 $k_{\mathrm{B}}$ 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:

$$\begin{aligned} \beta \tilde{F}(\theta) &= \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \ln \frac{Q(\mathbf{x}; \theta)}{P(\mathbf{x} | \beta, \mathbf{J})} - \ln Z(\beta, \mathbf{J}) & (34.8) \\ &= D_{\mathrm{KL}}(Q || P) + \beta F, & (34.9) \end{aligned}$$

where $F$ is the true free energy, defined by

$$\beta F \equiv -\ln Z(\beta, \mathbf{J}), \tag{34.10}$$

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's 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 $\theta$ 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 $\beta F$. Equivalently, $\tilde{Z} \equiv e^{-\beta \tilde{F}(\theta)}$ is a lower bound for $Z$.

*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

$$Z = \sum_{\mathbf{x}} \exp\left(-\beta E(\mathbf{x};\mathbf{J})\right), \tag{34.11}$$

the energy

$$\langle E \rangle_P = \frac{1}{Z} \sum_{\mathbf{x}} E(\mathbf{x};\mathbf{J}) \exp\left(-\beta E(\mathbf{x};\mathbf{J})\right), \tag{34.12}$$

and the entropy

$$S \equiv \sum_{\mathbf{x}} P(\mathbf{x}|\beta,\mathbf{J}) \ln \frac{1}{P(\mathbf{x}|\beta,\mathbf{J})} \tag{34.13}$$

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 $\mathbf{x}$ (34.5), 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.

## 34.2 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 (34.3), which we can approximate with a *separable* approximating distribution,

$$Q(\mathbf{x};\mathbf{a}) = \frac{1}{Z_Q} \exp\left(\sum_n a_n x_n\right). \tag{34.14}$$

The variational parameters $\theta$ of the variational free energy (34.5) are the components of the vector $\mathbf{a}$. To evaluate the variational free energy we need the entropy of this distribution,

$$S_Q = \sum_{\mathbf{x}} Q(\mathbf{x};\theta) \ln \frac{1}{Q(\mathbf{x};\theta)} \tag{34.15}$$

and the mean of the energy,

$$\langle E(\mathbf{x};\mathbf{J}) \rangle_Q = \sum_{\mathbf{x}} Q(\mathbf{x};\theta) E(\mathbf{x};\mathbf{J}). \tag{34.16}$$

The entropy of the separable approximating distribution is simply the sum of the entropies of the individual spins (exercise 9.1, p.168),

$$S_Q = \sum_n H_2^{(e)}(q_n), \tag{34.17}$$

where $q_n$ is the probability that spin $n$ is $+1$,

$$q_n = \frac{e^{a_n}}{e^{a_n} + e^{-a_n}} = \frac{1}{1+\exp(-2a_n)}, \tag{34.18}$$

and

$$H_2^{(e)}(q) = q \ln \frac{1}{q} + (1-q) \ln \frac{1}{(1-q)}. \tag{34.19}$$

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

$$\bar{x}_n = \frac{e^{a_n} - e^{-a_n}}{e^{a_n} + e^{-a_n}} = \tanh(a_n) = 2q_n - 1, \tag{34.20}$$

we obtain

$$\begin{aligned} \langle E(\mathbf{x}; \mathbf{J}) \rangle_Q &= \sum_{\mathbf{x}} Q(\mathbf{x}; \theta) \left[ -\frac{1}{2} \sum_{m,n} J_{mn} x_m x_n - \sum_n h_n x_n \right] & (34.21) \\ &= -\frac{1}{2} \sum_{m,n} J_{mn} \bar{x}_m \bar{x}_n - \sum_n h_n \bar{x}_n. & (34.22) \end{aligned}$$

So the variational free energy is given by

$$\beta \tilde{F}(\mathbf{a}) = \beta \langle E(\mathbf{x}; \mathbf{J}) \rangle_Q - S_Q = \beta \left( -\frac{1}{2} \sum_{m,n} J_{mn} \bar{x}_m \bar{x}_n - \sum_n h_n \bar{x}_n \right) - \sum_n H_2^{(e)}(q_n). \tag{34.23}$$

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

$$\frac{\partial}{\partial q} H_2^e(q) = \ln \frac{1-q}{q} = -2a. \tag{34.24}$$

So we obtain

$$\begin{aligned} \frac{\partial}{\partial a_m} \beta \tilde{F}(\mathbf{a}) &= \beta \left[ -\sum_n J_{mn} \bar{x}_n - h_m \right] \left( 2 \frac{\partial q_m}{\partial a_m} \right) - \ln \left( \frac{1 - q_m}{q_m} \right) \left( \frac{\partial q_m}{\partial a_m} \right) \\ &= 2 \left( \frac{\partial q_m}{\partial a_m} \right) \left[ -\beta \left( \sum_n J_{mn} \bar{x}_n + h_m \right) + a_m \right]. & (34.25) \end{aligned}$$

This derivative is equal to zero when

$$a_m = \beta \left( \sum_n J_{mn} \bar{x}_n + h_m \right). \tag{34.26}$$

So $\tilde{F}(\mathbf{a})$ is extremized at any point that satisfies equation (34.26) and

$$\bar{x}_n = \tanh(a_n) \tag{34.27}$$

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 (34.26) and (34.27). 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 (34.26), one at a time. This *asynchronous updating of the parameters* is guaranteed to decrease $\beta \tilde{F}(\mathbf{a})$.

Equations (34.26) and (34.27) 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 (34.27) describes the mean response of spin $n$, and equation (34.26) 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.

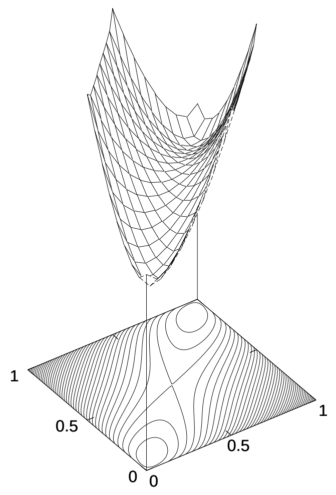


Figure 34.1. The variational free energy of the two-spin system whose energy is $E(\mathbf{x}) = -x_1 x_2$, as a function of the two variational parameters $q_1$ and $q_2$. The inverse-temperature is $\beta = 1.44$. The function plotted is

$$\beta \tilde{F} = -\beta \bar{x}_1 \bar{x}_2 - H_2^{(e)}(q_1) - H_2^{(e)}(q_2),$$

where $\bar{x}_n = 2q_n - 1$. Notice that for fixed $q_2$ the function is convex with respect to $q_1$, and for fixed $q_1$ it is convex with respect to $q_2$.

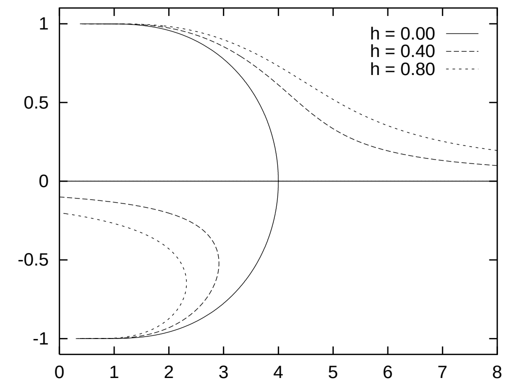


Figure 34.2. Solutions of the variational free energy extremization problem for the Ising model. Horizontal axis: temperature $T = 1/\beta$. Vertical axis: magnetization $\bar{x}$. The critical temperature found by mean field theory is $T_c^{\rm mft} = 4$.

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 $\theta$ 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.

## 34.3 Example: mean field theory for the ferromagnetic Ising model

In the simple Ising model studied in chapter 32, 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

$$Q(\mathbf{x};a) = \frac{1}{Z_Q}\exp\left(\sum_n a x_n\right) \tag{34.28}$$

in which all spins are independent and have the same probability

$$q_n = \frac{1}{1+\exp(-2a)} \tag{34.29}$$

of being up. The mean magnetization is

$$\bar{x} = \tanh(a) \tag{34.30}$$

and the equation (34.26) which defines the minimum of the variational free energy becomes

$$a = \beta\left(CJ\bar{x} + h\right), \tag{34.31}$$

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 (34.30) and (34.31) for $\bar{x}$ numerically — in fact, it is easiest to vary $\bar{x}$ and solve for $\beta$ — and obtain graphs of the free energy minima and maxima as a function of temperature as shown in figure 34.2. 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 $T_c^{\text{mft}}$. [A pitchfork bifurcation is a transition like the one shown by the solid lines in figure 34.2, from a system with one minimum as a function of $a$ (on the right) to a system (on the left) with two minima and one maximum; the maximum is the middle one of the three lines. The solid lines look like a pitchfork.] 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 34.2. 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 $T_c^{\text{mft}}$, 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

$$\beta\tilde{F} = \beta\left(-\frac{C}{2}J\bar{x}^2 - h\bar{x}\right) - H_2^{(e)}\left(\frac{\bar{x}+1}{2}\right). \qquad (34.32)$$

Exercise 34.1:$^{A2}$ 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 34.2 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 $T_c^{\text{mft}}$ 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 $\beta$ and find the entropy, heat capacity and fluctuations of the approximating distribution and compare them with those of a real $8 \times 8$ fragment using the matrix method of chapter 32. As shown in figure 34.3, 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.

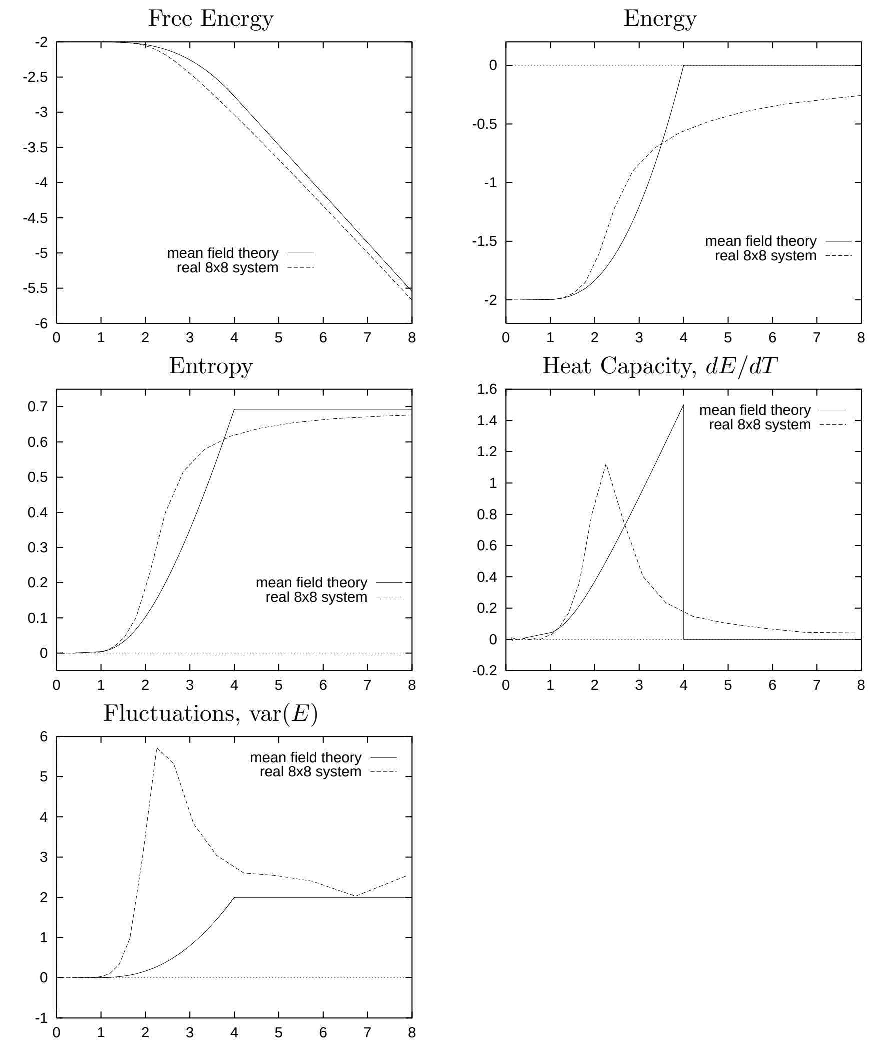


Figure 34.3. Comparison of approximating distribution's properties with those of a real $8 \times 8$ fragment. Notice that the variational free energy of the approximating distribution is indeed an upper bound on the free energy of the real system. All quantities are shown 'per spin'.

## 34.4 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 $\mathcal{H}$, $P(\mathbf{w}|D,\mathcal{H})$.

$$P(\mathbf{w}|D,\mathcal{H}) = \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})}{P(D|\mathcal{H})}. \tag{34.33}$$

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 $\theta$) 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

$$\tilde{F}(\theta) = \int d^k\mathbf{w}\, Q(\mathbf{w};\theta) \ln \frac{Q(\mathbf{w};\theta)}{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\mathcal{H})}. \tag{34.34}$$

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 $-\ln P(D|\mathcal{H})$ and the relative entropy between $Q(\mathbf{w};\theta)$ and $P(\mathbf{w}|D,\mathcal{H})$. $\tilde{F}(\theta)$ is bounded below by $-\ln P(D|\mathcal{H})$ 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 1995). 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.

## 34.5 The case of an unknown Gaussian: approximating the posterior distribution of $\mu$ and $\sigma$

We will fit an approximating ensemble $Q(\mu,\sigma)$ to the posterior distribution that we studied in chapter 26,

$$P(\mu,\sigma|\{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N|\mu,\sigma)P(\mu,\sigma)}{P(\{x_n\}_{n=1}^N)} \tag{34.35}$$

$$= \frac{\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N(\mu-\bar{x})^2+S}{2\sigma^2}\right) \frac{1}{\sigma_\mu}\frac{1}{\sigma}}{P(\{x_n\}_{n=1}^N)}. \tag{34.36}$$

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,

$$\tilde{F}(Q) = \int d\mu \, d\sigma \, Q_\mu(\mu) Q_\sigma(\sigma) \ln \frac{Q_\mu(\mu) Q_\sigma(\sigma)}{P(D|\mu,\sigma) P(\mu,\sigma)}. \tag{34.37}$$

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:

$$\begin{aligned} \tilde{F} &= -\int d\mu \, Q_\mu(\mu) \left[ \int d\sigma \, Q_\sigma(\sigma) \ln P(D|\mu,\sigma) + \ln[P(\mu)/Q_\mu(\mu)] \right] + \kappa \quad (34.38) \\ &= \int d\mu \, Q_\mu(\mu) \left[ \int d\sigma \, Q_\sigma(\sigma) N\beta \frac{1}{2}(\mu - \bar{x})^2 + \ln Q_\mu(\mu) \right] + \kappa', \quad (34.39) \end{aligned}$$

where $\beta \equiv 1/\sigma^2$ and $\kappa$ denotes constants that do not depend on $Q_\mu(\mu)$. The dependence on $Q_\sigma$ thus collapses down to a simple dependence on the mean

$$\bar{\beta} \equiv \int d\sigma \, Q_\sigma(\sigma) 1/\sigma^2. \tag{34.40}$$

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 $\int Q \ln(Q/P)$ is minimized by setting $Q = P$, we can immediately write down the distribution $Q_\mu^{\text{opt}}(\mu)$ that minimizes $\tilde{F}$ for fixed $Q_\sigma$:

$$Q_\mu^{\text{opt}}(\mu) = P(\mu|D, \bar{\beta}, \mathcal{H}) = \text{Normal}(\mu; \bar{x}, \sigma^2_{\mu|D}). \tag{34.41}$$

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):

$$\begin{aligned} \tilde{F} &= -\int d\sigma \, Q_\sigma(\sigma) \left[ \int d\mu \, Q_\mu(\mu) \ln P(D|\mu,\sigma) + \ln[P(\sigma)/Q_\sigma(\sigma)] \right] \quad (34.42) \\ &= \int d\sigma \, Q_\sigma(\sigma) \left[ (N\sigma^2_{\mu|D} + S)\beta/2 - \left(\tfrac{N}{2} - 1\right) \ln \beta + \ln Q_\sigma(\sigma) \right] \quad (34.43) \end{aligned}$$

where the integral over $\mu$ is performed assuming $Q_\mu(\mu) = Q_\mu^{\text{opt}}(\mu)$. Here, the $\beta$-dependent expression in the brackets can be recognized as the log of a gamma distribution over $\beta$ – see equation (26.2) – giving as the distribution that minimizes $\tilde{F}$ for fixed $Q_\mu$:

$$Q_\sigma^{\text{opt}}(\beta) = \Gamma(\beta; b', c'), \tag{34.44}$$

with

$$\frac{1}{b'} = \frac{1}{2}(N\sigma^2_{\mu|D} + S) \quad \text{and} \quad c' = \frac{N}{2}. \tag{34.45}$$

In figure 34.4, these two update rules (34.41,34.44) are applied alternately, starting from an arbitrary initial condition. The algorithm converges to the optimal approximating ensemble in a few iterations.

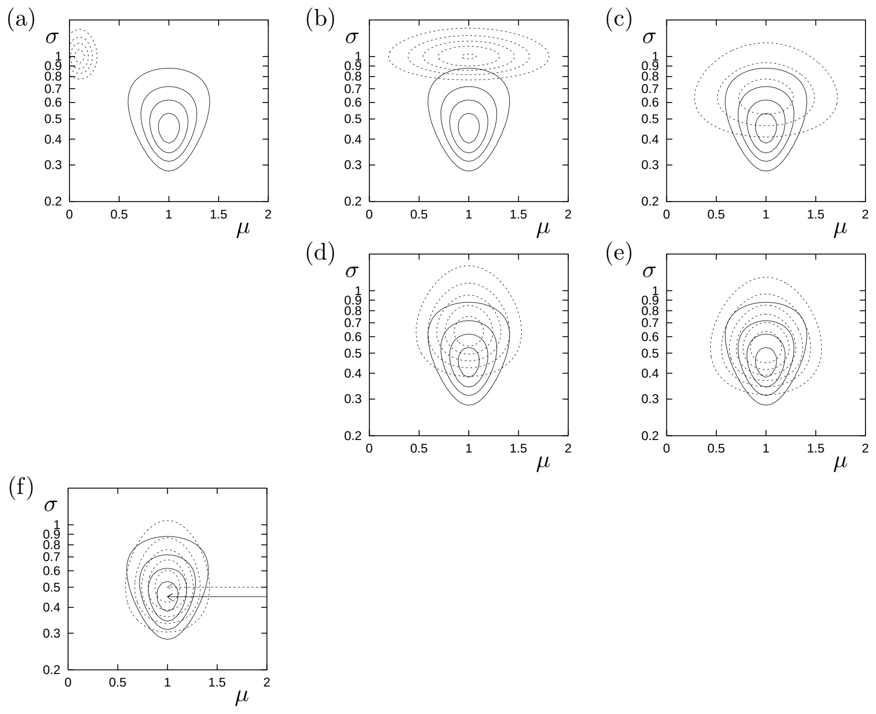


Figure 34.4. Optimization of an approximating distribution. The posterior distribution $P(\mu, \sigma | \{x_n\})$, which is the same as that in figure 26.1, is shown by solid contours. (a) Initial condition. The approximating distribution $Q(\mu, \sigma)$ (dotted contours) is an arbitrary separable distribution. (b) $Q_\mu$ has been updated, using equation (34.41). (c) $Q_\sigma$ has been updated, using equation (34.44). (d) $Q_\mu$ updated again. (e) $Q_\sigma$ 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 (34.41) and (34.44) define the optimum implicitly. We must simultaneously have $\sigma^2_{\mu|D} = 1/(N\bar{\beta})$, and $\bar{\beta} = b'c'$. The solution is:

$$1/\bar{\beta} = S/(N-1). \tag{34.46}$$

This is similar to the true posterior distribution of $\sigma$, 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 $\beta$ 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 $\beta$ 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 $\mu$ is a Gaussian with mean $\bar{x}$ and standard deviation $\sigma_{N-1}/\sqrt{N}$.

## 34.6 Variational methods other than free energy minimization

Given a complicated distribution $P(\mathbf{x})$, there are other strategies for approximation in addition to those based on minimizing the relative entropy between an approximating distribution $Q$ and $P$. One approach pioneered by Jaakkola and Jordan is to create adjustable upper and lower bounds $Q^U$ and $Q^L$ to $P$.

The lower bound can be adjusted to maximize

$$\sum_{\mathbf{x}} Q^L(\mathbf{x}), \tag{34.47}$$

and the upper bound can be adjusted to minimize

$$\sum_{\mathbf{x}} Q^U(\mathbf{x}). \tag{34.48}$$

Further reading on such methods can be found in the references (Jaakkola and Jordan 1996a; Jaakkola and Jordan 2000; Jaakkola and Jordan 1996b; Gibbs and MacKay 2000).

## 34.7 Interlude

One of my students asked

> How do you ever come up with a useful approximating distribution, given that the true distribution is so complex you can't compute it directly?

Let's answer this question in the context of Bayesian data modelling. Let the 'true' distribution of interest be the posterior probability distribution over a set of parameters $\mathbf{x}$, $P(\mathbf{x}|D)$. A standard data modelling practice is to find a single, 'best-fit' setting of the parameters, $\mathbf{x}^*$, for example, by finding the maximum of the likelihood function $P(D|\mathbf{x})$, or of the posterior distribution. One interpretation of this standard practice is that the full description of our knowledge about $\mathbf{x}$, $P(\mathbf{x}|D)$, is being approximated by a delta-function, a probability distribution concentrated on $\mathbf{x}^*$. From this perspective, *any* approximating distribution $Q(\mathbf{x};\theta)$, no matter how crummy it is, has to be an improvement on the spike produced by the standard method! So even if we use a simple Gaussian approximation, we are doing well.

Another motivation for studying variational methods is that they will pop up unexpectedly when we are studying unsupervised neural networks.

We now study an application of the variational approach to a realistic example – data clustering.

## 34.8 K-means clustering and the Expectation-Maximization algorithm as a variational method

In chapter 22, we introduced the soft K-means clustering algorithm, version 1. In chapter 24, we introduced versions 2 and 3 of this algorithm, and motivated the algorithm as a maximum likelihood algorithm.

> K-means clustering is an example of an 'Expectation-Maximization' ('E-M') algorithm, with the two steps, which we called 'assignment' and 'update', being known as the 'E-step' and the 'M-step' respectively.

We now give a more general view of K-means clustering, due to Neal and Hinton (1993), in which the algorithm is shown to optimize a variational objective function.

*The probability of everything*

Let the parameters of the mixture model – the means, standard deviations, and weights – be denoted by $\theta$. For each data point, there is a missing variable (also known as a latent variable), the class label $k_n$ for that point. The probability of everything, given our assumed model $\mathcal{H}$, is

$$P(\{\mathbf{x}^{(n)}, k_n\}_{n=1}^N, \theta|\mathcal{H}) = P(\theta|\mathcal{H}) \prod_{n=1}^{N} \left[ P(\mathbf{x}^{(n)}|k_n, \theta) P(k_n|\theta) \right]. \tag{34.49}$$

The posterior probability of everything given the data is proportional to the probability of everything:

$$P(\{k_n\}_{n=1}^N, \theta|\{\mathbf{x}^{(n)}\}_{n=1}^N, \mathcal{H}) = \frac{P(\{\mathbf{x}^{(n)}, k_n\}_{n=1}^N, \theta|\mathcal{H})}{P(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathcal{H})}. \tag{34.50}$$

We now approximate this posterior distribution by a separable distribution

$$Q_k(\{k_n\}_{n=1}^N) Q_\theta(\theta), \tag{34.51}$$

and define a variational free energy in the usual way:

$$\tilde{F}(Q_k, Q_\theta) = \sum_{\{k_n\}} \int \mathrm{d}^D\theta Q_k(\{k_n\}_{n=1}^N) Q_\theta(\theta) \log \frac{Q_k(\{k_n\}_{n=1}^N) Q_\theta(\theta)}{P(\{\mathbf{x}^{(n)}, k_n\}_{n=1}^N, \theta|\mathcal{H})}. \tag{34.52}$$

The variational free energy is bounded below by the evidence, $\log P(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathcal{H})$. We can now make an iterative algorithm with an 'assignment' step and an 'update' step. In the assignment step, $Q_k(\{k_n\}_{n=1}^N)$ is adjusted to reduce $\tilde{F}$, for fixed $Q_\theta$; in the update step, $Q_\theta$ is adjusted to reduce $\tilde{F}$, for fixed $Q_k$.

To obtain exactly the soft K-means algorithm, we must impose a further constraint on our approximating distribution: $Q_\theta$ is constrained to take the form of a delta-function centred on a point estimate of $\theta$, $\theta = \theta^*$:

$$Q_\theta(\theta) = \delta(\theta - \theta^*). \tag{34.53}$$

Unfortunately, this distribution gives an infinitely large value to the integral $\int \mathrm{d}^D\theta\, Q_\theta(\theta) \log Q_\theta(\theta)$, so we'd better leave that term out of $\tilde{F}$ and hope it doesn't come back to haunt us. Using a delta-function $Q_\theta$ is not a good idea if our aim is to minimize $\tilde{F}$! Moving on, our aim is to derive the soft K-means algorithm.

Exercise 34.2:$^{B2}$ Show that, given $Q_\theta(\theta) = \delta(\theta - \theta^*)$, the optimal $Q_k$, in the sense of minimizing $\tilde{F}$, is a separable distribution in which the probability the $k_n = k$ is given by the responsibility $r_k^{(n)}$.

Exercise 34.3:$^{B4}$ Show that, given a separable $Q_k$ as described above, the optimal $\theta^*$, in the sense of minimizing $\tilde{F}$, is obtained by the update step of the soft K-means algorithm. (Assume a uniform prior on $\theta$.)

Exercise 34.4:$^{D4}$ We can instantly improve on the infinitely large value of $\tilde{F}$ achieved by soft K-means clustering by allowing $Q_\theta$ to be a more general distribution than a delta-function. Derive an update step in which $Q_\theta$ is allowed to be a separable distribution, a product of $Q_\mu(\{\mu\})$, $Q_\sigma(\{\sigma\})$, and

$Q_\pi(\pi)$. Discuss whether this generalized algorithm still suffers from soft K-means's 'kaboom' problem, where the algorithm glues an ever-shrinking Gaussian to one data point.

This approach to data-modelling is known as variational Bayes, or ensemble learning. Sadly, while it sounds like a promising generalization of the algorithm to allow $Q_\theta$ to be a non-delta-function, and the 'kaboom' problem goes away, it has been our experience that other artefacts arise in this approximate inference method. For further reading, see (MacKay 1997; MacKay 2001).

## 34.9 Summary

Key point: variational methods (or rather, the VFE minimization approach) is parameterization-independent; gets round the problem of parameterization-dependence of MAP methods and Laplace's method.

## 34.10 Exercises

Exercise 34.5:$^{A2}$ 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 $\sigma_1^2$ and $\sigma_2^2$. What is the optimal variance if this distribution is approximated by a *spherical* Gaussian with variance $\sigma_Q^2$, optimized by variational free energy minimization? If we instead optimized the objective function

$$G = \int d\mathbf{x}\, P(\mathbf{x}) \ln_2 \frac{P(\mathbf{x})}{Q(\mathbf{x}; \sigma^2)}, \tag{34.54}$$

what would be the optimal value of $\sigma^2$? 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 34.6:$^{A2}$ 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 34's exercises

Solution to exercise 34.5 (p.457): This problem rests on knowing the relative entropy between two one-dimensional Gaussian distributions:

$$\int dx\, \mathrm{Normal}(x;0,\sigma_Q) \log \frac{\mathrm{Normal}(x;0,\sigma_Q)}{\mathrm{Normal}(x;0,\sigma_P)}$$

$$= \int dx\, \mathrm{Normal}(x;0,\sigma_Q) \left[ \log \frac{\sigma_P}{\sigma_Q} - \frac{1}{2} x^2 \left( \frac{1}{\sigma_Q^2} - \frac{1}{\sigma_P^2} \right) \right] \tag{34.55}$$

$$= \frac{1}{2} \left( \log \frac{\sigma_P^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_P^2} \right). \tag{34.56}$$

So, if we approximate $P$, whose variances are $\sigma_1^2$ and $\sigma_2^2$, by $Q$, whose variances are both $\sigma_Q^2$, we find

$$F(\sigma_Q^2) = \frac{1}{2} \left( \log \frac{\sigma_1^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_1^2} + \log \frac{\sigma_2^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_2^2} \right); \tag{34.57}$$

differentiating,

$$\frac{d}{d \log(\sigma_Q^2)} F = \frac{1}{2} \left[ -2 + \left( \frac{\sigma_Q^2}{\sigma_1^2} + \frac{\sigma_Q^2}{\sigma_2^2} \right) \right], \tag{34.58}$$

which is zero when

$$\frac{1}{\sigma_Q^2} = \frac{1}{2} \left( \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right). \tag{34.59}$$

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 $\sigma_2$, pretty much *independent* of the value of the larger standard deviation $\sigma_1$. *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:

$$G(\sigma_Q^2) = \frac{1}{2} \left( \log \sigma_Q^2 + \frac{\sigma_1^2}{\sigma_Q^2} + \log \sigma_Q^2 + \frac{\sigma_2^2}{\sigma_Q^2} \right) + \text{constant}, \tag{34.60}$$

where the constant depends on $\sigma_1$ and $\sigma_2$ only. Differentiating,

$$\frac{d}{d \log \sigma_Q^2} G = \frac{1}{2} \left( 2 - \frac{\sigma_1^2}{\sigma_Q^2} + \frac{\sigma_2^2}{\sigma_Q^2} \right), \tag{34.61}$$

which is zero when

$$\sigma_Q^2 = \frac{1}{2} \left( \sigma_1^2 + \sigma_2^2 \right). \tag{34.62}$$

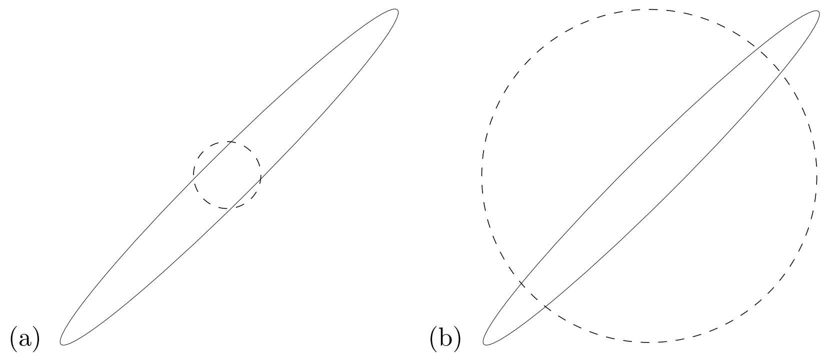


Figure 34.5. Two separable Gaussian approximations (dotted lines) to a bivariate Gaussian distribution (solid line). (a) The approximation that minimizes the variational free energy. (b) The approximation that minimizes the objective function $G$. In each figure, the lines show the contours at which $\mathbf{x}\mathbf{A}\mathbf{x} = 1$, 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 $\sigma_1$, independent of the value of $\sigma_2$.

The two approximations are shown to scale in figure 34.5.

Solution to exercise 34.6 (p.457): 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.

# 35

# *Independent Component Analysis and Latent Variable Modelling* *

## 35.1 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 $\mathbf{x}$.

### *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

$$\mathbf{x} = \mathbf{G}\mathbf{s}. \tag{35.1}$$

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

$$\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}, \tag{35.2}$$

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 $\sigma_i^2$.

### *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 35.1.

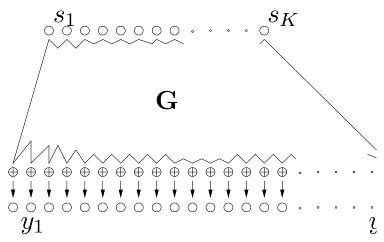


Figure 35.1. Error-correcting codes as latent variable models.

# 36

# Further exercises on inference

Exercise 36.1:$^{A2}$ 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 |
| | 1 | 1/8 | 1/8 | 0 | 0 |
| $y$ | 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.

Exercise 36.2:$^{B2}$ 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.

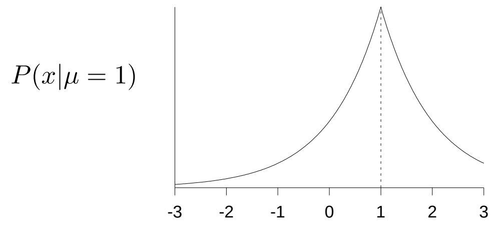
A 'parsing' of a string is a grammatical explanation interpretation of the string. For example, 'Punch bores' could be parsed as 'Punch (noun) bores (verb)', or 'Punch (imperative verb) bores (plural noun)'.

Solutions on p.464

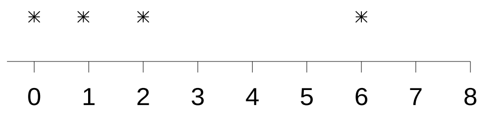
Exercise 36.3:$^{B2}$ In an experiment, the measured quantities $\{x_n\}$ come independently from a bi-exponential distribution with mean $\mu$,

$$P(x|\mu) = \frac{1}{Z} \exp\left(-|x-\mu|\right),$$

where $Z$ is the normalizing constant, $Z = 2$. The mean $\mu$ is not known. An example of this distribution, with $\mu = 1$, is shown below.



Assuming the four datapoints are



$\{x_n\} = \{0, 0.9, 2, 6\}$,

what do these data tell us about $\mu$? Include detailed sketches in your answer. Give a range of plausible values of $\mu$.

Solution to exercise 36.2 (p.462): 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 (*c.f.* section 38), 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,

$$P(n|\mathcal{H}_a) = \begin{cases} \frac{1}{Z}\frac{1}{n} & n \geq 1 \\ \frac{1}{Z}\frac{1}{10} & n = 0 \end{cases}, \tag{36.1}$$

where $Z = 2.93$, and second, for terminal digits,

$$P(n|\mathcal{H}_b) = \frac{1}{10}. \tag{36.2}$$

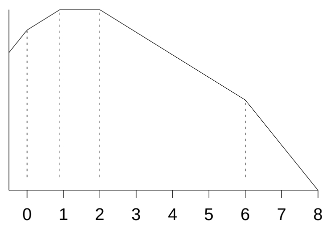
Then the probability of the given 43 digits is

$$P(\{n\}|\mathcal{H}_a) = 2.71 \times 10^{-28}. \tag{36.3}$$

$$P(\{n\}|\mathcal{H}_b) = 10^{-43}. \tag{36.4}$$

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 $\mathcal{H}_a$.



