

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)?

## ► 12.7 Solutions

**Solution to exercise 12.1 (p.194).** 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 2.34, p.38).

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 are very similar to the search key, so that a lengthy sequence of comparisons is needed to find each 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 12.2 (p.197).** 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

$$\frac{P(\text{Datum} | \mathcal{H}_0)}{P(\text{Datum} | \mathcal{H}_1)} = \frac{1}{1/2} = 2. \quad (12.5)$$

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 evidence given by the prior information that the hash entry  $s$  has been found in the table at  $\mathbf{h}(\mathbf{x})$ . This fact gives further evidence in favour of  $\mathcal{H}_0$ .]

**Solution to exercise 12.3 (p.198).** Let the hash function have an output alphabet of size  $T = 2^M$ . 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 particular pair of entries collide under a random hash function is  $1/T$ . The number of pairs is  $S(S-1)/2$ . So the expected number of collisions between pairs is exactly

$$S(S-1)/(2T). \quad (12.6)$$

If we would like this to be smaller than 1, then we need  $T > S(S-1)/2$  so

$$M > 2 \log_2 S. \quad (12.7)$$

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  $T > S/f$  (since the probability that one particular name is collided-with is  $f \simeq S/T$ ) so

$$M > \log_2 S + \log_2[1/f], \quad (12.8)$$

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  $T$  with  $S$  in the two cases (12.7, 12.8). If we want the hash function to be collision-free, then we must have  $T$  greater than  $\sim S^2$ . If we are happy to have a small frequency of collisions, then  $T$  needs to be of order  $S$  only.

**Solution to exercise 12.5 (p.198).** 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 12.7 (p.199).** 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-check bits used by a successful error-correcting code would have to be at least this number, by the counting argument of exercise 1.10 (solution, p.20).

**Solution to exercise 12.10 (p.201).** 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; including duration and stress will increase it again. [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?] The number of recorded compositions is probably less than a million. 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 12.11 (p.201).** (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 that there is no other occurrence of the target sequence in the whole genome, of length  $N$  nucleotides, is roughly

$$(1 - (1/4)^L)^N \simeq \exp(-N(1/4)^L), \quad (12.9)$$

which is close to one only if

$$N4^{-L} \ll 1, \quad (12.10)$$

that is,

$$L > \log N / \log 4. \quad (12.11)$$

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 3.4 nm, so a contiguous sequence of sixteen nucleotides has a length of 5.4 nm. The diameter of the protein must therefore be about 5.4 nm or greater. Egg-white lysozyme is a small globular protein with a length of 129 amino acids and a diameter of about 4 nm. Assuming that volume is proportional to sequence length and that volume scales as the cube of the diameter, a protein of diameter 5.4 nm 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 that recognizes only the sub-sequence, and that binds to the DNA strongly only if it can form a *dimer*, both halves of which are bound to the recognized sequence. Halving the diameter of the protein, we now only need a protein whose length is greater than  $324/8 = 40$  amino acids. A protein of length smaller than this cannot by itself serve as a regulatory protein specific to one gene, because it’s simply too small to be able to make a sufficiently specific match – its available surface does not have enough information content.

In *guess that tune*, one player chooses a melody, and sings a gradually-increasing number of its notes, while the other participants try to guess the whole melody.

The *Parsons code* is a related hash function for melodies: each pair of consecutive notes is coded as U (‘up’) if the second note is higher than the first, R (‘repeat’) if the pitches are equal, and D (‘down’) otherwise. You can find out how well this hash function works at <http://musipedia.org/>.

---

## About Chapter 13

In Chapters 8–11, we 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 the special case of channels with binary inputs. Constraining ourselves to these channels simplifies matters, and leads us into an exceptionally rich world, which we will only taste in this book.

One of the aims of this chapter is to point out a contrast between Shannon’s aim of achieving reliable communication over a noisy channel and the apparent aim of many in the world of coding theory. Many coding theorists take as their fundamental problem the task of packing as many spheres as possible, with radius as large as possible, into an  $N$ -dimensional space, *with no spheres overlapping*. Prizes are awarded to people who find packings that squeeze in an extra few spheres. While this is a fascinating mathematical topic, we shall see that the aim of maximizing the distance between codewords in a code has only a tenuous relationship to Shannon’s aim of reliable communication.

# 13

## 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 the special case of 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.

### ► 13.1 Distance properties of a code

The *distance* of a code is the smallest separation between two of its codewords.

**Example 13.1.** The (7, 4) Hamming code (p.8) has distance  $d = 3$ . All pairs of its codewords differ in at least 3 bits. The maximum 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  $d_{\min}$ .

We'll now constrain our attention to linear codes. In a linear code, all codewords have identical distance properties, so we can summarize all the distances between the code's codewords by counting the distances from the all-zero codeword.

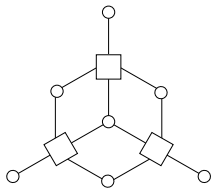
The *weight enumerator function* of a code,  $A(w)$ , is defined to be the number of codewords in the code that have weight  $w$ . The weight enumerator function is also known as the *distance distribution* of the code.

**Example 13.2.** The weight enumerator functions of the (7, 4) Hamming code and the dodecahedron code are shown in figures 13.1 and 13.2.

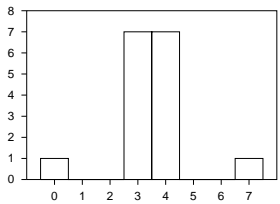
### ► 13.2 Obsession with distance

Since the maximum 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.

**Example:**  
 The Hamming distance  
 between 00001111  
 and 11001101  
 is 3.



$w$	$A(w)$
0	1
3	7
4	7
7	1
Total	16



**Figure 13.1.** The graph of the (7, 4) Hamming code, and its weight enumerator function.

$d = 2t + 1$  if  $d$  is odd, and  
 $d = 2t + 2$  if  $d$  is even.

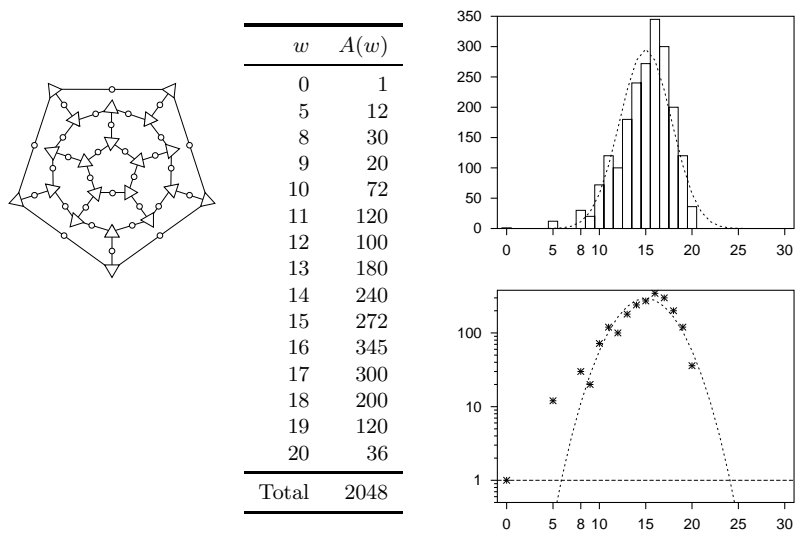


Figure 13.2. The graph defining the (30, 11) dodecahedron code (the circles are the 30 transmitted bits and the triangles are the 20 parity checks, one of which is redundant) and the weight enumerator function (solid lines). The dotted lines show the average weight enumerator function of all random linear codes with the same size of generator matrix, which will be computed shortly. The lower figure shows the same functions on a log scale.

**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 reach 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.

*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.183):

- 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.

**Example 13.3.** A *low-density generator-matrix code* is a linear code whose  $K \times N$  generator matrix  $\mathbf{G}$  has a small number  $d_0$  of 1s per row, regardless of how big  $N$  is. The minimum distance of such a code is at most  $d_0$ , so low-density generator-matrix codes have ‘very bad’ distance.

While having large distance is no bad thing, we’ll see, later on, why an emphasis on distance can be unhealthy.

\*

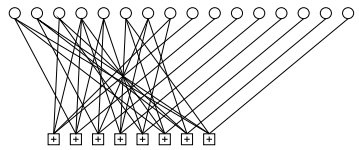


Figure 13.3. The graph of a rate- $1/2$  low-density generator-matrix code. The rightmost  $M$  of the transmitted bits are each connected to a single distinct parity constraint. The leftmost  $K$  transmitted bits are each connected to a small number of parity constraints.

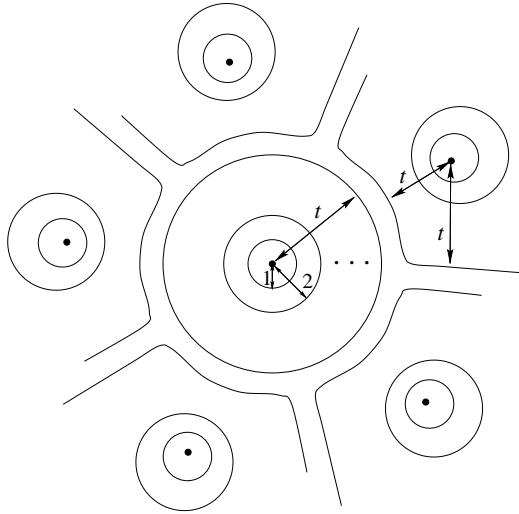


Figure 13.4. Schematic picture of part of Hamming space perfectly filled by  $t$ -spheres centred on the codewords of a perfect code.

### ► 13.3 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 13.4.)

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

$$\sum_{w=0}^t \binom{N}{w}. \quad (13.1)$$

For a code to be perfect with these parameters, we require  $S$  times the number of points in the  $t$ -sphere to equal  $2^N$ :

$$\text{for a perfect code, } 2^K \sum_{w=0}^t \binom{N}{w} = 2^N \quad (13.2)$$

$$\text{or, equivalently, } \sum_{w=0}^t \binom{N}{w} = 2^{N-K}. \quad (13.3)$$

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

$$1 + \binom{7}{1} = 2^3. \quad (13.4)$$

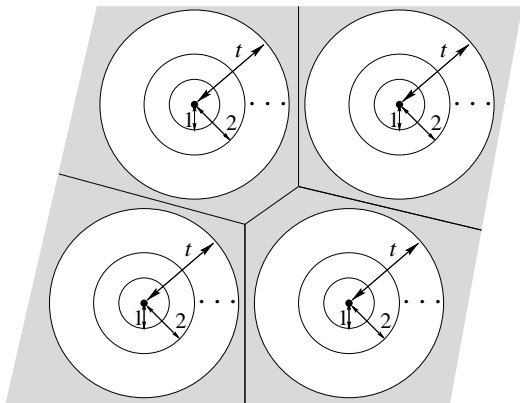


Figure 13.5. Schematic picture of Hamming space not perfectly filled by  $t$ -spheres centred on the codewords of a code. The grey regions show points that are at a Hamming distance of more than  $t$  from any codeword. This is a misleading picture, as, for any code with large  $t$  in high dimensions, the grey space between the spheres takes up almost all of Hamming space.

*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 precise numerological coincidences like those satisfied by the parameters of the Hamming code (13.4) and the Golay code,

$$1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2^{11}, \quad (13.5)$$

are rare? Are there plenty of 'almost-perfect' codes for which the  $t$ -spheres fill *almost* the whole space?

No. In fact, the picture of Hamming spheres centred on the codewords *almost* filling Hamming space (figure 13.5) 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 13.5).

Having established this gloomy picture, we spend a moment filling in the properties of the perfect codes mentioned above.



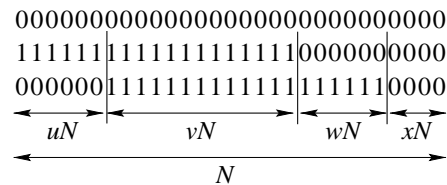


Figure 13.6. Three codewords.

*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 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 13.4.** [2, p.223] 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  $f$ ?

► **13.4 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 be 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 13.6. 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

### 13.5: Weight enumerator function of random linear codes

211

than  $2fN$ , so

$$u + v > 2f, \quad v + w > 2f, \quad \text{and} \quad u + w > 2f. \quad (13.6)$$

Summing these three inequalities and dividing by two, we have

$$u + v + w > 3f. \quad (13.7)$$

So if  $f > 1/3$ , we can deduce  $u + v + w > 1$ , so that  $x < 0$ , which is impossible. Such a code cannot exist. So the code cannot have *three* codewords, let alone  $2^{NR}$ .

We conclude that, whereas Shannon proved there are plenty of codes for communicating over a binary symmetric channel with  $f > 1/3$ , *there are no perfect codes that can do this.*

We now study a more general argument that indicates that there are no large perfect linear codes for general rates (other than 0 and 1). We do this by finding the typical distance of a random linear code.

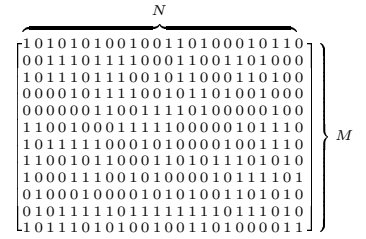


Figure 13.7. A random binary parity-check matrix.

### ► 13.5 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?

The weight enumerator of one particular code with parity-check matrix  $\mathbf{H}$ ,  $A(w)_{\mathbf{H}}$ , is the number of codewords of weight  $w$ , which can be written

$$A(w)_{\mathbf{H}} = \sum_{\mathbf{x}: |\mathbf{x}|=w} \mathbb{1}[\mathbf{H}\mathbf{x} = 0], \quad (13.8)$$

where the sum is over all vectors  $\mathbf{x}$  whose weight is  $w$  and the truth function  $\mathbb{1}[\mathbf{H}\mathbf{x} = 0]$  equals one if  $\mathbf{H}\mathbf{x} = 0$  and zero otherwise.

We can find the expected value of  $A(w)$ ,

$$\langle A(w) \rangle = \sum_{\mathbf{H}} P(\mathbf{H}) A(w)_{\mathbf{H}} \quad (13.9)$$

$$= \sum_{\mathbf{x}: |\mathbf{x}|=w} \sum_{\mathbf{H}} P(\mathbf{H}) \mathbb{1}[\mathbf{H}\mathbf{x} = 0], \quad (13.10)$$

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 that each of the  $M$  bits in the syndrome is zero. 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

$$\sum_{\mathbf{H}} P(\mathbf{H}) \mathbb{1}[\mathbf{H}\mathbf{x} = 0] = (1/2)^M = 2^{-M}, \quad (13.11)$$

independent of  $w$ .

The expected number of words of weight  $w$  (13.10) 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

$$\langle A(w) \rangle = \binom{N}{w} 2^{-M} \quad \text{for any } w > 0. \quad (13.12)$$

For large  $N$ , we can use  $\log \binom{N}{w} \simeq NH_2(w/N)$  and  $R \simeq 1 - M/N$  to write

$$\log_2 \langle A(w) \rangle \simeq NH_2(w/N) - M \quad (13.13)$$

$$\simeq N[H_2(w/N) - (1 - R)] \text{ for any } w > 0. \quad (13.14)$$

As a concrete example, figure 13.8 shows the expected weight enumerator function of a rate-1/3 random linear code with  $N = 540$  and  $M = 360$ .

#### *Gilbert–Varshamov 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 linear code will be close to the distance  $d_{GV}$  defined by

$$H_2(d_{GV}/N) = (1 - R). \quad (13.15)$$

**Definition.** This distance,  $d_{GV} \equiv NH_2^{-1}(1 - R)$ , is 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  $d_{GV}$ .

**Definition.** The *Gilbert–Varshamov rate*  $R_{GV}$  is the maximum rate at which you can reliably communicate with a bounded-distance decoder (as defined on p.207), assuming that the Gilbert–Varshamov conjecture is true.

#### *Why sphere-packing is a bad perspective, and an obsession with distance is inappropriate*

If one uses a bounded-distance decoder, the maximum tolerable noise level will flip a fraction  $f_{bd} = \frac{1}{2}d_{min}/N$  of the bits. So, assuming  $d_{min}$  is equal to the Gilbert distance  $d_{GV}$  (13.15), we have:

$$H_2(2f_{bd}) = (1 - R_{GV}). \quad (13.16)$$

$$R_{GV} = 1 - H_2(2f_{bd}). \quad (13.17)$$

Now, here's the crunch: what did Shannon say is achievable? He said the maximum possible rate of communication is the capacity,

$$C = 1 - H_2(f). \quad (13.18)$$

So for a given rate  $R$ , the maximum tolerable noise level, according to Shannon, is given by

$$H_2(f) = (1 - R). \quad (13.19)$$

Our conclusion: imagine a good code of rate  $R$  has been chosen; equations (13.16) and (13.19) respectively define the maximum noise levels tolerable by a bounded-distance decoder,  $f_{bd}$ , and by Shannon's decoder,  $f$ .

$$f_{bd} = f/2. \quad (13.20)$$

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 that fill Hamming space without overlapping.

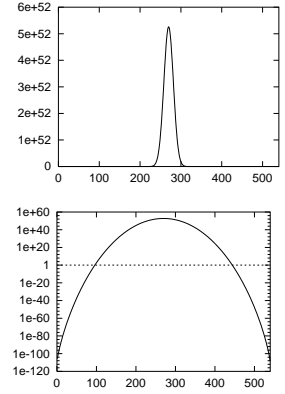


Figure 13.8. The expected weight enumerator function  $\langle A(w) \rangle$  of a random linear code with  $N = 540$  and  $M = 360$ . Lower figure shows  $\langle A(w) \rangle$  on a logarithmic scale.

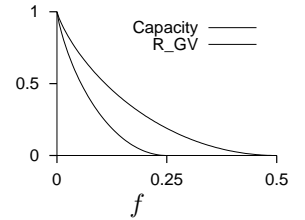


Figure 13.9. Contrast between Shannon's channel capacity  $C$  and the Gilbert rate  $R_{GV}$  – the maximum communication rate achievable using a bounded-distance decoder, as a function of noise level  $f$ . For any given rate,  $R$ , the maximum tolerable noise level for Shannon is twice as big as the maximum tolerable noise level for a 'worst-case-ist' who uses a bounded-distance decoder.

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 13.10, 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.

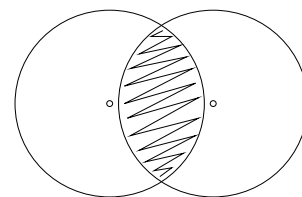


Figure 13.10. Two overlapping spheres whose radius is almost as big as the distance between their centres.

### ► 13.6 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 (figure 13.11). Each stalactite is analogous to the boundary between the home codeword and another codeword. The stalactite is like the shaded region in figure 13.10, 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, the bat is usually very close to the centre of the cave; 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.

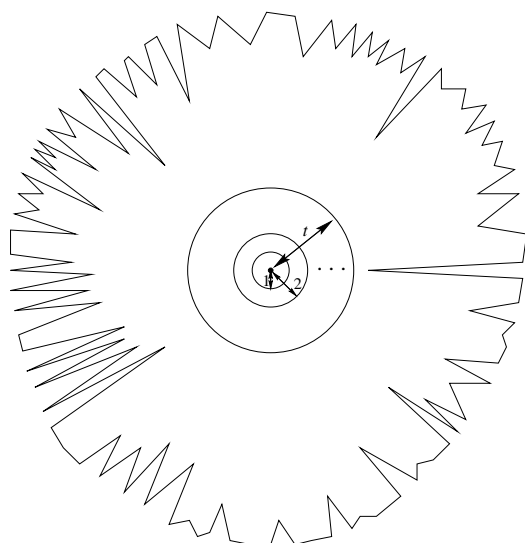


Figure 13.11. Berlekamp's schematic picture of Hamming space in the vicinity of a codeword. The jagged solid line encloses all points to which this codeword is the closest. The  $t$ -sphere around the codeword takes up a small fraction of this space.

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. Under these conditions, the bat's collision frequency has nothing to do with the distance from the centre to the closest stalactite.

### ► 13.7 Concatenation of Hamming codes

It is instructive to play some more with the concatenation of Hamming codes, a concept we first visited in figure 11.6, 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  $N$ .

$N = 2^M - 1$	blocklength
$K = N - M$	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

$$R_C = \prod_{c=1}^C \frac{N_c - M_c}{N_c} \quad (13.21)$$

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 (figure 13.12).

The blocklength  $N$  is a rapidly-growing function of  $C$ , so these codes are somewhat impractical. A further weakness of these codes is that their minimum distance is not very good (figure 13.13). 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 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 thus 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 11.4 to recall the definitions of the terms 'good' and 'very good'). Rather than prove this result, we will simply explore it numerically.

Figure 13.14 shows the bit error probability  $p_b$  of the concatenated codes assuming that the constituent codes are decoded in sequence, as described in section 11.4. [This one-code-at-a-time decoding is suboptimal, as we saw there.] The horizontal axis shows the rates of the codes. As the number of concatenations increases, the rate drops to 0.093 and the error probability drops towards zero. The channel assumed in the figure is the binary symmetric

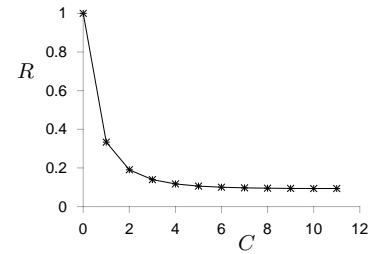


Figure 13.12. The rate  $R$  of the concatenated Hamming code as a function of the number of concatenations,  $C$ .

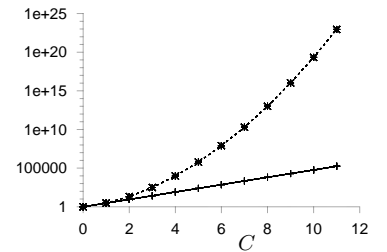


Figure 13.13. The blocklength  $N_C$  (upper curve) and minimum distance  $d_C$  (lower curve) of the concatenated Hamming code as a function of the number of concatenations  $C$ .

### 13.8: Distance isn't everything

215

channel with  $f = 0.0588$ . This is the highest noise level that can be tolerated using this concatenated 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 to Shannon's mission of achieving reliable communication over noisy channels.

- ▷ Exercise 13.5.<sup>[3]</sup> Prove that there exist families of codes with 'bad' distance that are 'very good' codes.

### ► 13.8 Distance isn't everything

Let's get a quantitative feeling for the effect of the minimum distance of a code, for the special case of a binary symmetric channel.

#### *The error probability associated with one low-weight codeword*

Let a binary code have blocklength  $N$  and just two codewords, which differ in  $d$  places. For simplicity, let's assume  $d$  is even. What is the error probability if this code is used on a binary symmetric channel with noise level  $f$ ?

Bit flips matter only in places where the two codewords differ. The error probability is dominated by the probability that  $d/2$  of these bits are flipped. What happens to the other bits is irrelevant, since the optimal decoder ignores them.

$$P(\text{block error}) \simeq \binom{d}{d/2} f^{d/2} (1-f)^{d/2}. \quad (13.22)$$

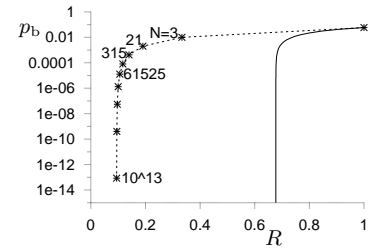
This error probability associated with a single codeword of weight  $d$  is plotted in figure 13.15. Using the approximation for the binomial coefficient (1.16), we can further approximate

$$P(\text{block error}) \simeq \left[ 2f^{1/2}(1-f)^{1/2} \right]^d \quad (13.23)$$

$$\equiv [\beta(f)]^d, \quad (13.24)$$

where  $\beta(f) = 2f^{1/2}(1-f)^{1/2}$  is called the Bhattacharyya parameter of the channel.

Now, consider a general linear code with distance  $d$ . Its block error probability must be at least  $\binom{d}{d/2} f^{d/2} (1-f)^{d/2}$ , independent of the blocklength  $N$  of the code. For this reason, a sequence of codes of increasing blocklength  $N$  and constant distance  $d$  (i.e., 'very bad' distance) cannot have a block error probability that tends to zero, on any binary symmetric channel. If we are interested in making superb error-correcting codes with tiny, tiny error probability, we might therefore shun codes with bad distance. However, being pragmatic, we should look more carefully at figure 13.15. In Chapter 1 we argued that codes for disk drives need an error probability smaller than about  $10^{-18}$ . If the raw error probability in the disk drive is about 0.001, the error probability associated with one codeword at distance  $d = 20$  is smaller than  $10^{-24}$ . If the raw error probability in the disk drive is about 0.01, the error probability associated with one codeword at distance  $d = 30$  is smaller than  $10^{-20}$ . For practical purposes, therefore, it is not essential for a code to have good distance. For example, codes of blocklength 10 000, known to have many codewords of weight 32, can nevertheless correct errors of weight 320 with tiny error probability.



I wouldn't want you to think I am *recommending* the use of codes with bad distance; in Chapter 47 we will discuss low-density parity-check codes, my favourite codes, which have both excellent performance and *good* distance.

### ► 13.9 The union bound

The error probability of a code on the binary symmetric channel can be bounded in terms of its weight enumerator function by adding up appropriate multiples of the error probability associated with a single codeword (13.24):

$$P(\text{block error}) \leq \sum_{w>0} A(w)[\beta(f)]^w. \quad (13.25)$$

This inequality, which is an example of a *union bound*, is accurate for low noise levels  $f$ , but inaccurate for high noise levels, because it overcounts the contribution of errors that cause confusion with more than one codeword at a time.

▷ Exercise 13.6.<sup>[3]</sup> Poor man's noisy-channel coding theorem.

Pretending that the union bound (13.25) is accurate, and using the average weight enumerator function of a random linear code (13.14) (section 13.5) as  $A(w)$ , estimate the maximum rate  $R_{\text{UB}}(f)$  at which one can communicate over a binary symmetric channel.

Or, to look at it more positively, using the union bound (13.25) as an inequality, show that communication at rates up to  $R_{\text{UB}}(f)$  is possible over the binary symmetric channel.

In the following chapter, by analysing the probability of error of *syndrome decoding* for a binary linear code, and using a union bound, we will prove Shannon's noisy-channel coding theorem (for symmetric binary channels), and thus show that *very good linear codes exist*.

### ► 13.10 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 p.10) is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad (13.26)$$

and its sixteen codewords were displayed in table 1.14 (p.9). 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

$$\mathbf{H}\mathbf{t} = \mathbf{0}. \quad (13.27)$$

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.

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 p.12):

$$\mathbf{H} = [\mathbf{P} \quad \mathbf{I}_3] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \quad (13.28)$$

The dual of the (7, 4) Hamming code  $\mathcal{H}_{(7,4)}$  is the code shown in table 13.16.

0000000	0101101	1001110	1100011
0010111	0111010	1011001	1110100

Table 13.16. The eight codewords of the dual of the (7, 4) Hamming code. [Compare with table 1.14, p.9.]

A possibly unexpected property of this pair of codes is that the dual,  $\mathcal{H}_{(7,4)}^\perp$ , 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:

$$\mathcal{H}_{(7,4)}^\perp \subset \mathcal{H}_{(7,4)}. \quad (13.29)$$

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? But when we work in modulo-two arithmetic, many non-zero vectors are indeed orthogonal to themselves!

- ▷ **Exercise 13.7.** [1, p.223] 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,

$$\mathbf{G} = [\mathbf{I}_K | \mathbf{P}^T], \quad (13.30)$$

where  $\mathbf{P}$  is a  $K \times M$  matrix, then its parity-check matrix is

$$\mathbf{H} = [\mathbf{P} | \mathbf{I}_M]. \quad (13.31)$$



Example 13.8. The repetition code  $R_3$  has generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}; \quad (13.32)$$

its parity-check matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}. \quad (13.33)$$

The two codewords are  $[1 \ 1 \ 1]$  and  $[0 \ 0 \ 0]$ .

The dual code has generator matrix

$$\mathbf{G}^\perp = \mathbf{H} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad (13.34)$$

or equivalently, modifying  $\mathbf{G}^\perp$  into systematic form by row additions,

$$\mathbf{G}^\perp = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \quad (13.35)$$

We call this dual code the *simple parity code*  $P_3$ ; 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. The last category is especially important: 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 13.9.<sup>[3]</sup> 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, then use the argument from p.215.]

Exercise 13.10.<sup>[5]</sup> Show that low-density parity-check codes are good, and have good distance. (For solutions, see Gallager (1963) and MacKay (1999b).)

### Self-dual codes

The  $(7, 4)$  Hamming code had the property that the dual was contained in the code itself. A code is *self-orthogonal* if it is contained in its dual. For example, the dual of the  $(7, 4)$  Hamming code is a self-orthogonal code. One way of seeing this is that the overlap between any pair of rows of  $\mathbf{H}$  is even. Codes that contain their duals are important in quantum error-correction (Calderbank and Shor, 1996).

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.

$$\mathcal{C}^\perp = \mathcal{C}. \quad (13.36)$$

Some properties of self-dual codes can be deduced:

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 13.11. [2, p.223] What property must the matrix  $\mathbf{P}$  satisfy, if the code with generator matrix  $\mathbf{G} = [\mathbf{I}_K | \mathbf{P}^T]$  is self-dual?

#### Examples of self-dual codes

1. The repetition code  $R_2$  is a simple example of a self-dual code.

$$\mathbf{G} = \mathbf{H} = \begin{bmatrix} 1 & 1 \end{bmatrix}. \quad (13.37)$$

2. The smallest non-trivial self-dual code is the following  $(8, 4)$  code.

$$\mathbf{G} = [\mathbf{I}_4 | \mathbf{P}^T] = \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array} \right]. \quad (13.38)$$

▷ Exercise 13.12. [2, p.223] Find the relationship of the above  $(8, 4)$  code to the  $(7, 4)$  Hamming code.

#### Duals and graphs

Let a code be 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).

#### Further reading

Duals are important in coding theory because functions involving a code (such as the posterior distribution over codewords) can be transformed by a Fourier transform into functions over the dual code. For an accessible introduction to Fourier analysis on finite groups, see Terras (1999). See also MacWilliams and Sloane (1977).

### ► 13.11 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 than  $u$ . Rather than using the word perfect, however, the conventional term for such a code is a 'maximum distance separable code', or MDS code.

As we already noted in exercise 11.10 (p.190), the  $(7, 4)$  Hamming code is *not* an MDS 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.

In a perfect  $u$ -error-correcting code for the binary erasure channel, the number of redundant bits must be  $N - K = u$ .

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 13.13. [5, p.224] 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. (For further reading, see Lin and Costello (1983).)

## ► 13.12 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 that the best codes 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.

The relationship between good codes and distance properties is discussed further in exercise 13.14 (p.220).

## ► 13.13 Further exercises

Exercise 13.14. [3, p.224] 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  $\mathbf{y}$ . We 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.

Consider optimal decoders for these two decoding problems. 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 theorem.

**Theorem 13.1** *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:*

$$p_B \geq p_b \geq \frac{1}{2} \frac{d_{\min}}{N} p_B. \quad (13.39)$$



Exercise 13.15.<sup>[1]</sup> What are the minimum distances of the (15, 11) Hamming code and the (31, 26) Hamming code?

▷ Exercise 13.16.<sup>[2]</sup> 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 13.17.<sup>[3C]</sup> A code with minimum distance greater than  $d_{GV}$ . A rather nice (15, 5) code is generated by this generator matrix, which is based on measuring the parities of all the  $\binom{5}{3} = 10$  triplets of source bits:

$$\mathbf{G} = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & \cdot & 1 & 1 & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot & 1 & 1 & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & \cdot & \cdot & 1 & 1 & \cdot & 1 & \cdot \end{bmatrix}. \quad (13.40)$$

Find the minimum distance and weight enumerator function of this code.

Exercise 13.18.<sup>[3C]</sup> Find the minimum distance of the ‘pentagonful’ low-density parity-check code whose parity-check matrix is

$$\mathbf{H} = \left[ \begin{array}{ccccc|ccccc|ccccc} 1 & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot \end{array} \right]. \quad (13.41)$$

Show that nine of the ten rows are independent, so the code has parameters  $N = 15$ ,  $K = 6$ . Using a computer, find its weight enumerator function.

▷ Exercise 13.19.<sup>[3C]</sup> Replicate the calculations used to produce figure 13.12. Check the assertion that the highest noise level that’s correctable is 0.0588. Explore alternative concatenated sequences of codes. Can you find a better sequence of concatenated codes – better in the sense that it has either higher asymptotic rate  $R$  or can tolerate a higher noise level  $f$ ?

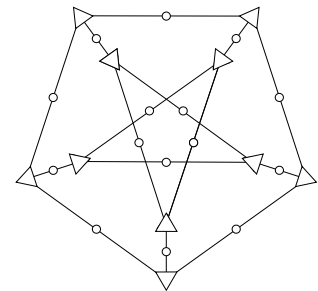


Figure 13.17. The graph of the pentagonful low-density parity-check code with 15 bit nodes (circles) and 10 parity-check nodes (triangles). [This graph is known as the Petersen graph.]



**Exercise 13.20.** [3, p.226] 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  $\mathbf{H}\mathbf{n} = \mathbf{z}$ , 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 13.21.** [2] Linear binary codes use the input symbols 0 and 1 with equal probability, implicitly treating the channel as a symmetric channel. Investigate how much loss in communication rate is caused by this assumption, if in fact the channel is a highly asymmetric channel. Take as an example a Z-channel. How much smaller is the maximum possible rate of communication using symmetric inputs than the capacity of the channel? [Answer: about 6%.]

**Exercise 13.22.** [2] Show that codes with 'very bad' distance are 'bad' codes, as defined in section 11.4 (p.183).

**Exercise 13.23.** [3] One linear code can be obtained from another by *puncturing*. Puncturing means taking each codeword and deleting a defined set of bits. Puncturing turns an  $(N, K)$  code into an  $(N', K)$  code, where  $N' < N$ .

Another way to make new linear codes from old is *shortening*. Shortening means constraining a defined set of bits to be zero, and then deleting them from the codewords. Typically if we shorten by one bit, half of the code's codewords are lost. Shortening typically turns an  $(N, K)$  code into an  $(N', K')$  code, where  $N - N' = K - K'$ .

Another way to make a new linear code from two old ones is to make the *intersection* of the two codes: a codeword is only retained in the new code if it is present in both of the two old codes.

Discuss the effect on a code's distance-properties of puncturing, shortening, and intersection. Is it possible to turn a code family with bad distance into a code family with good distance, or *vice versa*, by each of these three manipulations?

**Exercise 13.24.** [3, p.226] Todd Ebert's 'hat puzzle'.

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**.]

Exercise 13.25.<sup>[5]</sup> Estimate how many binary low-density parity-check codes have self-orthogonal duals. [Note that we don't expect a huge number, since almost all low-density parity-check codes are 'good', but a low-density parity-check code that contains its dual must be 'bad'.]

Exercise 13.26.<sup>[2C]</sup> In figure 13.15 we plotted the error probability associated with a single codeword of weight  $d$  as a function of the noise level  $f$  of a binary symmetric channel. Make an equivalent plot for the case of the Gaussian channel, showing the error probability associated with a single codeword of weight  $d$  as a function of the rate-compensated signal-to-noise ratio  $E_b/N_0$ . Because  $E_b/N_0$  depends on the rate, you have to choose a code rate. Choose  $R = 1/2, 2/3, 3/4$ , or  $5/6$ .

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?

## ► 13.14 Solutions

Solution to exercise 13.4 (p.210). The probability of block error to leading order is  $p_B = \frac{3}{N} \binom{N}{2} f^2$ .

Solution to exercise 13.7 (p.217). A binary vector is perpendicular to itself if it has even weight, i.e., an even number of 1s.

Solution to exercise 13.11 (p.219). The self-dual code has two equivalent parity-check matrices,  $\mathbf{H}_1 = \mathbf{G} = [\mathbf{I}_K | \mathbf{P}^T]$  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

$$[\mathbf{U}\mathbf{P} | \mathbf{U}\mathbf{I}_K] = [\mathbf{I}_K | \mathbf{P}^T]. \quad (13.42)$$

From the right-hand sides of this equation, we have  $\mathbf{U} = \mathbf{P}^T$ , so the left-hand sides become:

$$\mathbf{P}^T \mathbf{P} = \mathbf{I}_K. \quad (13.43)$$

Thus if a code with generator matrix  $\mathbf{G} = [\mathbf{I}_K | \mathbf{P}^T]$  is self-dual then  $\mathbf{P}$  is an *orthogonal* matrix, modulo 2, and *vice versa*.

Solution to exercise 13.12 (p.219). The (8, 4) and (7, 4) codes are intimately related. The (8, 4) code, whose parity-check matrix is

$$\mathbf{H} = [\mathbf{P} | \mathbf{I}_4] = \left[ \begin{array}{cccc|cccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right], \quad (13.44)$$

is obtained by (a) appending an extra parity-check bit which can be thought of as the parity of all seven bits of the (7, 4) Hamming code; and (b) reordering the first four bits.

**Solution to exercise 13.13 (p.220).** 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 C.1. The elements of the input alphabet are  $\{0, 1, A, B, C, D, E, F\}$  and the generator matrix of the code is

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & A & B & C & D & E & F \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (13.45)$$

The resulting 64 codewords are:

000000000	011111111	0AAAAAAA	0BBBBBBB	0CCCCCCC	0DDDDDDD	0EEEEEEE	0FFFFFFF
101ABCDEF	110BADCFE	1AB01EFC	1BA10FED	1CDEF01AB	1DCF01BA	1EFCDA01	1FEDCBA10
A0ACEB1FD	A1BDFAOEC	AA0EC1BDF	AB1FDOACE	ACE0AFDB1	ADF1BECA0	AECAODF1B	AFDB1CE0A
BOBEDFC1A	B1AFCEDOB	BA1CFDEB0	BB0DECFA1	BCFA1BODE	BDEB0A1CF	BED0B1AFC	BF01A0BED
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	FADFOBCE1	FBCE1ADF0	FCB1EDA0F	FDA0FCB1E	FE1BCF0AD	FF0ADE1BC

**Solution to exercise 13.14 (p.220).** 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:

$$p_B = \sum_{\mathbf{x} \neq 0} P(\mathbf{x} | \mathbf{r}). \quad (13.46)$$

The average bit error probability, averaging over all bits in the codeword, is:

$$p_b = \sum_{\mathbf{x} \neq 0} P(\mathbf{x} | \mathbf{r}) \frac{w(\mathbf{x})}{N}, \quad (13.47)$$

where  $w(\mathbf{x})$  is the weight of codeword  $\mathbf{x}$ . Now the weights of the non-zero codewords satisfy

$$1 \geq \frac{w(\mathbf{x})}{N} \geq \frac{d_{\min}}{N}. \quad (13.48)$$

Substituting the inequalities (13.48) into the definitions (13.46, 13.47), we obtain:

$$p_B \geq p_b \geq \frac{d_{\min}}{N} p_B, \quad (13.49)$$

which is a factor of two stronger, on the right, than the stated result (13.39). In making the proof watertight, I have weakened the result a little.