Sketch of likelihood function on a log scale.

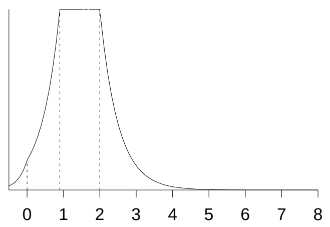
Solution to exercise 36.3 (p.463): Bayes's theorem:

$$P(\mu|\{x_n\}) = \frac{P(\mu)\prod_n P(x_n|\mu)}{P(\{x_n\})} \tag{36.5}$$

The likelihood function contains a complete summary of what the experiment tells us about $\mu$. Expression for the log likelihood,

$$L(\mu) = -\sum_n |x_n - \mu|. \tag{36.6}$$

Note gradient changes by 2 as you pass each data point. Gradients are 4, 2, 0, −2, −4.



Sketch of likelihood function on a linear scale. The exponential functions have lengthscales 1/4, 1/2, 1/2, 1/4.

The most probable values of $\mu$ are 0.9–2, and the posterior probability falls by a factor of $e^2$ once we reach −0.1 and 3, so a range of plausible 0 values for $\mu$ is $(-0.1, 3)$.

*Counting combinations*

Solutions on p.??

Exercise 36.4:$^{B2}$ 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 36.5:$^{B2}$ 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.465

Exercise 36.6:$^{C4}$ 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?

Solution to exercise 36.6 (p.465): The probablities of winning either of the first two bets is $6/11 = 0.54545$. The probability that you win the third bet is 0.4544. 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.

# 37

# *Decision theory*

Decision theory is trivial, apart from computational details.

You have a choice of various actions, $a$. The world may be in one of many states $\mathbf{x}$; which one occurs may be influenced by your action. The world's state is described by a probability distribution $P(\mathbf{x}|a)$. Finally, there is a utility function $U(\mathbf{x}, a)$ which specifies the payoff you receive when the world is in state $\mathbf{x}$ and you chose action $a$.

The task of decision theory is to select the action that maximizes the expected utility,

$$\mathcal{E}[U|a] = \int d^K\mathbf{x}\, U(\mathbf{x}, a) P(\mathbf{x}|a). \tag{37.1}$$

That's all. The computational problem is to maximize $\mathcal{E}[U|a]$ over $a$. [Pessimists may prefer to define a loss function $L$ instead of a utility function $U$ and minimize the expected loss.]

Is there anything more to be said about decision theory?

Well, in a real problem, the choice of an appropriate utility function may be quite difficult. Furthermore, when a sequence of actions is to be taken, with each action providing information about $\mathbf{x}$, we have to take into account the affect that this anticipated information may have on our subsequent actions. The resulting mixture of forward probability and inverse probability computations in a decision problem is distinctive. In a realistic problem such as playing a board game, the tree of possible cogitations and actions that must be considered becomes enormous, and 'doing the right thing' is not simple (Russell and Wefald 1991; Baum and Smith 1993; Baum and Smith 1997) because the expected utility of an action cannot be computed exactly.

Perhaps a simple example is worth exploring.

## Rational prospecting

Suppose you have the task of choosing the site for a Tanzanite mine. Your final action will be to select the site from a list of $N$ sites. The $n$th site has a net value called the return $x_n$ which is initially unknown, and will be found out exactly only after site $n$ has been chosen. [$x_n$ equals the revenue earned from selling the Tanzanite from that site, minus the costs of buying the site, paying the staff, and so forth.] At the outset, the return $x_n$ has a probability distribution $P(x_n)$, based on the information already available.

Tanzanite is a mineral found in East Africa.

Before you take your final action you have the opportunity to do some prospecting. Prospecting at the $n$th site has a cost $c_n$ and yields data $d_n$

which reduce the uncertainty about $x_n$. [We'll assume that the returns of the $N$ sites are unrelated to each other, and that prospecting at one site only yields information about that site and doesn't affect the return from that site.]

Your decision problem is:

> given the initial probability distributions $P(x_1)$, $P(x_2)$, $\ldots P(x_N)$, first, decide whether to prospect, and at which sites; then choose which site to mine.

For simplicity, let's make everything in the problem Gaussian and focus on the question of whether to prospect once or not. We'll assume our utility function is linear in $x_n$; we wish to maximize our expected return. The utility function is

$$U = x_{n_a}, \tag{37.2}$$

if no prospecting is done, where $n_a$ is the chosen 'action' site, and if prospecting is done the utility is

$$U = -c_{n_p} + x_{n_a}, \tag{37.3}$$

where $n_p$ is the site at which prospecting took place.

The notation $P(y) = \text{Normal}(y; \mu, \sigma^2)$ indicates that $y$ has Gaussian distribution with mean $\mu$ and variance $\sigma^2$.

The prior distribution of the return of site $n$ is

$$P(x_n) = \text{Normal}(x_n; \mu_n, \sigma_n^2). \tag{37.4}$$

If you prospect at site $n$, the datum $d_n$ is a noisy version of $x_n$:

$$P(d_n|x_n) = \text{Normal}(d_n; x_n, \sigma^2). \tag{37.5}$$

Exercise 37.1: Given these assumptions, show that the prior probability distribution of $d_n$ is

$$P(d_n) = \text{Normal}(d_n; \mu_n, \sigma^2 + \sigma_n^2) \tag{37.6}$$

[mnemonic: when independent variables add, variances add], and that the posterior distribution of $x_n$ given $d_n$ is

$$P(x_n|d_n) = \text{Normal}\left(x_n; \mu_n', \sigma_n^{2\prime}\right) \tag{37.7}$$

where

$$\mu_n' = \frac{d_n/\sigma^2 + \mu_n/\sigma_n^2}{1/\sigma^2 + 1/\sigma_n^2} \quad \text{and} \quad \frac{1}{\sigma_n^{2\prime}} = \frac{1}{\sigma^2} + \frac{1}{\sigma_n^2} \tag{37.8}$$

[mnemonic: when Gaussians multiply, precisions add].

To start with let's evaluate the expected utility if we do no prospecting (i.e., choose the site immediately); then we'll evaluate the expected utility if we first prospect at one site and then make our choice. From these two results we will be able to decide whether to prospect once or zero times, and, if we prospect once, at which site.

So, first we consider the expected utility without any prospecting.

Exercise 37.2: Show that the optimal action, assuming no prospecting, is to select the site with biggest mean

$$n_a = \underset{n}{\text{argmax}}\ \mu_n, \tag{37.9}$$

and the expected utility of this action is

$$\mathcal{E}[U|\text{optimal } n] = \max_n \mu_n. \tag{37.10}$$

[If your intuition says 'surely the optimal decision should take into account the different uncertainties $\sigma_n$ too?', the answer to this question is 'reasonable – if so, then the utility function should be *nonlinear* in $x$'.]

Now the exciting bit. Should we prospect? Once we have prospected at site $n_p$, we will choose the site using the descision rule (37.9) with the value of mean $\mu_{n_p}$ replaced by the updated value $\mu'_n$ given by (37.8). What makes the problem exciting is that we don't yet know the value of $d_n$, so we don't know what our action $n_a$ will be; indeed the whole value of doing the prospecting comes from the fact that the outcome $d_n$ may alter the action from the one that we would have taken in the abscence of the experimental information.

From the expression for the new mean in terms of $d_n$ (37.8), and the known variance of $d_n$ (37.6) we can compute the probability distribution of the key quantity, $\mu'_n$, and can work out the expected utility by integrating over all possible outcomes and their associated actions.

Exercise 37.3: Show that the probability distribution of the new mean $\mu'_n$ (37.8) is Gaussian with mean $\mu_n$ and variance

$$s^2 \equiv \sigma_n^2 \frac{\sigma_n^2}{\sigma^2 + \sigma_n^2}. \tag{37.11}$$

Consider prospecting at site $n$. Let the biggest mean of the other sites be $\mu_1$. When we obtain the new value of the mean, $\mu'_n$, we will choose site $n$ and get an expected return of $\mu'_n$ if $\mu'_n > \mu_1$, and we will choose site 1 and get an expected return of $\mu_1$ if $\mu'_n < \mu_1$.

So the expected utility of prospecting at site $n$, then picking the best site, is

$$\mathcal{E}[U|\text{prospect at } n] = -c_n + P(\mu'_n < \mu_1)\,\mu_1 + \int_{\mu_1}^{\infty} d\mu'_n\, \mu'_n \,\text{Normal}(\mu'_n; \mu_n, s^2). \tag{37.12}$$

The difference in utility between prospecting and not prospecting is a quantity of interest, and it depends on what we would have done without prospecting. If $\mu_1$ is not only the biggest of the rest, but is also bigger than $\mu_n$, then we would have chosen $\mu_1$; if $\mu_n$, we would have chosen $n$.

$$\mathcal{E}[U|\text{no prospecting}] = \begin{cases} -\mu_1 & \text{if } \mu_1 \geq \mu_n \\ -\mu_n & \text{if } \mu_1 \leq \mu_n \end{cases} \tag{37.13}$$

So

$$\mathcal{E}[U|\text{prospect at } n] - \mathcal{E}[U|\text{no prospecting}]$$
$$= \begin{cases} -c_n + \int_{\mu_1}^{\infty} d\mu'_n\, (\mu'_n - \mu_1)\,\text{Normal}(\mu'_n; \mu_n, s^2) & \text{if } \mu_1 \geq \mu_n \\ -c_n + \int_{-\infty}^{\mu_1} d\mu'_n\, (\mu_1 - \mu'_n)\,\text{Normal}(\mu'_n; \mu_n, s^2) & \text{if } \mu_1 \leq \mu_n. \end{cases} \tag{37.14}$$

We can plot the change in expected utility due to prospecting (omitting $c_n$) as a function of (horizontal axis) the difference $(\mu_n - \mu_1)$ and (vertical axis) the initial standard deviation $\sigma_n$. In the figure the noise variance is $\sigma^2 = 1$.

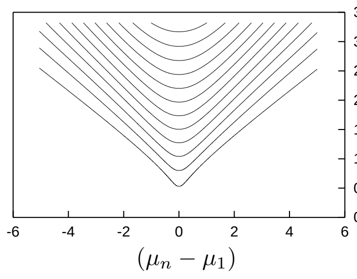


Figure 37.1. The gain in expected utility due to prospecting. The contours are equally spaced from 0.1 to 1.2 in steps of 0.1. To decide whether it is worth prospecting at site $n$, find the contour equal to $c_n$; all points $[(\mu_n - \mu_1), \sigma_n]$ above that contour are worthwhile.

# 38

# What Do You Know if You Are Ignorant?

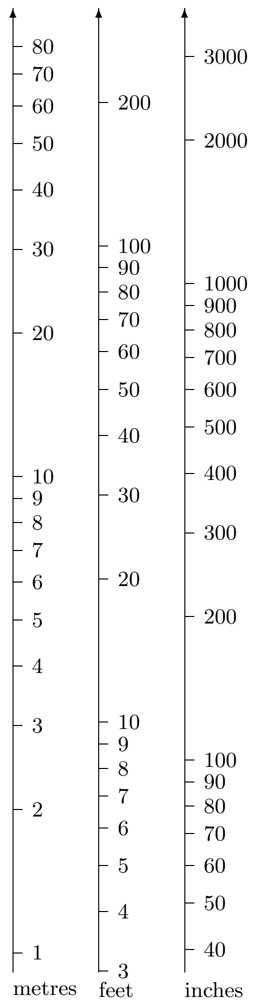


Figure 38.1. When viewed on a logarithmic scale, scales using different units are simply translated relative to each other.

Example 38.1: A real variable $x$ is measured in an accurate experiment. For example, $x$ might be the half-life of the neutron, the wavelength of light emitted by a firefly, the depth of Lake Vostok, or the mass of Jupiter's moon Io.

What is the probability that the value of $x$ starts with a '`1`', like the charge of the electron (in S.I. units),

$$e = 1.602\ldots \times 10^{-19}\,\mathrm{C},$$

and the Boltzmann constant,

$$k = 1.38066\ldots \times 10^{-23}\,\mathrm{J}\ \ \mathrm{K}^{-1}?$$

And what is the probability that it starts with a '`9`', like the Faraday constant,

$$\mathcal{F} = 9.648\ldots \times 10^{4}\mathrm{C} \quad \mathrm{mol}^{-1}?$$

What about the second digit? What is the probability that the mantissa of $x$ starts '`1.1...`', and what is the probability that $x$ starts '`9.9...`'?

Solution: An expert on neutrons, fireflies, Antarctica, or Jove might be able to predict the value of $x$, and thus predict the first digit with some confidence, but what about someone with no knowledge of the topic? What is the probability distribution corresponding to 'knowing nothing'?

One way to attack this question is to notice that the units of $x$ have not been specified. If the half-life of the neutron were measured in fortnights instead of seconds, the number $x$ would be divided by 1209600; or if it were measured in years, it would be divided by $3 \times 10^7$. Now, the question is, is our knowledge about $x$, and, in particular, our knowledge of its first digit, affected by the change in units? For the expert, the answer is yes; but let us take someone truly ignorant, for whom the answer is no, their predictions about the first digit of $x$ are independent of the units. The arbitrariness of the units corresponds to *invariance* of the probability distribution when $x$ is *multiplied* by any number.

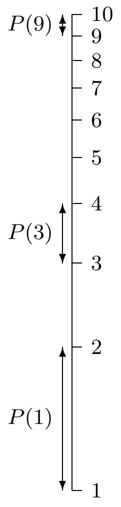
If you don't know the units that a quantity is measured in, the probability of the first digit must be proportional the length of the corresponding piece of logarithmic scale. The probability that the first digit of a number is `1` is thus

$$p_1 = \frac{\log 2 - \log 1}{\log 10 - \log 1} = \frac{\log 2}{\log 10} \qquad (38.1)$$

Now, $2^{10} = 1024 \simeq 10^3 = 1000$, so without a calculator, we have $10 \log 2 = 3 \log 10$ and

$$p_1 \simeq \frac{3}{10}. \tag{38.2}$$

This observation about initial digits is known as Benford's law. Ignorance does not correspond to a uniform probability distribution.



Example 38.2: A pin is thrown tumbling in the air. What is the probability distribution of the angle $\theta$ between the pin and the vertical?

A coin is thrown tumbling in the air. What is the probability distribution of the angle $\theta$ between the plane of the coin and the vertical?

# 39

# *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 of interest given the particular data set. As you have probably gathered, I strongly recommend the use of Bayesian methods.

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. Untrained readers – and perhaps, more worryingly, the authors of many papers – usually interpret 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. We have already seen such an example in exercise 3.5 (p.65), where a sampling theorist got a $p$-value smaller than 7%, and viewed this as strong evidence *against* the null hypothesis, whereas the data actually *favoured* the null hypothesis over the simplest alternative.

This chapter is only provided for those readers who are curious about the sampling theory / Bayesian methods debate. If you find any of this chapter tough to understand, please skip it. There is no point trying to understand the debate. Just use Bayesian methods – they are much easier to understand than the debate itself!

## 39.1 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 of human suffering or financial cost, depending on one's philosophy of health care.)

### *Sampling theory has a go*

The standard sampling theory approach to the question 'is A better than B?' is to construct a *statistical test*. The test usually compares a hypothesis such as

If you don't understand the sampling theory approach, don't worry. It is only important that you understand Bayesian methods.

$\mathcal{H}_1$: 'A and B have different effectivenesses'

with a null hypothesis such as

$\mathcal{H}_0$: 'A and B have exactly the same effectivenesses as each other'.

A novice might object 'no, no, I want to compare the hypothesis 'A is better than B' with the alternative 'B is better than A'!' but such objections are not welcome in sampling theory.

Once the two hypotheses have been defined, the first hypothesis is scarcely mentioned again – attention focuses solely on the null hypothesis. It makes me laugh to write this, but it's true! One chooses a *statistic* which measures how much a data set deviates from the null hypothesis. In the example here, the standard statistic to use would be one called $\chi^2$ (chi-squared). To compute $\chi^2$, we take the difference between each data measurement and its *expected* value *assuming the null hypothesis to be true*, and divide the square of that difference by the *variance* of the measurement, *assuming the null hypothesis to be true*. In the present problem, the four data measurements are the integers $F_{A+}$, $F_{A-}$, $F_{B+}$, and $F_{B-}$, that is, the number of subjects given treatment $A$ who contracted *microsoftus* ($F_{A+}$), the number of subjects given treatment $A$ who didn't ($F_{A-}$), and so forth. The definition of $\chi^2$ is:

$$\chi^2 = \sum_i \frac{(F_i - \langle F_i \rangle)^2}{\langle F_i \rangle}. \tag{39.1}$$

Actually in (Spiegel 1988) I find Yates's correction:

If you want to know about Yates's correction, read a sampling theory textbook. The point of this chapter is not to teach sampling theory; I merely mention Yates's correction because it is what a professional sampling theorist would use.

$$\chi^2 = \sum_i \frac{(|F_i - \langle F_i| - 0.5 \rangle)^2}{\langle F_i \rangle}. \tag{39.2}$$

In this case, given the null hypothesis that treatments $A$ and $B$ are equally effective, and have rates $f_+$ and $f_-$ for the two outcomes,

$$\langle F_{A+} \rangle = f_+ N_A \tag{39.3}$$
$$\langle F_{A-} \rangle = f_- N_A \tag{39.4}$$
$$\langle F_{B+} \rangle = f_+ N_B \tag{39.5}$$
$$\langle F_{B-} \rangle = f_- N_B \tag{39.6}$$

The test will determine whether we reject the null hypothesis on the basis of how big $\chi^2$ is. To make this test precise, and give it a 'significance level', we have to work out what the *sampling distribution* of $\chi^2$ is, taking into account the fact that the four data points are not independent (they satisfy the two constraints $F_{A+} + F_{A-} = N_A$ and $F_{B+} + F_{B-} = N_B$) and the fact that the parameters $f_\pm$ are not known. These three constraints reduce the 'number of degrees of freedom' in the data from four to one. [If you want to learn more about computing the 'number of degrees of freedom', read a sampling theory book; in Bayesian methods we don't need to know all that, and quantities equivalent to the number of degrees of freedom pop straight out of a Bayesian analysis when they are appropriate.] These sampling distributions are tabulated by sampling theory gnomes and come accompanied by warnings about the conditions under which they are accurate. For example, standard tabulated distributions for $\chi^2$ are only accurate if the expected numbers $F_i$ are about 5 or more.

The sampling distribution of a statistic is the probability distribution of its value under repetitions of the experiment, assuming that the null hypothesis is true.

Once the data arrive, we estimate the unknown parameters $f_\pm$ of the null hypothesis from the data:

$$\hat{f}_+ = \frac{F_{A+} + F_{B+}}{N_A + N_B} \tag{39.7}$$

$$\hat{f}_- = \frac{F_{A-} + F_{B-}}{N_A + N_B} \tag{39.8}$$

and we evaluate $\chi^2$. At this point, the sampling theory school divides itself into two camps. One camp uses the following protocol: first, before looking at the data, pick the significance level of the test (e.g., 5%), and determine the critical value of $\chi^2$ above which the null hypothesis will be rejected. [The significance level is the fraction of times that the statistic $\chi^2$ would exceed the critical value, if the null hypothesis were true.] Then evaluate $\chi^2$, compare with the critical value, and declare the outcome of the test, and its significance level (which was fixed beforehand).

The second camp looks at the data, finds $\chi^2$, then looks in the table of $\chi^2$-distributions for the significance level, $p$, for which the observed value of $\chi^2$ would be the critical value. The result of the test is then reported by giving this value of $p$, which is the fraction of times that a result as extreme as the one observed, or more extreme, would be expected to arise if the null hypothesis were true.

Let's apply these two methods. First camp: let's pick 5% as our significance level. The critical value for $\chi^2$ with one degree of freedom is $\chi^2_{.95} = 3.84$. The estimated values of $f_\pm$ are

$$f_+ = 1/10, \quad f_- = 9/10. \tag{39.9}$$

The expected values of the four measurements are

$$\begin{aligned}\langle F_{A+}\rangle &= 3 & (39.10)\\ \langle F_{A-}\rangle &= 27 & (39.11)\\ \langle F_{B+}\rangle &= 1 & (39.12)\\ \langle F_{B-}\rangle &= 9 & (39.13)\end{aligned}$$

and $\chi^2$ is

$$\chi^2 = 5.93 \tag{39.14}$$

Since this value exceeds 3.84, we reject the null hypothesis that the two treatments are equivalent at the 0.05 significance level. However, if we use Yates's correction, we find $\chi^2 = 3.33$, and therefore accept the null hypothesis.

Camp two runs a finger across the $\chi^2$ table found at the back of any good sampling theory book and finds $\chi^2_{.90} = 2.71$. Interpolating between $\chi^2_{.90}$ and $\chi^2_{.95}$, camp two reports '$p = 0.93$'.

Notice that this answer does not say how much more effective $A$ is than $B$, it simply says that $A$ is 'significantly' different from $B$. And here, 'significant' means only 'statistically significant', not practically significant.

The man in the street, reading the statement that 'the treatments was significantly different from the control ($p = 0.93$)', might come to the conclusion that 'there is a 93% chance that the treatments differ in effectiveness'. But what '$p = 0.93$' actually means is 'if you did this experiment many times, and the two treatments had equal effectiveness, then 7% of the time you would find a value of $\chi^2$ more extreme than the one that happened here'.

### *Let me through, I'm a Bayesian*

OK, now let's *infer* what we really want to know. Let's scrap the hypothesis that the two treatments have exactly equal effectivenesses, since we do not believe it. There are two unknown parameters, $p_{A+}$ and $p_{B+}$, which are the probabilities that people given treatments $A$ and $B$, respectively, contract the disease.

Given the data, we can infer these two probabilities, and we can answer questions of interest by examining the posterior distribution.

The posterior distribution is

$$P(p_{A+}, p_{B+} | \{F_i\}) = \frac{P(\{F_i\} | p_{A+}, p_{B+}) P(p_{A+}, p_{B+})}{P(\{F_i\})} \tag{39.15}$$

The likelihood function is

$$\begin{aligned} P(\{F_i\} | p_{A+}, p_{B+}) &= \binom{N_A}{F_{A+}} p_{A+}^{F_{A+}} p_{A-}^{F_{A-}} \binom{N_B}{F_{B+}} p_{B+}^{F_{B+}} p_{B-}^{F_{B-}}. && (39.16) \\ &= \binom{30}{1} p_{A+}^{1} p_{A-}^{29} \binom{10}{3} p_{B+}^{3} p_{B-}^{7}. && (39.17) \end{aligned}$$

What prior distribution should we use? The prior distribution gives us the opportunity to include knowledge from other experiments, or a prior belief that the two parameters $p_{A+}$ and $p_{B+}$, while different from each other, are expected to have similar values.

Here we will use the simplest vanilla prior distribution, a uniform distribution over each parameter.

$$P(p_{A+}, p_{B+}) = 1. \tag{39.18}$$

We can now plot the posterior distribution. Given the assumption of a separable prior on $p_{A+}$ and $p_{B+}$, the posterior distribution is actually separable:

$$P(p_{A+}, p_{B+} | \{F_i\}) = P(p_{A+} | F_{A+}, F_{A-}) P(p_{B+} | F_{B+}, F_{B-}) \tag{39.19}$$

The two posterior distributions are shown in figure 39.1 (except the graphs are not normalized) and the joint posterior probability is shown in figure 39.2.

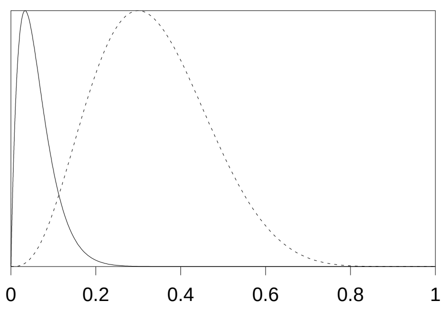


Figure 39.1. Posterior probabilities of the two effectivenesses. A – solid line; B – dotted line.

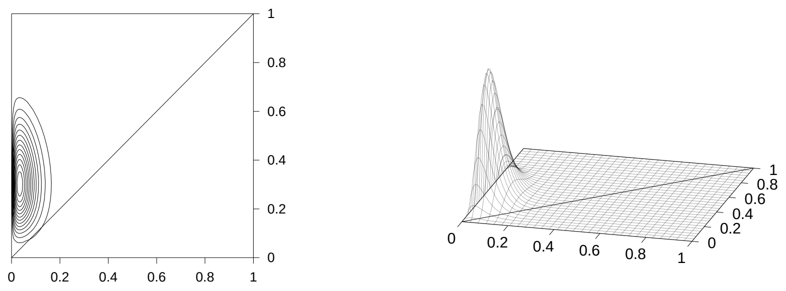


Figure 39.2. Joint posterior probability of the two effectivenesses – contour plot and surface plot.

If we want to know the answer to the question 'how probable is it that $p_{A+}$ is smaller than $p_{B+}$?', we can answer exactly that question by computing the posterior probability

$$P(p_{A+} < p_{B+}|\text{Data}), \tag{39.20}$$

which is the integral of the joint posterior probability $P(p_{A+}, p_{B+}|\text{Data})$ shown in figure 39.2 over the region in which $p_{A+} < p_{B+}$. That region is the upper left triangle in figure 39.2. The value of this integral (obtained by a straightforward numerical integration of the likelihood function (39.17) over the relevant region) is $P(p_{A+} < p_{B+}|\text{Data}) = 0.990$.

Thus there is a 99% chance, given the data and our prior assumptions, that treatment $A$ is superior to treatment $B$.

## *Model comparison*

If there were a situation in which we wanted to compare the two hypotheses $\mathcal{H}_1 : p_{A+} = p_{B+}$ and $\mathcal{H}_2 : p_{A+} \neq p_{B+}$, we can of course do this directly with Bayesian methods also. As an example, consider the data set:

$D$: One subject, given treatment A, subsequently contracted *microsoftus*.
One subject, given treatment B, did not.

How strongly does this data set favour $\mathcal{H}_2$ over $\mathcal{H}_1$?

We answer this question by computing the evidence for each hypothesis. Let's assume uniform priors over the unknown parameters of the models. The first hypothesis $\mathcal{H}_1 : p_{A+} = p_{B+}$ has just one unknown parameter, let's call it $p$.

$$P(p|\mathcal{H}_1) = 1 \;\; p \in (0,1) \tag{39.21}$$

We'll use the uniform prior over the two parameters of model $\mathcal{H}_2$ that we used before:

$$P(p_{A+}, p_{B+}|\mathcal{H}_2) = 1 \;\; p_{A+} \in (0,1), p_{B+} \in (0,1). \tag{39.22}$$

Now, the probability of the data $D$ under model $\mathcal{H}_1$ is the normalizing constant from the inference of $p$ given $D$:

$$\begin{aligned} P(D|\mathcal{H}_1) &= \int dp\, P(D|p)P(p|\mathcal{H}_1) && (39.23) \\ &= \int dp\, p(1-p) \times 1 && (39.24) \\ &= 1/6. && (39.25) \end{aligned}$$

The probability of the data $D$ under model $\mathcal{H}_2$ is given by a simple two-dimensional integral:

$$\begin{aligned} P(D|\mathcal{H}_2) &= \int\!\!\int dp_{A+}\, dp_{B+}\, P(D|p_{A+}, p_{B+})P(p_{A+}, p_{B+}|\mathcal{H}_2) && (39.26) \\ &= \int dp_{A+} p_{A+} \int dp_{B+}(1-p_{B+}) && (39.27) \\ &= 1/2 \times 1/2 && (39.28) \\ &= 1/4. && (39.29) \\ & && (39.30) \end{aligned}$$

Thus the evidence ratio in favour of model $\mathcal{H}_2$, which asserts that the two effectivenesses are unequal, is

$$\frac{P(D|\mathcal{H}_2)}{P(D|\mathcal{H}_1)} = \frac{1/4}{1/6} = \frac{0.6}{0.4}. \tag{39.31}$$

So if the prior probability over the two hypotheses was 50:50, the posterior probability is 60:40 in favour of $\mathcal{H}_2$.

Is it not easy to get sensible answers to well-posed questions using Bayesian methods?

[The sampling theory answer to this question would involve the identical significance test that was used in the preceding problem; that test would yield a 'not significant' result. I think it is greatly preferable to acknowledge what is obvious to the intuition, namely that the data $D$ do give evidence in favour of $\mathcal{H}_2$, and that that evidence is very weak. Bayesian methods quantify how weak the evidence is.]

## 39.2 Confidence intervals

In an experiment in which data $D$ are obtained from a system with an unknown parameter $\theta$, a standard concept in sampling theory is the idea of a *confidence interval* for $\theta$. Such an interval $(\theta_{\min}(D), \theta_{\max}(D)$ has associated with it a *confidence level*[1] such as 95% which is informally interpreted as 'the probability that $\theta$ lies in this interval'.

[1]check terminology

Let's make precise what the confidence level really means, then give an example. A confidence interval is a rule for obtaining an interval $(\theta_{\min}(D), \theta_{\max}(D))$ from a data set $D$. The confidence level of the confidence interval is a property that we can compute before the data arrives. We imagine generating many data sets from a particular true value of $\theta$, and calculating the interval $(\theta_{\min}(D), \theta_{\max}(D))$, and then checking whether the true value of $\theta$ lies in that interval. If, averaging over all these imagined repetitions of the experiment, the true value of $\theta$ lies in the confidence interval a fraction $f$ of the time, and this property holds for all true values of $\theta$, then the confidence level of the confidence interval is $f$.

For example, if $\theta$ is the mean of a Gaussian distribution which is known to have standard deviation 1, and $D$ is a sample from that Gaussian, then $(\theta_{\min}(D), \theta_{\max}(D)) = D-2, D+2$ is a 95% confidence interval for $\theta$.

Let us now look at a simple example where the meaning of the confidence level becomes clearer. Let the parameter $\theta$ be an integer, and let the data be a pair of points $x_1, x_2$, drawn independently from the following distribution:

$$P(x|\theta) = \begin{cases} 1/2 & x = \theta \\ 1/2 & x = \theta + 1 \\ 0 & \text{for other values of } x \end{cases} \tag{39.32}$$

For example, if $\theta$ were 39, then we could expect the following data sets:

$$\begin{aligned} D = (x_1, x_2) &= (39, 39) \quad \text{with probability } 1/4; \\ (x_1, x_2) &= (39, 40) \quad \text{with probability } 1/4; \\ (x_1, x_2) &= (40, 39) \quad \text{with probability } 1/4; \\ (x_1, x_2) &= (40, 40) \quad \text{with probability } 1/4. \end{aligned} \tag{39.33}$$

We now consider the following confidence interval:

$$[\theta_{\min}(D), \theta_{\max}(D)] = [\min(x_1, x_2), \min(x_1, x_2)]. \tag{39.34}$$

For example, if $(x_1, x_2) = (40, 39)$, then the confidence interval for $\theta$ would be $[\theta_{\min}(D), \theta_{\max}(D)] = [39, 39]$.

Let's think about this confidence interval. What is its confidence level? By considering the four possibilities shown in (39.33), we can see that there is a 75% chance that the confidence interval will contain the true value. The confidence interval therefore has a confidence level of 75%, by definition.

Now, what if the data we acquire are $(x_1, x_2) = (29, 29)$? Well, we can compute the confidence interval, and it is $[29, 29]$. So shall we report this interval, and its associated confidence level, 75%? Does this make sense? What do we actually know in this case? Intuitively, or by Bayes's theorem, it is clear that $\theta$ could either be 29 or 28, and both possibilities are equally likely. The posterior probability of $\theta$ is 50% on 29 and 50% on 28.

What if the data are $(x_1, x_2) = (29, 30)$? In this case, the confidence interval is still $[29, 29]$, and its associated confidence level is 75%. But in this case, by Bayes's theorem, or common sense, we are 100% sure that $\theta$ is 29.

Thus

1. the way in which many people interpret the confidence levels of classical statistics is *incorrect*;

2. given some data, what people usually want to know (whether they know it or not) is a Bayesian posterior probability distribution.

# Part V

# Neural networks

# 40

# Introduction to neural networks

In the field of neural networks, we study the properties of networks of idealised 'neurons'.

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 book 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.

## 40.1 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 need 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 a memory without knowing the address. [Computer scientists have devoted effort to wrapping traditional address-based memories inside cunning software to produce 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.

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.

   It 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*. Given a person's name, we can often recall their face; and *vice versa*. 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 'a 20th century American actor who was 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 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 some functional specialization

– but in the parts of the brain where memories are stored, it seems that 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.

## 40.2 Terminology

Each time we describe a neural network algorithm we will typically specify three things. [If any of this terminology is hard to understand, it's probably best to dive straight into the next chapter.]

**Architecture.** The architecture specifies what variables are involved in the model and their topological relationships – for example, the variables involved in a neural net might 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: local rules define how 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 *teacher*'s specification 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.

# 41

# The single neuron as a classifier

## 41.1 The single neuron

We will study a single neuron for two reasons. First, many 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 41.1.) 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,
$$a = \sum_i w_i x_i, \qquad (41.1)$$
where the sum is over $i = 0, \ldots, I$ if there is a bias and $i = 1, \ldots, I$ otherwise.