**Careful proof.** The 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  $P(x_n = a|\mathbf{r}) = \sum_{\mathbf{x}} P(\mathbf{x}|\mathbf{r}) \mathbb{1}[x_n = a]$ .

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  $p_B$  (13.39).

The left-hand inequality in (13.39) 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 that:

- (a) the bit-guessing decoder is nearly as good as the optimal bit decoder:

$$p_b^G \leq 2p_b. \quad (13.50)$$

- (b) the bit-guessing decoder's error probability is related to the block-guessing decoder's by

$$p_b^G \geq \frac{d_{\min}}{N} p_B^G. \quad (13.51)$$

Then since  $p_B^G \geq p_B$ , we have

$$p_b > \frac{1}{2} p_b^G \geq \frac{1}{2} \frac{d_{\min}}{N} p_B^G \geq \frac{1}{2} \frac{d_{\min}}{N} p_B. \quad (13.52)$$

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

$$P^{\text{optimal}} = \min(p_0, p_1). \quad (13.53)$$

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,

$$P^{\text{guess}} = 2p_0p_1 \leq 2\min(p_0, p_1) = 2P^{\text{optimal}}. \quad (13.54)$$

Since  $p_b^G$  is the average of many such error probabilities,  $P^{\text{guess}}$ , and  $p_b$  is the average of the corresponding optimal error probabilities,  $P^{\text{optimal}}$ , we obtain the desired relationship (13.50) between  $p_b^G$  and  $p_b$ .  $\square$



**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. □

**Solution to exercise 13.20 (p.222).** 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

$$M \geq NH_2(f), \quad (13.55)$$

or, in terms of the rate  $R = 1 - M/N$ ,

$$R \leq 1 - H_2(f), \quad (13.56)$$

a bound which agrees precisely with the capacity of the channel.

This argument is turned into a proof in the following chapter.

**Solution to exercise 13.24 (p.222).** 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. 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 you have not figured out these winning strategies for teams of 7 and 15, I recommend thinking about the solution to the three-player game in terms

of the locations of the winning and losing states on the three-dimensional hypercube, then thinking laterally.

If the number of players,  $N$ , is  $2^r - 1$ , the optimal strategy can be defined using a Hamming code of length  $N$ , and the probability of winning the prize is  $N/(N + 1)$ . Each player is identified with a number  $n \in 1 \dots N$ . 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. It's quite easy to train seven players to follow the optimal strategy if the cyclic representation of the  $(7, 4)$  Hamming code is used (p.19).

---

## About Chapter 14

In this chapter we will draw together several ideas that we've encountered so far in one nice short proof. We will simultaneously prove both Shannon's noisy-channel coding theorem (for symmetric binary channels) and his source coding theorem (for binary sources). While this proof has connections to many preceding chapters in the book, it's not essential to have read them all.

On the noisy-channel coding side, our proof will be more constructive than the proof given in Chapter 10; there, we proved that almost any random code is 'very good'. Here we will show that almost any *linear* code is very good. We will make use of the idea of typical sets (Chapters 4 and 10), and we'll borrow from the previous chapter's calculation of the weight enumerator function of random linear codes (section 13.5).

On the source coding side, our proof will show that *random linear hash functions* can be used for compression of compressible binary sources, thus giving a link to Chapter 12.

# 14

---

## *Very Good Linear Codes Exist*

In this chapter we'll use a single calculation to prove simultaneously the source coding theorem and the noisy-channel coding theorem for the binary symmetric channel.

Incidentally, this proof works for much more general channel models, not only the binary symmetric channel. For example, the proof can be reworked for channels with non-binary outputs, for time-varying channels and for channels with memory, as long as they have binary inputs satisfying a symmetry property, cf. section 10.6.

### ► 14.1 A simultaneous proof of the source coding and noisy-channel coding theorems

We consider a linear error-correcting code with binary parity-check matrix  $\mathbf{H}$ . The matrix has  $M$  rows and  $N$  columns. Later in the proof we will increase  $N$  and  $M$ , keeping  $M \propto N$ . The rate of the code satisfies

$$R \geq 1 - \frac{M}{N}. \quad (14.1)$$

If all the rows of  $\mathbf{H}$  are independent then this is an equality,  $R = 1 - M/N$ . In what follows, we'll assume the equality holds. Eager readers may work out the expected rank of a random binary matrix  $\mathbf{H}$  (it's very close to  $M$ ) and pursue the effect that the difference ( $M - \text{rank}$ ) has on the rest of this proof (it's negligible).

A codeword  $\mathbf{t}$  is selected, satisfying

$$\mathbf{H}\mathbf{t} = \mathbf{0} \bmod 2, \quad (14.2)$$

and a binary symmetric channel adds noise  $\mathbf{x}$ , giving the received signal

$$\mathbf{r} = \mathbf{t} + \mathbf{x} \bmod 2. \quad (14.3)$$

In this chapter  $\mathbf{x}$  denotes the noise added by the channel, not the input to the channel.

The receiver aims to infer both  $\mathbf{t}$  and  $\mathbf{x}$  from  $\mathbf{r}$  using a syndrome-decoding approach. Syndrome decoding was first introduced in section 1.2 (p.10 and 11). The receiver computes the syndrome

$$\mathbf{z} = \mathbf{H}\mathbf{r} \bmod 2 = \mathbf{H}\mathbf{t} + \mathbf{H}\mathbf{x} \bmod 2 = \mathbf{H}\mathbf{x} \bmod 2. \quad (14.4)$$

The syndrome only depends on the noise  $\mathbf{x}$ , and the decoding problem is to find the most probable  $\mathbf{x}$  that satisfies

$$\mathbf{H}\mathbf{x} = \mathbf{z} \bmod 2. \quad (14.5)$$

This best estimate for the noise vector,  $\hat{\mathbf{x}}$ , is then subtracted from  $\mathbf{r}$  to give the best guess for  $\mathbf{t}$ . Our aim is to show that, as long as  $R < 1 - H(X) = 1 - H_2(f)$ , where  $f$  is the flip probability of the binary symmetric channel, the optimal decoder for this syndrome-decoding problem has vanishing probability of error, as  $N$  increases, for random  $\mathbf{H}$ .

We prove this result by studying a sub-optimal strategy for solving the decoding problem. Neither the optimal decoder nor this *typical-set decoder* would be easy to implement, but the typical-set decoder is easier to analyze. The typical-set decoder examines the typical set  $T$  of noise vectors, the set of noise vectors  $\mathbf{x}'$  that satisfy  $\log 1/P(\mathbf{x}') \simeq NH(X)$ , checking to see if any of those typical vectors  $\mathbf{x}'$  satisfies the observed syndrome,

$$\mathbf{H}\mathbf{x}' = \mathbf{z}. \quad (14.6)$$

If exactly one typical vector  $\mathbf{x}'$  does so, the typical set decoder reports that vector as the hypothesized noise vector. If no typical vector matches the observed syndrome, or more than one does, then the typical set decoder reports an error.

The probability of error of the typical-set decoder, for a given matrix  $\mathbf{H}$ , can be written as a sum of two terms,

$$P_{\text{TS}|\mathbf{H}} = P^{(I)} + P_{\text{TS}|\mathbf{H}}^{(II)}, \quad (14.7)$$

where  $P^{(I)}$  is the probability that the true noise vector  $\mathbf{x}$  is itself not typical, and  $P_{\text{TS}|\mathbf{H}}^{(II)}$  is the probability that the true  $\mathbf{x}$  is typical and at least one other typical vector clashes with it. The first probability vanishes as  $N$  increases, as we proved when we first studied typical sets (Chapter 4). We concentrate on the second probability. To recap, we're imagining a true noise vector,  $\mathbf{x}$ ; and if *any* of the typical noise vectors  $\mathbf{x}'$ , different from  $\mathbf{x}$ , satisfies  $\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0$ , then we have an error. We use the truth function

$$\mathbb{1}[\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0], \quad (14.8)$$

whose value is one if the statement  $\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0$  is true and zero otherwise. We can bound the number of type II errors made when the noise is  $\mathbf{x}$  thus:

$$[\text{Number of errors given } \mathbf{x} \text{ and } \mathbf{H}] \leq \sum_{\mathbf{x}': \substack{\mathbf{x}' \in T \\ \mathbf{x}' \neq \mathbf{x}}} \mathbb{1}[\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0]. \quad (14.9)$$

The number of errors is either zero or one; the sum on the right-hand side may exceed one, in cases where several typical noise vectors have the same syndrome.

Equation (14.9) is a union bound.

We can now write down the probability of a type-II error by averaging over  $\mathbf{x}$ :

$$P_{\text{TS}|\mathbf{H}}^{(II)} \leq \sum_{\mathbf{x} \in T} P(\mathbf{x}) \sum_{\mathbf{x}': \substack{\mathbf{x}' \in T \\ \mathbf{x}' \neq \mathbf{x}}} \mathbb{1}[\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0]. \quad (14.10)$$

Now, we will find the average of this probability of type-II error over all linear codes by averaging over  $\mathbf{H}$ . By showing that the *average* probability of type-II error vanishes, we will thus show that there exist linear codes with vanishing error probability, indeed, that almost all linear codes are very good.

We denote averaging over all binary matrices  $\mathbf{H}$  by  $\langle \dots \rangle_{\mathbf{H}}$ . The average probability of type-II error is

$$\bar{P}_{\text{TS}}^{(II)} = \sum_{\mathbf{H}} P(\mathbf{H}) P_{\text{TS}|\mathbf{H}}^{(II)} = \left\langle P_{\text{TS}|\mathbf{H}}^{(II)} \right\rangle_{\mathbf{H}} \quad (14.11)$$

We'll leave out the  $\epsilon$ s and  $\beta$ s that make a typical-set definition rigorous. Enthusiasts are encouraged to revisit section 4.4 and put these details into this proof.

$$= \left\langle \sum_{\mathbf{x} \in T} P(\mathbf{x}) \sum_{\mathbf{x}': \substack{\mathbf{x}' \in T \\ \mathbf{x}' \neq \mathbf{x}}} \mathbb{1}[\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0] \right\rangle_{\mathbf{H}} \quad (14.12)$$

$$= \sum_{\mathbf{x} \in T} P(\mathbf{x}) \sum_{\mathbf{x}': \substack{\mathbf{x}' \in T \\ \mathbf{x}' \neq \mathbf{x}}} \langle \mathbb{1}[\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0] \rangle_{\mathbf{H}}. \quad (14.13)$$

Now, the quantity  $\langle \mathbb{1}[\mathbf{H}(\mathbf{x}' - \mathbf{x}) = 0] \rangle_{\mathbf{H}}$  already cropped up when we were calculating the expected weight enumerator function of random linear codes (section 13.5): for any non-zero binary vector  $\mathbf{v}$ , the probability that  $\mathbf{H}\mathbf{v} = 0$ , averaging over all matrices  $\mathbf{H}$ , is  $2^{-M}$ . So

$$\bar{P}_{\text{TS}}^{(II)} = \left( \sum_{\mathbf{x} \in T} P(\mathbf{x}) \right) (|T| - 1) 2^{-M} \quad (14.14)$$

$$\leq |T| 2^{-M}, \quad (14.15)$$

where  $|T|$  denotes the size of the typical set. As you will recall from Chapter 4, there are roughly  $2^{NH(X)}$  noise vectors in the typical set. So

$$\bar{P}_{\text{TS}}^{(II)} \leq 2^{NH(X)} 2^{-M}. \quad (14.16)$$

This bound on the probability of error either vanishes or grows exponentially as  $N$  increases (remembering that we are keeping  $M$  proportional to  $N$  as  $N$  increases). It vanishes if

$$H(X) < M/N. \quad (14.17)$$

Substituting  $R = 1 - M/N$ , we have thus established the noisy-channel coding theorem for the binary symmetric channel: very good linear codes exist for any rate  $R$  satisfying

$$R < 1 - H(X), \quad (14.18)$$

where  $H(X)$  is the entropy of the channel noise, per bit.  $\square$

Exercise 14.1.<sup>[3]</sup> Redo the proof for a more general channel.

## ► 14.2 Data compression by linear hash codes

The decoding game we have just played can also be viewed as an *uncompression* game. The world produces a binary noise vector  $\mathbf{x}$  from a source  $P(\mathbf{x})$ . The noise has redundancy (if the flip probability is not 0.5). We compress it with a linear compressor that maps the  $N$ -bit input  $\mathbf{x}$  (the noise) to the  $M$ -bit output  $\mathbf{z}$  (the syndrome). Our uncompression task is to recover the input  $\mathbf{x}$  from the output  $\mathbf{z}$ . The rate of the compressor is

$$R_{\text{compressor}} \equiv M/N. \quad (14.19)$$

[We don't care about the possibility of linear redundancies in our definition of the rate, here.] The result that we just found, that the decoding problem can be solved, for almost any  $\mathbf{H}$ , with vanishing error probability, as long as  $H(X) < M/N$ , thus instantly proves a source coding theorem:

Given a binary source  $X$  of entropy  $H(X)$ , and a required compressed rate  $R > H(X)$ , there exists a linear compressor  $\mathbf{x} \rightarrow \mathbf{z} = \mathbf{H}\mathbf{x} \bmod 2$  having rate  $M/N$  equal to that required rate  $R$ , and an associated uncompressor, that is virtually lossless.

This theorem is true not only for a source of independent identically distributed symbols but also for any source for which a typical set can be defined: sources with memory, and time-varying sources, for example; all that's required is that the source be ergodic.

*Notes*

This method for proving that codes are good can be applied to other linear codes, such as low-density parity-check codes (MacKay, 1999b; Aji *et al.*, 2000). For each code we need an approximation of its expected weight enumerator function.

# 15

---

## Further Exercises on Information Theory

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.<sup>[2]</sup> Let  $X$  be an ensemble with  $\mathcal{A}_X = \{0, 1\}$  and  $\mathcal{P}_X = \{0.995, 0.005\}$ . Consider source coding using 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.
- If the assigned codewords are all of the same length, find the minimum length required to provide the above set with distinct codewords.
  - Calculate the probability of getting an  $\mathbf{x}$  that will be ignored.
- ▷ Exercise 15.2.<sup>[2]</sup> Let  $X$  be an ensemble with  $\mathcal{P}_X = \{0.1, 0.2, 0.3, 0.4\}$ . The ensemble is encoded using the symbol code  $\mathcal{C} = \{0001, 001, 01, 1\}$ . Consider the codeword corresponding to  $\mathbf{x} \in X^N$ , where  $N$  is large.
- Compute the entropy of the fourth bit of transmission.
  - Compute the conditional entropy of the fourth bit given the third bit.
  - Estimate the entropy of the hundredth bit.
  - Estimate the conditional entropy of the hundredth bit given the ninety-ninth bit.



Exercise 15.3.<sup>[2]</sup> 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.<sup>[2]</sup> How can you use a coin to draw straws among 3 people?
- ▷ Exercise 15.5.<sup>[2]</sup> In a magic trick, there are three participants: the magician, an assistant, and a volunteer. The assistant, who claims to have paranormal abilities, is in a soundproof room. The magician gives the volunteer six blank cards, five white and one blue. The volunteer writes a different integer from 1 to 100 on each card, as the magician is watching. The volunteer keeps the blue card. The magician arranges the five white cards in some order and passes them to the assistant. The assistant then announces the number on the blue card.

How does the trick work?



▷ Exercise 15.6.<sup>[3]</sup> How does *this* trick work?

‘Here’s an ordinary pack of cards, shuffled into random order. Please choose five cards from the pack, any that you wish. Don’t let me see their faces. No, don’t give them to me: pass them to my assistant Esmerelda. She can look at them.  
 ‘Now, Esmerelda, show me four of the cards. Hmm... nine of spades, six of clubs, four of hearts, ten of diamonds. The hidden card, then, must be the queen of spades!’

The trick can be performed as described above for a pack of 52 cards. Use information theory to give an upper bound on the number of cards for which the trick can be performed.

▷ Exercise 15.7.<sup>[2]</sup> Find a probability sequence  $\mathbf{p} = (p_1, p_2, \dots)$  such that  $H(\mathbf{p}) = \infty$ .

▷ Exercise 15.8.<sup>[2]</sup> 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  $4^8 = 65\,536$  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 4.29) where  $N = 8$  and  $\beta = 0.1$ .

▷ Exercise 15.9.<sup>[2]</sup> 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

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2/3 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1/3 & 0 \end{bmatrix}. \quad (15.1)$$

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 15.10.<sup>[2]</sup> 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 15.11.<sup>[2]</sup> Find the capacity and optimal input distribution for the three-input, three-output channel whose transition probabilities are:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 1/3 & 2/3 \end{bmatrix}. \quad (15.2)$$

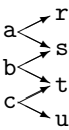


**Exercise 15.12.** [3, p.239] The input to a channel  $Q$  is a word of 8 bits. The output is also a word of 8 bits. Each time it is used, the channel flips *exactly one* of the transmitted bits, but the receiver does not know which one. The other seven bits are received without error. All 8 bits are 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 15.13.** [2] A channel with input  $x \in \{a, b, c\}$  and output  $y \in \{r, s, t, u\}$  has conditional probability matrix:

$$Q = \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix}.$$



What is its capacity?

▷ **Exercise 15.14.** [3] The ten-digit number on the cover of a book known as the ISBN incorporates an error-detecting code. The number consists of nine source digits  $x_1, x_2, \dots, x_9$ , satisfying  $x_n \in \{0, 1, \dots, 9\}$ , and a tenth check digit whose value is given by

$$x_{10} = \left( \sum_{n=1}^9 nx_n \right) \bmod 11.$$

Here  $x_{10} \in \{0, 1, \dots, 9, 10\}$ . If  $x_{10} = 10$  then the tenth digit is shown using the roman numeral X.

Show that a valid ISBN satisfies:

$$\left( \sum_{n=1}^{10} nx_n \right) \bmod 11 = 0.$$

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

$$x_{10} = \left( \sum_{n=1}^9 nx_n \right) \bmod 10,$$

why would the code not work so well? (Discuss the detection of both modifications of single digits and transpositions of digits.)

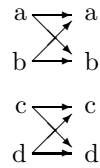
0-521-64298-1  
 1-010-00000-4

**Table 15.1.** Some valid ISBNs. [The hyphens are included for legibility.]



Exercise 15.15.<sup>[3]</sup> 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 15.16.<sup>[2]</sup> What are the differences in the redundancies needed in an error-detecting code (which can reliably detect that a block of data has been corrupted) and an error-correcting code (which can detect and correct errors)?

### Further tales from information theory

The following exercises give you the chance to discover for yourself the answers to some more surprising results of information theory.

Exercise 15.17.<sup>[3]</sup> **Communication of information from correlated sources.** 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 dependent, so their joint information content is only a little greater than the marginal information content of either of them. For example, C is a weather collator who wishes to receive a string of reports saying whether it is raining in Allerton ( $x^{(A)}$ ) and whether it is raining in Bognor ( $x^{(B)}$ ). The joint probability of  $x^{(A)}$  and  $x^{(B)}$  might be

$$P(x^{(A)}, x^{(B)}): \begin{array}{c|cc} & \begin{matrix} x^{(A)} \\ 0 \quad 1 \end{matrix} \\ \begin{matrix} x^{(B)} \\ 0 \quad 1 \end{matrix} & \begin{bmatrix} 0.49 & 0.01 \\ 0.01 & 0.49 \end{bmatrix} \end{array} \quad (15.3)$$

The weather collator would like to know  $N$  successive values of  $x^{(A)}$  and  $x^{(B)}$  exactly, but, since he has to pay for every bit of information he receives, he is interested in the possibility of avoiding buying  $N$  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?

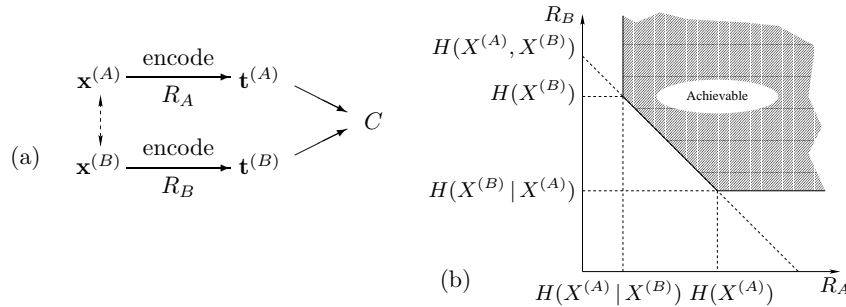


Figure 15.2. Communication of information from dependent sources. (a)  $x^{(A)}$  and  $x^{(B)}$  are dependent sources (the dependence is represented by the dotted arrow). Strings of values of each variable are encoded using codes of rate  $R_A$  and  $R_B$  into transmissions  $\mathbf{t}^{(A)}$  and  $\mathbf{t}^{(B)}$ , which are communicated over noise-free channels to a receiver  $C$ . (b) The achievable rate region. Both strings can be conveyed without error even though  $R_A < H(X^{(A)})$  and  $R_B < H(X^{(B)})$ .

The answer, which you should demonstrate, is indicated in figure 15.2. In the general case of two dependent 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 entropy  $H(X^{(A)}, X^{(B)})$  (Slepian and Wolf, 1973).

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, for example  $R_A = 0.6$ ,  $R_B = 0.6$ . 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 15.18.**<sup>[3]</sup> **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  $x^{(A)}$  and  $x^{(B)}$  and a ternary output  $y$  equal to the arithmetic sum of the two inputs, that's 0, 1 or 2. There is no noise. Users  $A$  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 signals? How fast can  $A$  and  $B$  communicate?

Clearly the total information rate from  $A$  and  $B$  to the receiver cannot be two bits. On the other hand it is easy to achieve a total information rate  $R_A + R_B$  of one bit. Can reliable communication be achieved at rates  $(R_A, R_B)$  such that  $R_A + R_B > 1$ ?

The answer is indicated in figure 15.3.

Some practical codes for multi-user channels are presented in Ratzer and MacKay (2003).

**Exercise 15.19.**<sup>[3]</sup> **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-sharing (time-division signaling). If the capacities of the two channels, considered separately,

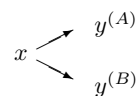


Figure 15.4. The broadcast channel.  $x$  is the channel input;  $y^{(A)}$  and  $y^{(B)}$  are the outputs.

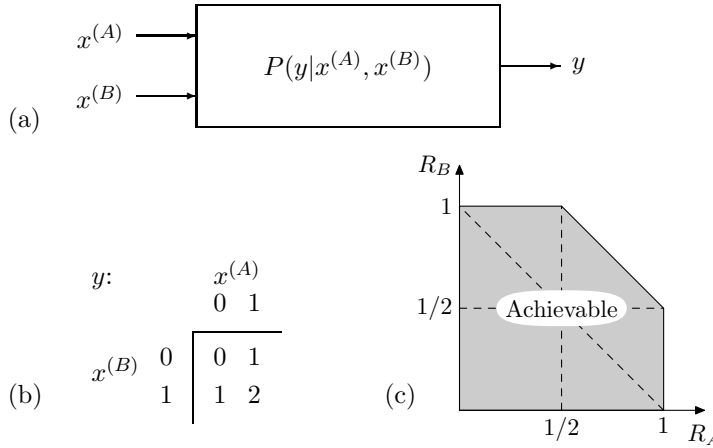


Figure 15.3. Multiple access channels. (a) A general multiple access channel with two transmitters and one receiver. (b) A binary multiple access channel with output equal to the sum of two inputs. (c) The achievable region.

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 Belarusian; you are fluent in American and in Belarusian, but 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 Belarusian 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.

An example of a broadcast channel consists of two binary symmetric channels with a common input. The two halves of the channel have flip probabilities  $f_A$  and  $f_B$ . 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,

$$x \rightarrow y^{(A)} \rightarrow y^{(B)}, \quad (15.4)$$

i.e.,  $y^{(B)}$  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  $R_B$ : the 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  $A$ .

The following exercise is equivalent to this one, and a solution to it is illustrated in figure 15.8.

**Exercise 15.20.**<sup>[3]</sup> **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, imagine that a channel is known to be a binary symmetric channel with noise level either  $f_A$  or  $f_B$ . Let  $f_B > f_A$ , and 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

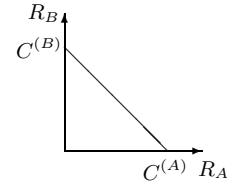


Figure 15.5. Rates achievable by simple timesharing.

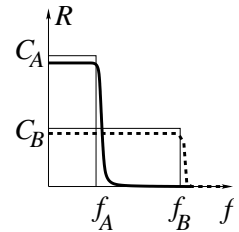


Figure 15.6. Rate of reliable communication  $R$ , as a function of noise level  $f$ , for Shannonesque codes designed to operate at noise levels  $f_A$  (solid line) and  $f_B$  (dashed line).

would be a catastrophic failure to communicate information reliably (solid line in figure 15.6).

A conservative approach would design the encoding system for the worst-case scenario, installing a code with rate  $R_B \simeq C_B$  (dashed line in figure 15.6). In the event that the lower noise level,  $f_A$ , holds true, the managers would have a feeling of regret because of the wasted capacity difference  $C_A - R_B$ .

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 15.7? This code 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  $R_A$ .

An illustrative answer is shown in figure 15.8, for the case  $f_A = 0.01$  and  $f_B = 0.1$ . (This figure also shows the achievable region for a broadcast channel whose two half-channels have noise levels  $f_A = 0.01$  and  $f_B = 0.1$ .) I admit I find the gap between the simple time-sharing solution and the cunning solution disappointingly small.

In Chapter 50 we will discuss codes for a special class of broadcast channels, namely erasure channels, where every symbol is either received without error or erased. These codes have the nice property that they are *rateless* – the number of symbols transmitted is determined on the fly such that reliable communication is achieved, whatever the erasure statistics of the channel.

**Exercise 15.21.**<sup>[3]</sup> **Multiterminal information networks** are both important practically and intriguing theoretically. Consider the following example of a two-way binary channel (figure 15.9a,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 hear the signal transmitted by the other person only if you are transmitting a zero. What simultaneous information rates 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 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 15.9c. There exist codes that can achieve rates up to the boundary shown. There may exist better codes too.

## Solutions

**Solution to exercise 15.12** (p.235).  $C(Q) = 5$  bits.

Hint for the last part: a solution exists that involves a simple (8, 5) code.

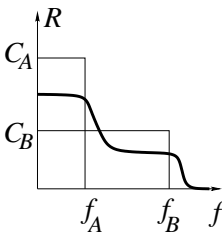


Figure 15.7. Rate of reliable communication  $R$ , as a function of noise level  $f$ , for a desired variable-rate code.

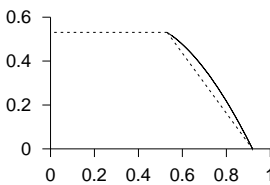


Figure 15.8. An achievable region for the channel with unknown noise level. Assuming the two possible noise levels are  $f_A = 0.01$  and  $f_B = 0.1$ , the dashed lines show the rates  $R_A, R_B$  that are achievable using a simple time-sharing approach, and the solid line shows rates achievable using a more cunning approach.

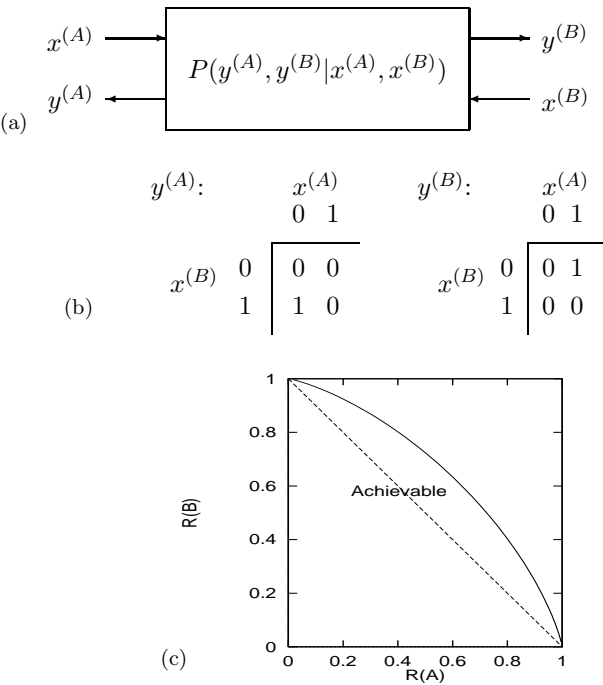


Figure 15.9. (a) A general two-way channel. (b) The rules for a binary two-way channel. The two tables show the outputs  $y^{(A)}$  and  $y^{(B)}$  that result for each state of the inputs. (c) Achievable region for the two-way binary channel. Rates below the solid line are achievable. The dotted line shows the ‘obviously achievable’ region which can be attained by simple time-sharing.

# 16

---

## 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.

► **16.1 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 expensive 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 must 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:

1. If you are the front soldier in the line, say the number ‘one’ to the soldier behind you.
  2. If you are the rearmost 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.

Algorithm 16.1. Message-passing 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).

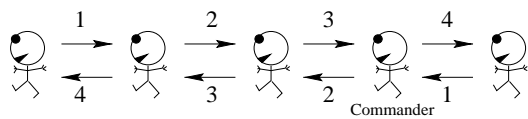


Figure 16.2. A line of soldiers counting themselves using message-passing rule-set A. The commander can add ‘3’ from the soldier in front, ‘1’ from the soldier behind, and ‘1’ for himself, and deduce that there are 5 soldiers in total.

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*, because the two groups are separated by the commander.

If the soldiers were not arranged in a line but were travelling in a swarm, then it would not be easy to separate them into two groups in this way. The

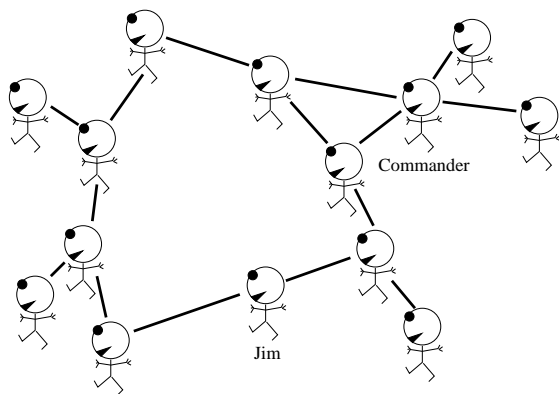


Figure 16.3. A swarm of guerillas.

guerillas in figure 16.3 could not be counted using the above message-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*.

Rule-set B is a message-passing algorithm for counting a swarm of guerillas whose connections form a *cycle-free graph*, also known as a *tree*, as illustrated in figure 16.4. Any guerilla can deduce the total in the tree from the messages that they receive.

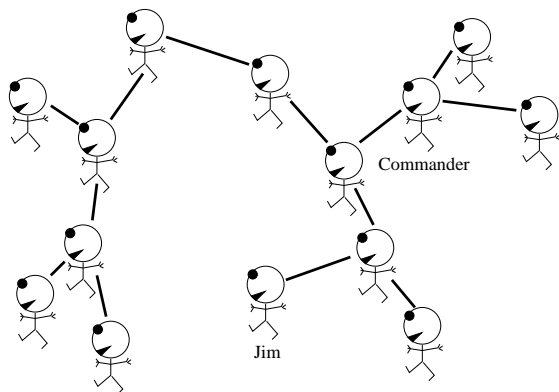


Figure 16.4. A swarm of guerillas whose connections form a tree.

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  $v_1, v_2, \dots, v_N$  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  $V + 1 - v_n$ .  
}

Algorithm 16.5. Message-passing rule-set B.

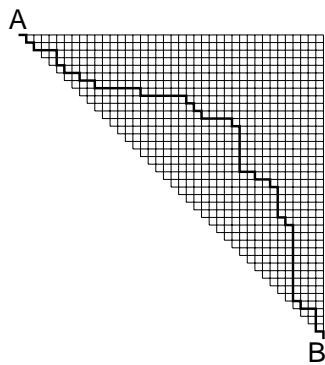


Figure 16.6. A triangular  $41 \times 41$  grid. How many paths are there from A to B? One path is shown.

► 16.2 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 16.6 shows a rectangular grid, and a path through the grid, connecting points A and B. A valid path is one that starts from A and proceeds to B by rightward and downward moves. Our questions are:

- 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 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 were a diagonal strip only three nodes wide, there would still be about  $2^{N/2}$  possible paths.

The computational breakthrough is to realize that to find the *number* of paths, we do not have to enumerate all the paths explicitly. 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 above or to the left). 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.

This message-passing algorithm is illustrated in figure 16.8 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 16.9 shows the five distinct paths from A to B.

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; and if we divide that by the total number of paths, we obtain the probability that a randomly selected path passes through that node. Figure 16.10 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 16.11 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 the corresponding node.

*Random path sampling*



**Exercise 16.1.** [1, p.247] 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

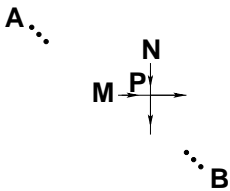


Figure 16.7. Every path from A to P enters P through an upstream neighbour of P, either M or N; so we can find the number of paths from A to P by adding the number of paths from A to M to the number from A to N.

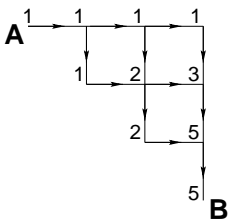


Figure 16.8. Messages sent in the forward pass.

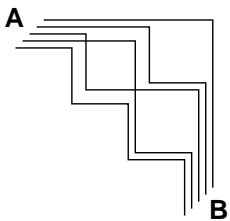


Figure 16.9. The five paths.

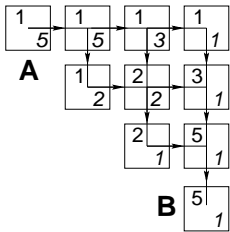


Figure 16.10. Messages sent in the forward and backward passes.

the resulting path a uniform random sample from the set of all paths?  
[Hint: imagine trying it for the grid of figure 16.8.]

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



**Exercise 16.2.** [2, p.247] 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? (Figure 16.11.)

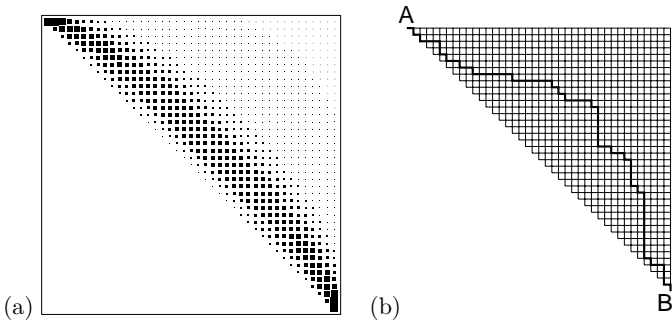


Figure 16.11. (a) The probability of passing through each node, and (b) a randomly chosen path.

The message-passing algorithm we used to count the paths to B is an example of the *sum-product algorithm*. The ‘sum’ takes place at each node when it adds together the messages coming from its predecessors; the ‘product’ was not mentioned, but you can think of the sum as a weighted sum in which all the summed terms happened to have weight 1.

► **16.3 Finding the lowest-cost path**

Imagine you wish to travel as quickly as possible from Ambridge (A) to Bognor (B). The various possible routes are shown in figure 16.12, along with the cost in hours of traversing each edge in the graph. For example, the route A–I–L–N–B has a cost of 8 hours. We would like to find the lowest-cost path without explicitly evaluating the cost of all paths. We can do this efficiently by finding for each node what the cost of the lowest-cost path to that node from A is. These quantities can be computed by message-passing, starting from node A. The message-passing algorithm is called the *min-sum algorithm* or *Viterbi algorithm*.

For brevity, we'll call the cost of the lowest-cost path from node A to node  $x$  ‘the cost of  $x$ ’. Each node can broadcast its cost to its descendants once it knows the costs of all its possible predecessors. Let's step through the algorithm by hand. The cost of A is zero. We pass this news on to H and I. As the message passes along each edge in the graph, the cost of that edge is *added*. We find the costs of H and I are 4 and 1 respectively (figure 16.13a). Similarly then, the costs of J and L are found to be 6 and 2 respectively, but what about K? Out of the edge H–K comes the message that a path of cost 5 exists from A to K via H; and from edge I–K we learn of an alternative path of cost 3 (figure 16.13b). The min-sum algorithm sets the cost of K equal to the minimum of these (the ‘min’), and records which was the smallest-cost route into K by retaining only the edge I–K and pruning away the other edges leading to K (figure 16.13c). Figures 16.13d and e show the remaining two iterations of the algorithm which reveal that there is a path from A to B with cost 6. [If the min-sum algorithm encounters a tie, where the minimum-cost

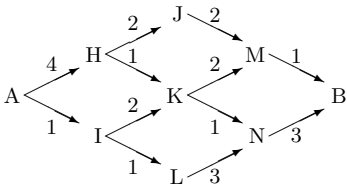


Figure 16.12. Route diagram from Ambridge to Bognor, showing the costs associated with the edges.

path to a node is achieved by more than one route to it, then the algorithm can pick any of those routes at random.]

We can recover this lowest-cost path by backtracking from B, following the trail of surviving edges back to A. We deduce that the lowest-cost path is A–I–K–M–B.

#### Other applications of the min-sum algorithm

Imagine that you manage the production of a product from raw materials via a large set of operations. You wish to identify the *critical path* in your process, that is, the subset of operations that are holding up production. If any operations on the critical path were carried out a little faster then the time to get from raw materials to product would be reduced.

The critical path of a set of operations can be found using the min-sum algorithm.

In Chapter 25 the min-sum algorithm will be used in the decoding of error-correcting codes.

### ► 16.4 Summary and related ideas

Some global functions 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). Such functions can be computed efficiently by message-passing. Other functions 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 travelling salesman could take that visits every soldier in a troop.