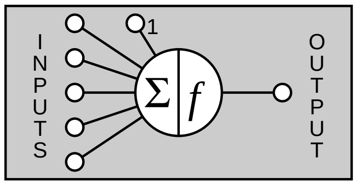


Figure 41.1. A single neuron

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*; here are the most popular.

$$\begin{array}{ccc} \text{activation} & & \text{activity} \\ & \rightarrow & \\ a & & f(a) \end{array}$$

   (a) Deterministic activation functions:

      i. Linear.

$$y(a) = a \tag{41.2}$$

      ii. Sigmoid (logistic function).

$$y(a) = \frac{1}{1+e^{-a}} \qquad (y \in (0,1)) \tag{41.3}$$

      iii. Sigmoid (tanh).

$$y(a) = \tanh(a) \qquad (y \in (-1,1)) \tag{41.4}$$

      iv. Threshold function.

$$y(a) = \Theta(a) \equiv \begin{cases} 1 & a > 0 \\ -1 & a \leq 0 \end{cases} \tag{41.5}$$

   (b) Stochastic activation functions: $y$ is stochastically selected from $\pm 1$.

      i. Heat bath.

$$y(a) = \begin{cases} 1 & \text{with probability } \dfrac{1}{1+e^{-a}} \\ -1 & \text{otherwise} \end{cases} \tag{41.6}$$

      ii. The Metropolis rule produces the output in a way that depends on the previous output state $y$: we compute $\Delta = ay$.

$$\begin{array}{l} \textsf{If } \Delta \leq 0, \text{ flip } y \text{ to the other state} \\ \textsf{Else } \text{flip } y \text{ to the other state with probability } e^{-\Delta}. \end{array} \tag{41.7}$$

## 41.2 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 $\mathbf{x}$:

$$y(\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}} \tag{41.8}$$

Exercise 41.1:$^{A1}$ 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 12.2 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

$$P(s=1|\mathbf{y}) = \frac{1}{1+\exp(-a(\mathbf{y}))}, \tag{41.9}$$

where $a(\mathbf{y})$ was a linear function of the received vector,

$$a(\mathbf{y}) = \mathbf{w}^{\mathsf{T}}\mathbf{y} + \theta, \tag{41.10}$$

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:

$$y(\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-(w_1x_1+w_2x_2)}}. \tag{41.11}$$

Figure 41.2 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 41.3. 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 41.2. 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 a way as to

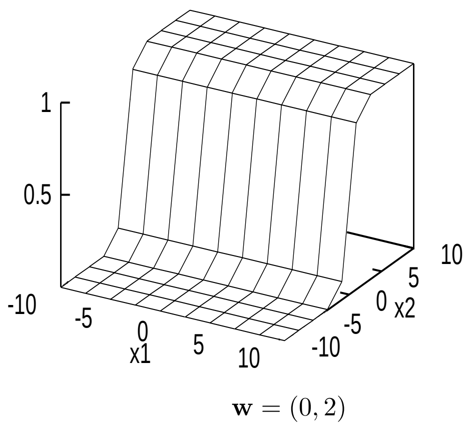


Figure 41.2. Output of a simple neural network as a function of its input

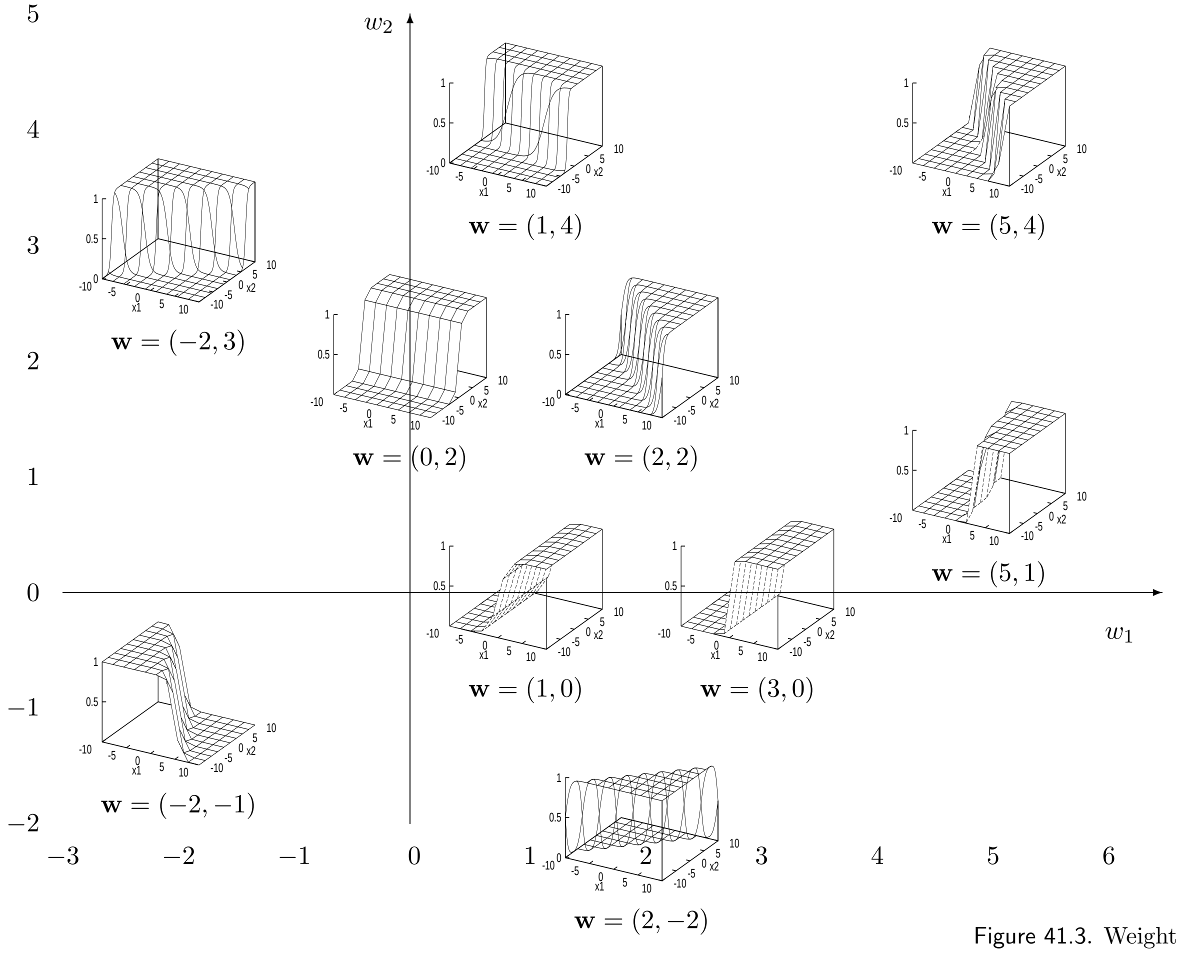


Figure 41.3. Weight space.

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 $\mathbf{w}$.

## 41.3 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*:

$$G(\mathbf{w}) = -\sum_{n}\left[t^{(n)}\log y(\mathbf{x}^{(n)};\mathbf{w}) + (1-t^{(n)})\log(1-y(\mathbf{x}^{(n)};\mathbf{w}))\right]. \quad (41.12)$$

Each term in this objective function may be recognized as the *information content* of the outcome. It may also be described as the 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 41.2:$^{A2}$ The backpropagation algorithm. Show that the derivative $\mathbf{g} = \partial G/\partial\mathbf{w}$ is given by:

$$g_j = \frac{\partial G}{\partial w_j} = \sum_{n=1}^{N} -(t^{(n)} - y^{(n)})x_j^{(n)}. \quad (41.13)$$

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, since a gradient has dimensions [1/parameter], whereas a change in a parameter has dimensions [parameter]), so equation (41.13) motivates the following learning algorithm. Since the derivative $\partial G/\partial\mathbf{w}$ is a sum of terms $\mathbf{g}^{(n)}$ defined by

$$g_j^{(n)} \equiv -(t^{(n)} - y^{(n)})x_j^{(n)} \quad (41.14)$$

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,

$$a = \sum_{i} w_i x_i, \quad (41.15)$$

```
global x ;        # x is an N * I 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
```

Algorithm 41.1. `Octave` source code for a gradient descent optimizer of a single neuron, batch learning, 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 $(N \times I)$ *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`. $L$ is the total number of gradient descent steps made.

where the sum is over $i = 0, \ldots, I$ if there is a bias and $i = 1, \ldots, I$ otherwise.

2. Second, the *output* $y$ is set as a sigmoid function of the activation.

$$y(a) = \frac{1}{1 + e^{-a}} \tag{41.16}$$

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*

$$e = t - y \tag{41.17}$$

then adjust the weights $\mathbf{w}$ in a direction that would reduce the magnitude of this error:

$$\Delta w_i = \eta e x_i, \tag{41.18}$$

where $\eta$ is the 'learning rate'. Commonly $\eta$ is set by trial and error to a constant value or to a decreasing function of simulation time $\tau$ such as $\eta_0/\tau$.

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.

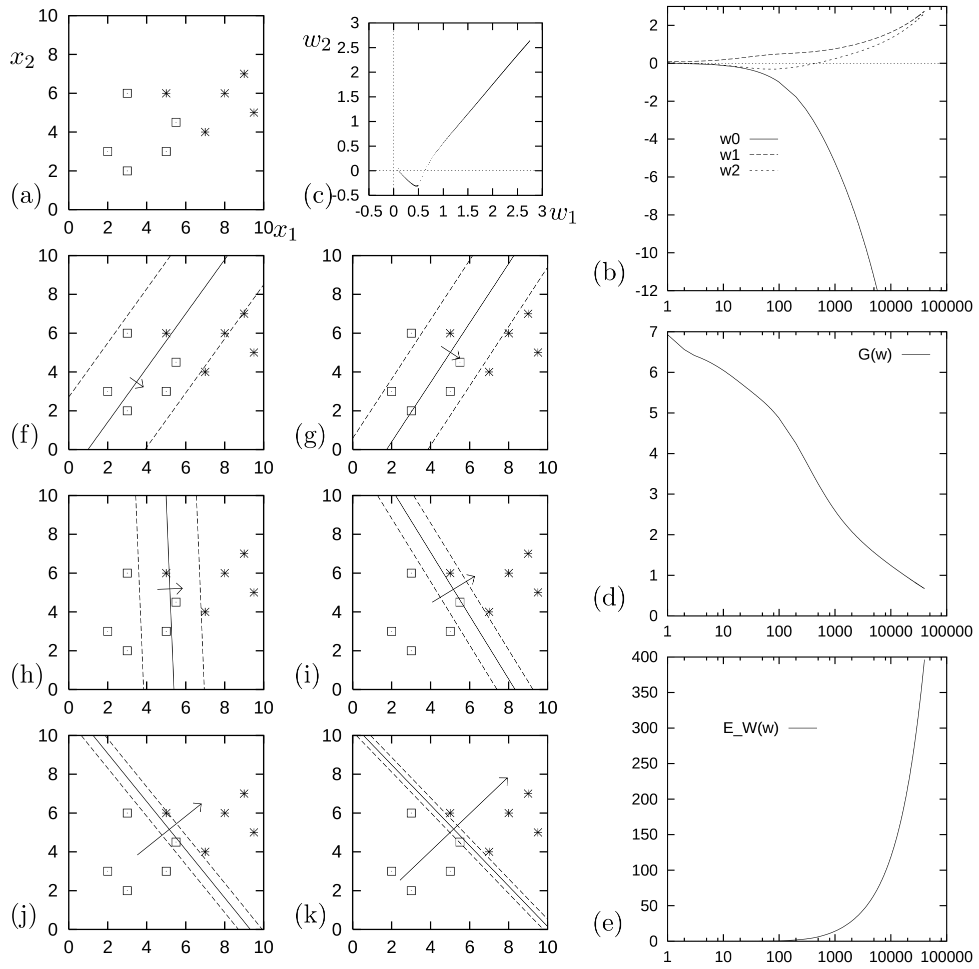


Figure 41.4. A single neuron learning to classify by gradient descent. The neuron has two weights $w_1$ and $w_2$ and a bias $w_0$. The learning rate was set to $\eta = 0.01$ and batch–mode gradient descent was performed using the code displayed in algorithm 41.1. (a) The training data. (b) Evolution 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.

*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 (41.18) 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

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp\left(-\sum_i w_i x_i\right)}, \tag{41.19}$$

define $e^{(n)} = t^{(n)} - y^{(n)}$, and compute for each weight $w_i$

$$g_i^{(n)} = -e^{(n)} x_i^{(n)}. \tag{41.20}$$

**Then** let

$$\Delta w_i = -\eta \sum_n g_i^{(n)}. \tag{41.21}$$

Source code implementing batch learning gradient descent is given in algorithm 41.1. This algorithm is demonstrated in algorithm 41.4 for a neuron with two inputs with weights $w_1$ and $w_2$ and a bias $w_0$, performing the function

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}. \tag{41.22}$$

The bias $w_0$ is included, in contrast to the figure 41.3, where it was omitted. The neuron is trained on a data set of ten labelled examples.

## 41.4 Beyond descent on the error function: regularization

If the parameter $\eta$ 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 41.4(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 in figure 41.4(k); this sharp boundary is associated with large weight values, so we

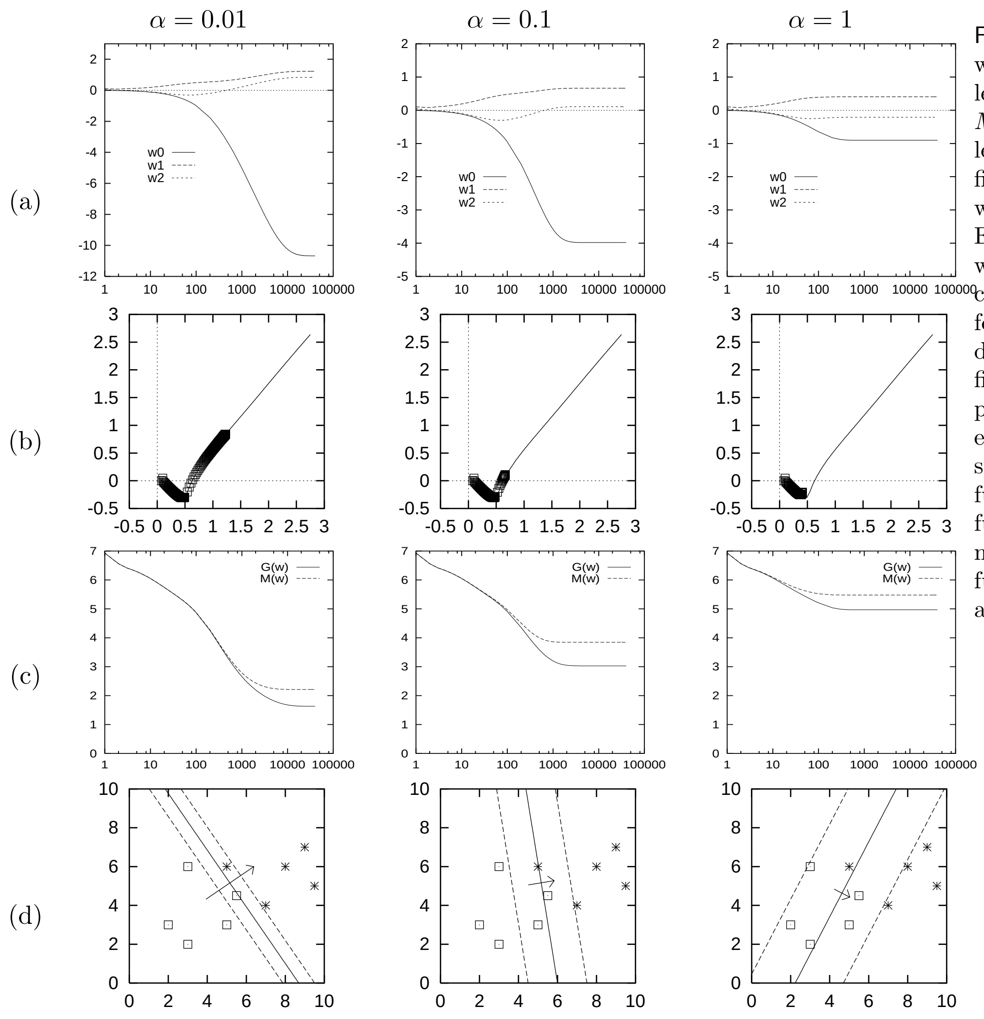


Figure 41.5. The influence of weight decay on a single neuron's learning. The objective function is $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$. The learning method was as in figure 41.4. (a) Evolution of weights $w_0$, $w_1$ and $w_2$. (b) Evolution of weights $w_1$ and $w_2$ in weight space shown by points, contrasted with the trajectory followed in the case of zero weight decay, shown by a thin line (from figure 41.4). 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.

use a regularizer that penalizes large weight values. We modify the objective function to:

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{41.23}$$

where the simplest choice of regularizer is the '*weight decay*' regularizer

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_i w_i^2. \tag{41.24}$$

The regularization constant $\alpha$ 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 $\alpha$ 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 41.3:$^{A2}$ 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 algorithm 41.1 implements weight decay. This gradient descent algorithm is demonstrated in figure 41.5 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 $\eta$ is in general *not* the most efficient way to minimize a function. A modification of gradient descent known as *momentum*, while improving convergence, is also not recommended. Most neural network experts use more advanced optimizers such as conjugate gradient algorithms. [An aside to those who are familiar with momentum: please do not confuse momentum, which is sometimes given the symbol $\alpha$, with weight decay.]

## 41.5 Exercises

### *Motivations for the linear neuron*

Exercise 41.4:$^{B2}$ Consider the task of recognizing which of two Gaussian distributions a vector $\mathbf{z}$ comes from. Unlike the case studied in section 12.2, 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

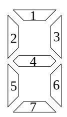
$$P(\mathbf{z}|s) = \prod_{i=1}^{I} \text{Normal}(z_i; 0, \sigma_{si}^2), \tag{41.25}$$

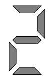
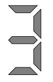
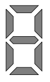
where $\sigma_{si}^2$ is the variance of $z_i$ when the source symbol is $s$. Show that $P(s=1|\mathbf{z})$ can be written in the form

$$P(s=1|\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{w}^{\mathsf{T}}\mathbf{x}+\theta)}, \tag{41.26}$$

where $x_i$ is an appropriate function of $z_i$, $x_i = g(z_i)$.

Exercise 41.5:$^{A2}$ **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

$$P(s = 2|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^{\mathsf{T}}\mathbf{x} + \theta)}, \tag{41.27}$$

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

$$P(s|\mathbf{x}) = \frac{e^{a_s}}{\sum_{s'} e^{a_{s'}}}, \tag{41.28}$$

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?

Exercise 41.6:$^{B2}$ A (3,1) code consists of the two codewords $\mathbf{x}^{(1)} = (1, 0, 0)$ and $\mathbf{x}^{(2)} = (0, 0, 1)$. A source bit $s \in \{1, 2\}$ having probability distribution $\{p_1, p_2\}$ is used to select one of the two codewords for transmission over a binary symmetric channel with noise level $f$. The received vector is $\mathbf{r}$. Show that the posterior probability of $s$ given $\mathbf{r}$ can be written in the form

$$P(s = 1|\mathbf{r}) = \frac{1}{1 + \exp\left(-w_0 - \sum_{n=1}^{3} w_n r_n\right)},$$

and give expressions for the coefficients $\{w_n\}_{n=1}^{3}$ and the bias, $w_0$. [5]

Describe, with a diagram, how this optimal decoder can be expressed in terms of a 'neuron'.

# *Solutions to Chapter 41's exercises*

Solution to exercise 41.1 (p.484): One answer, given in the text on page 484, 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 41.2 (p.487): ...

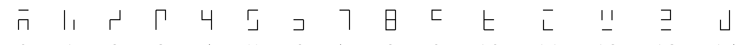
Solution to exercise 41.3 (p.492): ...

Solution to exercise 41.4 (p.492): ...

Solution to exercise 41.5 (p.493): 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 |

Solution to exercise 41.6 (p.493):

$$\begin{aligned}
\log \frac{P(s=1|\mathbf{r})}{P(s=2|\mathbf{r})} &= \log \frac{P(\mathbf{r}|s=1)P(s=1)}{P(\mathbf{r}|s=2)P(s=2)} \\
&= \log \left(\frac{1-f}{f}\right)^{2r_1-1} + \log \left(\frac{1-f}{f}\right)^{-(2r_3-1)} + \log \frac{P(s=1)}{P(s=2)} \\
&= w_1 r_1 + w_3 r_3 + w_0,
\end{aligned}$$

where

$$\begin{aligned}
w_1 &= 2\log\left(\frac{1-f}{f}\right) \\
w_3 &= -2\log\left(\frac{1-f}{f}\right) \\
w_0 &= \log \frac{P(s=1)}{P(s=2)} \\
(w_2 &= 0)
\end{aligned}$$

which we can rearrange to give

$$P(s=1|\mathbf{r}) \quad = \quad \frac{1}{1+\exp\left(-w_0 - \sum_{n=1}^{3} w_n r_n\right)}.$$

This can be viewed as a neuron with two or three inputs, one from $r_1$ with a positive weight, and one from $r_3$ with a negative weight, and a bias. (Picture here of blob with three lines.)

# *Problems to look at before Chapter 42*

Exercise 41.7: What is $\sum_{K=0}^{N} \binom{N}{K}$? [The symbol $\binom{N}{K}$ means the combination $\frac{N!}{K!(N-K)!}$.]

Exercise 41.8: 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 41.9: 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 41.1** *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 41.10: 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?

This chapter's material is originally due to Polya (1954) and 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.

# 42

# *Capacity of a single neuron*

## 42.1 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

$$\begin{tikzcd}
\{t_n\}_{n=1}^N \arrow[r] & \begin{array}{c}\text{Learning}\\ \text{Algorithm}\end{array} \arrow[r] & \mathbf{w} \arrow[r] & \mathbf{w} \arrow[r] & \{\hat{t}_n\}_{n=1}^N \\
& \{\mathbf{x}_n\}_{n=1}^N \arrow[u] & & \{\mathbf{x}_n\}_{n=1}^N \arrow[u] &
\end{tikzcd}$$

Figure 42.1. 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?

## 42.2 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 weight.* 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 42.1** *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

$$y = f\left(\sum_{k=1}^{K} w_k x_k\right). \tag{42.1}$$

where

$$f(a) = \begin{cases} 1 & a > 0 \\ 0 & a \leq 0 \end{cases} \tag{42.2}$$

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. (These input points would not then be in general position, but it is actually sufficient for the original input vectors to be in general position.)

## 42.3 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 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 360 degrees we reproduce the function we started from. Because the points are in general position, the separating plane (line) 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 $\mathbf{x} = (\pm 1, \pm 1)$. [These points are not in general position, but you may confirm that the function remains unrealisable if the points are perturbed into general position.]

### *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 42.2(a) shows a single point in our two–dimensional $\mathbf{x}$–space, and figure 42.2(b) shows the two

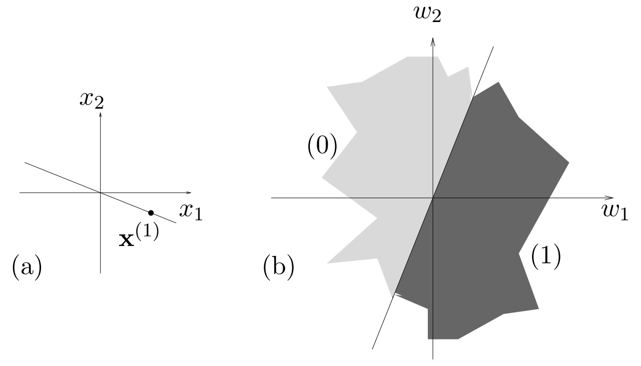


Figure 42.2. One data point in a two–dimensional input space, and the two regions of weight space that give the two alternative labellings of that point.

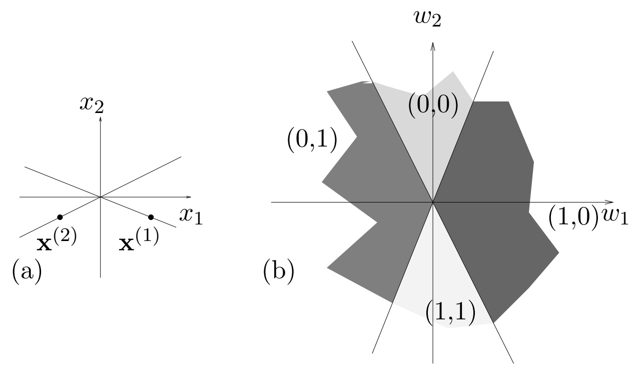


Figure 42.3. Two data points in a two–dimensional input space, and the four regions of weight space that give the four alternative labellings.

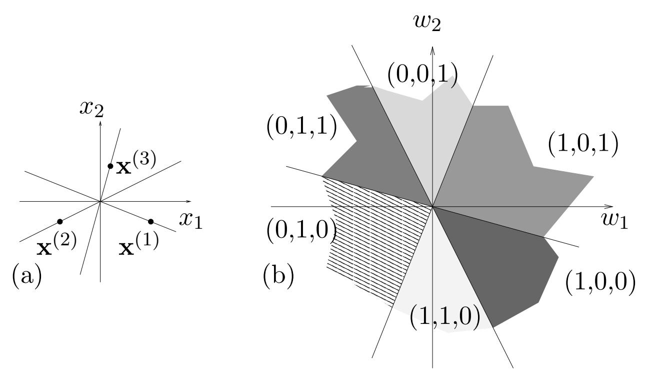


Figure 42.4. Three data points in a two–dimensional input space, and the six regions of weight space that give alternative labellings of those points. In this case, the labellings (0,0,0) and (1,1,1) cannot be realised. For any three points in general position there are always two labellings that cannot be realised.

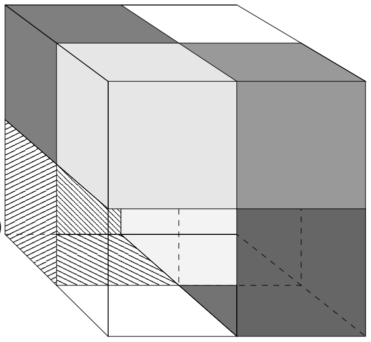
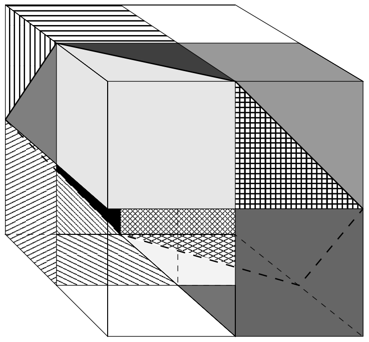
(a)

(b)


Figure 42.5. 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 42.5a: 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 42.1. Values of $T(N,K)$ deduced by hand

corresponding sets of points in **w**–space. One set of weight vectors occupy the half space

$$\mathbf{x}^{(n)} \cdot \mathbf{w} > 0, \tag{42.3}$$

and the others occupy $\mathbf{x}^{(n)} \cdot \mathbf{w} < 0$. In figure 42.3(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 42.3(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 42.4), 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 **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 **w** space, so that there are 8 distinguishable regions. $T(3,3) = 8$. The three bisecting planes are shown in figure 42.5a.

At this point matters become slightly more tricky. The fourth plane in the three–dimensional **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 42.2. Pascal's triangle

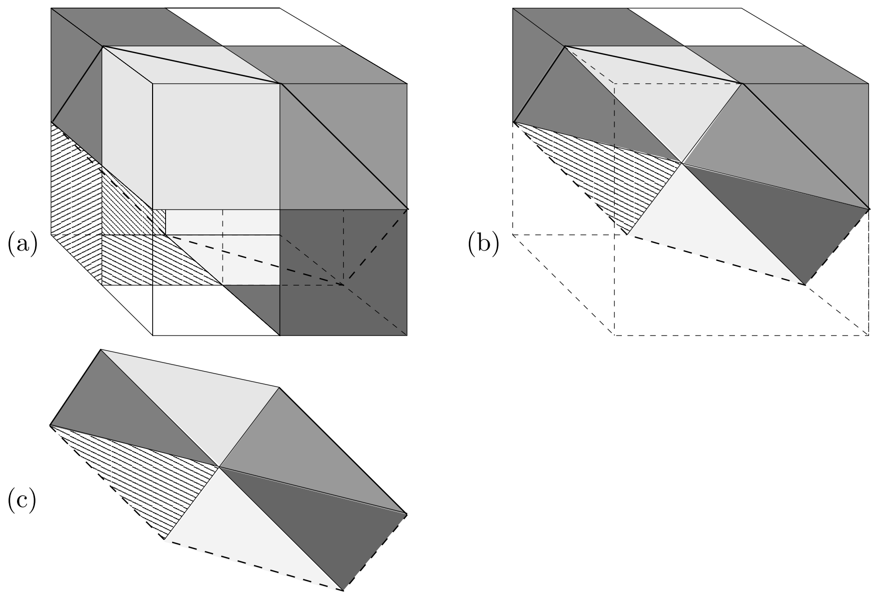


Figure 42.6. Illustration of the cutting process going from $T(3,3)$ to $T(4,3)$. (a) The eight regions of figure 42.5a with one added hyperplane. All of the regions that are not coloured white have been cut into two. (b) Here, the hollow cube has been made solid, so we can see which regions are cut 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 42.6a which is cut into two by this new hyperplane. This shows that $T(4,3) - T(3,3) = 6$. Figure 42.6c should be compared with figure 42.4b.

(figure 42.5b). 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 42.1. 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 42.6a). Equivalently, if we look in the $K-1$ dimensional subspace that is the $N$th hyperplane, that subspace is divided into six regions by the $N-1$ previous hyperplanes (figure 42.6c). Now this is a concept we have met before. Compare figure 42.6c with figure 42.4b. 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

$$T(4,3) = T(3,3) + T(3,2). \tag{42.4}$$

## *Recurrence relation for any $N, K$*

Generalizing this picture, we see that when we add an $N$th 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. Therefore, the total number of regions obtained after adding the $N$th hyperplane is $2T(N-1,K-1)$ (since $T(N-1,K-1)$ out of $T(N-1,K)$ regions are split in two) plus the remaining $T(N-1,K) - T(N-1,K-1)$ regions not split by the Nth hyperplane, which gives the following equation for $T(N,K)$:

$$T(N,K) = T(N-1,K) + T(N-1,K-1). \tag{42.5}$$

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 (42.5) 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

$$C(N,K) \equiv \binom{N}{K} \equiv \frac{N!}{(N-K)!K!}. \tag{42.6}$$

So combinations $\binom{N}{K}$ satisfy the equation

$$C(N,K) = C(N-1,K-1) + C(N-1,K), \quad \text{for all } N > 0. \tag{42.7}$$

[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 (42.5). 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 42.2 and 42.1 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:

$$T(N,K) = 2\sum_{k=0}^{K-1} \binom{N-1}{k} \tag{42.8}$$

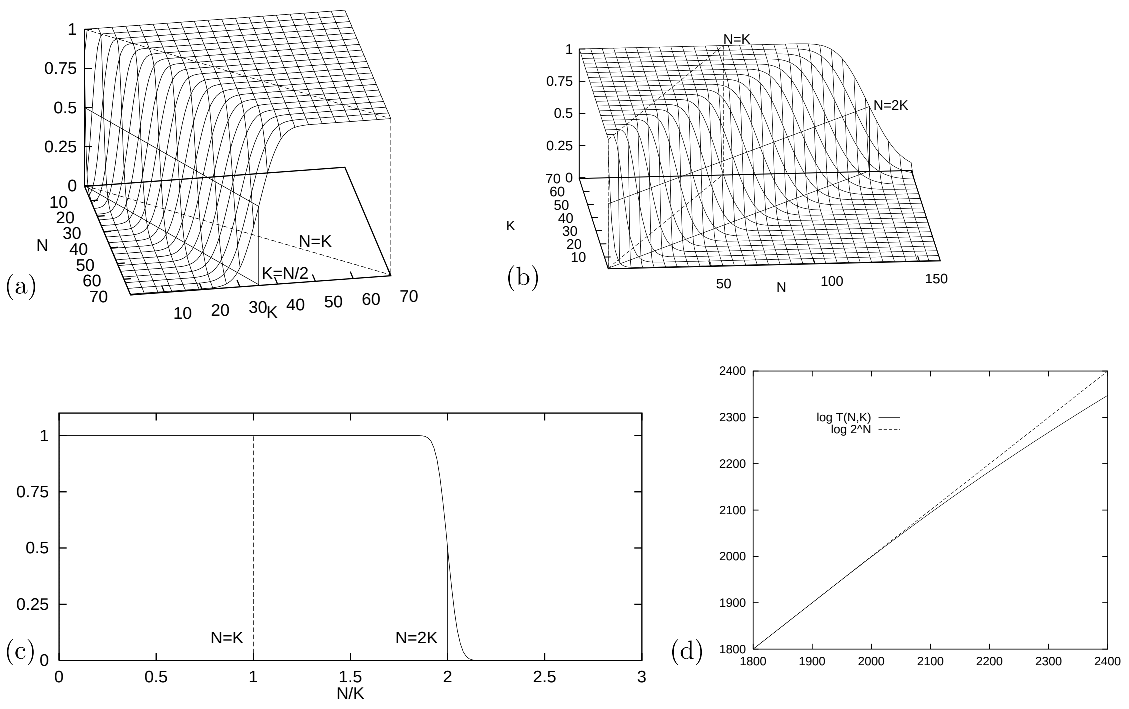


Figure 42.7. The fraction of functions on $N$ points in $K$ dimensions that are linear threshold functions, $T(N,K)/2^N$, shown from various viewpoints. In (a) we see the dependence on $K$, which is approximately an error function passing through 0.5 at $K = N/2$; the fraction reaches 1 at $K = N$. In (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.

Using the fact that the $N$th 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 $K-1 \geq N-1$.

$$T(N,K) = \begin{cases} 2^N & K \geq N \\ 2\sum_{k=0}^{K-1} \binom{N-1}{k} & K < N \end{cases} \tag{42.9}$$

## *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 (42.9) is well approximated by the error function,

$$\sum_{0}^{K} \binom{N}{k} \simeq 2^N \, \Phi\left( \frac{K-(N/2)}{\sqrt{N}/2} \right), \tag{42.10}$$

where $\Phi(z) \equiv \int_{-\infty}^{z} \exp(-z^2/2)/\sqrt{2\pi}$. Figure 42.7 shows the fraction $T(N,K)/2^N$ as a function of $N$ and $K$. The take-home message is shown in figure 42.7(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 weight.

A single neuron can almost certainly memorize up to $N = 2K$ random binary labels, but will almost certainly fail to memorize more.

## 42.4 Problems

Exercise 42.1:$^{A2}$ 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 42.2:$^{A2}$ 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 42.3:$^{B2}$ 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 42's exercises*

Solution to exercise 42.1 (p.505): The probability that all 4 points lie on a single hemisphere is

$$T(4,3)/2^4 = 14/16 = 7/8. \tag{42.11}$$

Solution to exercise 42.2 (p.505):

(a) $\mathbf{w} = (1,1,1)$.

(b) $\mathbf{w} = (1/4, 1/4, -1)$.

The two unrealizable labellings are $\{0,0,0,1\}$ and $\{1,1,1,0\}$.

# 43

# *Learning as Inference*

## 43.1 Neural network learning as inference

In chapter 41 we trained a simple neural network as a classifier by minimizing an objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{43.1}$$

made up of an error function

$$G(\mathbf{w}) = -\sum_n \left[ t^{(n)} \log y(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \log(1 - y(\mathbf{x}^{(n)}; \mathbf{w})) \right] \tag{43.2}$$

and a regularizer

$$E_W(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 . \tag{43.3}$$

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

$$\begin{aligned} P(t = 1|\mathbf{w}, \mathbf{x}) &= y \\ P(t = 0|\mathbf{w}, \mathbf{x}) &= 1 - y \end{aligned} \tag{43.4}$$

can be rewritten as the single equation

$$P(t|\mathbf{w}, \mathbf{x}) = y^t (1 - y)^{1-t} = \exp\left[t \log y + (1 - t) \log(1 - y)\right] . \tag{43.5}$$

So the error function $G$ can be interpreted as minus the log likelihood:

$$P(D|\mathbf{w}) = \exp[-G(\mathbf{w})]. \tag{43.6}$$

Similarly the regularizer can be interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{43.7}$$

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:

$$\begin{aligned} P(\mathbf{w}|D,\alpha) &= \frac{P(D|\mathbf{w})P(\mathbf{w}|\alpha)}{P(D|\alpha)} && (43.8) \\ &= \frac{e^{-G(\mathbf{w})}\, e^{-\alpha E_W(\mathbf{w})}/Z_W(\alpha)}{P(D|\alpha)} && (43.9) \\ &= \frac{1}{Z_M} \exp(-M(\mathbf{w})). && (43.10) \end{aligned}$$

*So the* $\mathbf{w}$ *found by (locally) minimizing* $M(\mathbf{w})$ *can be interpreted as the (locally) most probable parameter vector,* $\mathbf{w}^*$.

I also refer to $\mathbf{w}^*$ as $\mathbf{w}_{\rm MP}$. Need to rationalize this.

Why is it natural to interpret the error functions as *log* probabilities? Error functions are usually additive. For example, $G$ is a *sum* of information contents, 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.

## 43.2 Illustration for a neuron with two weights

In the case of a neuron with just two inputs and no bias,

$$y(\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-(w_1 x_1 + w_2 x_2)}}, \tag{43.11}$$

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 43.1. Each data point consists of a two dimensional input vector $\mathbf{x}$ and a $t$ value indicated by $\times$ $(t=1)$ or $\square$ $(t=0)$. 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 (43.11).

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 43.1). 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 43.1). The posterior ensemble (within a multiplicative constant) is shown in the third column of figure 43.1, 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 $\mathbf{w}^*$.

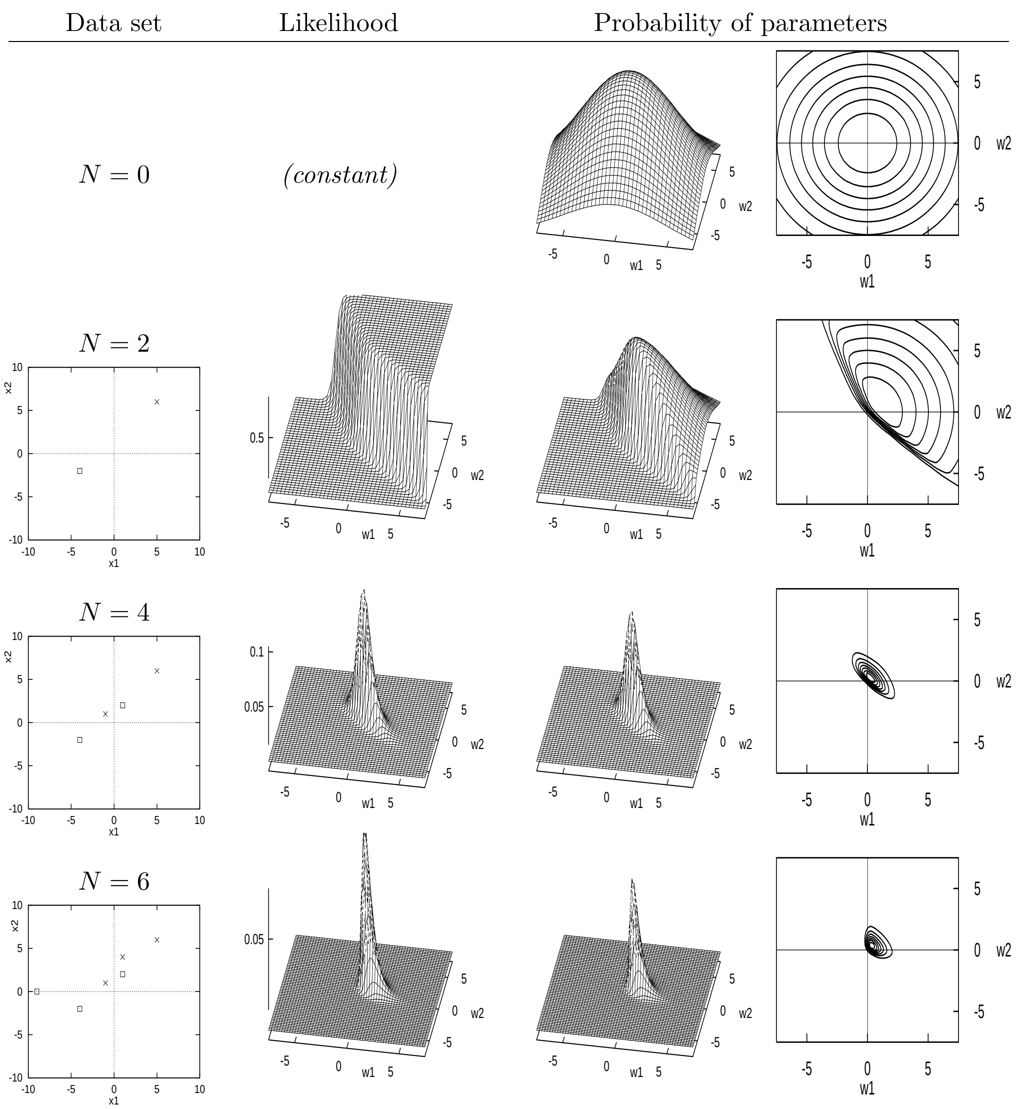


Figure 43.1. The Bayesian interpretation and generalization of traditional neural network learning. Evolution of the probability distribution over parameters as data arrive.

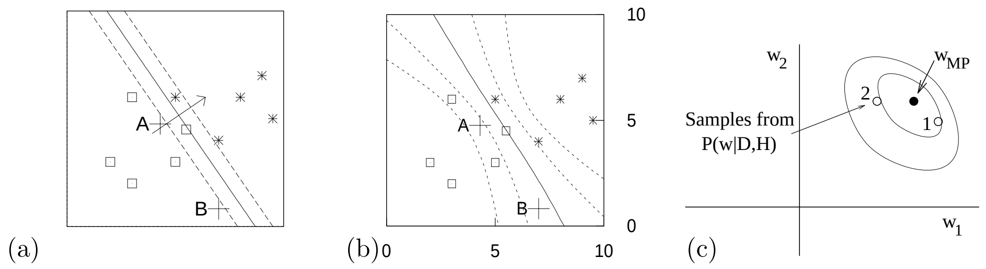


Figure 43.2. Making predictions. (a) The function performed by an optimized neuron (shown by three of its contours) trained with weight decay, $\alpha = 0.01$ (from figure 41.5). The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and $0.73$. (b) Are these predictions 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 43.4.

## 43.3 Beyond optimization: making predictions

Let us consider the task of making predictions with the neuron which we trained as a classifier in section 41.3. This was a neuron with two inputs and a bias.

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}} \tag{43.12}$$

When we last played with it, we trained it by minimizing the objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}). \tag{43.13}$$

The resulting optimized function for the case $\alpha = 0.01$ is reproduced in figure 43.2(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 43.2a. 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 43.2(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 43.2(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 $\alpha$ 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:

$$P(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, D, \alpha) = \int d^k\mathbf{w}\, P(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) P(\mathbf{w}|D, \alpha). \tag{43.14}$$

Thus the predictions are obtained by weighting the prediction for each possible $\mathbf{w}$,

$$\begin{array}{rcl} P(\mathbf{t}^{(N+1)}{=}1 \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & y(\mathbf{x}^{(N+1)}; \mathbf{w}) \\ P(\mathbf{t}^{(N+1)}{=}0 \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & 1 - y(\mathbf{x}^{(N+1)}; \mathbf{w}) \end{array}, \tag{43.15}$$

with a weight given by the posterior probability of $\mathbf{w}$, $P(\mathbf{w}|D, \alpha)$, which we most recently wrote down in equation (43.10). This posterior probability is

$$P(\mathbf{w}|D, \alpha) = \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{43.16}$$

where

$$Z_M = \int \mathrm{d}^k\mathbf{w} \exp(-M(\mathbf{w})). \tag{43.17}$$

In summary, we can get the Bayesian predictions if we can find a way of computing the integral

$$P(\mathbf{t}^{(N+1)}{=}1|\mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^k\mathbf{w}\ \ y(\mathbf{x}^{(N+1)}; \mathbf{w})\, \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{43.18}$$

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 (43.18)? 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 27), by Monte Carlo sampling (chapter 30), or by deterministic approximate methods, for example, methods that make Gaussian approximations to $P(\mathbf{w}|D, \alpha)$ using Taylor expansions (chapter 28) or variational methods (chapter 34). 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).

## 43.4 Monte Carlo implementation of a single neuron*

First we will use a Monte Carlo approach in which the task of evaluating the integral (43.18) is solved by treating $y(\mathbf{x}^{(N+1)}; \mathbf{w})$ as a function $f$ of $\mathbf{w}$ whose mean we compute using

$$\langle f(\mathbf{w}) \rangle \simeq \frac{1}{R} \sum_r f(\mathbf{w}^{(r)}). \tag{43.19}$$

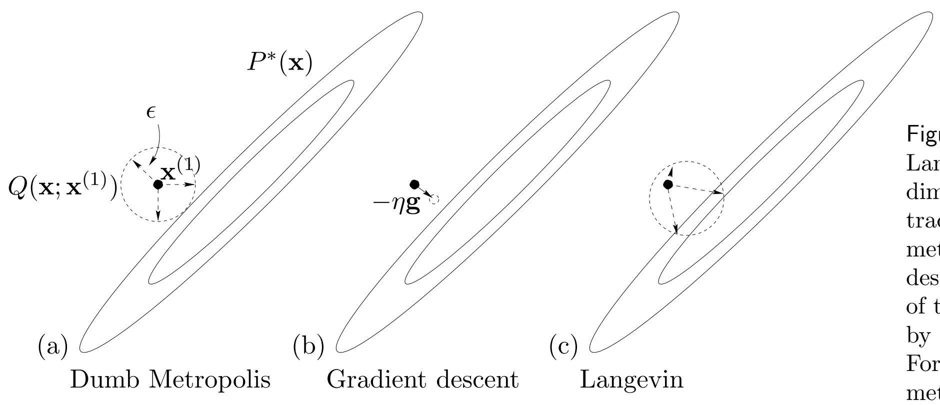


Figure 43.3. One step of the Langevin method in two dimensions (c), contrasted with a traditional 'dumb' Metropolis method (a) and with gradient descent (b). The proposal density of the Langevin method is given by 'gradient descent with noise'. For a complete description of the method see the text.

where $\{\mathbf{w}^{(r)}\}$ are random samples from the posterior distribution $\frac{1}{Z_M}\exp(-M(\mathbf{w}))$ (c.f. equation (30.4)). We will obtain the samples using a Metropolis method (section **??**). 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 algorithm 43.1. It may be summarised as 'gradient descent with added noise', as shown pictorially in figure 43.3. 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

$$\Delta\mathbf{w} = -\tfrac{1}{2}\epsilon^2\mathbf{g} + \epsilon\mathbf{p}. \tag{43.20}$$

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.

The Langevin method has one free parameter, $\epsilon$, which controls the typical step size. If $\epsilon$ is set to too large a value, moves may be rejected. If it is set to a very small value, progress around the state space will be small.

## *Demonstration of Langevin method*

The Langevin method is demonstrated in figures 43.4, 43.5 and 43.6. 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 41.5). 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.

```
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
```

Algorithm 43.1. `Octave` source code for the Langevin Monte Carlo method. To obtain a 'hybrid Monte Carlo' method, we repeat the four lines marked * multiple times (algorithm 43.2).

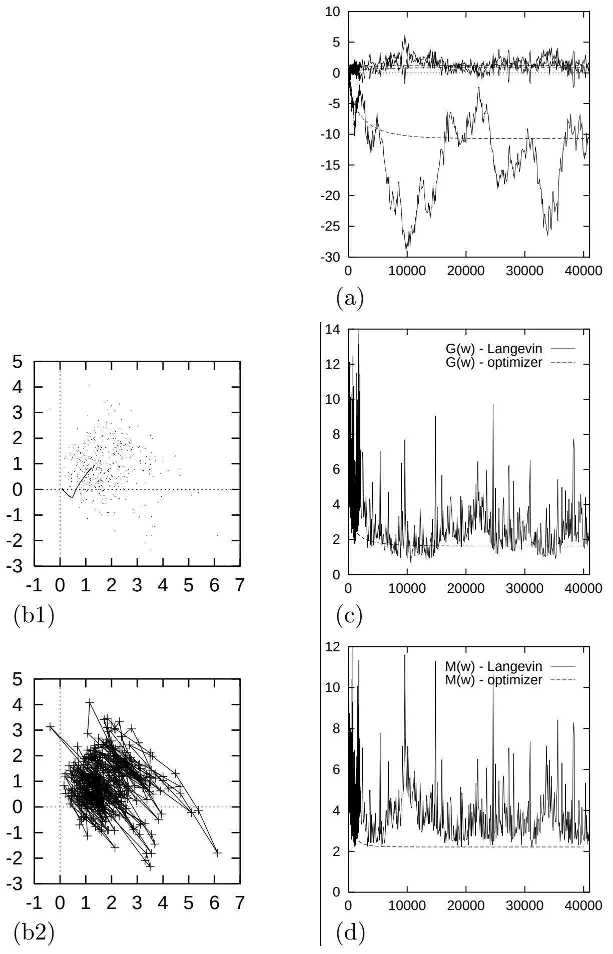


Figure 43.4. A single neuron learning under the Langevin Monte Carlo method. (a) Evolution of weights $w_0$, $w_1$ and $w_2$ as a function of number of iterations. (b) Evolution of weights $w_1$ and $w_2$ in weight space. Also shown by a line is the evolution of the weights using the optimizer of figure 41.5. 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 41.5. (d) The objective function $M(\mathbf{x})$ as a function of number of iterations. See also figures 43.5 and 43.6.

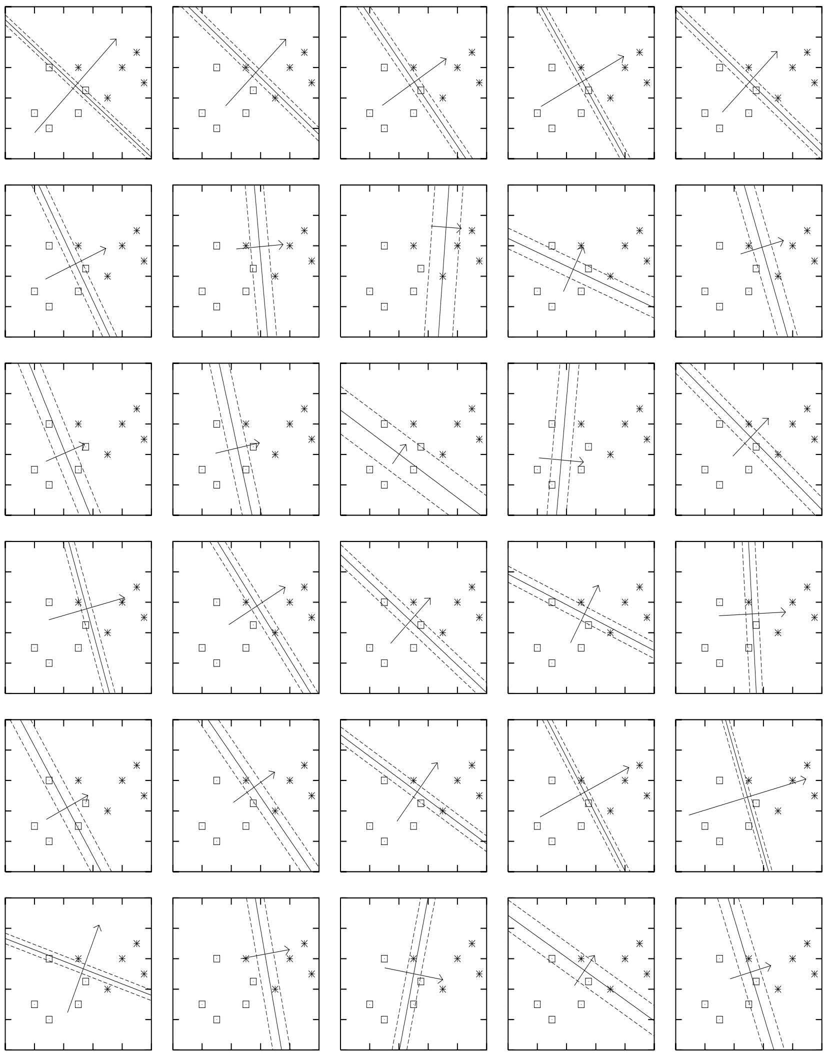


Figure 43.5. Samples obtained by the Langevin Monte Carlo method. The learning rate was set to $\eta = 0.01$ and the weight decay rate to $\alpha = 0.01$. The step size is given by $\epsilon = \sqrt{2\eta}$. The function performed by the neuron is shown by three of its contours every 1000 iterations 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 $(w_1, w_2)$.

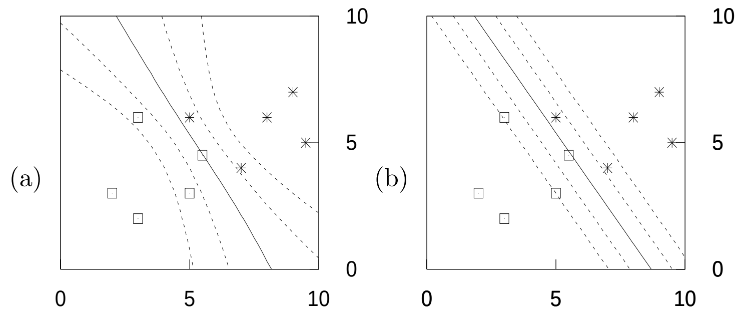


Figure 43.6. Bayesian predictions found by the Langevin Monte Carlo method compared with the predictions using the optimized parameters. (a) The predictive function obtained by averaging the predictions for 30 samples uniformly spaced between iterations 10000 and 40000, shown in figure 43.5. The contours 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 $M(\mathbf{w})$.

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 $\epsilon$ had been used, but the value was chosen so that the 'descent rate' $\eta = \frac{1}{2}\epsilon^2$ matched the step size of the earlier simulations.

## *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 43.5. 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 43.6 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 43.4). 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*

```
wnew = w ;
gnew = g ;
for tau = 1:Tau

   p = p - epsilon * gnew / 2 ;     # make half-step in p
   wnew = wnew + epsilon * p ;      # make step in w

   gnew = gradM ( wnew ) ;          # find new gradient
   p = p - epsilon * gnew / 2 ;     # make half-step in p

endfor
```

Algorithm 43.2. `Octave` source code for the hybrid Monte Carlo method. The code is identical to that for the Langevin method in algorithm 43.1, except for the replacement of the four lines marked `*` in that figure by the above fragment.

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 overconfident.

### *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 43.2. 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 $\epsilon$ 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 43.7. 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%.

## 43.5 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

$$P(\mathbf{t}^{(N+1)} = 1 \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int d^k\mathbf{w}\; y(\mathbf{x}^{(N+1)}; \mathbf{w}) \frac{1}{Z_M} \exp(-M(\mathbf{w})), \quad (43.21)$$

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-

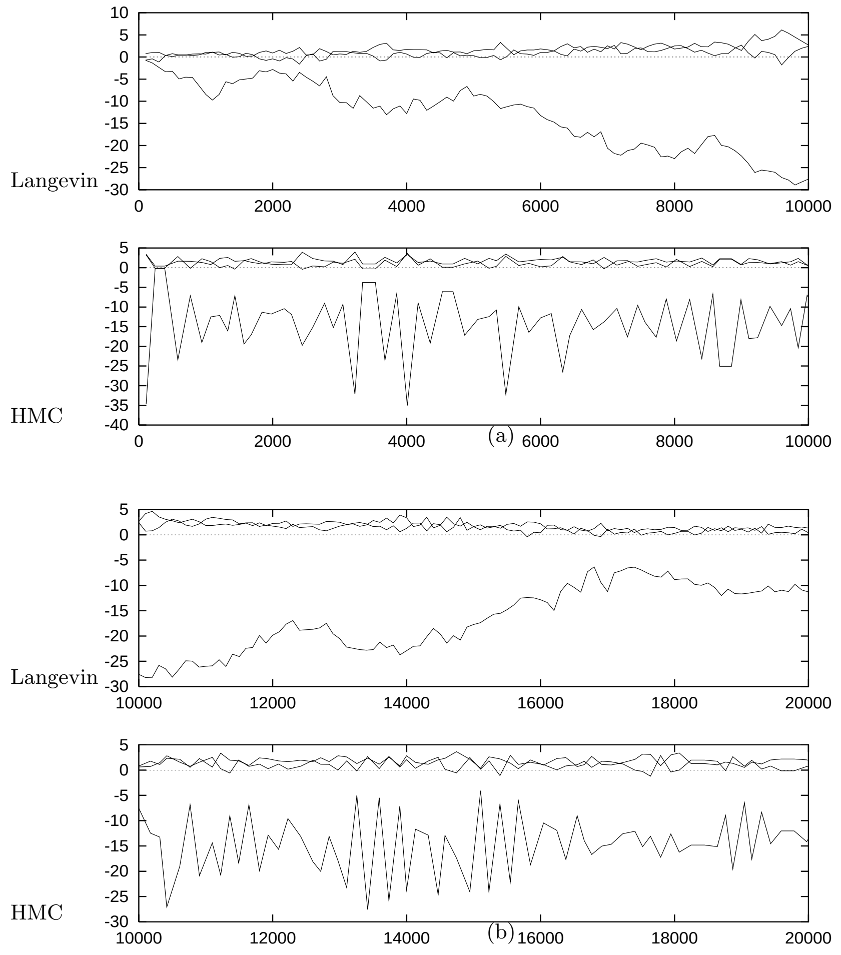


Figure 43.7. Comparison of sampling properties of the Langevin Monte Carlo method and the hybrid Monte Carlo (HMC) method. The horizontal axis is the number of gradient evalutations made. (a) The weights during the first 10000 iterations. (b) The weights during iterations 10000–20000.

expand $M$ there:

$$M(\mathbf{w}) \simeq M(\mathbf{w}_{\rm MP}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_{\rm MP})^{\mathsf{T}}\mathbf{A}(\mathbf{w} - \mathbf{w}_{\rm MP}) + \ldots, \tag{43.22}$$

where $\mathbf{A}$ is the matrix of second derivatives, also known as the **Hessian**, defined by

$$A_{ij} \equiv \left.\frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w})\right|_{\mathbf{w}=\mathbf{w}_{\rm MP}}. \tag{43.23}$$

We thus define our Gaussian approximation:

$$Q(\mathbf{w}; \mathbf{w}_{\rm MP}, \mathbf{A}) = [\det(\mathbf{A}/2\pi)]^{1/2} \exp\left[-\frac{1}{2}(\mathbf{w} - \mathbf{w}_{\rm MP})^{\mathsf{T}}\mathbf{A}(\mathbf{w} - \mathbf{w}_{\rm MP})\right]. \tag{43.24}$$

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 43.1:$^{A2}$ Show that the second derivative of $M(\mathbf{w})$ with respect to $\mathbf{w}$ is given by

$$\frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) = \sum_{n=1}^{N} f'(a^{(n)}) x_i^{(n)} x_j^{(n)} + \alpha\delta_{ij}, \tag{43.25}$$

where $f'(a)$ is the first derivative of

$$f(a) \equiv 1/(1 + e^{-a}) \tag{43.26}$$

which is given by

$$f'(a) = \frac{d}{da} f(a) = f(a)(1 - f(a)), \tag{43.27}$$

and

$$a^{(n)} = \sum_j w_j x_j^{(n)}. \tag{43.28}$$

Having computed the Hessian, our task is then to perform the integral (43.21) 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 $\mathbf{w} = \mathbf{w}_{\rm MP} + \Delta\mathbf{w}$, $P(\mathbf{w} \,|\, D, \alpha) \simeq (1/Z_Q)\exp(-\frac{1}{2}\Delta\mathbf{w}^{\mathsf{T}}\mathbf{A}\Delta\mathbf{w})$. 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 43.2:$^{B2}$ 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

$$P(a \,|\, \mathbf{x}, D, \alpha) = \text{Normal}(a_{\rm MP}, s^2) = \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{(a - a_{\rm MP})^2}{2s^2}\right), \tag{43.29}$$

where $a_{\rm MP} = a(\mathbf{x}; \mathbf{w}_{\rm MP})$ and $s^2 = \mathbf{x}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{x}$.

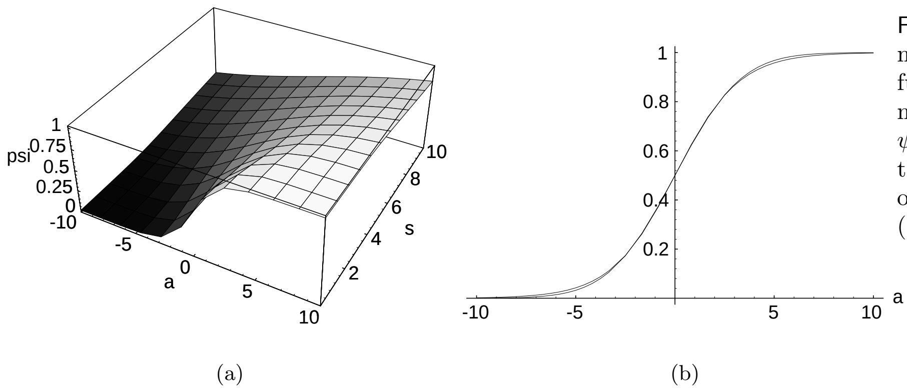


Figure 43.8. Approximation to the marginalized probability. (a) The function $\psi(a, s^2)$, evaluated numerically. In (b) the functions $\psi(a, s^2)$ and $\phi(a, s^2)$ defined in the text are shown as a function of $a$ for $s^2 = 4$. From MacKay (1992b).

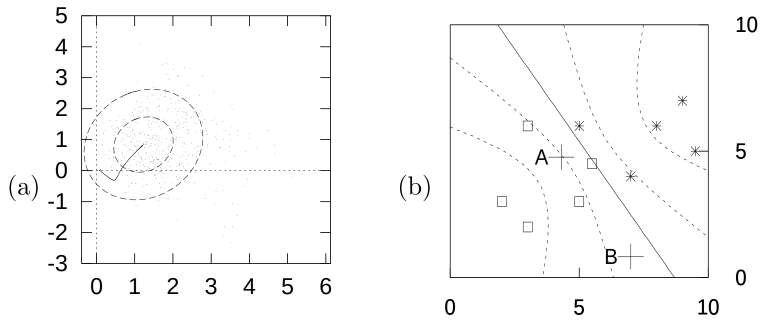


Figure 43.9. The Gaussian approximation and its approximate predictions. (a) A projection of the Gaussian approximation onto the $(w_1, w_2)$ plane of weight space. The one- and two-standard-deviation contours are shown. Also shown are the trajectory of the optimizer, and the Monte Carlo method's samples. 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 (43.31). The contours shown are those corresponding to $a = 0, \pm 1, \pm 2$, namely $y = 0.5, 0.27$, $0.73$, $0.12$ and $0.88$.

This means that the marginalized output is:

$$P(t{=}1 \mid \mathbf{x}, D, \alpha) = \psi(a_{\rm MP}, s^2) \equiv \int da\, f(a)\, \mathrm{Normal}(a_{\rm MP}, s^2). \tag{43.30}$$

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:

$$\psi(a_{\rm MP}, s^2) \simeq \phi(a_{\rm MP}, s^2) \equiv f(\kappa(s) a_{\rm MP}) \tag{43.31}$$

with $\kappa = 1/\sqrt{1 + \pi s^2/8}$ (figure 43.8). The errors in the approximation are most significant when $a \gg s$.

## *Demonstration*

Figure 43.9 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 (43.31) to make predictions. Comparing these predictions with those of the Langevin Monte Carlo method (figure 43.6) 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 43.3:$^{B2}$ Is the Gaussian approximation to $P(\mathbf{w} \mid D, \alpha)$ too heavy-tailed or too light-tailed, or both? It may help to consider $P(\mathbf{w} \mid D, \alpha)$ 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 'I guess 0' and 'I guess 1'. And even if there are just the two choices '0' and '1', if the costs associated with error are unequal, then the decision boundary will be some contour other than the 0.5 contour, and the boundary will be affected by marginalization.

# *Solutions to Chapter 43's exercises*

Solution to exercise 43.2 (p.519): 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

$$\begin{aligned} \left\langle (a - a(\mathbf{x}; \mathbf{w}_{\text{MP}}))^2 \right\rangle &= \langle \mathbf{x} \cdot (\mathbf{w} - \mathbf{w}_{\text{MP}})(\mathbf{w} - \mathbf{w}_{\text{MP}}) \cdot \mathbf{x} \rangle \qquad (43.32) \\ &= \mathbf{x}^{\mathsf{T}} \left\langle (\mathbf{w} - \mathbf{w}_{\text{MP}})(\mathbf{w} - \mathbf{w}_{\text{MP}})^{\mathsf{T}} \right\rangle \mathbf{x} \;=\; \mathbf{x}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{x}. \end{aligned}$$

Finally, we know $a$ must be Gaussian because the marginals of a multivariate Gaussian are Gaussian.

Solution to exercise 43.3 (p.521): 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 43.10a. 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

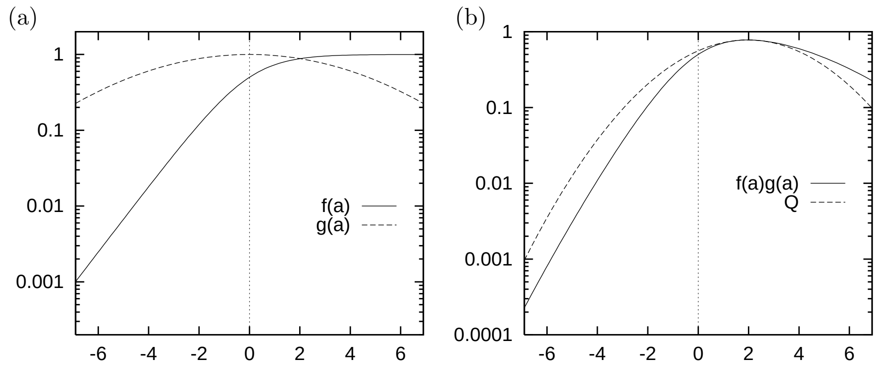


Figure 43.10. (a) The log of the sigmoid function $f(a) = 1/(1 + e^{-a})$ and the log of a Gaussian $g(a) \propto \text{Normal}(0, 4^2)$. (b) The product $P = f(a)g(a)$ and a Gaussian approximation to it, fitted at its mode. Notice that for a range of negative values of $a$, the Gaussian approximation $Q$ is bigger than $P$, while for values of $a$ beyond the mode, $Q$ is smaller than $P$.

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.

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 43.10b. 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 43.1). 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.

# *Postscript on Supervised Neural Networks*

One of my students, Robert, asked:

> Maybe I'm missing something fundamental, but supervised neural networks seem equivalent to fitting a pre-defined function to some given data, then extrapolating – what's the difference?

I agree with Robert. The supervised neural networks we have studied so far are simply parameterized nonlinear functions which can be fitted to data.

Hopefully you will agree with another comment that Robert made:

> Unsupervised networks seem much more interesting than their supervised counterparts. I'm amazed that it works!

# 44

# *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.

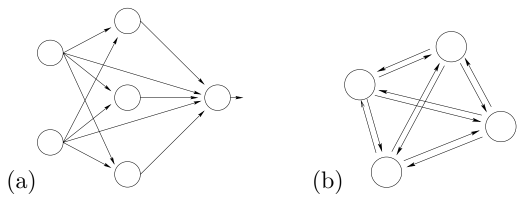


Figure 44.1. (a) A feedforward network. (b) A feedback network.

**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 first discuss the idea of associative memory, also known as content-addressable memory.

## 44.1 Hebbian learning

In chapter 40, 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 the weights between neurons whose activities are *positively* *correlated* are *increased*:

$$\frac{dw_{ij}}{dt} \sim \text{Correlation}(x_i, x_j). \tag{44.1}$$

Now imagine that when stimulus $m$ is present (for example, the smell of a banana), the activity of neuron $m$ increases; and that neuron $n$ is associated with another stimulus, $n$ (for example, the sight of a yellow object). If these two stimuli – a yellow sight and a banana smell – co-occur in the environment, then the Hebbian learning rule (44.1) will increase 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'. No 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.

## 44.2 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

$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases} \tag{44.2}$$

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

$$a_i = \sum_j w_{ij} x_j \tag{44.3}$$

then update their states simultaneously to

$$x_i = \Theta(a_i); \tag{44.4}$$

**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**,

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)}, \tag{44.5}$$

where $\eta$ is an unimportant constant. To prevent the largest possible weight from growing with $N$ we might choose to set $\eta = 1/N$.

Exercise 44.1:$^{A1}$ Explain why the value of $\eta$ is not important for the Hopfield network defined above.