One of the challenges of machine learning is to find low-cost solutions to problems like these. The problem of finding a large subset of variables that are approximately equal can be solved with a neural network approach (Hopfield and Brody, 2000; Hopfield and Brody, 2001). A neural approach to the travelling salesman problem will be discussed in section 42.9.

### ► 16.5 Further exercises

- ▷ Exercise 16.3.<sup>[2]</sup> Describe the asymptotic properties of the probabilities depicted in figure 16.11a, for a grid in a triangle of width and height  $N$ .
- ▷ Exercise 16.4.<sup>[2]</sup> In image processing, the *integral image*  $I(x, y)$  obtained from an image  $f(x, y)$  (where  $x$  and  $y$  are pixel coordinates) is defined by

$$I(x, y) \equiv \sum_{u=0}^x \sum_{v=0}^y f(u, v). \quad (16.1)$$

Show that the integral image  $I(x, y)$  can be efficiently computed by message passing.

Show that, from the integral image, some simple functions of the image can be obtained. For example, give an expression for the sum of the image intensities  $f(x, y)$  for all  $(x, y)$  in a rectangular region extending from  $(x_1, y_1)$  to  $(x_2, y_2)$ .

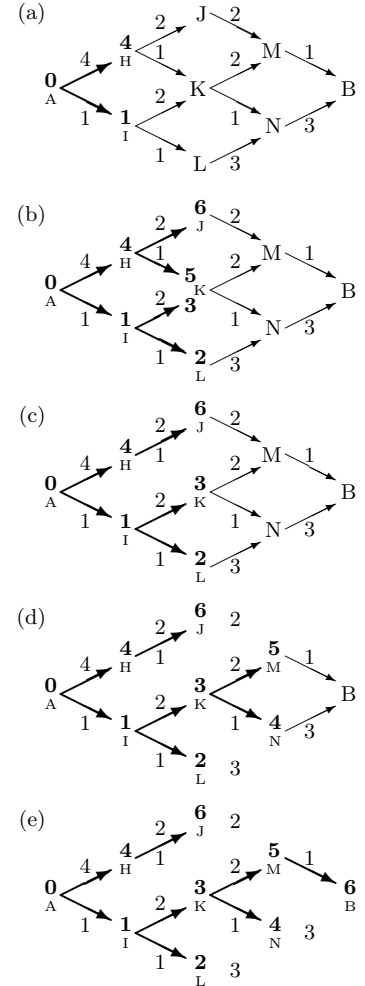
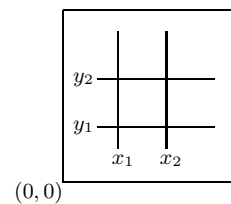


Figure 16.13. Min-sum message-passing algorithm to find the cost of getting to each node, and thence the lowest cost route from A to B.



## ► 16.6 Solutions

Solution to exercise 16.1 (p.244). Since there are five paths through the grid of figure 16.8, 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 16.2 (p.245). To make a uniform random walk, each forward 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 16.11b was generated.

# 17

---

## 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.

We make use of the idea introduced in Chapter 16, that *global properties of graphs can be computed by a local message-passing algorithm*.

### ► 17.1 Three examples of constrained binary channels

A constrained channel can be defined by rules that define which strings are permitted.

**Example 17.1.** In Channel A every 1 must be followed by at least one 0.

A valid string for this channel is

$$00100101001010100010. \quad (17.1)$$

As a motivation for this model, consider a channel in which 1s are represented by pulses of electromagnetic energy, and the device that produces those pulses requires a recovery time of one clock cycle after generating a pulse before it can generate another.

**Example 17.2.** Channel B has the rule that all 1s must come in groups of two or more, and all 0s must come in groups of two or more.

A valid string for this channel is

$$00111001110011000011. \quad (17.2)$$

As a motivation for this model, consider a disk drive in which successive bits are written onto neighbouring points in a track along the disk surface; the values 0 and 1 are represented by two opposite magnetic orientations. 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 17.3.** Channel C has the rule that the largest permitted runlength is two, that is, each symbol can be repeated at most once.

A valid string for this channel is

$$10010011011001101001. \quad (17.3)$$

**Channel A:**  
the substring 11 is forbidden.

**Channel B:**  
101 and 010 are forbidden.

**Channel C:**  
111 and 000 are forbidden.

A physical motivation for this model is a disk drive in which the rate of rotation of the disk is not known accurately, so it is difficult to distinguish between a string of two 1s and a string of three 1s, which are represented by oriented magnetizations of duration  $2\tau$  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 1s and the string of three 0s.

All three of these channels are examples of *runlength-limited channels*. The rules constrain the minimum and maximum numbers of successive 1s and 0s.

Channel	Runlength of 1s		Runlength of 0s	
	minimum	maximum	minimum	maximum
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 0s may be of any length but runs of 1s 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 conveyed reliably per channel-use.]

Some codes for a constrained channel

Let us concentrate for a moment on channel A, in which runs of 0s may be of any length but runs of 1s are restricted to length 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,  $C_1$ . 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?

$C_1$  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  $C_1$ .

$C_2$  is such a *variable-length* code. If the source symbols are used with equal frequency then the average transmitted length per source bit is

$$L = \frac{1}{2}1 + \frac{1}{2}2 = \frac{3}{2}, \tag{17.4}$$

so the average communication rate is

$$R = 2/3, \tag{17.5}$$

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

Code  $C_1$

$s$	$t$
0	00
1	10

Code  $C_2$

$s$	$t$
0	0
1	10



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,

$$L(f) = (1 - f) + 2f = 1 + f. \quad (17.6)$$

Thus

$$R(f) = \frac{H_2(f)}{L(f)} = \frac{H_2(f)}{1 + f}. \quad (17.7)$$

The original code  $C_2$ , without preprocessor, corresponds to  $f = 1/2$ . What happens if we perturb  $f$  a little towards smaller  $f$ , setting

$$f = \frac{1}{2} + \delta, \quad (17.8)$$

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 17.4.**<sup>[1]</sup> 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 17.1 shows these functions;  $R(f)$  does indeed increase as  $f$  decreases and has a maximum of about 0.69 bits per channel use at  $f \simeq 0.38$ .

By this argument we have shown that the capacity of channel A is at least  $\max_f R(f) = 0.69$ .

- ▷ **Exercise 17.5.**<sup>[2, p.257]</sup> 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 is 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.

## ► 17.2 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

$$C = \lim_{N \rightarrow \infty} \frac{1}{N} \log S(N). \quad (17.9)$$

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 9.6), ran over messages  $s = 1, \dots, S$ , is about to take on a new role: labelling the states of our channel;

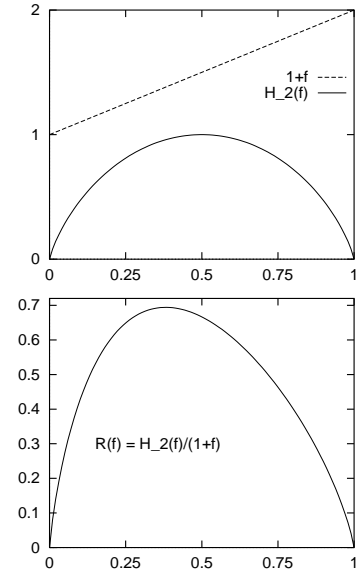


Figure 17.1. Top: The information content per source symbol and mean transmitted length per source symbol as a function of the source density. Bottom: The information content per transmitted symbol, in bits, as a function of  $f$ .

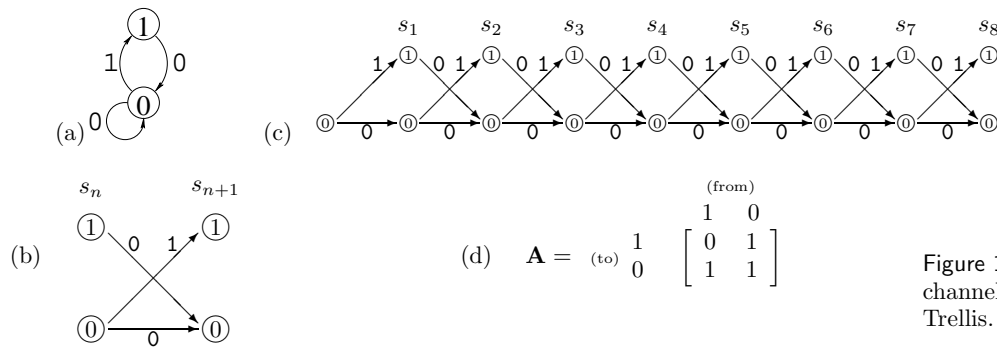


Figure 17.2. (a) State diagram for channel A. (b) Trellis section. (c) Trellis. (d) Connection matrix.

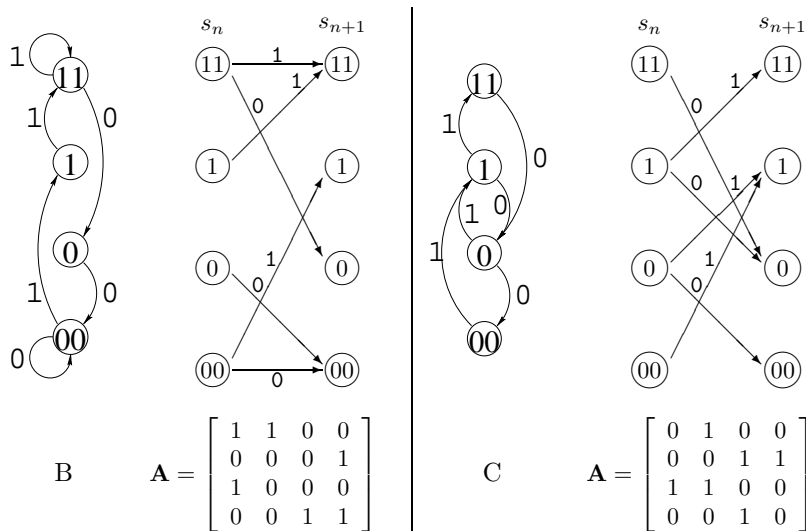


Figure 17.3. State diagrams, trellis sections and connection matrices for channels B and C.

so in this chapter we will denote the number of distinguishable messages of length  $N$  by  $M_N$ , and define the capacity to be:

$$C = \lim_{N \rightarrow \infty} \frac{1}{N} \log M_N. \quad (17.10)$$

Once we have figured out the capacity of a channel we will return to the task of making a practical code for that channel.

### ► 17.3 Counting the number of possible messages

First let us introduce some representations of constrained channels. In a *state diagram*, states of the transmitter are represented by circles labelled with the name of the state. Directed edges from one state to another indicate that the transmitter is permitted to move from the first state to the second, and a label on that edge indicates the symbol emitted when that transition is made. Figure 17.2a shows the state diagram for channel A. It has two states, 0 and 1. When transitions to state 0 are made, a 0 is transmitted; when transitions to state 1 are made, a 1 is transmitted; transitions from state 1 to state 1 are not possible.

We can also represent the state diagram by a *trellis section*, which shows two successive states in time at two successive horizontal locations (figure 17.2b). The state of the transmitter at time  $n$  is called  $s_n$ . The set of possible state sequences can be represented by a *trellis* as shown in figure 17.2c. A valid sequence corresponds to a path through the trellis, and the number of

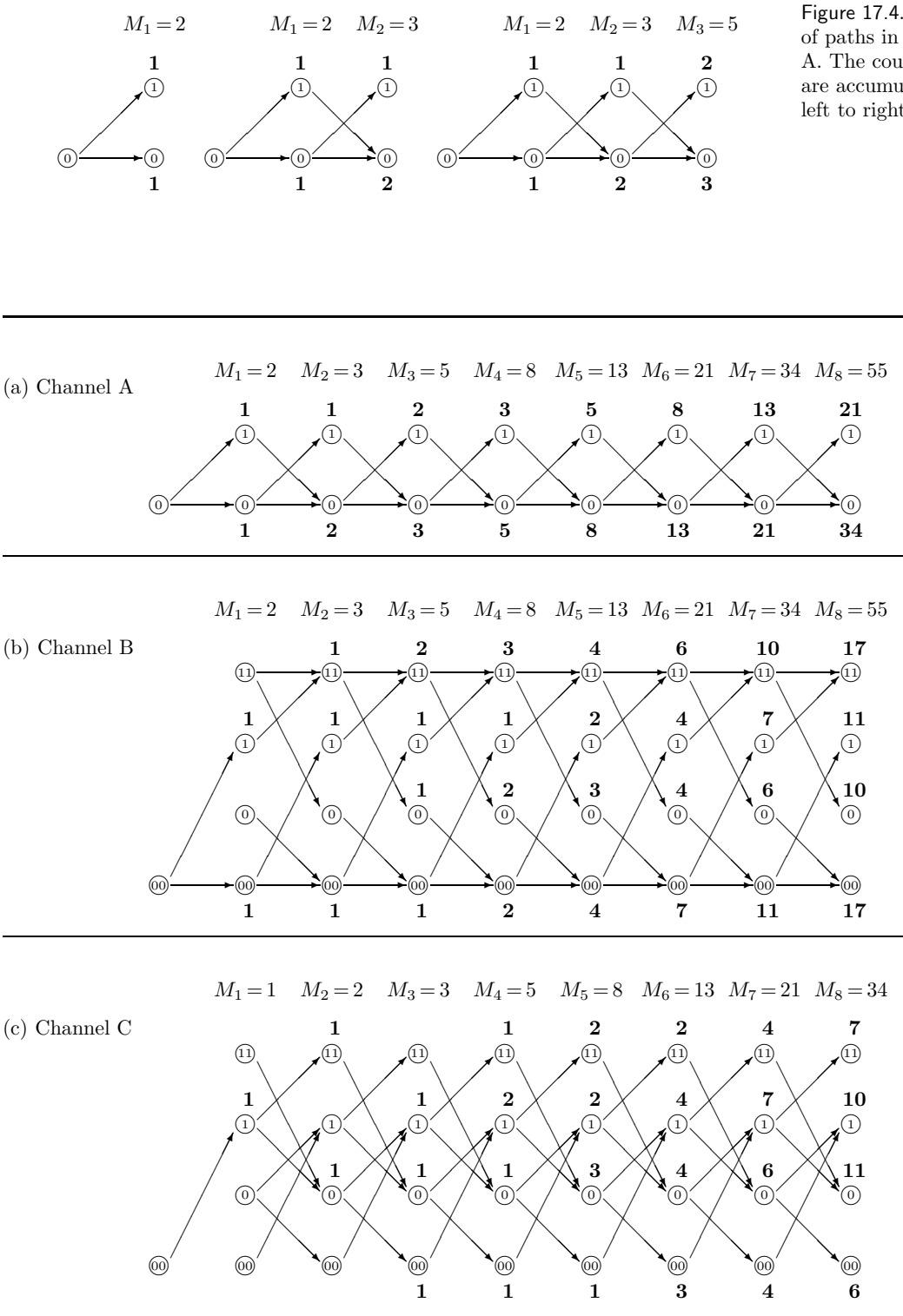


Figure 17.5. Counting the number of paths in the trellises of channels A, B, and C. We assume that at the start the first bit is preceded by 00, so that for channels A and B, any initial character is permitted, but for channel C, the first character must be a 1.

$n$	$M_n$	$M_n/M_{n-1}$	$\log_2 M_n$	$\frac{1}{n} \log_2 M_n$
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	$9 \times 10^{20}$	1.618	69.7	0.70
200	$7 \times 10^{41}$	1.618	139.1	0.70
300	$6 \times 10^{62}$	1.618	208.5	0.70
400	$5 \times 10^{83}$	1.618	277.9	0.69

Figure 17.6. Counting the number of paths in the trellis of channel A.

valid sequences is the number of paths. For the purpose of counting how many paths there are through the trellis, we can ignore the labels on the edges and summarize the trellis section by the *connection matrix*  $\mathbf{A}$ , in which  $A_{ss'} = 1$  if there is an edge from state  $s$  to  $s'$ , and  $A_{ss'} = 0$  otherwise (figure 17.2d). Figure 17.3 shows the state diagrams, trellis sections and connection matrices for channels B and C.

Let's count the number of paths for channel A by message-passing in its trellis. Figure 17.4 shows the first few steps of this counting process, and figure 17.5a shows the number of paths ending in each state after  $n$  steps for  $n = 1, \dots, 8$ . The total number of paths of length  $n$ ,  $M_n$ , is shown along the top. We recognize  $M_n$  as the Fibonacci series.

▷ **Exercise 17.6.**<sup>[1]</sup> Show that the ratio of successive terms in the Fibonacci series tends to the golden ratio,

$$\gamma \equiv \frac{1 + \sqrt{5}}{2} = 1.618. \quad (17.11)$$

Thus, to within a constant factor,  $M_N$  scales as  $M_N \sim \gamma^N$  as  $N \rightarrow \infty$ , so the capacity of channel A is

$$C = \lim_{N \rightarrow \infty} \frac{1}{N} \log_2 [\text{constant} \cdot \gamma^N] = \log_2 \gamma = \log_2 1.618 = 0.694. \quad (17.12)$$

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:

$$\mathbf{c}^{(n+1)} = \mathbf{A} \mathbf{c}^{(n)}. \quad (17.13)$$

So

$$\mathbf{c}^{(N)} = \mathbf{A}^N \mathbf{c}^{(0)}, \quad (17.14)$$

where  $\mathbf{c}^{(0)}$  is the state count before any symbols are transmitted. In figure 17.5 we assumed  $\mathbf{c}^{(0)} = [0, 1]^\top$ , 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  $\mathbf{A}$ .

$$\mathbf{c}^{(N)} \rightarrow \text{constant} \cdot \lambda_1^N \mathbf{e}_R^{(0)}. \quad (17.15)$$

Here,  $\lambda_1$  is the principal eigenvalue of  $\mathbf{A}$ .

So to find the capacity of any constrained channel, all we need to do is find the principal eigenvalue,  $\lambda_1$ , of its connection matrix. Then

$$C = \log_2 \lambda_1. \tag{17.16}$$

► **17.4 Back to our model channels**

Comparing figure 17.5a and figures 17.5b and c it looks as if channels B and C have the same capacity as channel A. The principal eigenvalues of the three trellises are the same (the eigenvectors for channels A and B are given at the bottom of table C.4, p.608). And indeed the channels are intimately related.

*Equivalence of channels A and B*

If we take any valid string  $\mathbf{s}$  for channel A and pass it through an *accumulator*, obtaining  $\mathbf{t}$  defined by:

$$\begin{aligned} t_1 &= s_1 \\ t_n &= t_{n-1} + s_n \bmod 2 \quad \text{for } n \geq 2, \end{aligned} \tag{17.17}$$

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*,

$$\begin{aligned} s_1 &= t_1 \\ s_n &= t_n - t_{n-1} \bmod 2 \quad \text{for } n \geq 2. \end{aligned} \tag{17.18}$$

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 17.7.<sup>[1, p.257]</sup> What is the relationship of channel C to channels A and B?

► **17.5 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 table 17.8 achieves a rate of  $3/5 = 0.6$ .

▷ Exercise 17.8.<sup>[1, p.257]</sup> 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?

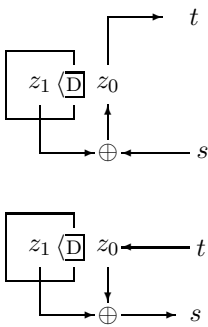


Figure 17.7. An accumulator and a differentiator.