## 44.3 Definition of the continuous Hopfield network

Using the identical architecture and learning rule 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

$$a_i = \sum_j w_{ij} x_j \tag{44.6}$$

and

$$x_i = \tanh(a_i). \tag{44.7}$$

The learning rule is the same as in the binary Hopfield network, but the value of $\eta$ becomes relevant. Alternatively, we may fix $\eta$ and introduce a **gain** $\beta \in (0, \infty)$ into the activation function:

$$x_i = \tanh(\beta a_i). \tag{44.8}$$

Exercise 44.2:$^{A1}$ Where have we encountered equations (44.6, 44.7 & 44.8) before?

## 44.4 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 44.2. 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.

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 (44.6,

(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 44.2. 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.

44.8) when we discussed variational methods: when we approximated the spin system whose energy function was

$$E(\mathbf{x};\mathbf{J}) = -\frac{1}{2}\sum_{m,n} J_{mn}x_m x_n - \sum_n h_n x_n \tag{44.9}$$

with a separable distribution

$$Q(\mathbf{x};\mathbf{a}) = \frac{1}{Z_Q}\exp\left(\sum_n a_n x_n\right) \tag{44.10}$$

and optimized the latter so as to minimize the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})E(\mathbf{x};\mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})\log\frac{1}{Q(\mathbf{x};\mathbf{a})}, \tag{44.11}$$

we found that the pair of iterative equations

$$a_m = \beta\left(\sum_n J_{mn}\bar{x}_n + h_m\right) \tag{44.12}$$

and

$$\bar{x}_n = \tanh(a_n) \tag{44.13}$$

were guaranteed to decrease the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\left(-\frac{1}{2}\sum_{m,n} J_{mn}\bar{x}_m\bar{x}_n - \sum_n h_n\bar{x}_n\right) - \sum_n H_2^{(e)}(q_n). \tag{44.14}$$

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

$$\beta \tilde{F}(\mathbf{x}) = -\beta \frac{1}{2} \mathbf{x}^{\mathsf{T}} \mathbf{W} \mathbf{x} - \sum_i H_2^{(e)}[(1 + x_i)/2]. \tag{44.15}$$

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 44.3:$^{A2}$ 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.

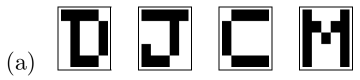
Exercise 44.4:$^{A2}$ Show by constructing an example that if a Hopfield network is updated synchronously that, from some initial conditions, it may fail to converge to a fixed point.

## 44.5 The associative memory in action

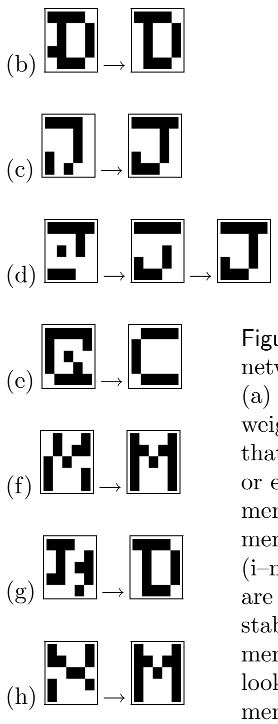
Figure 44.3 shows the dynamics of a 25-unit binary Hopfield network that has learnt four patterns by Hebbian learning. The four patterns are displayed as five by five binary images in figure 44.3(a). Parts (b-m) show the state of the network, iteration by iteration, all 25 units being updated asynchronously 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 44.3 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!



```
 .  0  0  0  0 -2  2 -2  2  2 -2  0  0  0  2  0  0 -2  0  2  2  0  0 -2 -2
 0  .  4  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  4  2 -2
 0  4  .  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  4  2 -2
 0  4  4  .  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  4  2 -2
 0  0  0  0  .  2 -2 -2  2 -2  2 -4  0  0 -2  4 -4 -2  0 -2  2  0  0 -2  2
-2 -2 -2 -2  2  .  0  0  0  0  4 -2  2 -2  0  2 -2  0 -2  0  0 -2 -2  0  4
 2 -2 -2 -2 -2  0  .  0  0  4  0  2  2 -2  4 -2  2  0 -2  4  0 -2 -2  0  0
-2 -2 -2 -2 -2  0  0  .  0  0  0  2  2  2  0 -2  2  4  2  0  0 -2 -2  0  0
 2 -2 -2 -2  2  0  0  0  .  0  0 -2  2  2  0  2 -2  0  2  0  4 -2 -2 -4  0
 2 -2 -2 -2 -2  0  4  0  0  .  0  2  2 -2  4 -2  2  0 -2  4  0 -2 -2  0  0
-2 -2 -2 -2  2  4  0  0  0  0  . -2  2 -2  0  2 -2  0 -2  0  0 -2 -2  0  4
 0  0  0  0 -4 -2  2  2 -2  2 -2  .  0  0  2 -4  4  2  0  2 -2  0  0  2 -2
 0 -4 -4 -4  0  2  2  2  2  2  2  0  .  0  2  0  0  2  0  2  2 -4 -4 -2  2
 0  0  0  0  0 -2 -2  2  2 -2 -2  0  0  . -2  0  0  2  4 -2  2  0  0 -2 -2
 2 -2 -2 -2 -2  0  4  0  0  4  0  2  2 -2  . -2  2  0 -2  4  0 -2 -2  0  0
 0  0  0  0  4  2 -2 -2  2 -2  2 -4  0  0 -2  . -4 -2  0 -2  2  0  0 -2  2
 0  0  0  0 -4 -2  2  2 -2  2 -2  4  0  0  2 -4  .  2  0  2 -2  0  0  2 -2
-2 -2 -2 -2 -2  0  0  4  0  0  0  2  2  2  0 -2  2  .  2  0  0 -2 -2  0  0
 0  0  0  0  0 -2 -2  2  2 -2 -2  0  0  4 -2  0  0  2  . -2  2  0  0 -2 -2
 2 -2 -2 -2 -2  0  4  0  0  4  0  2  2 -2  4 -2  2  0 -2  .  0 -2 -2  0  0
 2 -2 -2 -2  2  0  0  0  4  0  0 -2  2  2  0  2 -2  0  2  0  . -2 -2 -4  0
 0  4  4  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  .  4  2 -2
 0  4  4  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  .  2 -2
-2  2  2  2 -2  0  0  0 -4  0  0  2 -2 -2  0 -2  2  0 -2  0 -4  2  2  .  0
-2 -2 -2 -2  2  4  0  0  0  0  4 -2  2 -2  0  2 -2  0 -2  0  0 -2 -2  0  .
```



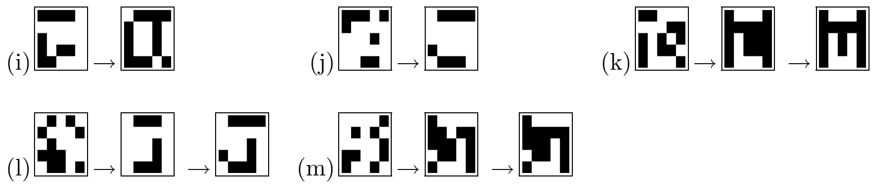
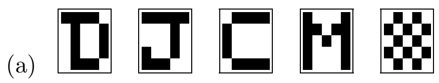


Figure 44.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to stable states other than the four memories; 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).



```
 .-1  1-1  1  x  x-3  3  x  x-1  1-1  x-1  1-3  x  1  3-1  1  x-1
-1  .  3  5-1-1-3-1-3-1-3  1  x  1-3  1-1-1-1-1-3  5  3  3-3
 1  3  .  3  1-3-1  x-1-3-1-1  x-1-1-1  1-3  1-3-1  3  5  1-1
-1  5  3  .-1-1-3-1-3-1-3  1-5  1-3  1-1-1-1-1-3  5  x  3-3
 1-1  1-1  .  1-1-3  x  x  3-5  1-1-1  3  x-3  1-3  3-1  1-3  3
 x-1-3-1  1  .-1  1-1  1  3-1  1-1-1  3-3  1  x  1  x-1-3  1  3
 x-3-1-3-1-1  .-1  1  3  1  1  3-3  5-3  3-1-1  x  1-3-1-1  1
-3-1  x-1-3  1-1  .-1  1-1  3  1  x-1-1  1  5  1  1-1  x-3  1-1
 3-3-1-3  x-1  1-1  .-1  1-3  3  1  1  1-1-1  3-1  5-3-1  x  1
 x-1-3-1  x  1  3  1-1  .-1  3  1-1  3-1  x  1-3  5-1-1-3  1-1
 x-3-1-3  3  3  1-1  1-1  .-3  3-3  1  1-1-1-1-1  1-3-1-1  5
-1  1-1  1-5-1  1  3-3  3-3  .-1  1  1-3  3  x-1  3-3  1-1  3-3
 1  x  x-5  1  1  3  1  3  1  3-1  .-1  3-1  1  1  1  1  3-5-3-3  3
-1  1-1  1-1-1-3  x  1-1-3  1-1  .  x  1-1  3  3-1  1  1-1-1-3
 x-3-1-3-1-1  5-1  1  3  1  1  3  x  .  x  3-1-1  3  1-3-1-1  1
-1  1-1  1  3  3-3-1  1-1  1-3-1  1  x  .-5-1-1-1  1  1-1-1  1
 1-1  1-1  x-3  3  1-1  x-1  3  1-1  3-5  .  1  1  1-1-1  1  1-1
-3-1-3-1-3  1-1  5-1  1-1  x  1  3-1-1  1  .  1  1-1-1-3  1-1
 x-1  1-1  1  x-1  1  3-3-1-1  1  3-1-1  1  1  .-3  3-1  1-3-1
 1-1-3-1-3  1  x  1-1  5-1  3  1-1  3-1  1  1-3  .  x-1-3  1-1
 3-3-1-3  3  x  1-1  5-1  1-3  3  1  1  1-1-1  3  x  .-3-1-5  1
-1  5  3  5-1-1-3  x-3-1-3  1-5  1-3  1-1-1-1-1-3  .  3  x-3
 1  3  5  x  1-3-1-3-1-3-1-1-3-1-1-1  1-3  1-3-1  3  .  1-1
 x  3  1  3-3  1-1  1  x  1-1  3-3-1-1-1  1  1-3  1-5  x  1  .-1
-1-3-1-3  3  3  1-1  1-1  5-3  3-3  1  1-1-1-1-1  1-3-1-1  .
```

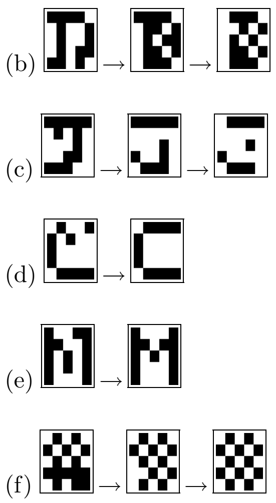


Figure 44.4. Hopfield network storing five memories, and suffering deletion of 26 of its 300 weights. (a) The five memories, and the weights of the network, with deleted weights shown by 'x'. (b–f) Initial states that differ by three random bits from a memory: some are restored, but some converge to other states.

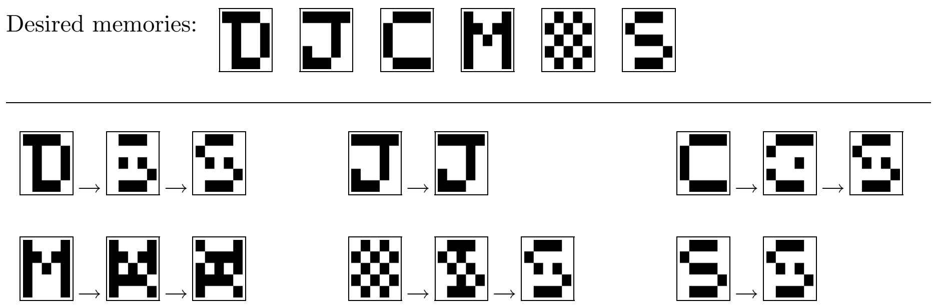


Figure 44.5. An overloaded Hopfield network trained on six memories, most of which are not stable.

| Desired memories | | Attracting stable states | |
|---|---|---|---|
| moscow______russia | | moscow______russia | |
| lima__________peru | | lima__________peru | |
| london_____england | | londog_____englard | (1) |
| tokyo________japan | | tonco________japan | (1) |
| edinburgh_scotland | | edinburgh_scotland | |
| ottawa______canada | $\rightarrow$ **W** $\rightarrow$ | | (2) |
| oslo________norway | | oslo________norway | |
| stockholm___sweden | | stockholm___sweden | |
| paris_______france | | paris_______france | |
| | | wzkmhewn__xqwqwpoq | (3) |
| | | paris_______sweden | (4) |
| | | ecnarf_______sirap | (4) |

Figure 44.6. 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 44.4(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 than before: figure 44.4(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. When we add a sixth pattern, as shown in figure 44.5, only one of the patterns is stable; the others all flow into one of two spurious stable states.

## 44.6 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

$$a_i(t) = \sum_j w_{ij} x_j(t). \tag{44.16}$$

The neuron's response to its activation is assumed to be mediated by the differential equation:

$$\frac{d}{dt} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)), \tag{44.17}$$

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 $\tau$.

Now, here is the nice result: as long as the weight matrix is symmetric, this system has the variational free energy (44.15) as its Lyapunov function.

Exercise 44.5:$^{B1}$ By computing $\frac{d}{dt}\tilde{F}$, prove that the variational free energy $\tilde{F}(\mathbf{x})$ is a Lyapunov function for the continuous–time Hopfield network.

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 the case of the continuous-time continuous Hopfield network, 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 $\tilde{F}(\mathbf{x})$.

## 44.7 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 44.6). A list of desired memories is encoded into a set of weights $\mathbf{W}$ using the Hebb rule of equation (44.5), 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 as 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 now study these alternative definitions of the capacity.

## *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

$$a_i = \sum_j w_{ij} x_j^{(n)}, \tag{44.18}$$

where the weights are, for $i \neq j$,

$$w_{ij} = x_i^{(n)} x_j^{(n)} + \sum_{m \neq n} x_i^{(m)} x_j^{(m)}. \tag{44.19}$$

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

$$\begin{aligned} a_i &= \sum_{j \neq i} x_i^{(n)} x_j^{(n)} x_j^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)} & (44.20) \\ &= (I-1) x_i^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)}. & (44.21) \end{aligned}$$

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

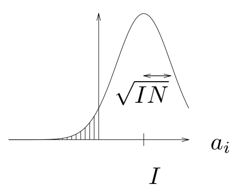


Figure 44.7. The probability density of the activation $a_i$ in the case $x_i^{(n)}=1$; the probability that bit $i$ becomes flipped is the area of the tail.

$$P(i \text{ unstable}) = \Phi\left(-\frac{I}{\sqrt{IN}}\right) = \Phi\left(-\frac{1}{\sqrt{N/I}}\right), \tag{44.22}$$

where

$$\Phi(z) \equiv \int_{-\infty}^{z} dz\, \tfrac{1}{\sqrt{2\pi}} e^{-z^2/2}. \tag{44.23}$$

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 44.6:$^{B2}$ 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 $\epsilon$. Using the approximation to the error function for large $z$,

$$\Phi(-z) \simeq \frac{1}{\sqrt{2\pi}} \frac{e^{-z^2/2}}{z}, \tag{44.24}$$

show that the maximum number of patterns that can be stored, $N_{\max}$, is

$$N_{\max} \simeq \frac{I}{4\log I + 2\log(1/\epsilon)}. \tag{44.25}$$

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 (44.22) 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 each desired memory. For the limit of large $I$, Amit *et al.* (1985) have used methods from statistical physics

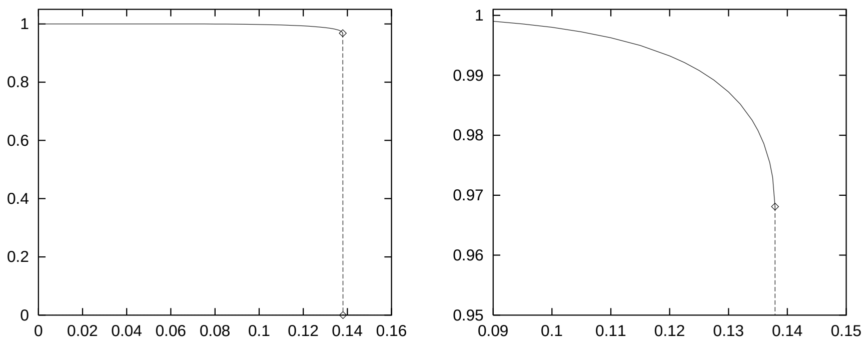


Figure 44.8. Overlap between a desired memory and the stable state nearest to it as a function of the loading fraction $N/I$. The overlap is defined to be the scaled inner product $\sum_i x_i x_i^{(n)}/I$, which is 1 when recall is perfect and zero when the stable state has 50% of the bits flipped. There 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

$$N_{\text{crit}} = 0.138I. \tag{44.26}$$

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 44.8 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

$$0.138I^2 \times (1 - H_2(0.016)) = 0.122I^2 \text{ bits.} \tag{44.27}$$

Since there are $I^2/2$ weights in the network, we can also express the capacity as *0.24 bits per weight.*

## 44.8 Improving on the capacity of the Hebb rule

The capacities discussed in the previous section are the capacities of the Hopfield network whose weights are set using the Hebbian learning rule. We can do better than the Hebb rule by defining an objective function that measures how well the network stores all the memories, and minimizing it.

For an associative memory to be useful, it must be able to correct at least one flipped bit. Let's make an objective function that measures whether flipped bits tend to be restored correctly. Our intention is that, for every neuron $i$ in the network, the weights to that neuron should satisfy this rule:

> for every pattern $\mathbf{x}^{(n)}$, if the neurons other than $i$ are set correctly to $x_j = x_j^{(n)}$, then the activation of neuron $i$ should be such that its preferred output is $x_i = x_i^{(n)}$.

Is this rule a familiar idea? Yes, it is precisely what we wanted the single neuron of chapter 41 to do! Each pattern $\mathbf{x}^{(n)}$ defines an input, target pair for the single neuron $i$. And it defines an input, target pair for all the other neurons too.

So, just as we defined an objective function (41.12) for the training of a single neuron as a classifier, we can define

$$G(\mathbf{W}) = -\sum_i \sum_n t_i^{(n)} \log y_i^{(n)} + (1 - t_i^{(n)}) \log(1 - y_i^{(n)}) \tag{44.28}$$

where

$$t_i^{(n)} = \begin{cases} 1 & x_i^{(n)} = 1 \\ 0 & x_i^{(n)} = -1 \end{cases} \tag{44.29}$$

and

$$y_i^{(n)} = \frac{1}{1 + \exp(-a_i^{(n)})}, \text{ where } a_i^{(n)} = \sum w_{ij} x_j^{(n)} . \tag{44.30}$$

We can then steal the algorithm (41.1) which we wrote for the single neuron. The convenient syntax of `Octave` requires very few changes; the extra lines enforce the constraints that the self-weights $w_{ii}$ should all be zero and that the weight matrix should be symmetrical ($w_{ij} = w_{ji}$) – see algorithm 44.1.

As expected, this learning algorithm does a better job than the one-shot Hebbian learning rule. When the six patterns of figure 44.5, which cannot be memorized by the Hebb rule, are learned using algorithm 44.1, all six patterns become stable states.

## 44.9 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.

```
w = data' * data ;  # initialize the weights using Hebb rule

for l = 1:L         # loop L times

        for i=1:I       #
          w(i,i) = 0 ;  #  ensure the self-weights are zero.
        end             #

        a  = x * w      ;    # compute all activations
        y  = sigmoid(a) ;    # compute all outputs
        e  = t - y      ;    # compute all errors
        gw = x' * e     ;    # compute the gradients
        gw = gw + gw'   ;    # symmetrize gradients

        w  = w + eta * ( gw - alpha * w ) ;  # make step

endfor
```

Algorithm 44.1. `Octave` source code for optimizing the weights of a Hopfield network, so that it works as an associative memory. *c.f.* algorithm 41.1. The `data` matrix has $I$ columns and $N$ rows.

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 44.9(a).

The weights in the Hopfield network play two roles. First, they must define an energy function which is minimized only when the state of the network represents a **valid** tour. A valid state is one that 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 44.9(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 dis-

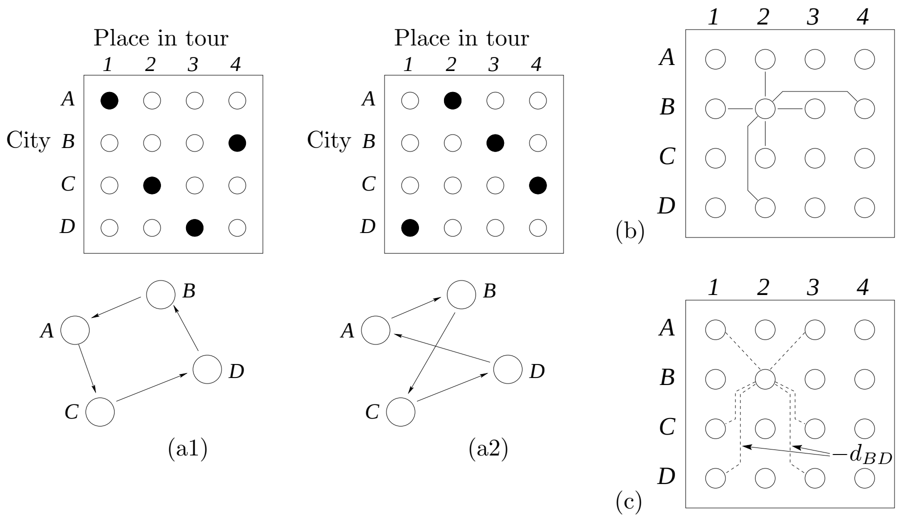
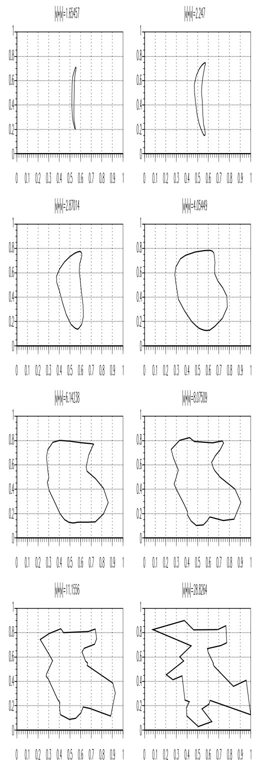


Figure 44.9. Hopfield network for solving a travelling salesman problem with $K = 4$ cities. (a1,2) Two solution states of the 16-neuron network, with activites represented by black = 1, white = 0; and the tours corresponding to these network states. (b) The negative weights between node $B2$ and other nodes; these weights enforce validity of a tour. (c) The negative weights that embody the distance objective function.

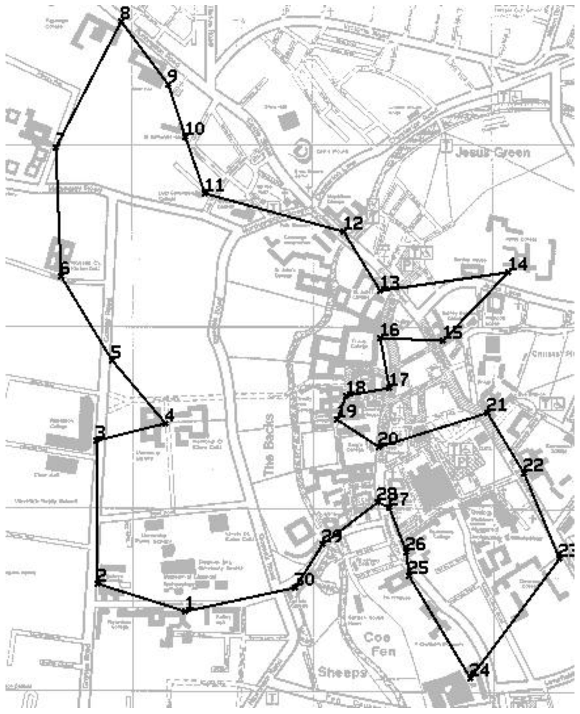
tance. This can be done by putting negative weights proportional to the appropriate distances between the nodes in adjacent columns. For example, between the $B$ and $D$ nodes in adjacent columns, the weight would be $-d_{BD}$. The negative weights that are connected to neuron $B2$ are shown in figure 44.9(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 $\beta$ of the neurons in the network from 0 to $\infty$, 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 44.10(a).

A solution to the 'travelling scholar problem' found by Aiyer using a con-



(a)



(b)

Figure 44.10. (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).

tinuous Hopfield network is shown in figure 44.10(b).

## 44.10 Exercises

Solutions on p.540

Exercise 44.7:$^{C3}$ Hopfield network as a collection of binary classifiers This exercise explores the link between unsupervised networks to supervised networks. If a Hopfield network's desired memories are all attracting stable states, then every neuron in the network has weights going to it that solve a classification problem personal to that neuron. Take the set of memories and write them in the form $\mathbf{x}'^{(n)}, x_i^{(n)}$, where $\mathbf{x}'$ denotes all the components $x_{i'}$ for all $i' \neq i$, and let $\mathbf{w}'$ denote the vector of weights $w_{ii'}$, for $i' \neq i$.

Using what we know about the capacity of the single neuron, show that it is impossible to store more than $2I$ memories in a Hopfield network of $I$ neurons.

# *Solutions to Chapter 44's exercises*

Solution to exercise 44.3 (p.529): 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 44.4 (p.529): 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.

# 45 From Hopfield networks to Boltzmann machines

## 45.1 The Boltzmann machine

We have noticed that the binary Hopfield network minimizes an energy function

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x} \tag{45.1}$$

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,

$$P(\mathbf{x}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})}\exp[-E(\mathbf{x})] = \frac{1}{Z(\mathbf{W})}\exp\left[\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x}\right]. \tag{45.2}$$

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* (Hinton and Sejnowski 1986) has the following activity rule:

**Activity rule of Boltzmann machine:** after computing the activation $a_n$,

$$\begin{array}{l} \text{set } x_n = +1 \text{ with probability } \dfrac{1}{1+e^{-2a}} \\ \text{else set } x_n = -1. \end{array} \tag{45.3}$$

This rule implements Gibbs sampling for the probability distribution (45.2).

### *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

$$P(\mathbf{x}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})}\exp\left[\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x}\right] \tag{45.4}$$

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:

$$P(\mathbf{W}|\{\mathbf{x}^{(n)}\}_1^N) = \frac{\left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)}|\mathbf{W})\right] P(\mathbf{W})}{P(\{\mathbf{x}^{(n)}\}_1^N)}. \tag{45.5}$$

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. It is convenient to differentiate the logarithm of the likelihood,

$$\log \left[ \prod_{n=1}^{N} P(\mathbf{x}^{(n)}|\mathbf{W}) \right] = \sum_{n=1}^{N} \left[ \frac{1}{2} \mathbf{x}^{(n)\mathsf{T}} \mathbf{W} \mathbf{x}^{(n)} - \log Z(\mathbf{W}) \right]. \tag{45.6}$$

We differentiate with respect to $w_{ij}$, bearing in mind that $\mathbf{W}$ is defined to be symmetric with $w_{ji} = w_{ij}$.

Exercise 45.1:$^{A2}$ Show that the derivative of $\log Z(\mathbf{W})$ with respect to $w_{ij}$ is

$$\frac{\partial}{\partial w_{ij}} \log Z(\mathbf{W}) = \sum_{\mathbf{x}} x_i x_j P(\mathbf{x}|\mathbf{W}) = \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})}. \tag{45.7}$$

The derivative of the log likelihood is therefore:

$$\begin{aligned} \frac{\partial}{\partial w_{ij}} \log P(\{\mathbf{x}^{(n)}\}_1^N\}|\mathbf{W}) &= \sum_{n=1}^{N} \left[ x_i^{(n)} x_j^{(n)} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \right] & (45.8) \\ &= N \left[ \langle x_i x_j \rangle_{\text{Data}} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \right]. & (45.9) \end{aligned}$$

This gradient is thus proportional to a difference of two terms. The first term is the *empirical* correlation between $x_i$ and $x_j$,

$$\langle x_i x_j \rangle_{\text{Data}} \equiv \frac{1}{N} \sum_{n=1}^{N} \left[ x_i^{(n)} x_j^{(n)} \right], \tag{45.10}$$

and the second term is the correlation between $x_i$ and $x_j$ under the current model,

$$\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{W})} \equiv \sum_{\mathbf{x}} x_i x_j P(\mathbf{x}|\mathbf{W}). \tag{45.11}$$

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 (45.3), 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 $\eta$, then, after one iteration, the weights will be

$$w_{ij} = \eta \sum_{n=1}^{N} \left[ x_i^{(n)} x_j^{(n)} \right], \tag{45.12}$$

precisely the value of the weights given by the Hebb rule, equation (16.5), with which we trained the Hopfield network.

## *Interpretation of Boltzmann machine learning*

One way of viewing the two terms in the gradient (45.9) is as 'waking' and 'sleeping' rules. While the network is 'awake', it measures the correlation

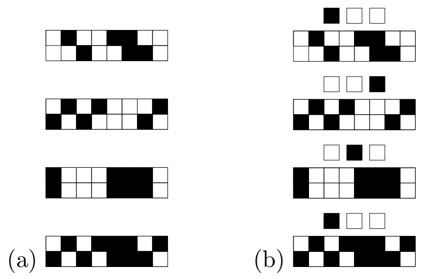


Figure 45.1. The 'shifter' ensembles. (a) Four samples from the plain shifter ensemble. (b) Four corresponding samples from the labelled shifter ensemble.

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 (45.4), 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 second-order statistics of an ensemble $P(\mathbf{x})$ are the expected values $\langle x_i x_j \rangle$ of all the pairwise products $x_i x_j$.] 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 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 45.1(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. The signals from the two eyes 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. The correlation between one pixel and any of the three pixels above it 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 45.1(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 label neuron and any image neuron is zero. We need models that can capture higher-order statistics of an environment.

So, how can we develop such models? One idea might be to create models that directly capture higher-order correlations, such as:

$$P'(\mathbf{x}|\mathbf{W},\mathbf{V},\ldots) = \frac{1}{Z'} \exp\left( \frac{1}{2}\sum_{ij} w_{ij} x_i x_j + \frac{1}{6}\sum_{ij} v_{ijk} x_i x_j x_k + \ldots \right). \quad (45.13)$$

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 45.2:$^{B2}$ 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 $128 \times 128$ 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.

## 45.2 Boltzmann machine with hidden units

We now add *hidden neurons* to our stochastic model. These are neurons that do not correspond to observed variables; they are free to play any role in the probabilistic model defined by equation (45.4). They might actually take on interpretable roles, effectively performing 'feature extraction'.

### *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 $\mathbf{y} \equiv (\mathbf{x}, \mathbf{h})$. The state of the network when the visible neurons are clamped in state $\mathbf{x}^{(n)}$ is $\mathbf{y}^{(n)} \equiv (\mathbf{x}^{(n)}, \mathbf{h})$.

The likelihood of $\mathbf{W}$ given a single data example $\mathbf{x}^{(n)}$ is

$$P(\mathbf{x}^{(n)}|\mathbf{W}) = \sum_{\mathbf{h}} P(\mathbf{x}^{(n)}, \mathbf{h}|\mathbf{W}) = \sum_{\mathbf{h}} \frac{1}{Z(\mathbf{W})} \exp\left[\frac{1}{2}[\mathbf{y}^{(n)}]^{\mathsf{T}}\mathbf{W}\mathbf{y}^{(n)}\right], \quad (45.14)$$

where

$$Z(\mathbf{W}) = \sum_{\mathbf{x},\mathbf{h}} \exp\left[\frac{1}{2}\mathbf{y}^{\mathsf{T}}\mathbf{W}\mathbf{y}\right]. \quad (45.15)$$

Equation (45.14) may also be written

$$P(\mathbf{x}^{(n)}|\mathbf{W}) = \frac{Z_{\mathbf{x}^{(n)}}(\mathbf{W})}{Z(\mathbf{W})} \quad (45.16)$$

where

$$Z_{\mathbf{x}^{(n)}}(\mathbf{W}) = \sum_{\mathbf{h}} \exp\left[\frac{1}{2}[\mathbf{y}^{(n)}]^{\mathsf{T}}\mathbf{W}\mathbf{y}^{(n)}\right]. \quad (45.17)$$

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,

$$\frac{\partial}{\partial w_{ij}} \log P(\{\mathbf{x}^{(n)}\}_1^N|\mathbf{W}) = \sum_n \left\{\langle y_i y_j\rangle_{P(\mathbf{h}|\mathbf{x}^{(n)},\mathbf{W})} - \langle y_i y_j\rangle_{P(\mathbf{x},\mathbf{h}|\mathbf{W})}\right\} \quad (45.18)$$

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; Hinton 2000).

In my opinion, Hinton and Sejnowski's conception of the Boltzmann machine is an important step towards understanding how the brain works.

# 46

# *Supervised learning in multilayer networks*

## 46.1 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:

$$\text{Hidden layer:} \quad a_j^{(1)} = \sum_l w_{jl}^{(1)} x_l + \theta_j^{(1)}; \quad h_j = f^{(1)}(a_j^{(1)}) \tag{46.1}$$

$$\text{Output layer:} \quad a_i^{(2)} = \sum_j w_{ij}^{(2)} h_j + \theta_i^{(2)}; \quad y_i = f^{(2)}(a_i^{(2)}) \tag{46.2}$$

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' $\theta$ 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 46.1(a). The entire network is shown in figure 46.1(b).

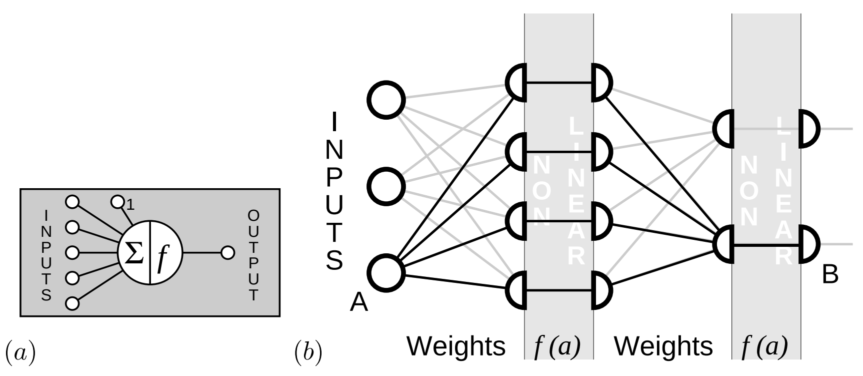


Figure 46.1. Graphical representation of a neural network. (*a*) A single neuron. (*b*) A two layer network with 3 inputs, four hidden units and 2 outputs. The weights from input A to the hidden layer have been highlighted, and the weights from the hidden layer to output B.