$s$	$c(s)$
1	00000
2	10000
3	01000
4	00100
5	00010
6	10100
7	01010
8	10010

Table 17.8. A runlength-limited code for channel A.

### 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 showed that a sparse source with density  $f = 0.38$ , driving code  $C_2$ , would achieve capacity. And we know how to make sparsifiers (Chapter 6): 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 given as an exercise.

**Exercise 17.9.**<sup>[3]</sup> 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  $\mathbf{e}^{(L)\top}\mathbf{A} = \lambda\mathbf{e}^{(L)\top}$  with largest eigenvalue  $\lambda$ . Then construct a matrix  $\mathbf{Q}$  whose invariant distribution is proportional to  $e_i^{(R)}e_i^{(L)}$ , namely

$$Q_{s'|s} = \frac{e_{s'}^{(L)}A_{s's}}{\lambda e_s^{(L)}}. \quad (17.19)$$

[Hint: exercise 16.2 (p.245) might give helpful cross-fertilization here.]

▷ **Exercise 17.10.**<sup>[3, p.258]</sup> Show that when sequences are generated using the optimal transition probability matrix (17.19), the entropy of the resulting sequence is asymptotically  $\log_2 \lambda$  per symbol. [Hint: 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 example, a disk drive is easier to control if all blocks of 512 bytes are known to take exactly the same amount of disk real-estate. For some constrained channels we can make a simple modification to our variable-length encoding and offer such a guarantee, as follows. 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*. Then to transmit a string  $\mathbf{x}$  we encode the whole string 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 17.11.**<sup>[3C, p.258]</sup> How many valid sequences of length 8 starting with a 0 are there for the run-length-limited channels shown in figure 17.9?

What are the capacities of these channels?

Using a computer, find the matrices  $\mathbf{Q}$  for generating a random path through the trellises of the channel A, and the two run-length-limited channels shown in figure 17.9.

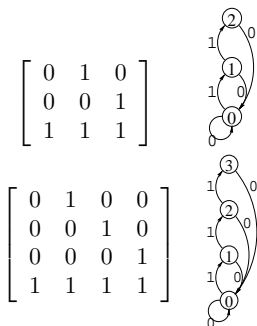


Figure 17.9. State diagrams and connection matrices for channels with maximum runlengths for 1s equal to 2 and 3.

- ▷ Exercise 17.12.<sup>[3, p.258]</sup> Consider the run-length-limited channel in which any length of run of 0s is permitted, and the maximum run length of 1s is a large number  $L$  such as nine or ninety.

Estimate the capacity of this channel. (Give the first two terms in a series expansion involving  $L$ .)

What, roughly, is the form of the optimal matrix  $\mathbf{Q}$  for generating a random path through the trellis of this channel? Focus on the values of the elements  $Q_{1|0}$ , the probability of 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 maximum runlength of 1s is nine.

## ► 17.6 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 encountered an unconstrained noiseless channel with variable symbol durations in exercise 6.18 (p.125). Solve that problem, and you've done this topic. The task is to determine the optimal frequencies with which the symbols should be used, given their durations.

There is a nice analogy between this task and the task of designing an optimal symbol code (Chapter 4). When we make a binary symbol code for a source with unequal probabilities  $p_i$ , the optimal message lengths are  $l_i^* = \log_2 1/p_i$ , so

$$p_i = 2^{-l_i^*}. \quad (17.20)$$

Similarly, 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

$$p_i^* = 2^{-\beta l_i}, \quad (17.21)$$

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 17.13.<sup>[3]</sup> 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 in bits per unit time assuming (a) that all four symbols have equal durations; or (b) that the symbol durations are 2, 4, 3 and 6 time units respectively.

Exercise 17.14.<sup>[4]</sup> How well-designed is Morse code for English (with, say, the probability distribution of figure 2.1)?

Exercise 17.15.<sup>[3C]</sup> How difficult is it to get DNA into a narrow tube?

To an information theorist, the entropy associated with a constrained channel reveals how much information can be conveyed over it. In statistical physics, the same calculations are done for a different reason: to predict the thermodynamics of polymers, for example.

As a toy example, consider a polymer of length  $N$  that can either sit in a constraining tube, of width  $L$ , or in the open where there are no constraints. In the open, the polymer adopts a state drawn at random from the set of one dimensional random walks, with, say, 3 possible directions per step. The entropy of this walk is  $\log 3$  per step, i.e., a total of  $N \log 3$ . [The free energy of the polymer is defined to be  $-kT$  times this, where  $T$  is the temperature.] In the tube, the polymer's one-dimensional walk can go in 3 directions unless the wall is in the way, so the connection matrix is, for example (if  $L = 10$ ),

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ & & & & \ddots & \ddots & \ddots & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Now, what is the entropy of the polymer? What is the *change* in entropy associated with the polymer entering the tube? If possible, obtain an expression as a function of  $L$ . Use a computer to find the entropy of the walk for a particular value of  $L$ , e.g. 20, and plot the probability density of the polymer's transverse location in the tube.

Notice the difference in capacity between two channels, one constrained and one unconstrained, is directly proportional to the force required to pull the DNA into the tube.

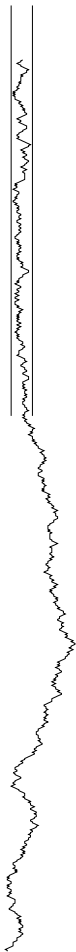


Figure 17.10. Model of DNA squashed in a narrow tube. The DNA will have a tendency to pop out of the tube, because, outside the tube, its random walk has greater entropy.

► 17.7 Solutions

Solution to exercise 17.5 (p.250). A file transmitted by  $C_2$  contains, on average, one-third 1s and two-thirds 0s.

If  $f = 0.38$ , the fraction of 1s is  $f/(1 + f) = (\gamma - 1.0)/(2\gamma - 1.0) = 0.2764$ .

Solution to exercise 17.7 (p.254). A valid string for channel C can be obtained from a valid string for channel A by first inverting it [ $1 \rightarrow 0$ ;  $0 \rightarrow 1$ ], then passing it through an accumulator. These operations are invertible, so any valid string for C can also be mapped onto a valid string for A. The only proviso here comes from the edge effects. If we assume that the first character transmitted over channel C is preceded by a string of zeroes, so that the first character is forced to be a 1 (figure 17.5c) then the two channels are exactly equivalent only if we assume that channel A's first character must be a zero.

Solution to exercise 17.8 (p.254). With  $N = 16$  transmitted bits, the largest integer number of source bits that can be encoded is 10, so the maximum rate of a fixed length code with  $N = 16$  is 0.625.



Solution to exercise 17.10 (p.255). Let the invariant distribution be

$$P(s) = \alpha e_s^{(L)} e_s^{(R)}, \quad (17.22)$$

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 4,  $S_t$  denotes the ensemble whose random variable is the state  $s_t$ .

$$H(S_t|S_{t-1}) = - \sum_{s,s'} P(s)P(s'|s) \log P(s'|s) \quad (17.23)$$

$$= - \sum_{s,s'} \alpha e_s^{(L)} e_s^{(R)} \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}} \log \frac{e_{s'}^{(L)} A_{s's}}{\lambda e_s^{(L)}} \quad (17.24)$$

$$= - \sum_{s,s'} \alpha e_s^{(R)} \frac{e_{s'}^{(L)} A_{s's}}{\lambda} \left[ \log e_{s'}^{(L)} + \log A_{s's} - \log \lambda - \log e_s^{(L)} \right]. \quad (17.25)$$

Now,  $A_{s's}$  is either 0 or 1, so the contributions from the terms proportional to  $A_{s's} \log A_{s's}$  are all zero. So

$$\begin{aligned} H(S_t|S_{t-1}) &= \log \lambda + -\frac{\alpha}{\lambda} \sum_{s'} \left( \sum_s A_{s's} e_s^{(R)} \right) e_{s'}^{(L)} \log e_{s'}^{(L)} + \\ &\quad \frac{\alpha}{\lambda} \sum_s \left( \sum_{s'} e_{s'}^{(L)} A_{s's} \right) e_s^{(R)} \log e_s^{(L)} \end{aligned} \quad (17.26)$$

$$= \log \lambda - \frac{\alpha}{\lambda} \sum_{s'} \lambda e_{s'}^{(R)} e_{s'}^{(L)} \log e_{s'}^{(L)} + \frac{\alpha}{\lambda} \sum_s \lambda e_s^{(L)} e_s^{(R)} \log e_s^{(L)} \quad (17.27)$$

$$= \log \lambda. \quad (17.28)$$

Solution to exercise 17.11 (p.255). 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.

Solution to exercise 17.12 (p.256). 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  $111\dots 1$  by  $111\dots 10$ , 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 6.18 (p.125). The capacity is the value of  $\beta$  such that the equation

$$Z(\beta) = \sum_{l=1}^{L+1} 2^{-\beta l} = 1 \quad (17.29)$$

is satisfied. The  $L+1$  terms in the sum correspond to the  $L+1$  possible strings that can be emitted,  $0, 10, 110, \dots, 11\dots 10$ . The sum is exactly given by:

$$Z(\beta) = 2^{-\beta} \frac{(2^{-\beta})^{L+1} - 1}{2^{-\beta} - 1}. \quad (17.30)$$

$$\left[ \text{Here we used } \sum_{n=0}^N ar^n = \frac{a(r^{N+1} - 1)}{r - 1} \right]$$

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  $\ln(1+x) \simeq x$ ,

$$Z(\beta) = 1 \quad (17.31)$$

$$\Rightarrow \beta \simeq 1 - 2^{-(L+2)} / \ln 2. \quad (17.32)$$

We evaluated the true capacities for  $L = 2$  and  $L = 3$  in an earlier exercise. The 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),

$$\begin{aligned} H(\text{the next } N \text{ chars}) &\simeq H_2(f) + [(N - 1)(1 - f) + (N - 2)f]C \\ &= H_2(f) + NC - fC \simeq H_2(f) + N - f. \end{aligned} \quad (17.33)$$

Differentiating and setting to zero to find the optimal  $f$ , we obtain:

$$\log_2 \frac{1-f}{f} \simeq 1 \Rightarrow \frac{1-f}{f} \simeq 2 \Rightarrow f \simeq 1/3. \quad (17.34)$$

The probability of emitting a 1 thus decreases from about 0.5 to about  $1/3$  as the number of emitted 1s increases.

Here is the optimal matrix:

$$\begin{bmatrix} 0 & .3334 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .4287 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .4669 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .4841 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .4923 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .4963 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4983 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4993 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4998 \\ 1 & .6666 & .5713 & .5331 & .5159 & .5077 & .5037 & .5017 & .5007 & .5002 \end{bmatrix}. \quad (17.35)$$

Our rough theory works.

# 18

## Crosswords and Codebreaking

In this chapter we make a random walk through a few topics related to language modelling.

### ► 18.1 Crosswords

The rules of crossword-making may be thought of as defining a constrained channel. The fact that *many* valid crosswords can be made demonstrates that this constrained channel has a capacity greater than zero.

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. In a language with high redundancy, on the other hand, it is hard to make crosswords (except perhaps a small number of trivial ones). The possibility of making crosswords in a language thus demonstrates a *bound on the redundancy* of that language. Crosswords are not normally written in genuine English. They are written in ‘word-English’, the language consisting of strings of words from a dictionary, separated by spaces.

- Exercise 18.1.<sup>[2]</sup> Estimate the capacity of word-English, in bits per character. [Hint: think of word-English as defining a constrained channel (Chapter 17) and see exercise 6.18 (p.125).]

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 4.1 which consists of  $W$  words all of length  $L$ . The entropy of such a language, per character, including inter-word spaces, is:

$$H_W \equiv \frac{\log_2 W}{L + 1}. \quad (18.1)$$

D	A	T	A	S	C	H	M	O	S	A	S	S
U	F	O	S	T	I	E	U	P	I	L	I	A
F	A	T	H	E	R	T	I	M	E	S	O	R
F	R	O	V	E	E	R	E	T	H	E	R	
			M	I	S	S	A	P	P	E	A	S
S	T	O	O	L	S	S	T	A	I	R		
T	I	L	T	S			U	N	L	U	C	K
U	T	A	H		S	T	E	A	L		E	R
D	O	V	E	C	O	T	E	S		C	N	O
			R	U	L	E	R		M	A	N	N
G	A	R	G	L	E	R		M	I	R	Y	
I	D	I	O	T			C	A	S	T		T
L	I	D	O		B	R	O	T	H	E	R	R
D	E	E	S		A	O	R	T	A		A	E
S	U	R	E		S	T	E	E	P		H	E

B	A	N	G	E	R		B	A	K	E	R	I	E	S	
V		A		O	R		I		O		L				
P	A	R	L	I	A	M	E	N	T		C	A	T	S	
	L		L		S		M		E	L	K		O		
V	A	L	E	N	T	I	N	E	S		E	T	N	A	
N		O		B		E			T						
C	A	N	O	E		R	H	A	P	S	O	D	Y		
H				E				U							
J	E	N	N	I	F	E	R		S	T	E	P	S		
		E				O		T		X		P			
D	U	E	T		N	U	T		C	R	A	C	K	E	R
S		T	W	O		A		A		U		R			
P	H	I	L		B	A	T	T	L	E	S	T	A	R	
E		E		E		E		I		E		T			
B	R	I	S	T	L	E	S		A	U	S	T	E	N	

Figure 18.1. Crosswords of types A (American) and B (British).

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  $f_1$  have roughly the values in table 18.2.

We 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 distribution then the entropy is 4.2. The redundancy of Wenglish stems from two sources: it tends to use some letters more than others; and there are only  $W$  words in the dictionary.

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 this 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

$$|T| = 2^{f_1 S H_0}. \quad (18.2)$$

The probability that one word of length  $L$  is validly filled-in is

$$\beta = \frac{W}{2^{L H_0}}, \quad (18.3)$$

and the probability that the whole crossword, made of  $f_w S$  words, is validly filled-in by a single typical in-filling is approximately

$$\beta^{f_w S}. \quad (18.4)$$

So the log of the number of valid crosswords of size  $S$  is estimated to be

$$\log \beta^{f_w S} |T| = S [(f_1 - f_w L) H_0 + f_w \log W] \quad (18.5)$$

$$= S [(f_1 - f_w L) H_0 + f_w (L + 1) H_W], \quad (18.6)$$

which is an increasing function of  $S$  only if

$$(f_1 - f_w L) H_0 + f_w (L + 1) H_W > 0. \quad (18.7)$$

So arbitrarily many crosswords can be made only if there's enough words in the Wenglish dictionary that

$$H_W > \frac{(f_w L - f_1)}{f_w (L + 1)} H_0. \quad (18.8)$$

Plugging in the values of  $f_1$  and  $f_w$  from table 18.2, we find the following.

Crossword type	A	B
Condition for crosswords	$H_W > \frac{1}{2} \frac{L}{L+1} H_0$	$H_W > \frac{1}{4} \frac{L}{L+1} H_0$

If we set  $H_0 = 4.2$  bits and assume there are  $W = 4000$  words in a normal English-speaker's dictionary, all with length  $L = 5$ , then we find that the condition for crosswords of type B is satisfied, but the condition for crosswords of type A is *only just* satisfied. This fits with my experience that crosswords of type A usually contain more obscure words.

	A	B
$f_w$	$\frac{2}{L+1}$	$\frac{1}{L+1}$
$f_1$	$\frac{L}{L+1}$	$\frac{3}{4} \frac{L}{L+1}$

Table 18.2. Factors  $f_w$  and  $f_1$  by which the number of words and number of letter-squares respectively are smaller than the total number of squares.

This calculation underestimates the number of valid Wenglish crosswords by counting only crosswords filled with 'typical' strings. If the monogram distribution is non-uniform then the true count is dominated by 'atypical' fillings-in, in which crossword-friendly words appear more often.

Further reading

These observations about crosswords were first made by Shannon (1948); I learned about them from Wolf and Siegel (1998). The topic is closely related to the capacity of two-dimensional constrained channels. An example of a two-dimensional constrained channel is a two-dimensional bar-code, as seen on parcels.

Exercise 18.2.<sup>[3]</sup> A two-dimensional channel is defined by the constraint that, of the eight neighbours of every interior pixel in an  $N \times N$  rectangular grid, four must be black and four white. (The counts of black and white pixels around boundary pixels are not constrained.) A binary pattern satisfying this constraint is shown in figure 18.3. What is the capacity of this channel, in bits per pixel, for large  $N$ ?

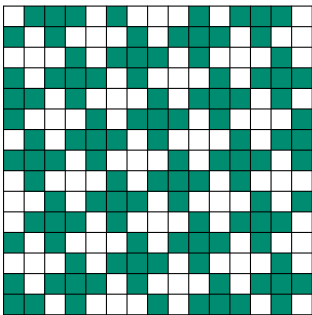


Figure 18.3. A binary pattern in which every pixel is adjacent to four black and four white pixels.

► 18.2 Simple language models

The Zipf–Mandelbrot distribution

The crudest model for a language is the monogram model, which asserts that each successive word is drawn independently from a distribution over words. What is the nature of this distribution over words?

Zipf’s law (Zipf, 1949) asserts that the probability of the  $r$ th most probable word in a language is approximately

$$P(r) = \frac{\kappa}{r^\alpha}, \tag{18.9}$$

where the exponent  $\alpha$  has a value close to 1, and  $\kappa$  is a constant. According to Zipf, a log–log plot of frequency versus word-rank should show a straight line with slope  $-\alpha$ .

Mandelbrot’s (1982) modification of Zipf’s law introduces a third parameter  $v$ , asserting that the probabilities are given by

$$P(r) = \frac{\kappa}{(r + v)^\alpha}. \tag{18.10}$$

For some documents, such as Jane Austen’s *Emma*, the Zipf–Mandelbrot distribution fits well – figure 18.4.

Other documents give distributions that are not so well fitted by a Zipf–Mandelbrot distribution. Figure 18.5 shows a plot of frequency versus rank for the L<sup>A</sup>T<sub>E</sub>X source of this book. Qualitatively, the graph is similar to a straight line, but a curve is noticeable. To be fair, this source file is not written in pure English – it is a mix of English, maths symbols such as ‘ $x$ ’, and L<sup>A</sup>T<sub>E</sub>X commands.

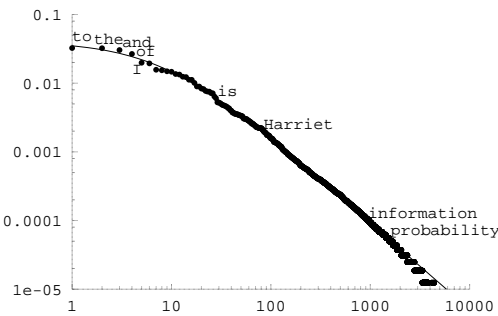


Figure 18.4. Fit of the Zipf–Mandelbrot distribution (18.10) (curve) to the empirical frequencies of words in Jane Austen’s *Emma* (dots). The fitted parameters are  $\kappa = 0.56$ ;  $v = 8.0$ ;  $\alpha = 1.26$ .

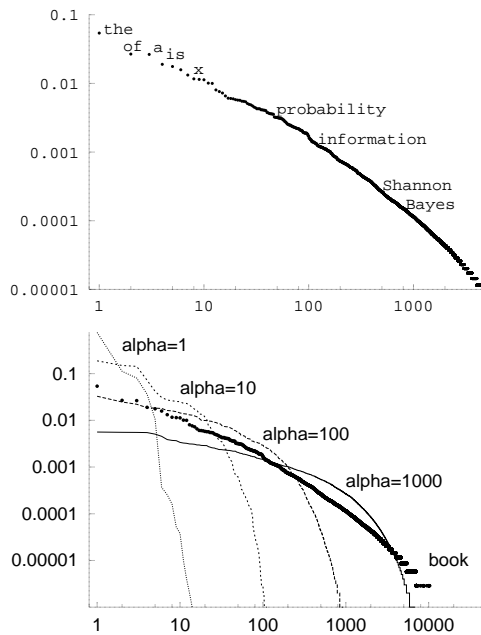


Figure 18.5. Log-log plot of frequency versus rank for the words in the L<sup>A</sup>T<sub>E</sub>X file of this book.

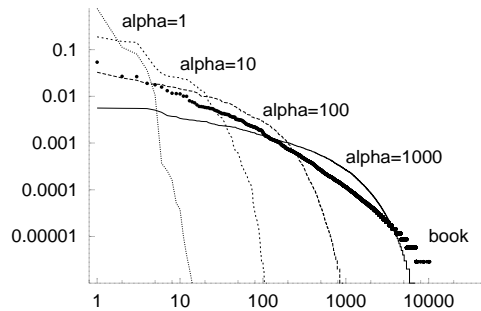


Figure 18.6. Zipf plots for four 'languages' randomly generated from Dirichlet processes with parameter  $\alpha$  ranging from 1 to 1000. Also shown is the Zipf plot for this book.

### The Dirichlet process

Assuming we are interested in monogram models for languages, what model should we use? One difficulty in modelling a language is the unboundedness of vocabulary. The greater the sample of language, the greater the number of words encountered. A generative model for a language should emulate this property. If asked 'what is the next word in a newly-discovered work of Shakespeare?' our probability distribution over words must surely include some non-zero probability for *words that Shakespeare never used before*. Our generative monogram model for language should also satisfy a consistency rule called *exchangeability*. If we imagine generating a new language from our generative model, producing an ever-growing corpus of text, all statistical properties of the text should be homogeneous: the probability of finding a particular word at a given location in the stream of text should be the same everywhere in the stream.

The Dirichlet process model is a model for a stream of symbols (which we think of as 'words') that satisfies the exchangeability rule and that allows the vocabulary of symbols to grow without limit. The model has one parameter  $\alpha$ . As the stream of symbols is produced, we identify each new symbol by a unique integer  $w$ . When we have seen a stream of length  $F$  symbols, we define the probability of the next symbol in terms of the counts  $\{F_w\}$  of the symbols seen so far thus: the probability that the next symbol is a new symbol, never seen before, is

$$\frac{\alpha}{F + \alpha}. \quad (18.11)$$

The probability that the next symbol is symbol  $w$  is

$$\frac{F_w}{F + \alpha}. \quad (18.12)$$

Figure 18.6 shows Zipf plots (i.e., plots of symbol frequency versus rank) for million-symbol 'documents' generated by Dirichlet process priors with values of  $\alpha$  ranging from 1 to 1000.

It is evident that a Dirichlet process is not an adequate model for observed distributions that roughly obey Zipf's law.

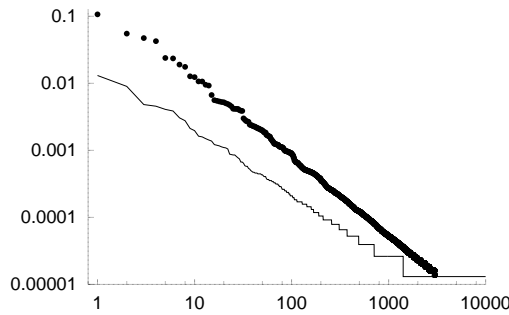


Figure 18.7. Zipf plots for the words of two ‘languages’ generated by creating successive characters from a Dirichlet process with  $\alpha = 2$ , and declaring one character to be the space character. The two curves result from two different choices of the space character.

With a small tweak, however, Dirichlet processes can produce rather nice Zipf plots. Imagine generating a language composed of elementary symbols using a Dirichlet process with a rather small value of the parameter  $\alpha$ , so that the number of reasonably frequent symbols is about 27. If we then declare one of those symbols (now called ‘characters’ rather than words) to be a space character, then we can identify the strings between the space characters as ‘words’. If we generate a language in this way then the frequencies of words often come out as very nice Zipf plots, as shown in figure 18.7. Which character is selected as the space character determines the slope of the Zipf plot – a less probable space character gives rise to a richer language with a shallower slope.

### ► 18.3 Units of information content

The information content of an outcome,  $x$ , whose probability is  $P(x)$ , is defined to be

$$h(x) = \log \frac{1}{P(x)}. \quad (18.13)$$

The entropy of an ensemble is an average information content,

$$H(X) = \sum_x P(x) \log \frac{1}{P(x)}. \quad (18.14)$$

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,

$$\text{‘log evidence for } \mathcal{H}_i \text{’} = \log P(D | \mathcal{H}_i), \quad (18.15)$$

or, in the case where just two hypotheses are being compared, we evaluate the ‘log odds’,

$$\log \frac{P(D | \mathcal{H}_1)}{P(D | \mathcal{H}_2)}, \quad (18.16)$$

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.

The names that have been given to these units are shown in table 18.8.

Unit	Expression that has those units
bit	$\log_2 p$
nat	$\log_e p$
ban	$\log_{10} p$
deciban (db)	$10 \log_{10} p$

Table 18.8. Units of measurement of information content.

The *bit* is the unit that we use most in this book. Because the word ‘bit’ has other meanings, a backup name for this unit is the *shannon*. A *byte* is 8 bits. A megabyte is  $2^{20} \simeq 10^6$  bytes. If one works in natural logarithms, information contents and weights of evidence are measured in *nats*. The most interesting units are the *ban* and the *deciban*.

*The history of the ban*

Let me tell you why a factor of ten in probability is called a ban. When Alan Turing and the other codebreakers at Bletchley Park were breaking each new day’s Enigma code, their task was a huge inference problem: to infer, given the day’s cyphertext, 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 that 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.

► 18.4 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.<sup>1</sup>

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 what the settings of the wheels were.

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  $26^3$  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 one character of a message, there would be another machine *in the same state* while sending a particular character in another message. Because the evolution of the machine’s state was deterministic, the two machines would remain in the same state as each other

<sup>1</sup>I’ve been most helped by descriptions given by Tony Sale (<http://www.codesandciphers.org.uk/lectures/>) and by Jack Good (1979), who worked with Turing at Bletchley.



for the rest of the transmission. 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.

*How to detect that two messages came from machines with a common state sequence*

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 cyphertexts  $\mathbf{x}$  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 cyphertexts are given by

$$\mathbf{x} = x_1x_2x_3 \dots = c_1(u_1)c_2(u_2)c_3(u_3) \dots \quad (18.17)$$

and

$$\mathbf{y} = y_1y_2y_3 \dots = c'_1(v_1)c'_2(v_2)c'_3(v_3) \dots, \quad (18.18)$$

where the codes  $c_t$  and  $c'_t$  are two unrelated time-varying permutations of the alphabet, and  $u_1u_2u_3 \dots$  and  $v_1v_2v_3 \dots$  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 cyphertexts is uniform. All cyphertexts are equally likely.

$$P(\mathbf{x}, \mathbf{y} | \mathcal{H}_0) = \left(\frac{1}{A}\right)^{2T} \text{ for all } \mathbf{x}, \mathbf{y} \text{ of length } T. \quad (18.19)$$

What about  $\mathcal{H}_1$ ? This hypothesis asserts that a *single* time-varying permutation  $c_t$  underlies both

$$\mathbf{x} = x_1x_2x_3 \dots = c_1(u_1)c_2(u_2)c_3(u_3) \dots \quad (18.20)$$

and

$$\mathbf{y} = y_1y_2y_3 \dots = c_1(v_1)c_2(v_2)c_3(v_3) \dots \quad (18.21)$$

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  $u_1u_2u_3 \dots$  and  $v_1v_2v_3 \dots$  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 equal to  $P(\mathbf{x}, \mathbf{y} | \mathcal{H}_0)$ , and the two hypotheses  $\mathcal{H}_0$  and  $\mathcal{H}_1$  would be indistinguishable.

We make progress by assuming that the plaintext is not completely random. Both plaintexts are written in a language, and that language has redundancies. Assume for example that 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; if  $\mathcal{H}_1$  is true, two synchronized letters from the two cyphertexts 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 bigrams and trigrams at the same time as each other, giving rise, if  $\mathcal{H}_1$  is true,

<b>u</b>	LITTLE-JACK-HORNER-SAT-IN-THE-CORNER-EATING-A-CHRISTMAS-PIE--HE-PUT-IN-H
<b>v</b>	RIDE-A-COCK-HORSE-TO-BANBURY-CROSS-TO-SEE-A-FINE-LADY-UPON-A-WHITE-HORSE
matches:	. * . . . . * . . . . . * . . . . . * . . . . . * . . . . . * . . . . . * . . . . . *

to a little burst of 2 or 3 identical letters. Table 18.9 shows such a coincidence in two plaintext messages that are unrelated, except that they are both written in English.

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. This method was first used by the Polish codebreaker Rejewski.

Let’s look at the simple case of a monogram language model and estimate how long a message is needed to be able to decide whether two machines are in the same state. 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 nonuniform: consider two single characters,  $x_t = c_t(u_t)$  and  $y_t = c_t(v_t)$ ; the probability that they are identical is

$$\sum_{u_t,v_t} P(u_t)P(v_t)\mathbb{1}[u_t=v_t] = \sum_i p_i^2 \equiv m. \tag{18.22}$$

We give this quantity the name  $m$ , for ‘match probability’; for both English and German,  $m$  is about  $2/26$  rather than  $1/26$  (the value that would hold for a completely random language). Assuming that  $c_t$  is an ideal random permutation, the probability of  $x_t$  and  $y_t$  is, by symmetry,

$$P(x_t,y_t|\mathcal{H}_1) = \begin{cases} \frac{m}{A} & \text{if } x_t = y_t \\ \frac{(1-m)}{A(A-1)} & \text{for } x_t \neq y_t. \end{cases} \tag{18.23}$$

Given a pair of cyphertexts  $\mathbf{x}$  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

$$\log \frac{P(\mathbf{x},\mathbf{y}|\mathcal{H}_1)}{P(\mathbf{x},\mathbf{y}|\mathcal{H}_0)} = M \log \frac{m/A}{1/A^2} + N \log \frac{\frac{(1-m)}{A(A-1)}}{1/A^2} \tag{18.24}$$

$$= M \log mA + N \log \frac{(1-m)A}{A-1}. \tag{18.25}$$

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 (18.25), 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

Table 18.9. Two aligned pieces of English plaintext, **u** and **v**, with matches marked by \*. Notice that there are twelve matches, including a run of six, whereas the expected number of matches in two completely random strings of length  $T = 74$  would be about 3. The two corresponding cyphertexts from two machines in identical states would also have twelve matches.

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 (20 decibans) in favour of one hypothesis or the other.

So, two English plaintexts have more matches than two random strings. Furthermore, because consecutive characters in English are not independent, the bigram and trigram statistics of English are nonuniform and the matches tend to occur in bursts of consecutive matches. [The same observations also apply to German.] Using better language models, the evidence contributed by runs of matches was more accurately computed. Such a 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 between two machines that were actually in unrelated states.

### Further reading

For further reading about Turing and Bletchley Park, see Hodges (1983) and Good (1979). For an in-depth read about cryptography, Schneier's (1996) book is highly recommended. It is readable, clear, and entertaining.

## ► 18.5 Exercises

- ▷ Exercise 18.3.<sup>[2]</sup> Another weakness in the design of the Enigma machine, which was intended to emulate a perfectly random time-varying permutation, is that it never mapped a letter to itself. When you press Q, what comes out is always a different letter from Q. How much information per character is leaked by this design flaw? How long a crib would be needed to be confident that the crib is correctly aligned with the cyphertext? And how long a crib would be needed to be able confidently to identify the correct key?

[A *crib* is a guess for what the plaintext was. Imagine that the Brits know that a very important German is travelling from Berlin to Aachen, and they intercept Enigma-encoded messages sent to Aachen. It is a good bet that one or more of the original plaintext messages contains the string OBERSTURMBANNFUEHRERXGRAFHEINRICHXVONXWEIZSAECKER, the name of the important chap. A crib could be used in a brute-force approach to find the correct Enigma key (feed the received messages through all possible Enigma machines and see if any of the putative decoded texts match the above plaintext). This question centres on the idea that the crib can also be used in a much less expensive manner: slide the plaintext crib along all the encoded messages until a perfect *mismatch* of the crib and the encoded message is found; if correct, this alignment then tells you a lot about the key.]

## 19

---

### *Why have Sex? Information Acquisition and Evolution*

Evolution has been happening on earth for about the last  $10^9$  years. Undeniably, *information has been acquired* during this process. Thanks to the tireless work of the Blind Watchmaker, some cells now carry within them all the information required to be outstanding spiders; other cells carry all the information required to make excellent octopuses. Where did this information come from?

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 (including the other organisms). The teaching signal is only a few bits per individual: an individual simply has a smaller or larger number of grandchildren, depending on the individual's fitness. 'Fitness' is a broad term that could cover

- the ability of an antelope to run faster than other antelopes and hence avoid being eaten by a lion;
- the ability of a lion to be well-enough camouflaged and run fast enough to catch one antelope per day;
- the ability of a peacock to attract a peahen to mate with it;
- the ability of a peahen to rear many young simultaneously.

The fitness of an organism is largely determined by its DNA – both the coding regions, or genes, and the non-coding regions (which play an important role in regulating the transcription of genes). We'll think of fitness as a function of the DNA sequence and the environment.

How does the DNA determine fitness, and how does information get from natural selection into the genome? Well, if the gene that codes for one of an antelope's proteins is defective, that antelope might get eaten by a lion early in life and have only two grandchildren rather than forty. The information content of natural selection is fully contained in a specification of which offspring survived to have children – an information content of *at most one bit per offspring*. 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.

So, how many bits per generation are acquired by the species as a whole by natural selection? How many bits has natural selection succeeded in conveying to the human branch of the tree of life, since the divergence between

Australopithecines and apes 4 000 000 years ago? Assuming a generation time of 10 years for reproduction, there have been about 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, as we noted, 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 allow for only 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. [One human genome contains about  $3 \times 10^9$  nucleotides.] It is certainly the case that the genomic overlap between apes and humans is huge, but is the difference that small?

In this chapter, we'll develop a crude model of the process of information acquisition through evolution, based on the assumption that a gene with two defects is typically likely to be more defective than a gene with one defect, and an organism with two defective genes is likely to be less fit than an organism with one defective gene. Undeniably, this is a crude model, since real biological systems are baroque constructions with complex interactions. Nevertheless, we persist with a simple model because it readily yields striking results.

What we find from this simple model is that

1. John Maynard Smith's figure of 1 bit per generation is correct for an *asexually-reproducing* population;
2. in contrast, *if the species reproduces sexually*, the rate of information acquisition can be as large as  $\sqrt{G}$  bits per generation, where  $G$  is the size of the genome.

We'll also find interesting results concerning the maximum mutation rate that a species can withstand.

## ► 19.1 The model

We study a simple model of a reproducing population of  $N$  individuals with a genome of size  $G$  bits: variation is produced by mutation or by recombination (i.e., sex) 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.

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:

$$F(\mathbf{x}) = \sum_{g=1}^G x_g. \quad (19.1)$$

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:

$$z_g = \begin{cases} x_g & \text{with probability } 1/2 \\ y_g & \text{with probability } 1/2. \end{cases} \quad (19.2)$$

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.

## ► 19.2 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 at most 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

$$\overline{\delta f}_{\text{child}} = (1 - 2m)\delta f \quad (19.3)$$

and variance

$$\frac{m(1-m)}{G} \simeq \frac{m}{G}. \quad (19.4)$$

If the population of parents has mean  $\delta f(t)$  and variance  $\sigma^2(t) \equiv \beta m/G$ , then the child population, before selection, will have mean  $(1-2m)\delta f(t)$  and variance  $(1+\beta)m/G$ . Natural selection chooses the upper half of this distribution, so the mean fitness and variance of fitness at the next generation are given by

$$\delta f(t+1) = (1-2m)\delta f(t) + \alpha \sqrt{(1+\beta)} \sqrt{\frac{m}{G}}, \quad (19.5)$$

$$\sigma^2(t+1) = \gamma(1+\beta) \frac{m}{G}, \quad (19.6)$$

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

$$\gamma(1+\beta) = \beta, \text{ so } (1+\beta) = \frac{1}{1-\gamma}, \quad (19.7)$$

and the factor  $\alpha\sqrt{(1+\beta)}$  in equation (19.5) 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:

$$\frac{df}{dt} \simeq -2m\delta f + \sqrt{\frac{m}{G}}, \quad (19.8)$$

which, assuming  $G(\delta f)^2 \gg 1$ , is maximized for

$$m_{\text{opt}} = \frac{1}{16G(\delta f)^2}, \quad (19.9)$$

at which point,

$$\left(\frac{df}{dt}\right)_{\text{opt}} = \frac{1}{8G(\delta f)}. \quad (19.10)$$

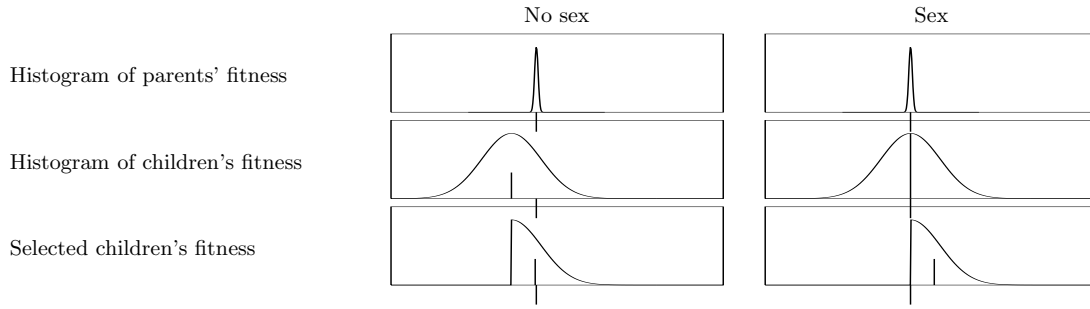
So the rate of increase of fitness  $F = fG$  is at most

$$\frac{dF}{dt} = \frac{1}{8(\delta f)} \text{ per generation.} \quad (19.11)$$

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 last only 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

$$\delta f(t) = \frac{1}{2\sqrt{mG}}(1 - ce^{-2mt}), \quad (19.12)$$



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  $F_{\text{eqm}} = (1/2 + 1/(2\sqrt{mG}))G$ .

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 described below.