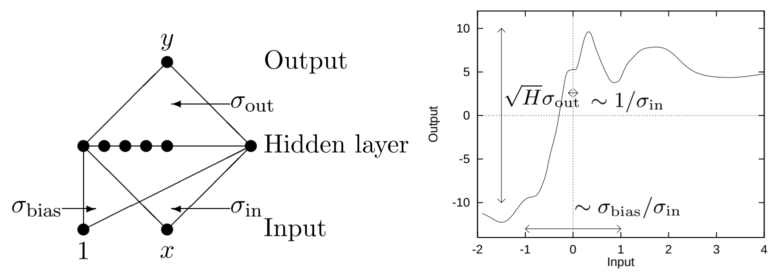


Figure 46.2. Properties of a function produced by a random network. The vertical scale of a typical function produced by the network with random weights is of order $\sqrt{H}\sigma_{\rm out}$; the horizontal range in which the function varies significantly is of order $\sigma_{\rm bias}/\sigma_{\rm in}$; 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 $\sigma_{\rm out} = 0.5$.

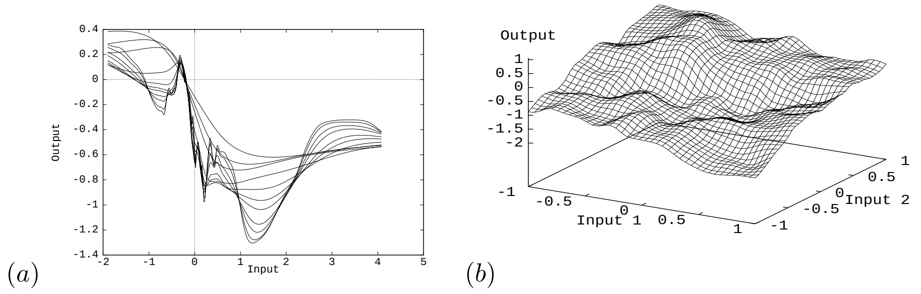


Figure 46.3. Samples from the prior of (*a*) a one input network; and (*b*) a two input network. Figure 46.3a shows one function for each of a sequence of values of $\sigma_{\rm bias}$ and $\sigma_{\rm in}$. with $H = 400$, $\sigma^w_{\rm out} = 0.05$. For each graph the parameter $\sigma^w_{\rm bias}$ 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 $\{H, \sigma^w_{\rm in}, \sigma^w_{\rm bias}, \sigma^w_{\rm out}\} = \{400, 8.0, 8.0, 0.05\}$.

*What sorts of functions can these networks implement?*

Just as we explored the weight space of the single neuron in chapter 41, examining the functions it could produce, let us explore the weight space of a multi-layer network. In figure 46.2 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 $\sigma_{\text{bias}}$; the input to hidden weights $w_{jl}^{(1)}$ to random values with standard deviation $\sigma_{\text{in}}$; and the bias and output weights $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values with standard deviation $\sigma_{\text{out}}$.

The sort of functions that we obtain depend on the values of $\sigma_{\text{bias}}$, $\sigma_{\text{in}}$ and $\sigma_{\text{out}}$ (Neal 1996) (see captions of figures 46.2 & 46.3). 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 46.3b 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.

## 46.2 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.,

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_n \sum_i \left( t_i^{(n)} - y_i(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 . \tag{46.3}$$

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:

$$M(\mathbf{w}) = \beta E_D + \alpha E_W \tag{46.4}$$

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).

## 46.3 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 43.]

The error function is interpreted as minus the log likelihood for a noise model:

$$P(D|\mathbf{w},\beta,\mathcal{H}) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D). \tag{46.5}$$

Thus, the use of the sum-squared error $E_D$ (46.3) corresponds to an assumption of Gaussian noise on the target variables, and the parameter $\beta$ 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:

$$P(\mathbf{w}|\alpha,\mathcal{H}) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{46.6}$$

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 (46.5), and the prior (46.6).

The objective function $M(\mathbf{w})$ then corresponds to the *inference* of the parameters $\mathbf{w}$, given the data:

$$\begin{aligned} P(\mathbf{w}|D,\alpha,\beta,\mathcal{H}) &= \frac{P(D|\mathbf{w},\beta,\mathcal{H})P(\mathbf{w}|\alpha,\mathcal{H})}{P(D|\alpha,\beta,\mathcal{H})} & (46.7)\\ &= \frac{1}{Z_M}\exp(-M(\mathbf{w})). & (46.8)\end{aligned}$$

The $\mathbf{w}$ found by (locally) minimizing $M(\mathbf{w})$ is then interpreted as the (locally) most probable parameter vector, $\mathbf{w}_{\text{MP}}$.

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 (46.1) and (46.2), with $f^{(2)}(a) = 1/(1+e^{-a})$. The error function $\beta E_D$ is replaced by the negative log likelihood:

$$G(\mathbf{w}) = -\left[\sum_n t^{(n)} \log y(\mathbf{x}^{(n)};\mathbf{w}) + (1-t^{(n)})\log(1-y(\mathbf{x}^{(n)};\mathbf{w}))\right]. \tag{46.9}$$

The total objective function is then $M = G + \alpha E_W$. Note that this includes no parameter $\beta$.

*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 (46.2) is replaced by:

$$y_i = \frac{e^{a_i}}{\sum_{i'} e^{a_{i'}}}. \tag{46.10}$$

The negative log likelihood in this case is

$$G(\mathbf{w}) = -\sum_n \sum_i t_i^{(n)} \log y_i(\mathbf{x}^{(n)}; \mathbf{w}). \tag{46.11}$$

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 (46.8). A variety of useful results can be built on this interpretation.

## 46.4 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 that 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).

## 46.5 Benefits of the Bayesian approach to supervised feedforward neural networks

From the statistical perspective, supervised neural networks are nothing more than 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 46.4. Consider a control parameter which influences the complexity of a model, for example a regularization constant $\alpha$ (weight decay parameter). As the control parameter is varied to increase the complexity of the model (descending from figure 46.4*a-c* and going from left to right across figure 46.4*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 multidimensional regression problem.

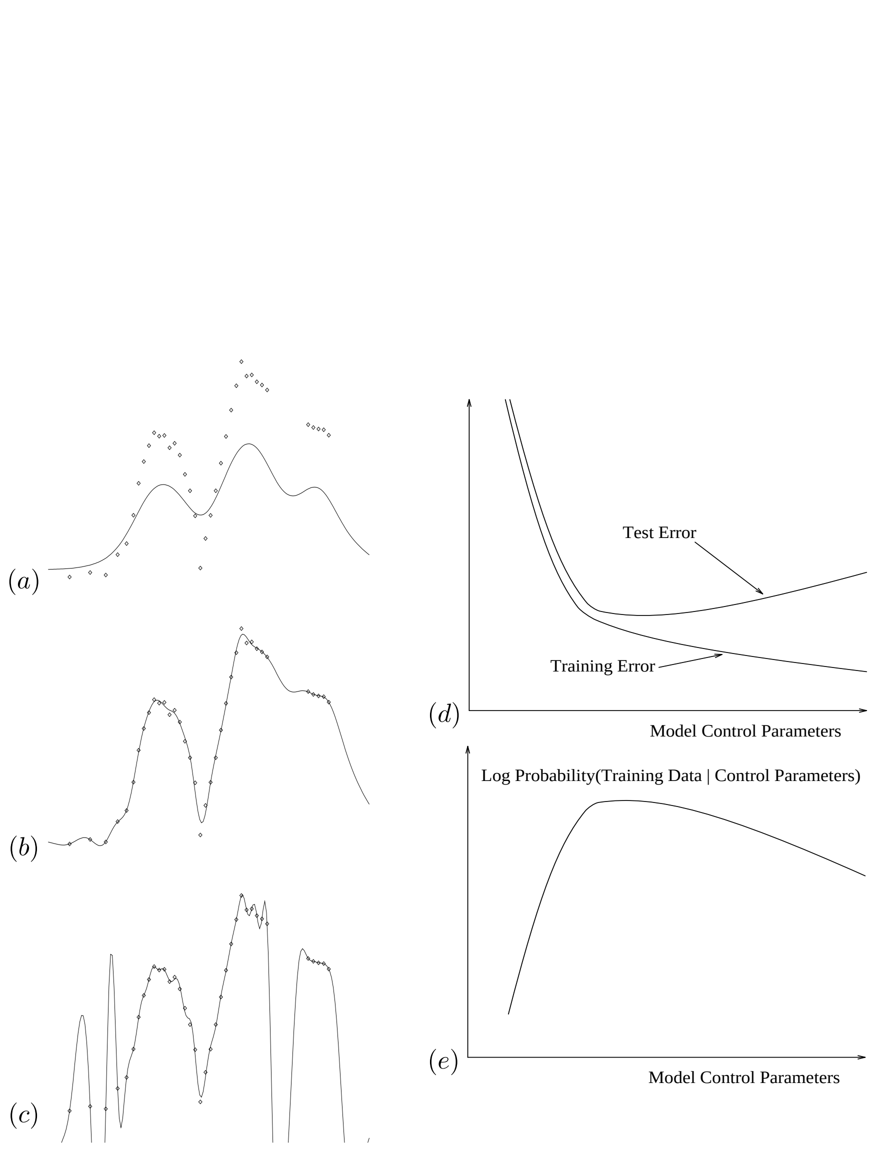


Figure 46.4. Optimization of model complexity. Figures *a*-*c* show a radial basis function model interpolating a simple data set with one input variable and one output variable. As the regularization constant is varied to increase the complexity of the model (from *a* to *c*), the interpolant is able 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.

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 46.4*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 $\alpha$ that maximizes this quantity is displayed in figure 46.4*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 43.

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.

47

# *Gaussian processes* *

A review of work by Neal, Williams, Rasmussen and Gibbs on Gaussian processes as a replacement for supervised neural networks.

48

# Deconvolution *

## 48.1 Traditional image reconstruction methods

*Optimal linear filters*

In many imaging problems, the data measurements $\{d_n\}$ are linearly related to the underlying image $\mathbf{f}$:

$$d_n = \sum_j R_{nj} f_j + \nu_n. \tag{48.1}$$

The vector $\nu$ 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 $n = 1, \ldots, N$ to run over data measurements, and the subscripts $k, k' = 1, \ldots, K$ 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 $\nu$ is Gaussian and independent, with a known standard deviation $\sigma_\nu$.

$$P(\mathbf{d}|\mathbf{f}, \sigma_\nu, \mathcal{H}) = \frac{1}{(2\pi\sigma_\nu^2)^{N/2}} \exp\left( -\sum_m \left(d_n - \textstyle\sum_k R_{nk} f_k\right)^2 \Big/ (2\sigma_\nu^2) \right) \tag{48.2}$$

We assume that the prior probability of the image is also Gaussian, with a standard deviation $\sigma_f$.

$$P(\mathbf{f}|\sigma_f, \mathcal{H}) = \frac{\det^{-\frac{1}{2}} \mathbf{C}}{(2\pi\sigma_f^2)^{k/2}} \exp\left( -\sum_{k,k'} f_k C_{kk'} f'_k / (2\sigma_f^2) \right) \tag{48.3}$$

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 $\mathbf{C} = [\mathbf{G}\mathbf{G}^{\mathsf{T}}]^{-1}$, 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 $\mathbf{d}$.

$$P(\mathbf{f}|\mathbf{d}, \sigma_\nu, \sigma_f, \mathcal{H}) = \frac{P(\mathbf{d}|\mathbf{f}, \sigma_\nu, \mathcal{H}) P(\mathbf{f}|\sigma_f, \mathcal{H}))}{P(\mathbf{d}|\sigma_\nu, \sigma_f, \mathcal{H})} \tag{48.4}$$

In words,

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}. \tag{48.5}$$

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 $\sigma_\nu$ and $\sigma_f$, 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*, $\mathbf{f}_{\text{MP}}$, and its covariance matrix:

$$\Sigma_{\mathbf{f}|\mathbf{d}} \equiv \left[-\nabla\nabla \log P(\mathbf{f}|\mathbf{d}, \sigma_\nu, \sigma_f, \mathcal{H})\right]^{-1}, \tag{48.6}$$

which defines the joint error bars on $\mathbf{f}$. In this equation, the symbol $\nabla$ denotes differentiation with respect to the parameters $\mathbf{f}$. We can find $\mathbf{f}_{\text{MP}}$ by differentiating the log of the posterior, and solving for the derivative being zero. We obtain:

$$\mathbf{f}_{\text{MP}} = \left[\mathbf{R}^{\mathsf{T}}\mathbf{R} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}\right]^{-1} \mathbf{R}^{\mathsf{T}}\mathbf{d}. \tag{48.7}$$

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 '$\mathbf{R}^{-1}$' $= \left[\mathbf{R}^{\mathsf{T}}\mathbf{R}\right]^{-1} \mathbf{R}^{\mathsf{T}}$. 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:

$$\text{Optimal linear filter} = \mathbf{C}^{-1}\mathbf{R}^{\mathsf{T}} \left[\mathbf{R}\mathbf{C}^{-1}\mathbf{R}^{\mathsf{T}} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{I}\right]^{-1}. \tag{48.8}$$

### *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:

$$\hat{\mathbf{f}} = \mathbf{W}\mathbf{d}. \tag{48.9}$$

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 $k$, $k'$, $n$ are implicit. The expectation $\langle\cdot\rangle$ is over both the statistics of the random variables $\{\nu_n\}$, 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 $\langle \nu_n \nu_{n'} \rangle = \sigma_\nu^2 \delta_{nn'}$.

$$\begin{aligned} \langle E \rangle &= \frac{1}{2} \left\langle (W_{kn} d_n - f_k)^2 \right\rangle & (48.10) \\ &= \frac{1}{2} \left\langle (W_{kn} R_{nj} f_j - f_k)^2 \right\rangle + \frac{1}{2} W_{kn} W_{kn} \sigma_\nu^2. & (48.11) \end{aligned}$$

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:

$$\mathbf{W}_{\rm opt} = \mathbf{F}\mathbf{R}^{\mathsf T} \left[ \mathbf{R}\mathbf{F}\mathbf{R}^{\mathsf T} + \sigma_\nu^2 \mathbf{I} \right]^{-1}. \qquad (48.12)$$

If we identify $\mathbf{F} = \sigma_f^2 \mathbf{C}^{-1}$, we obtain the optimal linear filter (48.8) 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 are spectacularly poorly matched to the real world. For example, the Gaussian prior (48.3) fails to specify that all pixel intensities in an image are positive.indexpositivity This omission 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 quality of the reconstruction is greatly enhanced.

The 'Classic maximum entropy' model assigns an entropic prior

$$P(\mathbf{f}|\alpha, \mathbf{m}, \mathcal{H}_{\rm Classic}) = \exp(\alpha S(\mathbf{f}, \mathbf{m}))/Z, \qquad (48.13)$$

where

$$S(\mathbf{f}, \mathbf{m}) = \sum_i (f_i \log(m_i/f_i) + f_i - m_i) \qquad (48.14)$$

(Skilling 1989). This model enforces positivity; the parameter $\alpha$ 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.

## 48.2 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 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 limited 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.

# 49

# *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).

## 49.1 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 10 are very simple *directed graphs* (figure 49.2b). And the Ising model of chapter 32 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.

### *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:

$$P(x, y) = P(x)P(y). \tag{49.1}$$

Exercise 49.1: 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:

$$P(x|y) = P(x). \tag{49.2}$$

**Conditional independence.** Two random variables $X$ and $Y$ are *conditionally independent* given $Z$ (written $X \perp Y|Z$) if and only if:

$$P(x, y|z) = P(x|z)P(y|z)\text{for all values of } z. \tag{49.3}$$

Exercise 49.2: Show that we could equivalently define conditional independence in the following ways:

- $X$ and $Y$ are conditionally independent given $Z$ if and only if:

$$P(x|y, z) = P(x|z)\text{for all values of } z. \tag{49.4}$$

- $X$ and $Y$ are conditionally independent given $Z$ if and only if the joint probability of $x$, $y$ and $z$ can be written:

$$P(x, y, z) = P(z)P(x|z)P(y|z). \tag{49.5}$$

  The directed graphical model corresponding to this joint probability factorization is `x <- z -> y`.

Exercise 49.3: If the joint probability of $x$, $y$ and $z$ can be written

$$P(x, y, z) = P(x)P(z|x)P(y|z), \tag{49.6}$$

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 49.1). The variables $\{x_i\}$ are

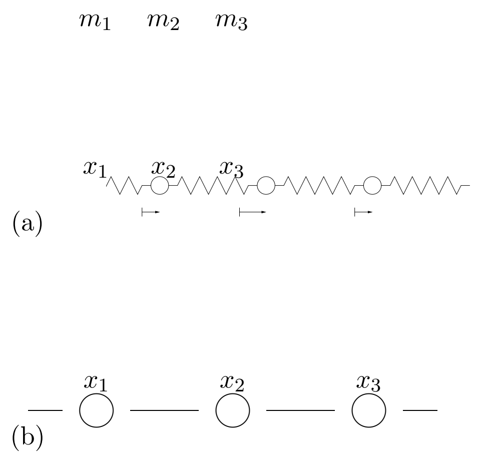


Figure 49.1. Masses and springs. (a) Physical structure. (b) Markov graph.

the displacements of the masses from equilibrium. The potential energy of a configuration is

$$E(\mathbf{x}) = \sum_i \frac{1}{2} k (x_i - x_{i+1})^2, \tag{49.7}$$

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

$$P(\mathbf{x}) = \frac{1}{Z} \exp[-\beta E(\mathbf{x})]. \tag{49.8}$$

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 49.2.

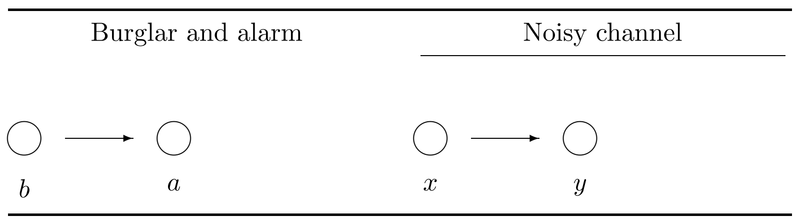


Figure 49.2. Examples of causal relationships.

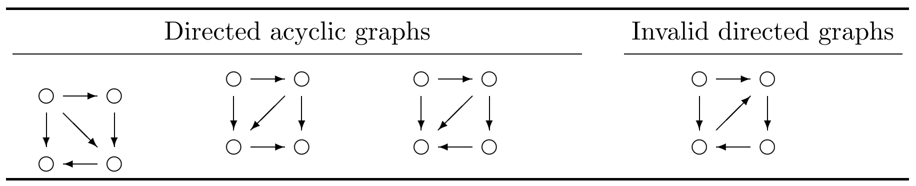


Figure 49.3. Directed graphs. The graphs on the left are directed acyclic graphs. The ones on the right are not, because there are directed cycles in them.

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 49.4.

In figure 49.4(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 49.4(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 49.4(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 49.4. 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 49.4(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 $b$.

## 49.2 Tanner graphs / factor graphs

# Part VI

# Complexity and Tractability *

# 50

# *Valiant, PAC* *

51

# *NP completeness* *

I think there are three meanings for computational complexity.

1. intractability – relates to NP completeness
2. Kolmogorov complexity – relates to description-length assuming you can use arbitrary computer languages
3.

There are several flavours of complexity theory. The flavours that we will study here are concerned with the *tractability* of a problem. We are interested in knowing how the *cost* of solving a problem scales with its size.

To make things precise, we will have to distinguish between *problems* and *instances* of problems. Here are some examples of *problems*.

**Problem 1:** factorize the integer $n$.

**Problem 2:** find the next prime greater than or equal to $n$.

**Problem 3:** Invert a real matrix $\mathbf{M}$.

**Problem 4:** Sort a list of integers $i_1, i_2, \ldots, i_N$.

**Problem 5 – syndrome decoding:** Given a binary matrix $\mathbf{H}$ and a syndrome $\mathbf{z}$, find the binary vector $\mathbf{x}$ of minimum weight such that $\mathbf{H}\mathbf{x} = \mathbf{z} \bmod 2$.

**Problem 6 – Graph colouring:** Given a graph, and $K$ colours, find whether it is possible to colour each vertex in the graph in such a way that no two adjacent vertices have the same colour.

An *instance* of problem 1 is:

Factorize the number $n = 2001$.

An *instance* of problem 2 is:

Find the next prime $\geq 2001$.

An *instance* of problem 3 is:

Invert the matrix $\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$.

An *instance* of problem 4 is:

Sort the list $2001, 2, 0, 1$ into increasing order.

We are interested in the computational time and space requirements of a problems, and how they scale with the size of the problem. For example, we might find it useful to know that

Matrix inversion: an $N \times N$ matrix can be inverted in time proportional to $N^3$.

We may be interested in the computational cost in the *worst case*, the cost of the *typical case*, or possibly, for problems that have approximate solutions, the cost of a solution correct to some tolerance $\epsilon$.

Much attention seems to focus on the cost of the worst case. In practice, I think it could be argued that actually the cost of the typical case is more important. For example, if I am making a cryptosystem that is intended to be very difficult to crack, it's not sufficient to know that cracking it takes exponentially long in the worst case – I need to know that it *always* (or almost always) takes a long time to crack.

For some problems, such as graph colouring, the worst case cost can be much greater than the typical cost of solving randomly-generated instances.

Nevertheless, we will now discuss worst-case tractability.

## *Complexity classes*

It would be nice if we could put every problem into a precise complexity class and make statements like

The cost of matrix inversion scales as $N^3$ (and you can't do it any faster).

However, there are two difficulties. First, it's hard to prove statements like that dangling clause *'you can't do it any faster'* – indeed, clever people are always coming up with better algorithms, and it is now known that

Matrix inversion can be done in $N^{8/3}$. [GET REFERENCE.]

Second, there are flocks of interesting problems, including graph colouring and syndrome decoding, for which no-one has yet been able to find an algorithm that scales better than exponentially in the problem size.

Consequently, researchers are sometimes content simply to establish which *complexity class* a problem belongs to. The three canonical complexity classes for computational requirements (in space or time) are *logarithmic*, *polynomial*, and *exponential*. When specifying the complexity class we need to specify what type of computer we are assuming; the two canonical types of computer are the deterministic Turing machine and the non-deterministic Turing machine, a beast that we will define in a moment.[1]

For example, since matrix inversion and sorting can both be solved on a deterministic Turing machine in a time that is bounded by polynomial in the size of the instance, we say that these two problems are 'in P'.

[1]For a rigorous discussion of this topic, see `http://hissa.nist.gov/dads/terms.html`.

So, what is a non-deterministic Turing machine, and what's this 'NP-completeness' business? There are three ways to describe these machines. A non-deterministic Turing machine can be viewed as a magical computer, which, on demand, is capable of guessing the key to the solution of a task, if such a key exists. For example, if the task is to establish whether a graph can be coloured with $K$ colours, we can imagine solving this problem in exponential time by trying all $|V|^K$ colourings ($V$ being the number of vertices) to see if one can be found that works; a non-deterministic computer is one which is able, by magic, to guess a valid colouring, so that all that needs to be done is check that the guess does indeed work. A second viewpoint is that the computer replicates itself an appropriately large number of times. The third, historical viewpoint, is that the machine is a regular computer that verifies a solution.

A problem is in *NP* if it can be solved in polynomial time on a non-deterministic computer.

S. Cook. "The complexity of theorem-proving procedures", Proceedings of the 3rd Annual Symposium on the Theory of Computing, ACM, New York, 151-158. J. Edmonds. Paths, trees, and flowers.Canadian Journal of Mathematics, 17, pp.449-467. M. R. Garey, D. S. Johnson. Computers and Intractability, W.H.Freeman & Co, 1979. L. Levin. Universal Search Problems, Probl.Pered.Inf. 9(3), 1973. A translation appears in B. A. Trakhtenbrot. A survey of Russian approaches to Perebor (brute-force search) algorithms, Annals of the History of Computing, 6(4):384-400, 1984

An example of a decision problem in NP is:

Decision Problem. Composite Number Instance. Binary encoding of a positive integer n. Language. All instances for which n is composite, i.e., not a prime number.

We can look at this as a language L by simply coding n in log n bits as a binary number, so every binary composite number is in L, and nothing else. We can show this problem is in NP by providing a polynomial time f(x,c) (also known as a "polynomial time proof system" for L). In this case, c can be the binary encoding of a non-trivial factor of n. Since c can be no bigger than n, the size of c is polynomial (at most linear) in the size of n. The function f simply checks to see whether c divides n evenly; if it does, then n is proved to be composite and f returns True. Since division can be done in time polynomial in the size of the operands, Composite Number is in NP.

(b) What is NP-hard?

An NP-hard problem is at least as hard as or harder than any problem in NP. Given a method for solving an NP-hard problem, we can solve any problem in NP with only polynomially more work.

Here's some more terminology. A language L' is polynomial time reducible to a language L if there exists a polynomial time function f(x) from strings to strings such that x is in L' if f(x) is in L. This means that if we can test strings for membership in L in time t, we can use f to test strings for membership in L' in a time polynomial in t. (hint) An example of this would be the relationship between Composite Number and Boolean Circuit Satisfiability. What is NP-complete?

Definition A decision problem L is NP-complete if it is both NP-hard and in NP.

So NP-complete problems are the hardest problems in NP. Since Cook's Theorem was proved in 196?, thousands of problems have been proved to be NP-complete. Probably the most famous example is the Travelling Salesman Problem:

Decision Problem. Travelling Salesman Problem (TSP) Instance. A set S of cities, a function f:S x S -¿ N giving the distances between the cities, and an integer k. Language. The travelling salesman departs from a starting city, goes through each city exactly once, and returns to the start. The language is all instances for which there exists such a tour through the cities of S of length less than or equal to k.

(Theoretical) Computer Science

## 51.1 Kolmogorov Complexity

This is worthless stuff, but let's mention it.

Definition The Kolmogorov Complexity K(x)of a string x is the length of the shortest program that outputs it.

One can actually show that the choice of programming language affects the Kolmogorov Complexity by at most a constant additive factor (i.e. such factor does not depend on x) for all but a finite number of strings x.

Definition A string is said to be incompressible (i.e. random or irregular) if K(x) ¿ length(x)+constant

## 51.2 Decoding linear codes is NP-complete *

# Part VII

# Sparse Graph Codes *

# 52

# *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.

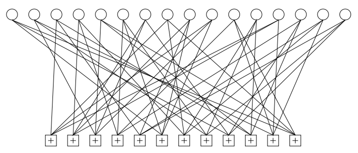
*This document outlines a proposed paper reviewing low-density parity-check codes, repeat-accumulate codes, and turbo codes, explaining how they work, and highlighting the significant steps of the last few years.*

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.

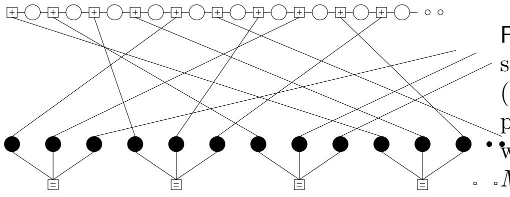
Designing a good error-correcting code is difficult because (a) it is hard to find an explicit set of well-spaced codewords; and (b) for a generic code, decoding, i.e., finding the closest codeword to a received signal, is intractable.

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.

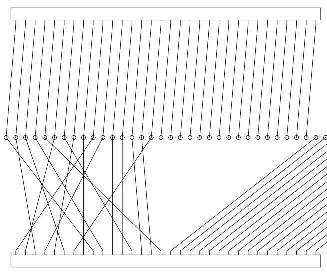
In a **sparse graph code**, the nodes in the graph represent the transmitted bits and the constraints they satisfy. For a linear code with a codeword length $N$ and rate $R = K/N$, the number of constraints is of order $M = N - K$ (there could be more constraints, if they happen to be redundant). Any 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 $N$.



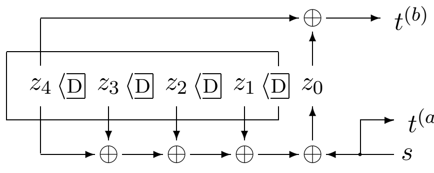
(a)



(b)



(c1)



(c2)

Figure 52.1. 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.

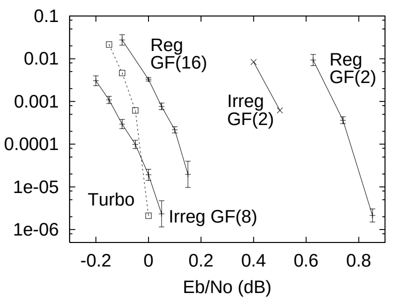
**Repeat-accumulate codes** (Divsalar *et al.* 1998) can be represented by a graph with four types of node: equality constraints $\boxed{=}$, intermediate binary variables (black circles), parity constraints $\boxed{+}$, and the transmitted bits (white circles). The encoder sets each group of intermediate 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; Richardson *et al.* 2001). The carefully constructed codes of Urbanke, Richard-

(a)


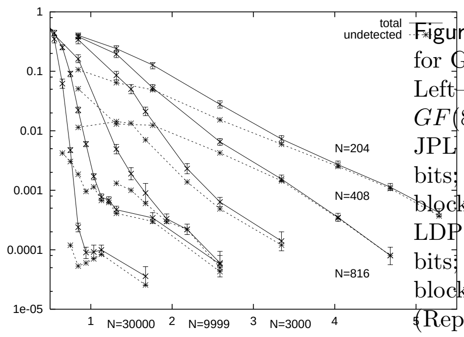
(b)


Figure 52.2. (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, versus $E_b/N_0$. The dotted lines show the frequency of undetected errors.

| 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 |

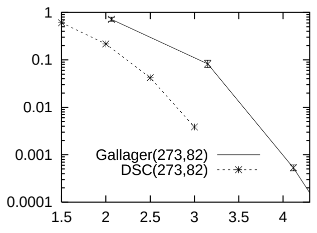


Figure 52.3. Low-density parity-check codes satisfying many redundant constraints outperform equivalent Gallager codes. Data on DSC code performance courtesy of R. Lucas and M. Fossorier. The table shows the $N$, $M$, $K$, distance $d$, and row weight $k$ of some difference-set cyclic codes, highlighting 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.

son and Shokrollahi outperform turbo codes at long blocklengths. Turbo 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 (Richardson *et al.* 2001), 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 52.3). 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**.

# 53

# *Low-density parity-check codes* *

These codes can be decoded by belief propagation. Their performance is excellent; as of 1999, they are the best known codes. 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.

# 54

# *Convolutional codes **

This chapter, which follows tightly on from chapter 27, 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.*

## 54.1 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.

## 54.2 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.

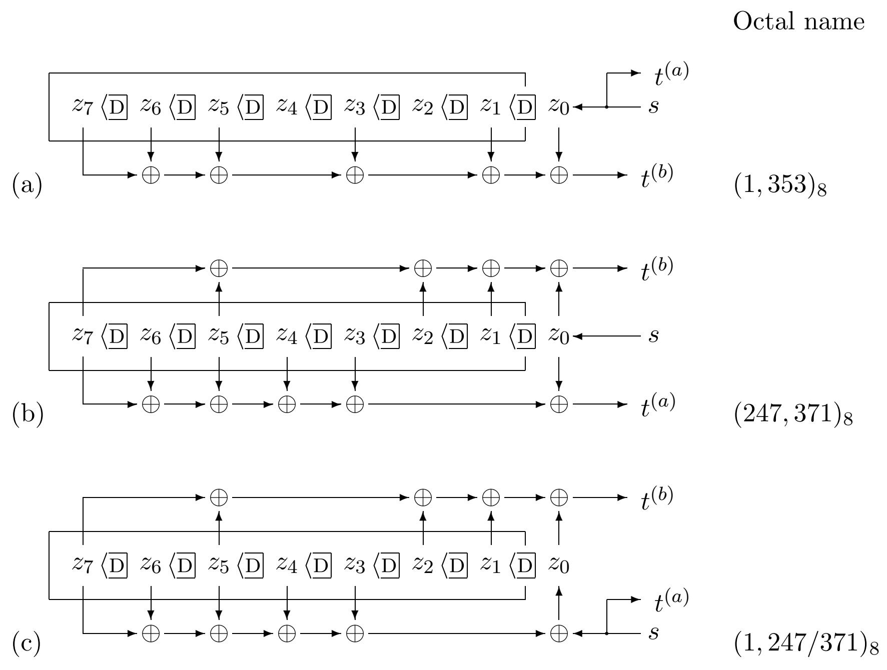


Figure 54.1. Linear feedback shift registers for generating convolutional codes with rate 1/2. After Frey (1998).

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 54.1 shows three linear feedback shift registers which could be used to define convolutional codes. The symbol $\langle\boxed{\mathrm{D}}$ 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 54.1 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 54.1(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 54.1(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 54.1(c) is similar to the nonsystematic nonrecursive filter shown in figure 54.1(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 54.2(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 54.2(a) we have

$$t^{(a)} = p \oplus z_0^{\text{nonrecursive}} \tag{54.1}$$

whereas is figure 54.2(b) we have

$$z_0^{\text{recursive}} = t^{(a)} \oplus p. \tag{54.2}$$

Substituting for $t^{(a)}$, and using $p \oplus p = 0$ we immediately find

$$z_0^{\text{recursive}} = z_0^{\text{nonrecursive}}. \tag{54.3}$$

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

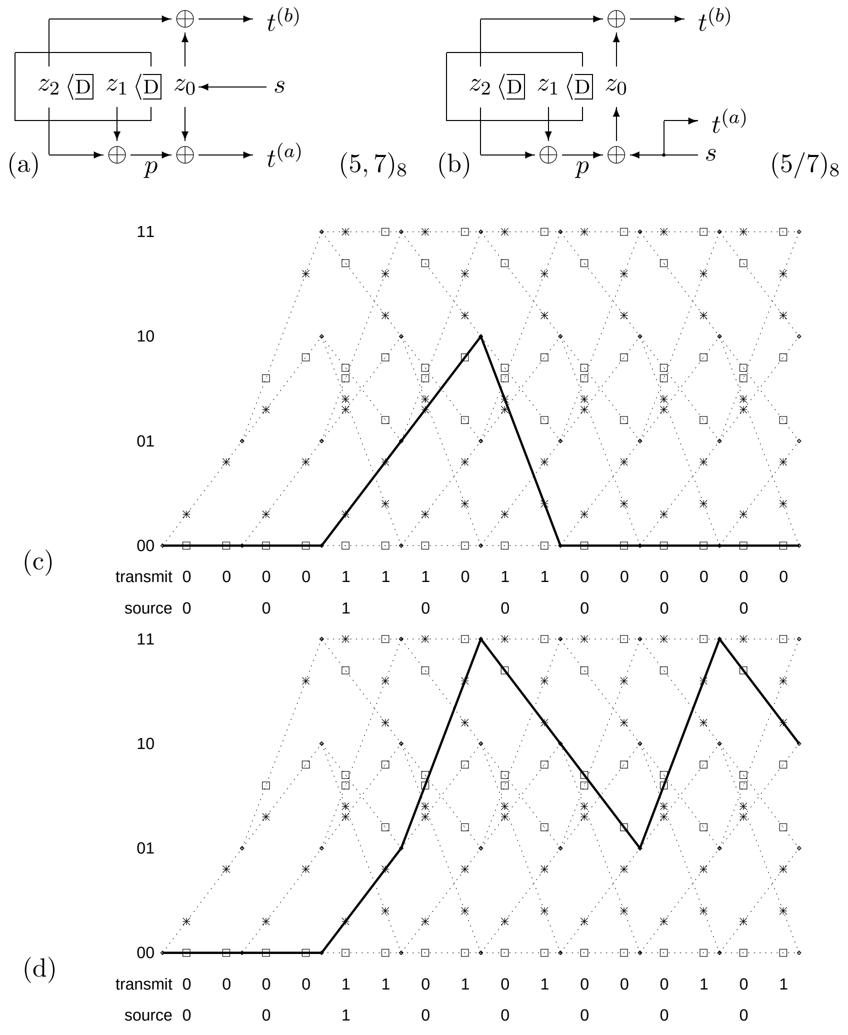


Figure 54.2. Trellis of $K = 2$ rate $1/2$ convolutional codes. $K = 2$. These two are compared to explain how recursive and non–recursive are equivalent. Comparison of trellises for NN and SR. Paths taken through trellis when source = 00100000. NN first transmission = 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 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.

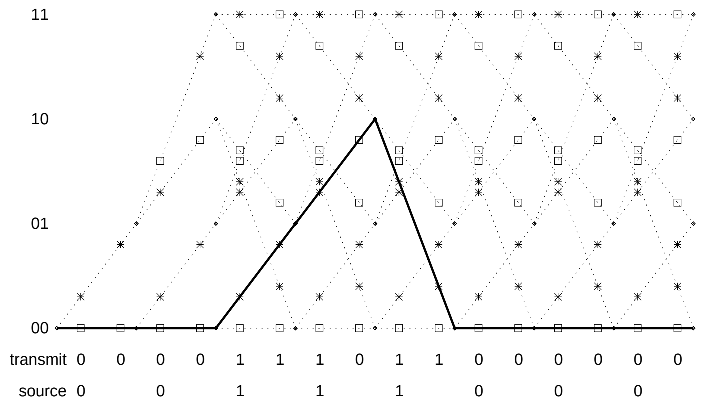
(b)


Figure 54.3. 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 54.2(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 54.2(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 54.2(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 54.2(d) shows the trellis of the recursive code of figure 54.2(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 54.2(c) and (d) are identical, and for any state sequence of the non-recursive 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, ...) into the recursive filter, as shown in figure 54.3, the state sequence and transmission are identical to those shown in figure 54.2(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.

## 54.3 The decoding problem

To keep things general, let's discuss the case where the initial start state is not known.

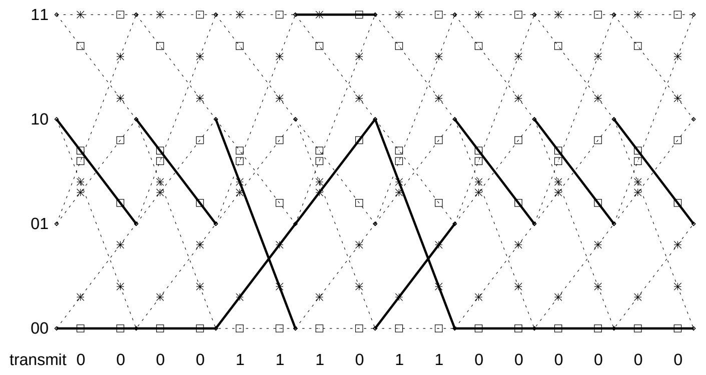
The receiver receives a bit stream, and wishes to infer the state sequence and thence the source stream. Let's first consider the noise–free decoding problem. Since it's systematic, can simply read out the relevant bits. What about recovering the initial condition? In principle there might be two paths through the trellis which have identical systematic bits but different parity bits and different initial condition. Figure 54.4(a) shows a selection of one path through the trellis of figure 54.4(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 54.4(b) shows the situation when the transmission of figure 54.4(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 54.2(c). Thus the Viterbi algorithm can correct some single errors.

Now look at the case shown in figure 54.5. Figure 54.6 shows two hypotheses useful for discussing figure 54.5. 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.

(a)



(b)

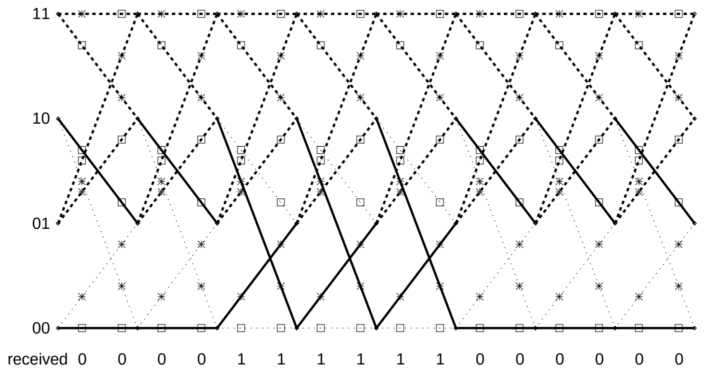
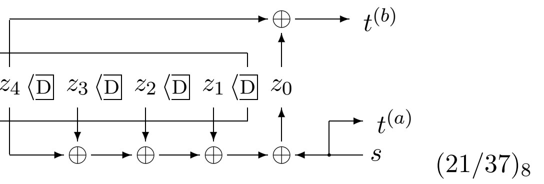


Figure 54.4. A transmission and a received vector, shown by the corresponding likelihoods for all the edges.

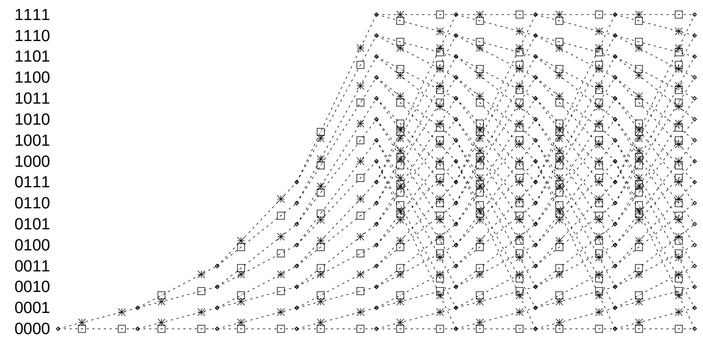
Figure 54.7 shows the likelihood function for the received vector equal to a random received vector, to illustrate how difficult the decoding problem gets when there is a lot of noise.

Termination is shown in figure 54.8. Note that terminating makes the protection uniform. Seems that the end bits get the best protection now.

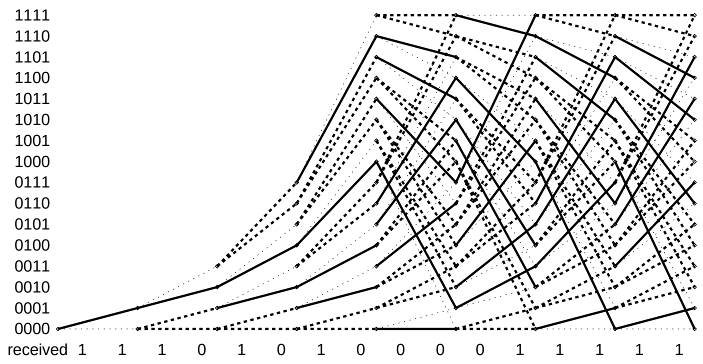
(z)



(a)



(b1)



(b2)

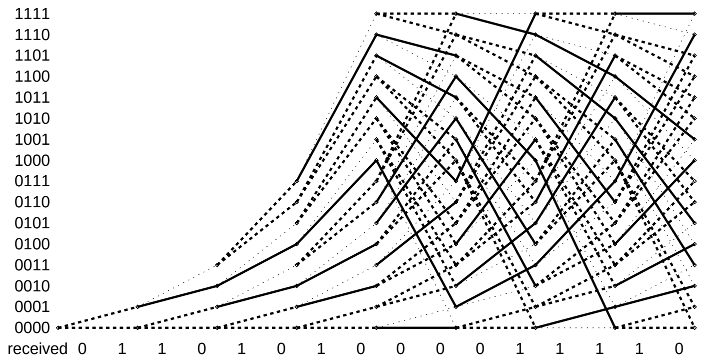
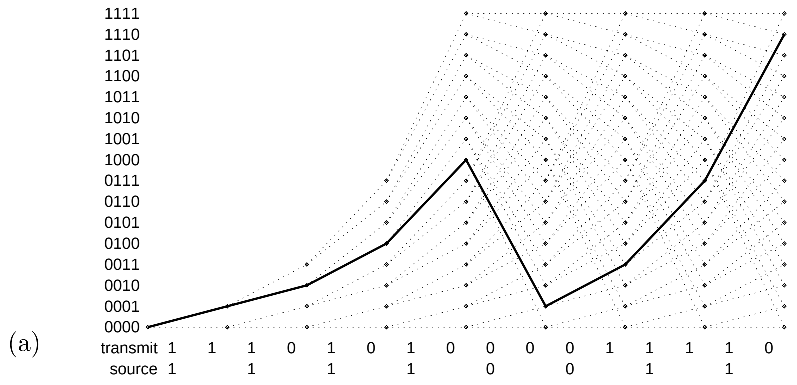


Figure 54.5. The trellis for a $K = 4$ code and (b1) the likelihood function for the received vector equal to a codeword with just one bit flipped; (b2) the likelihood function for the received vector equal to a codeword with a different one bit flipped. Notice also in b1 that there are pairs of paths which are identical for many successive bits. This figure illustrates the unequal protection of bits and motivates termination of the trellis.



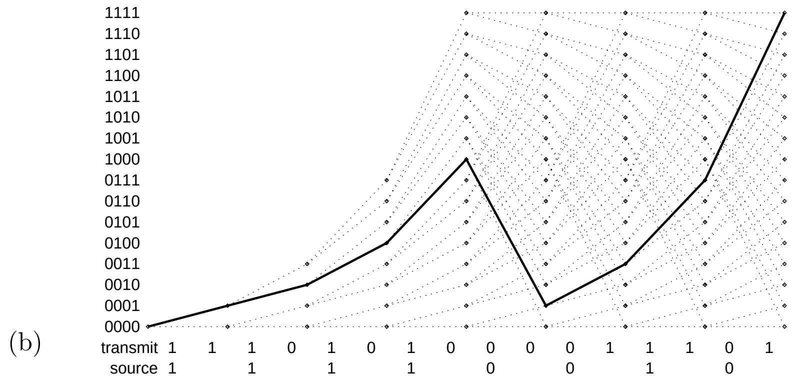


Figure 54.6. Two hypotheses useful for discussing figure 54.5. They differ in the last source bit only. The a figure is the only good hypothesis for b2, but in b1, both hypotheses are equally good.

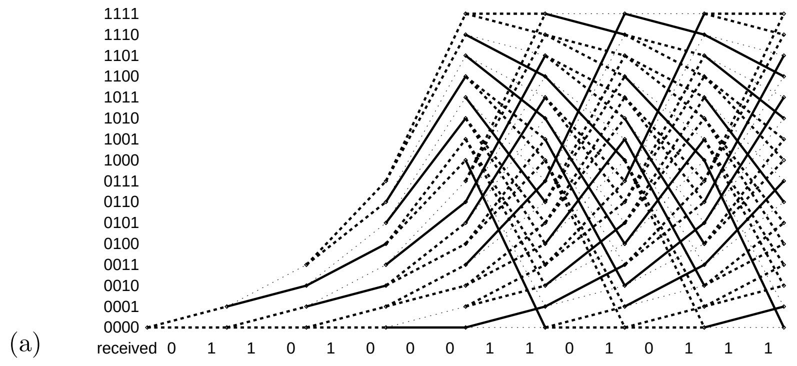
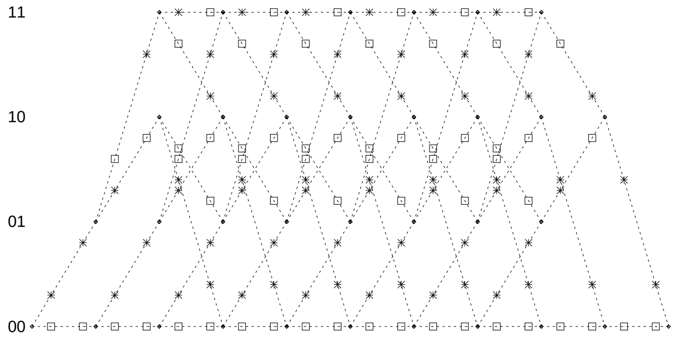


Figure 54.7. The likelihood function for the received vector equal to a random received vector, to illustrate how difficult the decoding problem gets when there is a lot of noise.

(a)



(b)

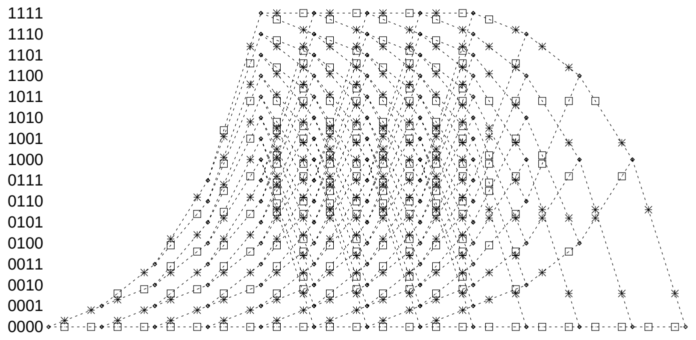


Figure 54.8. Terminated trellises

# 55

# *Turbo codes* *

In 1996, turbo codes were 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.

## 55.1 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 55.1); 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

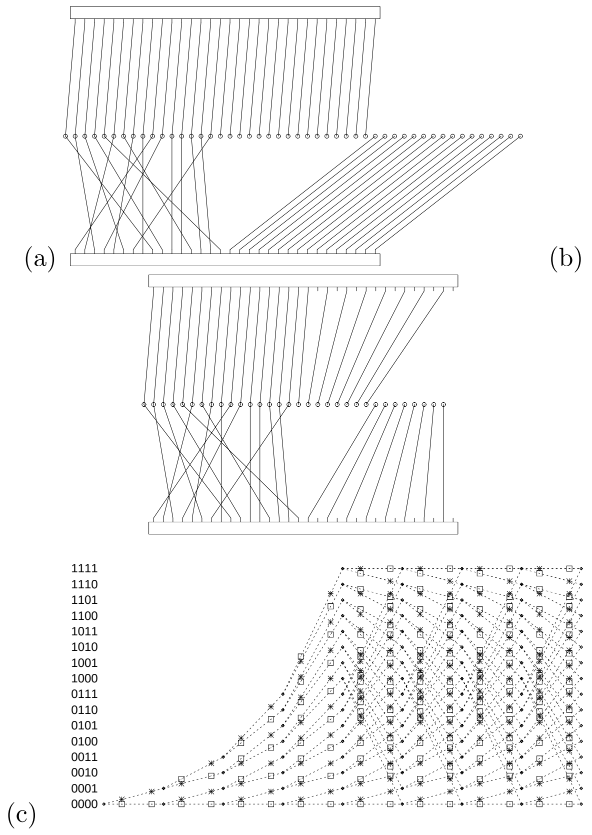


Figure 55.1. (a,b) A rate 1/3 (a) and rate 1/2 (b) Turbo code represented as trellis constrained codes. The circles represent the codeword bits. The two rectangles represent trellises of rate 1/2 convolutional codes, with the systematic bits occupying the left half of the rectangle and the parity bits 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 55.3 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 55.2.

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

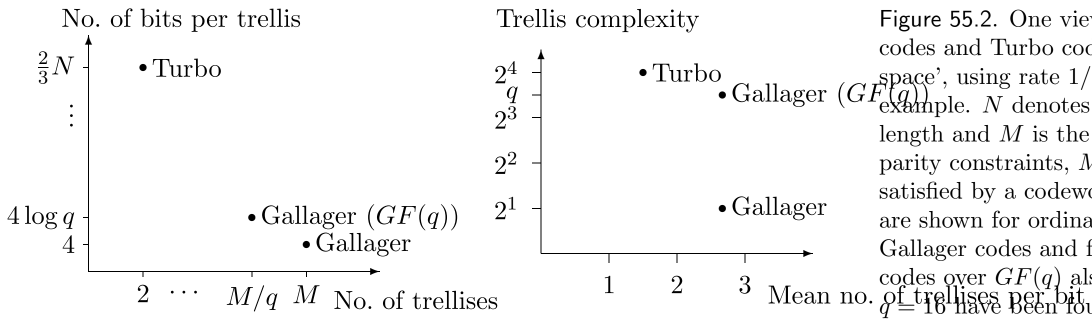


Figure 55.2. One view of Gallager codes and Turbo codes in 'code space', using rate 1/3 codes as an example. $N$ denotes the block length and $M$ is the number of parity constraints, $M = N - K$, satisfied by a codeword. Points are shown for ordinary binary Gallager codes and for Gallager codes over $GF(q)$ 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.

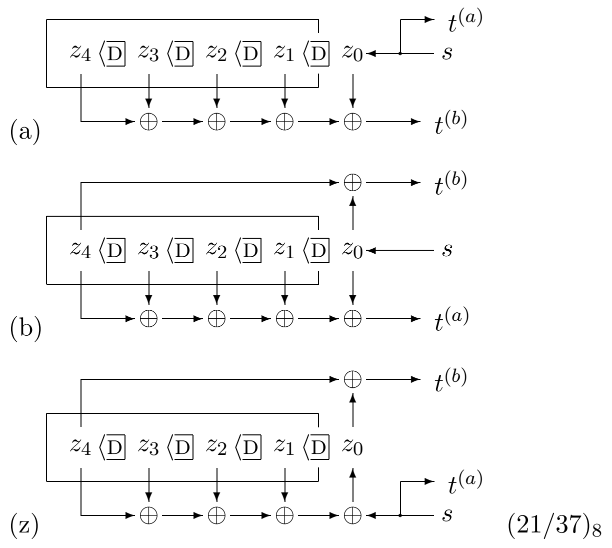


Figure 55.3. Linear feedback shift registers for generating convolutional codes. $K = 4$.

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.

## 55.2 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

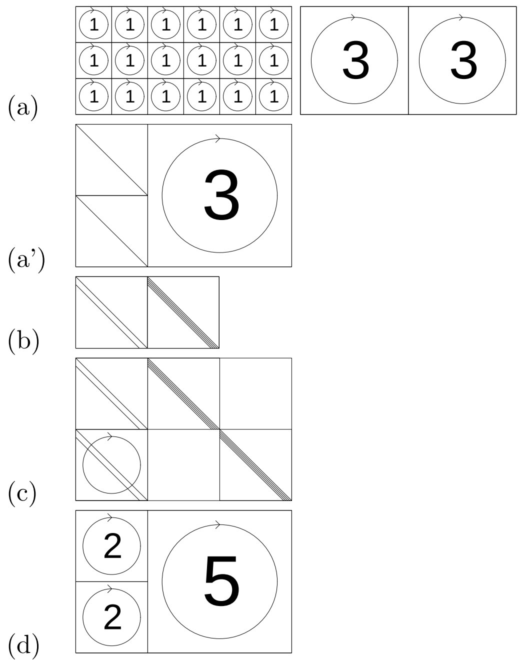


Figure 55.4. Schematic pictures of the parity check matrices of (a) a regular Gallager code, rate 1/2, (a') an almost regular Gallager code, rate 1/3, (b) a convolutional code, rate 1/2, and (c) a Turbo code, rate 1/3. Notation: A diagonal line represents an identity matrix. A band of diagonal lines 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.

## 55.3 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.

# 56

# *Repeat-accumulate codes*

# Part VIII

# Appendices

# A

# Notation

## A.1 Basic notation

**What does $P(A|B,C)$ mean?** $P(A|B,C)$ is pronounced 'the probability that $A$ is true *given that* $B$ is true *and* $C$ is true.

**What does $\hat{s}$ mean?** Usually, a 'hat' over a variable denotes a guess or estimator. So $\hat{s}$ is a guess at the value of $s$.

**What does $\prod_{n=1}^{N}$ mean?** This is like the summation $\sum_{n=1}^{N}$ but it denotes a product. It's pronounced 'product over $n$ from 1 to N' So, for example,

$$\prod_{n=1}^{N} n = 1 \times 2 \times 3 \times \ldots \times N = N! \tag{A.1}$$

and

$$\prod_{n=1}^{N} n = \exp\left[\sum_{n=1}^{N} \ln n\right]. \tag{A.2}$$

I like to choose the name of the free variable in a sum or a product – here, $n$ – to be the lower case version of the range of the sum. So $n$ usually runs from 1 to $N$, and $m$ usually runs from 1 to $M$. This is a habit I learnt from Yaser Abu-Mostafa, and I think it makes formulae easier to read.

**What does $\binom{N}{n}$ mean?** This is pronounced '$N$ choose $n$', and it is the number of ways of selecting an unordered set of $n$ objects from a set of size $N$.

$$\binom{N}{n} = \frac{N!}{(N-n)!n!} \tag{A.3}$$

This function is known as the combination function.

**What does $H_2^{-1}(1 - R/C)$ mean?** Just as $\sin^{-1}(s)$ denotes the inverse function to $s = \sin(x)$, so $H_2^{-1}(h)$ is the inverse function to $h = H_2(x)$.

There is potential confusion when people use $\sin^2 x$ to denote $(\sin x)^2$, since then we might expect $\sin^{-1} s$ to denote $1/\sin(s)$; I therefore like to avoid using the notation $\sin^2 x$.

**What does $f'(x)$ mean?** The answer depends on the context. Often, a 'prime' is used to denote differentiation:

$$f'(x) \equiv \frac{d}{dx} f(x); \tag{A.4}$$

similarly, a dot denotes differentiation with respect to time, $t$:

$$\dot{x} \equiv \frac{d}{dt} x. \tag{A.5}$$

However, the prime is also a useful indicator for 'another variable', for example 'a new value for a variable'. So, for example, $x'$ might denote 'the new value of $x$'. Also, if there are two integers that both range from 1 to $N$, I will often name those integers $n$ and $n'$.

So my rule is: if a prime occurs in an expression that could be a function, such as $f'(x)$ or $h'(y)$, then it denotes differentiation; otherwise it indicates 'another variable'.

**What is the error function?** Definitions of this function vary. I define it to be the cumulative probability of a standard (variance = 1) normal distribution,

$$\Phi(z) \equiv \int_{-\infty}^{z} \exp(-z^2/2)/\sqrt{2\pi}. \tag{A.6}$$

**What does $\mathbf{x}^{\mathsf{T}}$ mean?** The superscript $\mathsf{T}$ is pronounced 'transpose'. Transposing a row-vector turns it into a column vector:

$$(1, 2, 3)^{\mathsf{T}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \tag{A.7}$$

and vice versa.

Similarly, matrices can be transposed. If $M_{ij}$ is the entry in row $i$ and column $j$ of matrix $\mathbf{M}$, and $\mathbf{N} = \mathbf{M}^{\mathsf{T}}$, then $N_{ji} = M_{ij}$.

**What does $\mathcal{E}(r)$ mean?** $\mathcal{E}[r]$ is pronounced 'the expected value of $r$', and it is the mean value of $r$. Another symbol for 'expected value' is the pair of angle-brackets, $\langle r \rangle$.

**What does $|x|$ mean?** The vertical bars '$|\cdot|$' have two meanings. If $\mathcal{A}$ is a set, then $|\mathcal{A}|$ denotes the number of elements in the set; if $x$ is a number, then $|x|$ is the absolute value of $x$.

# B

# *Useful formulae, etc.*

## B.1 Representations

## B.2 How to represent a Gaussian

Representations are especially of interest if

1. they satisfy the symmetries and invariances of the situation;
2. the dimensionality of the parameters is the same as the true number of degrees of freedom;
3. any values of the parameters are valid;
4. there is no double-cover: any Gaussian can be instantiated by only one setting of the parameters;
5. there are no discontinuities: any two Gaussians that define similar probability distributions should have parameters that are close together.

*Special case: two-dimensional Euclidean space*

In two dimensions we can use the following representation of a two-by-two positive definite matrix with unit determinant

$$\mathbf{M} = \begin{bmatrix} \sqrt{1+a^2+b^2}+a & b \\ b & \sqrt{1+a^2+b^2}-a \end{bmatrix} \tag{B.1}$$

whose inverse is

$$\mathbf{M}^{-1} = \begin{bmatrix} \sqrt{1+a^2+b^2}-a & -b \\ -b & \sqrt{1+a^2+b^2}+a \end{bmatrix}, \tag{B.2}$$

to control the shape of the Gaussian. The parameter $a$ makes the Gaussian elongated in the horizontal or vertical directions, depending on its sign, and the parameter $b$ produces elongation in the diagonal directions.

This representation satisfies all the desiderata.

If you are interested in where if came from, it may be helpful to note that

$$\mathbf{M} = \begin{bmatrix} \cosh(u)+\sinh(u)\cos(2\,\theta) & \sinh(u)\sin(2\,\theta) \\ \sinh(u)\sin(2\,\theta) & \cosh(u)-\sinh(u)\cos(2\,\theta) \end{bmatrix} \tag{B.3}$$

where $u$ controls the degree of ellipticity and $\theta$ controls the orientation. The parameters are related by $\sinh(u)\cos(2\,\theta) = a$ and $\sinh(u)\sin(2\,\theta) = b$. The representation using $u, \theta$ violates the 'no double cover' desideratum.

To obtain a general variance-covariance matrix we need a third degree of freedom, the volume. We thus write

$$\mathbf{A}(a,b,c) = c \begin{bmatrix} \sqrt{1+a^2+b^2}+a & b \\ b & \sqrt{1+a^2+b^2}-a \end{bmatrix} \tag{B.4}$$

The derivatives with respect to $a$ and $b$ of the quadratic form $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}$ are

$$\frac{\partial \mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}}{\partial b} = \frac{(x^2+y^2)b}{\sqrt{1+a^2+b^2}} + 2\,xy \tag{B.5}$$

$$\frac{\partial \mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x}}{\partial b} = \frac{(x^2+y^2)a}{\sqrt{1+a^2+b^2}} + x^2 - y^2 \tag{B.6}$$

## B.3 Statistical physics

## B.4 About phase transitions

A system with states $\mathbf{x}$ in contact with a heat bath at temperature $T = 1/\beta$ has probability distribution

$$P(\mathbf{x}|\beta) = \frac{1}{Z(\beta)} \exp(-\beta E(\mathbf{x})). \tag{B.7}$$

The partition function is

$$Z(\beta) = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})). \tag{B.8}$$

The inverse temperature $\beta$ 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 $\beta$, since it is simply a sum of exponentials. Moreover, all the derivatives of $Z(\beta)$ with respect to $\beta$ are continuous too.

What phase transitions are all about, however, is this: phase transitions correspond to values of $\beta$ and $V$ at which the derivatives of $Z$ have discontinuities or divergences.

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 by simply grouping together $N$ independent systems whose partition function is $Z_{(1)}(\beta)$, then nothing interesting happens. The partition function is simply

$$Z_{(N)}(\beta) = [Z_{(1)}(\beta)]^N. \tag{B.9}$$

Now, while this function $Z_{(N)}(\beta)$ may be a very rapidly varying function of $\beta$, 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)

$$\log Z_{(N)}(\beta) = N \log Z_{(1)}(\beta). \tag{B.10}$$

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 derivatives 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 $\mathbf{x} \in \{0, 1\}^N$.

$$E(\mathbf{x}) = \begin{cases} -N\epsilon & \mathbf{x} = (0, 0, 0, \ldots, 0) \\ 0 & \text{otherwise} \end{cases} \tag{B.11}$$

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:

$$E(\mathbf{x}) = \epsilon \sum_n (2x_n - 1) \tag{B.12}$$

The partition function is given by

$$Z(\beta) = e^{\beta N \epsilon} + 2^N - 1. \tag{B.13}$$

The function

$$\log Z(\beta) = \log\left(e^{\beta N\epsilon} + 2^N - 1\right) \tag{B.14}$$

is sketched in figure B.1(a) along with its low temperature behaviour,

$$\log Z(\beta) \simeq N\beta\epsilon, \quad \beta \to \infty, \tag{B.15}$$

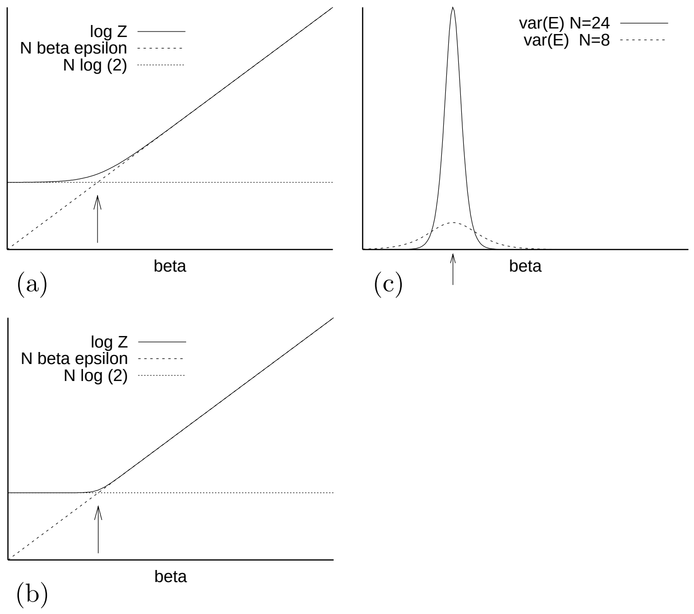


Figure B.1. (a) Partition function of toy system which shows a phase transition for large $N$. The arrow marks the point $\beta_c = \log 2/\epsilon$. (b) The same, for larger $N$.
(c) The variance of the energy of the system as a function of $\beta$ for two system sizes. As $N$ increases the variance has an increasingly sharp peak at the critical point $\beta_c$. Contrast with figure B.2.

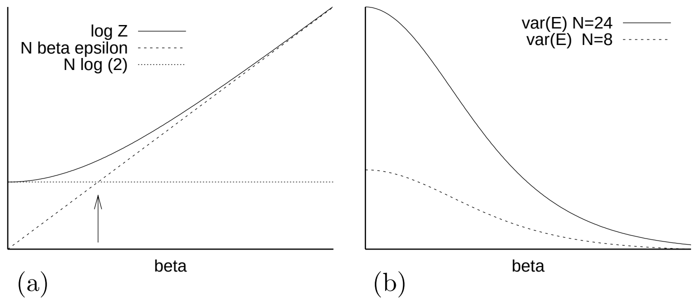


Figure B.2. The partition function (a) and energy-variance (b) of a system consisting of $N$ independent spins. The partition function changes gradually from one asymptote to the other, regardless of how large $N$ is; the variance of the energy does not have a peak. The fluctuations are largest at high temperature and scale linearly with system size $N$.

and its high temperature behaviour,

$$\log Z(\beta) \simeq N \log 2, \quad \beta \to 0. \tag{B.16}$$

The arrow marks the point

$$\beta = \frac{\log 2}{\epsilon} \tag{B.17}$$

at which these two asymptotes intersect. In the limit $N \to \infty$, the graph of $\log Z(\beta)$ becomes more and more sharply bent at this point (figure B.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

$$\frac{N^2\epsilon^2}{4}, \tag{B.18}$$

which corresponds to the system spending half of its time in the ground state and half its time in the other states.

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 into 'first-order' and 'continuous' transitions. In a first-order phase transition, there is a discontinuous change of an order-parameter; in a continuous transition, all order-parameters change continuously. [What's an order parameter? – a scalar function of the state of the system; or, to be precise, the expectation of such a function.]

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.

Remember that information content ($\log 1/P(\mathbf{x})$) and energy are very closely related. If typicality holds, then the system's energy has negligible fluctuations, and *vice versa*.

## *Replica theory*

The replica trick is

$$\lim_{n\to 0} \frac{Z^n - 1}{n} \to \log Z \tag{B.19}$$

Here is a mathematical identity involving the entropy function

$$\lim_{r\to 1} \frac{1 - \sum p_i^r}{r-1} = \sum p_i \log \frac{1}{p_i} \tag{B.20}$$

This identity is the replica trick in disguise:

$$1 - \sum_i p_i^r = \sum_i (p_i - p_i^r) = \sum_i p_i(1 - p_i^{r-1}) \tag{B.21}$$

make a change of variables n = r-1, divide by n and get

$$\sum_i p_i(1-p_i^n)/n \tag{B.22}$$

invoking the replica trick,

$$\sum_i p_i(1-p_i^n)/n = -\sum_i p_i(p_i^n - 1)/n \rightarrow H(\mathbf{p}) \tag{B.23}$$

Thanks to Tommi Jaakkola.

## B.5 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 '·'; multiplication of any element by 0 yields 0;
3. these operations satisfy the distributive rule $(a + b) \cdot c = a \cdot c + b \cdot c$.

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 |

| · | 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 |

| · | 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 |

[1] In fact an Abelian group, i.e., one that satisfies commutativity — $a + b = b + a$.
[2] Also an Abelian group.

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, $B \cdot B = x^2 + (1+1)x + 1 = x = A$.

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 uniform distribution is easiest to guarantee if these elements form a group under both addition and multiplication, because then these 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.

## B.6 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 |
| $D$ | $x^2+1$ | 101 |
| $E$ | $x^2+x$ | 110 |
| $F$ | $x^2+x+1$ | 111 |

(B.24)

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 | $E$ | $C$ | $B$ |

(B.25)

Here is a (9,2) code over $GF(8)$ generated by the generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & A & B & C & D & E & F \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{B.26}$$

```
000000000   011111111   0AAAAAAAA   0BBBBBBBB   0CCCCCCCC   0DDDDDDDD   0EEEEEEEE   0FFFFFFFF
101ABCDEF   110BADCFE   1AB01EFCD   1BA10FEDC   1CDEF01AB   1DCFE10BA   1EFCDAB01   1FEDCBA10
A0ACEB1FD   A1BDFA0EC   AA0EC1BDF   AB1FD0ACE   ACE0AFDB1   ADF1BECA0   AECA0DF1B   AFDB1CE0A
B0BEDFC1A   B1AFCED0B   BA1CFDEB0   BB0DECFA1   BCFA1B0DE   BDEB0A1CF   BED0B1AFC   BFC1A0BED
C0CBFEAD1   C1DAEFBC0   CAE1DC0FB   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 B.1**:** Is this code a perfect code?

Exercise B.2**:** Is this code a maximum distance separable code?

Solution to exercise B.1 (p.602): 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 B.2 (p.602): 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.

## B.7 Eigenvectors and eigenvalues

A *right-eigenvector* of a square matrix $\mathbf{A}$ is a non-zero vector $\mathbf{e}_{\mathsf{R}}$ that satisfies

$$\mathbf{A}\mathbf{e}_{\mathsf{R}} = \lambda \mathbf{e}_{\mathsf{R}}, \tag{B.27}$$

where $\lambda$ 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 $\mathbf{e}_{\mathsf{L}}$ that satisfies

$$\mathbf{e}_{\mathsf{L}}^{\mathsf{T}}\mathbf{A} = \lambda \mathbf{e}_{\mathsf{L}}^{\mathsf{T}}. \tag{B.28}$$

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 $\lambda^* = x - yi$.

*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,
$$\mathbf{e}^{(a)} \cdot \mathbf{e}^{(b)} = \delta_{ab}; \tag{B.29}$$
The matrix can be expressed as a weighted sum of outer products of the eigenvectors:
$$\mathbf{A} = \sum_a \lambda_a [\mathbf{e}^{(a)}][\mathbf{e}^{(a)}]^{\mathsf{T}}. \tag{B.30}$$

(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. Generically, there are $N$ eigenvalues, all distinct, and each has one left eigenvector and one right eigenvector. In cases where two or more eignevalues coincide, for each distinct eigenvalue that is non-zero, there is at least one left eigenvector and one right eigenvector.

Left and right eigenvectors that have different eigenvalue are orthogonal, that is,
$$\text{if } \lambda_a \neq \lambda_b \text{ then } \mathbf{e}_{\mathsf{L}}^{(a)} \cdot \mathbf{e}_{\mathsf{R}}^{(b)} = 0. \tag{B.31}$$

*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 $\geq 0$ which would be written $|\mathbf{C}| \geq 0$, nor with the statement that $\mathbf{C}$ is positive definite, for which I have no notation.)

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.

Generically, all the other eigenvalues are smaller in absolute magnitude. [There can be several eigenvalues of identical magnitude in special cases.]

| Matrix | Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$ |
|---|---|
| $\begin{bmatrix}1&2&0\\1&1&0\\0&0&1\end{bmatrix}$ | $\begin{array}{ccc} 2.41 & 1 & -0.41 \\ \begin{bmatrix}.58\\.82\\0\end{bmatrix}\begin{bmatrix}.82\\.58\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} & \begin{bmatrix}-.58\\.82\\0\end{bmatrix}\begin{bmatrix}-.82\\.58\\0\end{bmatrix} \end{array}$ |
| $\begin{bmatrix}0&0&1&2\\0&0&2&1\\1&2&0&0\\2&1&0&0\end{bmatrix}$ | $\begin{array}{cccc} 3 & 1 & -1 & -3 \\ \begin{bmatrix}.5\\.5\\.5\\.5\end{bmatrix}\begin{bmatrix}.5\\.5\\.5\\.5\end{bmatrix} & \begin{bmatrix}-.5\\.5\\.5\\-.5\end{bmatrix}\begin{bmatrix}-.5\\.5\\.5\\-.5\end{bmatrix} & \begin{bmatrix}.5\\-.5\\.5\\-.5\end{bmatrix}\begin{bmatrix}.5\\-.5\\.5\\-.5\end{bmatrix} & \begin{bmatrix}-.5\\-.5\\.5\\.5\end{bmatrix}\begin{bmatrix}-.5\\-.5\\.5\\.5\end{bmatrix} \end{array}$ |
| $\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&1&1&1\end{bmatrix}$ | $\begin{array}{cccc} 1.93 & -0.1-0.8i & -0.1+0.8i & -0.77 \\ \begin{bmatrix}.3\\.5\\.6\\.6\end{bmatrix}\begin{bmatrix}.1\\.2\\.4\\.9\end{bmatrix} & \begin{bmatrix}-.3-.3i\\.1-.7i\\.5-.1i\\-.2+.3i\end{bmatrix}\begin{bmatrix}-.5+.4i\\.3+.4i\\.3-.3i\\-.3-.2i\end{bmatrix} & \begin{bmatrix}-.3+.3i\\.1+.7i\\.5+.1i\\-.2-.3i\end{bmatrix}\begin{bmatrix}-.5-.4i\\.3-.4i\\.3+.3i\\-.3+.2i\end{bmatrix} & \begin{bmatrix}-.5\\.2\\-.7\\.4\end{bmatrix}\begin{bmatrix}-.7\\.5\\-.4\\.3\end{bmatrix} \end{array}$ |
| $\begin{bmatrix}0&1&0\\0&0&1\\1&1&1\end{bmatrix}$ | $\begin{array}{ccc} 1.84 & -0.4-0.6i & -0.4+0.6i \\ \begin{bmatrix}.4\\.6\\.7\end{bmatrix}\begin{bmatrix}.3\\.5\\.9\end{bmatrix} & \begin{bmatrix}-.5-.2i\\.2-.7i\\.1+.4i\end{bmatrix}\begin{bmatrix}-.5+.5i\\.5+.1i\\-.2-.4i\end{bmatrix} & \begin{bmatrix}-.5+.2i\\.2+.7i\\.1-.4i\end{bmatrix}\begin{bmatrix}-.5-.5i\\.5-.1i\\-.2+.4i\end{bmatrix} \end{array}$ |
| $\begin{bmatrix}0&1\\1&1\end{bmatrix}$ | $\begin{array}{cc} 1.62 & -0.62 \\ \begin{bmatrix}.53\\.85\end{bmatrix}\begin{bmatrix}.53\\.85\end{bmatrix} & \begin{bmatrix}.85\\-.53\end{bmatrix}\begin{bmatrix}.85\\-.53\end{bmatrix} \end{array}$ |
| $\begin{bmatrix}1&1&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&1\end{bmatrix}$ | $\begin{array}{cccc} 1.62 & 0.5+0.9i & 0.5-0.9i & -0.62 \\ \begin{bmatrix}.60\\.37\\.37\\.60\end{bmatrix}\begin{bmatrix}.60\\.37\\.37\\.60\end{bmatrix} & \begin{bmatrix}.1-.5i\\-.3-.4i\\.3+.4i\\-.1+.5i\end{bmatrix}\begin{bmatrix}.1-.5i\\.3+.4i\\-.3-.4i\\-.1+.5i\end{bmatrix} & \begin{bmatrix}.1+.5i\\-.3+.4i\\.3-.4i\\-.1-.5i\end{bmatrix}\begin{bmatrix}.1+.5i\\.3-.4i\\-.3+.4i\\-.1-.5i\end{bmatrix} & \begin{bmatrix}.37\\-.60\\-.60\\.37\end{bmatrix}\begin{bmatrix}.37\\-.60\\-.60\\.37\end{bmatrix} \end{array}$ |
| $\begin{bmatrix}0&1&0&0\\0&0&1&1\\1&1&0&0\\0&0&1&0\end{bmatrix}$ | $\begin{array}{cccc} 1.62 & -0.5+0.9i & -0.5-0.9i & -0.62 \\ \begin{bmatrix}.37\\.60\\.60\\.37\end{bmatrix}\begin{bmatrix}.37\\.60\\.60\\.37\end{bmatrix} & \begin{bmatrix}-.2+.5i\\.3+.4i\\-.3-.4i\\.2-.5i\end{bmatrix}\begin{bmatrix}.2-.5i\\.3+.4i\\-.3-.4i\\-.2+.5i\end{bmatrix} & \begin{bmatrix}-.2-.5i\\.3-.4i\\-.3+.4i\\.2+.5i\end{bmatrix}\begin{bmatrix}.2+.5i\\.3-.4i\\-.3+.4i\\-.2-.5i\end{bmatrix} & \begin{bmatrix}.60\\-.37\\-.37\\.60\end{bmatrix}\begin{bmatrix}.60\\-.37\\-.37\\.60\end{bmatrix} \end{array}$ |

Figure B.3. Some matrices and their eigenvectors

*Transition probability matrices*

An important example of a non-negative matrix is a transition probability matrix $\mathbf{Q}$.

*Definition*

A transition probability matrix $\mathbf{Q}$ has columns that are probability vectors, that is, it satisfies $\mathbf{Q} \geq 0$ and

$$\sum_i Q_{ij} = 1 \text{ for all } j. \tag{B.32}$$

*Properties*

This property can be rewritten in terms of the all-ones vector $\mathbf{n} = (1, 1, \ldots, 1)^{\mathsf{T}}$ thus:

$$\mathbf{n}^{\mathsf{T}}\mathbf{Q} = \mathbf{n}^{\mathsf{T}}. \tag{B.33}$$

So $\mathbf{n}$ is the principal left-eigenvector of $\mathbf{Q}$ with eigenvalue $\lambda_1 = 1$.

$$\mathbf{e}_{\mathsf{L}}^{(1)} = \mathbf{n}. \tag{B.34}$$

Because it is a non-negative matrix, $\mathbf{Q}$ has a principal right-eigenvector that is non-negative, $\mathbf{e}_{\mathsf{R}}^{(1)}$. This vector, if we normalize it such that $\mathbf{e}_{\mathsf{R}}^{(1)} \cdot \mathbf{n} = 1$, 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.

| Matrix | Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$ |
|---|---|
| $\begin{bmatrix} 0 & .5 & 0 \\ 1 & 0 & 1 \\ 0 & .5 & 0 \end{bmatrix}$ | $\begin{array}{ccc} 1 & 0.00 & -1 \\ \begin{bmatrix} .58 \\ .58 \\ .58 \end{bmatrix} \begin{bmatrix} .41 \\ .82 \\ .41 \end{bmatrix} & \begin{bmatrix} -.71 \\ -.00 \\ .71 \end{bmatrix} \begin{bmatrix} -.71 \\ -.00 \\ .71 \end{bmatrix} & \begin{bmatrix} .58 \\ -.58 \\ .58 \end{bmatrix} \begin{bmatrix} .41 \\ -.82 \\ .41 \end{bmatrix} \end{array}$ |
| $\begin{bmatrix} 0 & .33 & 0 & 0 \\ 1 & 0 & .67 & 0 \\ 0 & .67 & 0 & 1 \\ 0 & 0 & .33 & 0 \end{bmatrix}$ | $\begin{array}{cccc} 1 & 0.33 & -0.33 & -1 \\ \begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix} \begin{bmatrix} .22 \\ .67 \\ .67 \\ .22 \end{bmatrix} & \begin{bmatrix} .67 \\ .22 \\ -.22 \\ -.67 \end{bmatrix} \begin{bmatrix} .50 \\ .50 \\ -.50 \\ -.50 \end{bmatrix} & \begin{bmatrix} .67 \\ -.22 \\ -.22 \\ .67 \end{bmatrix} \begin{bmatrix} .50 \\ -.50 \\ -.50 \\ .50 \end{bmatrix} & \begin{bmatrix} -.50 \\ .50 \\ -.50 \\ .50 \end{bmatrix} \begin{bmatrix} -.22 \\ .67 \\ -.67 \\ .22 \end{bmatrix} \end{array}$ |
| $\begin{bmatrix} 0 & .25 & 0 & 0 & 0 \\ 1 & 0 & .50 & 0 & 0 \\ 0 & .75 & 0 & .75 & 0 \\ 0 & 0 & .50 & 0 & 1 \\ 0 & 0 & 0 & .25 & 0 \end{bmatrix}$ | $\begin{array}{ccccc} 1 & 0.50 & 0 & -0.50 & -1 \\ \begin{bmatrix} .4 \\ .4 \\ .4 \\ .4 \\ .4 \end{bmatrix} \begin{bmatrix} .1 \\ .5 \\ .7 \\ .5 \\ .1 \end{bmatrix} & \begin{bmatrix} -.6 \\ -.3 \\ -.0 \\ .3 \\ .6 \end{bmatrix} \begin{bmatrix} -.3 \\ -.6 \\ -.0 \\ .6 \\ .3 \end{bmatrix} & \begin{bmatrix} .7 \\ 0 \\ -.2 \\ -.0 \\ .7 \end{bmatrix} \begin{bmatrix} .4 \\ -.0 \\ -.8 \\ 0 \\ .4 \end{bmatrix} & \begin{bmatrix} -.6 \\ .3 \\ -.0 \\ -.3 \\ .6 \end{bmatrix} \begin{bmatrix} -.3 \\ .6 \\ -.0 \\ -.6 \\ .3 \end{bmatrix} & \begin{bmatrix} .4 \\ -.4 \\ .4 \\ -.4 \\ .4 \end{bmatrix} \begin{bmatrix} -.1 \\ .5 \\ -.7 \\ .5 \\ -.1 \end{bmatrix} \end{array}$ |
| $\begin{bmatrix} 0 & .2 & 0 & 0 & 0 & 0 \\ 1 & 0 & .4 & 0 & 0 & 0 \\ 0 & .8 & 0 & .6 & 0 & 0 \\ 0 & 0 & .6 & 0 & .8 & 0 \\ 0 & 0 & 0 & .4 & 0 & 1 \\ 0 & 0 & 0 & 0 & .2 & 0 \end{bmatrix}$ | $\begin{array}{cccccc} 1 & 0.60 & 0.20 & -0.20 & -0.60 & -1 \\ \begin{bmatrix} .4 \\ .4 \\ .4 \\ .4 \\ .4 \\ .4 \end{bmatrix} \begin{bmatrix} .1 \\ .3 \\ .6 \\ .6 \\ .3 \\ .1 \end{bmatrix} & \begin{bmatrix} .6 \\ .4 \\ .1 \\ -.1 \\ -.4 \\ -.6 \end{bmatrix} \begin{bmatrix} .2 \\ .6 \\ .4 \\ -.4 \\ -.6 \\ -.2 \end{bmatrix} & \begin{bmatrix} .7 \\ .1 \\ -.1 \\ -.1 \\ .1 \\ .7 \end{bmatrix} \begin{bmatrix} -.3 \\ -.3 \\ .6 \\ .6 \\ -.3 \\ -.3 \end{bmatrix} & \begin{bmatrix} -.7 \\ .1 \\ .1 \\ -.1 \\ -.1 \\ .7 \end{bmatrix} \begin{bmatrix} -.3 \\ .3 \\ .6 \\ -.6 \\ -.3 \\ .3 \end{bmatrix} & \begin{bmatrix} .6 \\ -.4 \\ .1 \\ .1 \\ -.4 \\ .6 \end{bmatrix} \begin{bmatrix} -.2 \\ .6 \\ -.4 \\ -.4 \\ .6 \\ -.2 \end{bmatrix} & \begin{bmatrix} -.4 \\ .4 \\ -.4 \\ .4 \\ -.4 \\ .4 \end{bmatrix} \begin{bmatrix} -.1 \\ .3 \\ -.6 \\ .6 \\ -.3 \\ .1 \end{bmatrix} \end{array}$ |

Table B.1. Random walk on hypercubes.

| Matrix | Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$ |
|---|---|
| $\begin{bmatrix} 0 & .38 \\ 1 & .62 \end{bmatrix}$ | $\begin{array}{cc} 1 & -0.38 \\ \begin{bmatrix} .71 \\ .71 \end{bmatrix} \begin{bmatrix} .36 \\ .93 \end{bmatrix} & \begin{bmatrix} -.93 \\ .36 \end{bmatrix} \begin{bmatrix} -.71 \\ .71 \end{bmatrix} \end{array}$ |
| $\begin{bmatrix} 0 & .35 & 0 \\ 0 & 0 & .46 \\ 1 & .65 & .54 \end{bmatrix}$ | $\begin{array}{ccc} 1 & -0.2-0.3i & -0.2+0.3i \\ \begin{bmatrix} .58 \\ .58 \\ .58 \end{bmatrix} \begin{bmatrix} .14 \\ .41 \\ .90 \end{bmatrix} & \begin{bmatrix} -.8+.1i \\ -.2-.5i \\ .2+.2i \end{bmatrix} \begin{bmatrix} .2-.5i \\ -.6+.2i \\ .4+.3i \end{bmatrix} & \begin{bmatrix} -.8-.1i \\ -.2+.5i \\ .2-.2i \end{bmatrix} \begin{bmatrix} .2+.5i \\ -.6-.2i \\ .4-.3i \end{bmatrix} \end{array}$ |

Table B.2. Transition probability matrices for generating random paths through trellises.

| Matrix | Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$ | | | |
|---|---|---|---|---|
| $\begin{bmatrix} .90 & .20 & 0 & 0 \\ .10 & .80 & 0 & 0 \\ 0 & 0 & .90 & .20 \\ 0 & 0 & .10 & .80 \end{bmatrix}$ | $1$ <br> $\begin{bmatrix} 0 \\ 0 \\ .71 \\ .71 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ .89 \\ .45 \end{bmatrix}$ | $1$ <br> $\begin{bmatrix} .71 \\ .71 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} .89 \\ .45 \\ 0 \\ 0 \end{bmatrix}$ | $0.70$ <br> $\begin{bmatrix} .45 \\ -.89 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} .71 \\ -.71 \\ 0 \\ 0 \end{bmatrix}$ | $0.70$ <br> $\begin{bmatrix} 0 \\ 0 \\ -.45 \\ .89 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -.71 \\ .71 \end{bmatrix}$ |
| $\begin{bmatrix} .90 & .20 & 0 & 0 \\ .10 & .79 & .02 & 0 \\ 0 & .01 & .88 & .20 \\ 0 & 0 & .10 & .80 \end{bmatrix}$ | $1$ <br> $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix} \begin{bmatrix} .87 \\ .43 \\ .22 \\ .11 \end{bmatrix}$ | $0.98$ <br> $\begin{bmatrix} -.18 \\ -.15 \\ .66 \\ .72 \end{bmatrix} \begin{bmatrix} -.66 \\ -.28 \\ .61 \\ .33 \end{bmatrix}$ | $0.70$ <br> $\begin{bmatrix} .20 \\ -.40 \\ -.40 \\ .80 \end{bmatrix} \begin{bmatrix} .63 \\ -.63 \\ -.32 \\ .32 \end{bmatrix}$ | $0.69$ <br> $\begin{bmatrix} -.19 \\ .41 \\ -.44 \\ .77 \end{bmatrix} \begin{bmatrix} -.61 \\ .65 \\ -.35 \\ .30 \end{bmatrix}$ |
| $\begin{bmatrix} 0 & 0 & .90 & .20 \\ 0 & 0 & .10 & .80 \\ .90 & .20 & 0 & 0 \\ .10 & .80 & 0 & 0 \end{bmatrix}$ | $1$ <br> $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix} \begin{bmatrix} .63 \\ .32 \\ .63 \\ .32 \end{bmatrix}$ | $0.70$ <br> $\begin{bmatrix} -.32 \\ .63 \\ -.32 \\ .63 \end{bmatrix} \begin{bmatrix} .50 \\ -.50 \\ .50 \\ -.50 \end{bmatrix}$ | $-0.70$ <br> $\begin{bmatrix} .32 \\ -.63 \\ -.32 \\ .63 \end{bmatrix} \begin{bmatrix} -.50 \\ .50 \\ .50 \\ -.50 \end{bmatrix}$ | $-1$ <br> $\begin{bmatrix} .50 \\ .50 \\ -.50 \\ -.50 \end{bmatrix} \begin{bmatrix} .63 \\ .32 \\ -.63 \\ -.32 \end{bmatrix}$ |
| $\begin{bmatrix} 0 & .01 & .88 & .20 \\ 0 & 0 & .10 & .80 \\ .90 & .20 & 0 & 0 \\ .10 & .79 & .02 & 0 \end{bmatrix}$ | $1$ <br> $\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix} \begin{bmatrix} .62 \\ .33 \\ .63 \\ .34 \end{bmatrix}$ | $0.68$ <br> $\begin{bmatrix} .34 \\ -.62 \\ .33 \\ -.63 \end{bmatrix} \begin{bmatrix} .50 \\ -.50 \\ .50 \\ -.50 \end{bmatrix}$ | $-0.70$ <br> $\begin{bmatrix} -.30 \\ .64 \\ .31 \\ -.64 \end{bmatrix} \begin{bmatrix} .51 \\ -.49 \\ -.51 \\ .49 \end{bmatrix}$ | $-0.98$ <br> $\begin{bmatrix} -.50 \\ -.51 \\ .49 \\ .51 \end{bmatrix} \begin{bmatrix} -.63 \\ -.31 \\ .64 \\ .30 \end{bmatrix}$ |

Table B.3. I
llustrative 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.

*Examples*

## B.8 Some numbers

| | | | |
|---|---|---|---|
| | $2^{8192}$ | $10^{2466}$ | Number of 1 kilobyte files |
| | $2^{1024}$ | $10^{308}$ | Number of states of a 2D Ising model with $32\times32$ spins |
| $2^{1000}$ | | $10^{301}$ | Number of binary strings of length 1000 |
| $2^{500}$ | | $3\times10^{150}$ | |
| | $2^{469}$ | $10^{141}$ | Number of binary strings of length 1000 having 100 `1`s and 900 `0`s |
| | $2^{266}$ | $10^{80}$ | Number of electrons in universe |
| $2^{200}$ | | $1.6\times10^{60}$ | |
| | $2^{190}$ | $10^{57}$ | Number of electrons in solar system |
| | $2^{171}$ | $3\times10^{51}$ | Number of electrons in the earth |
| $2^{100}$ | | $10^{30}$ | |
| | $2^{98}$ | $3\times10^{29}$ | Age of universe / picoseconds |
| | $2^{58}$ | $3\times10^{17}$ | Age of universe / seconds |
| $2^{50}$ | | $10^{15}$ | |
| $2^{40}$ | | $10^{12}$ | |
| | | $3\times10^{10}$ | Number of bits in the wheat genome |
| | | $6\times10^{9}$ | Number of bits to list one human genome |
| | $2^{32}$ | $5\times10^{9}$ | Population of earth |
| $2^{30}$ | | $10^{9}$ | |
| | | $2\times10^{8}$ | Number of bits in *C. Elegans*[3] genome |
| | | $2\times10^{8}$ | Number of bits in *Arabidopsis thaliana*[4] genome |
| | $2^{25}$ | $3\times10^{7}$ | One year / seconds |
| | | $2\times10^{7}$ | Number of bits in the compressed postscript file that is this book |
| | | $2\times10^{7}$ | Number of bits in `unix` kernel |
| | | $10^{7}$ | Number of bits to list one *E. Coli* genome |
| | | $6\times10^{6}$ | Number of years since human/chimpanzee divergence |
| $2^{20}$ | | $10^{6}$ | 1048576 |
| | | $2\times10^{5}$ | Number of generations since human/chimpanzee divergence |
| | | $3\times10^{4}$ | Number of genes in human genome |
| | | $3\times10^{4}$ | Number of genes in *Arabidopsis thaliana* genome |
| | | $1.5\times10^{3}$ | Number of base pairs in a gene |
| $2^{10}$ | | $10^{3}$ | 1024 |
| $2^{0}$ | | $10^{0}$ | 1 |
| $2^{-10}$ | | $10^{-3}$ | |
| $2^{-20}$ | | $10^{-6}$ | |
| | | $3\times10^{-8}$ | 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*

Abramson, N. (1963) *Information theory and coding*. New York: McGraw-Hill.

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.

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.

Baldwin, J. (1896) A new factor in evolution. *American Naturalist* **30**: 441–451.

Baum, E., Boneh, D., and Garrett, C. (1995) On genetic algorithms. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pp. 230–239, New York. ACM.

Baum, E. B., and Smith, W. D. (1993) Best play for imperfect players and game tree search. Technical report, NEC, Princeton, NJ.

Baum, E. B., and Smith, W. D. (1997) A Bayesian approach to relevance in game playing. *Artificial Intelligence* **97** (1-2): 195–242.

Bell, A. J., and Sejnowski, T. J. (1995) An information maximization approach to blind separation and blind deconvolution. *Neural Computation* **7** (6): 1129–1159.

Bentley, J. (2000) *Programming Pearls*. Reading, Massachusetts: Addison-Wesley, second edition.

Berger, J. (1985) *Statistical Decision theory and Bayesian Analysis*. Springer.

Berlekamp, E. R. (1980) The technology of error–correcting codes. *IEEE Transactions on Information Theory*.

Berlekamp, E. R., McEliece, R. J., and van Tilborg, H. C. A. (1978) On the intractability of certain coding problems. *IEEE Transactions on Information Theory* **24** (3): 384–386.

Berrou, C., and Glavieux, A. (1996) Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications* **44**: 1261–1271.

Bottou, L., Howard, P. G., and Bengio, Y. (1998) The Z-coder adaptive binary coder. In *Proceedings of the Data Compression Conference, Snowbird, Utah, March 1998*, pp. 13–22.

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.

Bulmer, M. (1985) *The Mathematical Theory of Quantitative Genetics*. Oxford: Oxford University Press.

Burrows, M., and Wheeler, D. J. (1994) A block-sorting lossless data compression algorithm. Technical report, Digital SRC. Research Report 124. 10th May 1994.

Byers, J., Luby, M., Mitzenmacher, M., and Rege, A. (1998) A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM '98, September 2-4, 1998*.

Carroll, L. (1998) *Alice's adventures in Wonderland; and, Through the looking-glass: and what Alice found there*. London: Macmillan Children's Books.

Childs, A. M., Patterson, R. B., and MacKay, D. J. C. (2001) Exact sampling from non-attractive distributions using summary states. *Physical Review E*.

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. (2001) Reliable communication over channels with insertions, deletions and substitutions. *IEEE Transactions on Information Theory* **47** (2): 687–698.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995) The Helmholtz machine. *Neural Computation* **7** (5): 889–904.

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.

Eyre-Walker, A., and Keightley, P. (1999) High genomic deleterious mutation rates in hominids. *Nature* **397**: 344–347.

Felsenstein, J. (1985) Recombination and sex: is Maynard Smith necessary? In *Evolution. Essays in honour of John Maynard Smith*, ed. by P. J. Greenwood, P. H. Harvey, and M. Slatkin, pp. 209–220. Cambridge: C.U.P.

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.

Fisher, R. A. (1930) *The genetical theory of natural selection*. Oxford: Clarendon.

Forney, Jr., G. D. (2001) Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory* **47** (2): 520–548.

Gallager, R. G. (1962) Low density parity check codes. *IRE Trans. Info. Theory* **IT-8**: 21–28.

Gallager, R. G. (1963) *Low Density Parity Check Codes*. Number 21 in Research monograph series. Cambridge, Mass.: MIT Press.

Gallager, R. G. (1968) *Information Theory and Reliable Communication*. New York: Wiley.

Gibbs, M. N., and MacKay, D. J. C. (2000) Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks* **11** (6): 1458–1464.

Gilks, W., Roberts, G., and George, E. (1994) Adaptive direction sampling. *Statistician* **43**: 179–189.

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. (1991a) Bayesian classification theory. Technical Report FIA–90-12-7-01, NASA Ames.

Hanson, R., Stutz, J., and Cheeseman, P. (1991b) 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.

Harvey, M., and Neal, R. M., (2000) Inference for belief networks using coupling from the past. submitted to UAI 2000.

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., (2000) Training products of experts by minimizing contrastive divergence.

Hinton, G., and Nowlan, S. (1987) How learning can guide evolution. *Complex Systems* **1**: 495–502.

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.

Jaakkola, T. S., and Jordan, M. I. (1996a) Bayesian logistic regression: a variational approach. Technical report, MIT.

Jaakkola, T. S., and Jordan, M. I. (1996b) Computing upper and lower bounds on likelihoods in intractable networks. In *Proceedings of the Twelfth Conference on Uncertainty in AI*. Morgan Kaufman.

Jaakkola, T. S., and Jordan, M. I. (2000) Bayesian parameter estimation via variational methods. *Statistics and Computing* **10** (1): 25–37.

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.

Kelling, M. J., and Rand, D. A. (1995) A spatial mechanism for the evolution and maintenance of sexual reproduction. *Oikos* **74**: 414–424.

Kimura, M. (1961) Natural selection as the process of accumulating genetic information in adaptive evolution. *Genetical Research Cambridge*.

Kondrashov, A. S. (1988) Deleterious mutations and the evolution of sexual reproduction. *Nature* **336** (6198): 435–440.

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.

Loredo, T. J. (1990) From Laplace to supernova SN 1987A: Bayesian inference in astrophysics. In *Maximum Entropy and Bayesian Methods, Dartmouth, U.S.A., 1989*, ed. by P. Fougere, pp. 81–142. Kluwer.

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.

Luttrell, S. P. (1989) Hierarchical vector quantisation. *Proc. IEE Part I* **136**: 405–413.

Luttrell, S. P. (1990) Derivation of a class of training algorithms. *IEEE Trans. on Neural Networks* **1** (2): 229–232.

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. (1995) Free energy minimization algorithm for decoding and cryptanalysis. *Electronics Letters* **31** (6): 446–447.

MacKay, D. J. C., (1997) Ensemble learning for hidden Markov models. `http://www.inference.phy.cam.ac.uk/mackay/abstracts/ensemblePaper.html`.

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., (2000) An alternative to runlength-limiting codes: Turn timing errors into substitution errors. available from `http://www.inference.phy.cam.ac.uk/mackay/`.

MacKay, D. J. C., (2001) A problem with variational free energy minimization. `http://www.inference.phy.cam.ac.uk/mackay/abstracts/minima.html`.

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.

MacKay, D. J. C., and Peto, L. (1995) A hierarchical Dirichlet language model. *Natural Language Engineering* **1** (3): 1–19.

Marinari, E., and Parisi, G. (1992) Simulated tempering — a new Monte-Carlo scheme. *Europhysics Letters* **19** (6): 451–458.

Maynard Smith, J. (1968) “Haldane’s dilemma” and the rate of evolution. *Nature* **219** (5159): 1114–1116.

Maynard Smith, J. (1978) *The Evolution of Sex*. Cambridge: C.U.P.

Maynard Smith, J. (1988) *Games, Sex and Evolution*. Hertfordshire: Harvester–Wheatsheaf.

Maynard Smith, J., and Száthmary, E. (1995) *The Major Transitions in Evolution*. Oxford: Freeman.

McEliece, R. J. (1977) *The Theory of Information and Coding: A Mathematical Framework for Communication*. Reading, Mass.: Addison-Wesley.

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.

Mühlenbein, H., and Schlierkamp-Voosen, D. (1993) Predictive models for the breeder genetic algorithm I. Continuous parameter optimization. *Evolutionary Computation* **1**: 25–50.

Neal, R. M. (1993) 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. (2001) Monte Carlo decoding of LDPC codes. Technical report, Dept. of Computer Science, University of Toronto. Presented at ICTP Workshop on Statistical Physics and Capacity-Approaching Codes.

Neal, R. M. (2002) Slice sampling. *Annals of Statistics*. In Press.

NEAL, R. M., and HINTON, G. E. (1993) A new view of the EM algorithm that justifies incremental, sparse, and other variants. *Biometrika*. submitted.

PINTO, R. L., and NEAL, R. M. (2001) Improving Markov chain Monte Carlo estimators by coupling to an approximating chain. Technical Report 0101, Dept. of Statistics, University of Toronto.

POLYA, G. (1954) *Induction and Analogy in Mathematics*. New Jersey: Princeton University Press. Volume 1 of Mathematics and Plausible Reasoning.

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.

REIF, F. (1965) *Fundamentals of Statistical and Thermal Physics*. McGraw–Hill.

RICHARDSON, T., SHOKROLLAHI, M. A., and URBANKE, R. (2001) Design of capacity-approaching irregular low-density parity check codes. *IEEE Transactions on Information Theory* **47** (2): 619–637.

RIDLEY, M. (1993) *The Red Queen*. London: Penguin.

RUMELHART, D. E., HINTON, G. E., and WILLIAMS, R. J. (1986) Learning representations by back–propagating errors. *Nature* **323**: 533–536.

RUSSELL, S., and WEFALD, E. (1991) *Do the Right Thing: Studies in Limited Rationality*. MIT Press.

SCHNEIER, B. (1996) *Applied Cryptography*. New York: Wiley.

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.

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., and MACKAY, D. J. (2001) Slice sampling – a binary implementation. *Annals of Statistics*. to appear.

SPIEGEL, M. R. (1988) *Statistics*. Schaum's outline series. New York: McGraw-Hill, 2 edition.

SPIELMAN, D. A. (1996) Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory* **42** (6.1): 1723–1731.

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.

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.

Ward, D. J., Blackwell, A. F., and MacKay, D. J. C. (2000) Dasher - A data entry interface using continuous gestures and language models. In *Proceedings of UIST 2000*, pp. 129–137.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987) Arithmetic coding for data compression. *Communications of the ACM* **30** (6): 520–540.

Worden, R. P. (1995) A speed limit for evolution. *Journal of Theoretical Biology* **176** (1): 137–152.

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*