### Theory of sex

The analysis of the sexual population becomes tractable with two approximations: first, we assume that the gene-pool 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 is proportional to this standard deviation, *the fitness increase per generation scales as the square root of the size of the genome,  $\sqrt{G}$* . As shown in box 19.2, the mean fitness  $\bar{F} = fG$  evolves in accordance with the differential equation:

$$\frac{d\bar{F}}{dt} \simeq \eta \sqrt{f(t)(1-f(t))G}, \quad (19.13)$$

where  $\eta \equiv \sqrt{2/(\pi+2)}$ . The solution of this equation is

$$f(t) = \frac{1}{2} \left[ 1 + \sin \left( \frac{\eta}{\sqrt{G}}(t+c) \right) \right], \quad \text{for } t+c \in \left( -\frac{\pi}{2}\sqrt{G}/\eta, \frac{\pi}{2}\sqrt{G}/\eta \right), \quad (19.14)$$

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.

### Simulations

Figure 19.3a shows the fitness of a sexual 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  $f(t)G$  from equation (19.14), which fits remarkably well.

In contrast, figures 19.3(b) and (c) show the evolving fitness when variation is produced by mutation at rates  $m = 0.25/G$  and  $m = 6/G$  respectively. Note the difference in the horizontal scales from panel (a).

**Figure 19.1.** Why sex is better than sex-free reproduction. 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 average 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  $O(\sqrt{G})$ .



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

$$F_{\text{child}} \sim \text{Normal} \left( \text{mean} = f(t)G, \text{variance} = \frac{1}{2}f(t)(1-f(t))G \right).$$

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.

If we 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 children's fitnesses are distributed as

$$F_{\text{child}} \sim \text{Normal} \left( \text{mean} = f(t)G, \text{variance} = \left(1 + \frac{\beta}{2}\right) \frac{1}{2}f(t)(1-f(t))G \right).$$

Natural selection selects the children on the upper side of this distribution. The mean increase in fitness will be

$$\bar{F}(t+1) - \bar{F}(t) = [\alpha(1 + \beta/2)^{1/2}/\sqrt{2}] \sqrt{f(t)(1-f(t))G},$$

and the variance of the surviving children will be

$$\sigma^2(t+1) = \gamma(1 + \beta/2) \frac{1}{2}f(t)(1-f(t))G,$$

where  $\alpha = \sqrt{2/\pi}$  and  $\gamma = (1 - 2/\pi)$ . If there is dynamic equilibrium [ $\sigma^2(t+1) = \sigma^2(t)$ ] then the factor in (19.2) is

$$\alpha(1 + \beta/2)^{1/2}/\sqrt{2} = \sqrt{\frac{2}{(\pi + 2)}} \simeq 0.62.$$

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*,

$$\frac{d\bar{F}}{dt} \simeq \eta \sqrt{f(t)(1-f(t))G} \text{ bits per generation.}$$

Box 19.2. Details of the theory of sex.

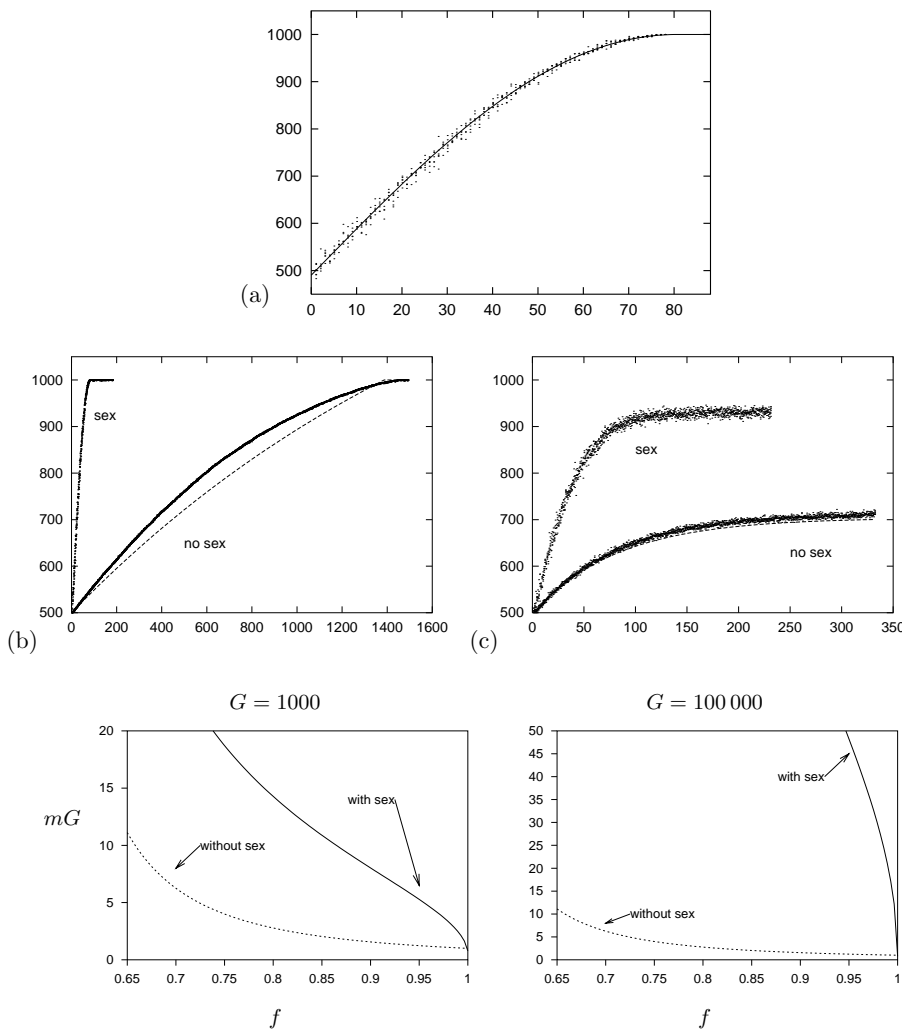


Figure 19.3. Fitness as a function of time. The genome size is  $G = 1000$ . The dots show the fitness of six randomly selected individuals from the birth population at each generation. The initial population of  $N = 1000$  had randomly generated genomes with  $f(0) = 0.5$  (exactly). (a) Variation produced by sex alone. Line shows theoretical curve (19.14) for infinite homogeneous population. (b,c) Variation produced by mutation, with and without sex, when the mutation rate is  $mG = 0.25$  (b) or 6 (c) bits per genome. The dashed line shows the curve (19.12).

Figure 19.4. Maximal tolerable mutation rate, shown as number of errors per genome ( $mG$ ), versus normalized fitness  $f = F/G$ . Left panel: genome size  $G = 1000$ ; right:  $G = 100000$ . Independent of genome size, a parthenogenetic species (no sex) can tolerate only of order 1 error per genome per generation; a species that uses recombination (sex) can tolerate far greater mutation rates.

Exercise 19.1.<sup>[3, p.280]</sup> Dependence on population size. How do the results for a sexual population depend on the population size? We anticipate that there is a minimum population size above which the theory of sex is accurate. How is that minimum population size related to  $G$ ?

Exercise 19.2.<sup>[3]</sup> Dependence on crossover mechanism. In the simple model of sex, each bit is taken at random from one of the two parents, that is, we allow crossovers to occur with probability 50% between any two adjacent nucleotides. How is the model affected (a) if the crossover probability is smaller? (b) if crossovers occur exclusively at *hot-spots* located every  $d$  bits along the genome?

### ► 19.3 The maximal tolerable mutation rate

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

$$\frac{df}{dt} \simeq -2m\delta f + \eta\sqrt{2}\sqrt{\frac{m + f(1-f)/2}{G}}, \quad (19.15)$$

which is positive if the mutation rate satisfies

$$m < \eta \sqrt{\frac{f(1-f)}{G}}. \quad (19.16)$$

Let us compare this rate with the result in the absence of sex, which, from equation (19.8), is that the maximum tolerable mutation rate is

$$m < \frac{1}{G} \frac{1}{(2\delta f)^2}. \quad (19.17)$$

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 mutation 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.

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  $G = 10^9$  coding nucleotides make at least occasional use of recombination. If the brood size is 12, then this number falls to  $G = 2.5 \times 10^8$ .

## ► 19.4 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.

We define the information acquired at an intermediate fitness to be the amount of selection (measured in bits) required to select the perfect 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

$$I = \sum_g \log_2 \frac{f_g}{1/2} \text{ bits}. \quad (19.18)$$

If all the fractions  $f_g$  are equal to  $F/G$ , then

$$I = G \log_2 \frac{2F}{G}, \quad (19.19)$$

which is well approximated by

$$\tilde{I} \equiv 2(F - G/2). \quad (19.20)$$

The rate of information acquisition is thus roughly two times the rate of increase of fitness in the population.

## ► 19.5 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.

### *‘The cost of males’ – stability of a gene for sex or parthenogenesis*

Why do people declare sex to be a mystery? The main motivation for being mystified is an idea called the ‘cost of males’. Sexual reproduction is disadvantageous compared with asexual reproduction, it’s argued, because of every two offspring produced by sex, one (on average) is a useless male, incapable of child-bearing, and only one is a productive female. In the same time, a parthenogenetic mother could give birth to *two* female clones. To put it another way, 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%. Thus if there were two versions of a species, one reproducing with and one without sex, the single mothers would be expected to outstrip their sexual cousins. The simple model presented thus far did not include either genders or the ability to convert from sexual reproduction to asexual, but we can easily modify the model.

We modify 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 results depend on the number of children had by a single parthenogenetic mother,  $K_p$  and the number of children born by a sexual couple,  $K_s$ . Both ( $K_p=2$ ,  $K_s=4$ ) and ( $K_p=4$ ,  $K_s=4$ ) are reasonable models. The former ( $K_p=2$ ,  $K_s=4$ ) would seem most appropriate in the case of unicellular organisms, where the cytoplasm of both parents goes into the children. The latter ( $K_p=4$ ,  $K_s=4$ ) 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 twice the expression (19.17) derived before for  $K_p=2$ . 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 19.5 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

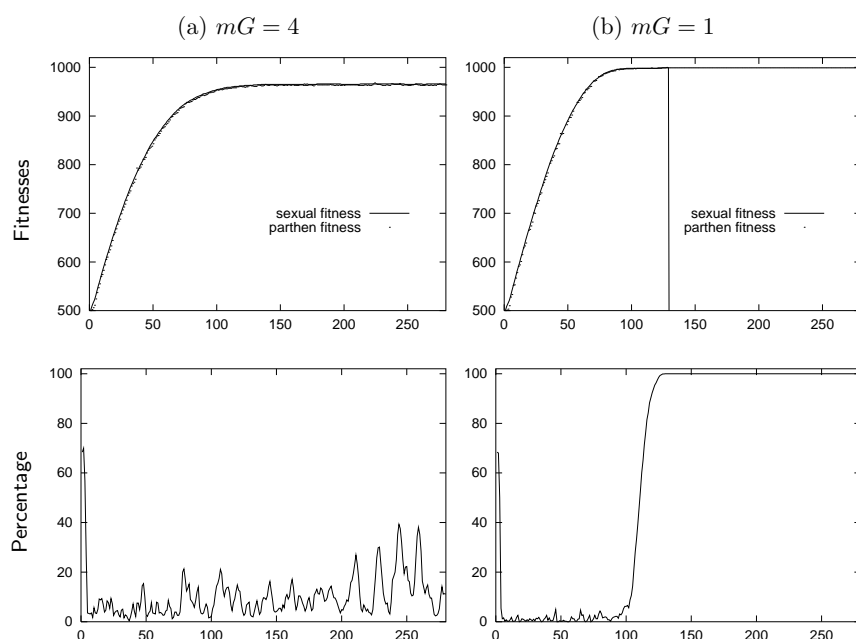


Figure 19.5. Results when there is a gene for parthenogenesis, and no interbreeding, and single mothers produce as many children as sexual couples.  $G = 1000$ ,  $N = 1000$ . (a)  $mG = 4$ ; (b)  $mG = 1$ . Vertical axes show the fitnesses of the two sub-populations, and the percentage of the population that is parthenogenetic.

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 the presence of a higher mutation rate ( $mG = 4$ ), however, the parthenogens 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 (2002) 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.

### Additive fitness function

Of course, our results depend 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, it seems plausible that the fitness would still involve a sum of such interacting terms, with the number of terms being some fraction of the genome size  $G$ .

**Exercise 19.3.**<sup>[3C]</sup> Investigate how fast sexual and asexual species evolve if they have a fitness function with interactions. For example, let the fitness be a sum of exclusive-ors of pairs of bits; compare the evolving fitnesses with those of the sexual and asexual species with a simple additive fitness function.

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. Assume the effectiveness of the peptide is a 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, accidentally produce a working enzyme; and it 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 I 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.

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, I 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 small, in-bred populations are the benefits of sex expected to be diminished.

In summary: Why have sex? Because sex is good for your bits!

### Further reading

How did a high-information-content self-replicating system ever emerge in the first place? In the general area of the origins of life and other tricky questions about evolution, I highly recommend Maynard Smith and Száthmary (1995), Maynard Smith and Száthmary (1999), Kondrashov (1988), Maynard Smith (1988), Ridley (2000), Dyson (1985), Cairns-Smith (1985), and Hopfield (1978).

## ► 19.6 Further exercises

**Exercise 19.4.**<sup>[3]</sup> 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. [See Appendix C.4.]

**Exercise 19.5.**<sup>[4]</sup> Given that DNA replication is achieved by bumbling Brownian motion and ordinary thermodynamics in a biochemical porridge at a temperature of 35 C, it's astonishing that the error-rate of DNA replication is about  $10^{-9}$  per replicated nucleotide. How can this reliability be achieved, given that the energetic difference between a correct base-pairing and an incorrect one is only one or two hydrogen bonds and the thermal energy  $kT$  is only about a factor of four smaller than the free energy associated with a hydrogen bond? If ordinary thermodynamics is what favours correct base-pairing, surely the frequency of incorrect base-pairing should be about

$$f = \exp(-\Delta E/kT), \quad (19.21)$$

where  $\Delta E$  is the free energy difference, i.e., an error frequency of  $f \simeq 10^{-4}$ ? How has DNA replication cheated thermodynamics?

The situation is equally perplexing in the case of protein synthesis, which translates an mRNA sequence into a polypeptide in accordance with the genetic code. Two specific chemical reactions are protected against errors: the binding of tRNA molecules to amino acids, and the production of the polypeptide in the ribosome, which, like DNA replication, involves base-pairing. Again, the fidelity is high (an error rate of about  $10^{-4}$ ), and this fidelity can't be caused by the energy of the 'correct' final state being especially low – the correct polypeptide sequence is not expected to be significantly lower in energy than any other sequence. How do cells perform error correction? (See Hopfield (1974), Hopfield (1980)).

**Exercise 19.6.**<sup>[2]</sup> While the genome acquires information through natural selection at a rate of a few bits per generation, 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.

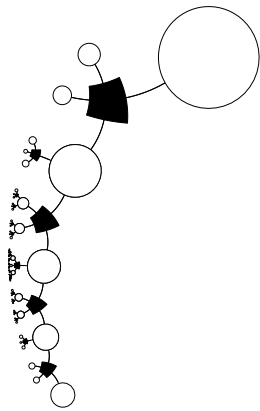
## ► 19.7 Solutions

**Solution to exercise 19.1 (p.275).** For small enough  $N$ , whilst the average fitness of the population increases, some unlucky bits become frozen into the bad state. (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 contain some 0-bits too. The smaller the population, the greater the number of frozen 0-bits is expected to be. How small can the population size  $N$  be if the theory of sex is accurate?

We find experimentally that the theory based on assuming homogeneity fits poorly only 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.

## Part IV

# Probabilities and Inference





---

## About 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 inferring 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 21, 24, 25, and 26.

**Approximate methods** can be subdivided into

1. **deterministic approximations**, which include maximum likelihood (Chapter 22), Laplace's method (Chapters 27 and 28) and variational methods (Chapter 33); and
2. **Monte Carlo methods** – techniques in which random numbers play an integral part – which will be discussed in Chapters 29, 30, and 32.

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 20 discusses the problem of clustering; subsequent chapters discuss the probabilistic interpretation of clustering as mixture modelling.

Chapter 21 discusses the option of dealing with probability distributions by completely enumerating all hypotheses. Chapter 22 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. Chapter 23 reviews the probability distributions that arise most often in Bayesian inference. Chapters 24, 25, and 26 discuss another way of avoiding the cost of complete enumeration: marginalization. Chapter 25 discusses message-passing methods appropriate for graphical models, using the decoding of error-correcting codes as an example. Chapter 26 combines these ideas with message-passing concepts from Chapters 16 and 17. These chapters are a prerequisite for the understanding of advanced error-correcting codes.

Chapter 27 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 28.

Chapter 29 discusses Monte Carlo methods. Chapter 30 gives details of state-of-the-art Monte Carlo techniques.

Chapter 31 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 32 describes ‘exact’ Monte Carlo methods and demonstrates their application to the Ising model.

Chapter 33 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 42) and the EM algorithm, which is an important method in latent-variable modelling. Chapter 34 discusses a particularly simple latent variable model called independent component analysis.

Chapter 35 discusses a ragbag of assorted inference topics. Chapter 36 discusses a simple example of decision theory. Chapter 37 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 4, the *most probable* outcome from a source is often not a *typical* outcome from that source. Similarly, the most probable hypothesis given some data may be atypical of the whole set of reasonably-plausible hypotheses.

*About Chapter 20*

Before reading the next chapter, exercise 2.17 (p.36) and section 11.2 (inferring the input to a Gaussian channel) are recommended reading.

## 20

---

### *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', and the objective function that measures how well the predictive model is working is the information content of the data,  $\log 1/P(\{\mathbf{x}\})$ .

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  $k_1, k_2, \dots, k_N$  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

$$D = \sum_{\mathbf{x}} P(\mathbf{x}) \frac{1}{2} \left[ \mathbf{m}^{(k(\mathbf{x}))} - \mathbf{x} \right]^2. \quad (20.1)$$

[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 satellite pictures of ocean surfaces, then maybe patches of image that are not well compressed by the vector quantizer are the patches that contain ships! 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  $K$  clusters compete with each other for the right to own the data points.

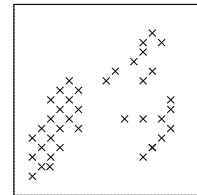


Figure 20.1.  $N = 40$  data points.

## ► 20.1 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.

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  $\mathbf{x}$  has  $I$  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,

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \sum_i (x_i - y_i)^2. \quad (20.2)$$

To start the K-means algorithm (algorithm 20.2), the  $K$  means  $\{\mathbf{m}^{(k)}\}$  are initialized in some way, for example to random values. K-means 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 20.3, 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 the assignments are unchanged so the means remain unmoved when updated. The K-means algorithm always converges to a fixed point.

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.

**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  $\hat{k}^{(n)}$ .

$$\hat{k}^{(n)} = \underset{k}{\operatorname{argmin}} \{d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\}. \quad (20.3)$$

An alternative, equivalent representation of this assignment of points to clusters is given by ‘responsibilities’, which are indicator variables  $r_k^{(n)}$ . 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.

$$r_k^{(n)} = \begin{cases} 1 & \text{if } \hat{k}^{(n)} = k \\ 0 & \text{if } \hat{k}^{(n)} \neq k. \end{cases} \quad (20.4)$$

*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.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \quad (20.5)$$

where  $R^{(k)}$  is the total responsibility of mean  $k$ ,

$$R^{(k)} = \sum_n r_k^{(n)}. \quad (20.6)$$

*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 20.2. The K-means clustering algorithm.

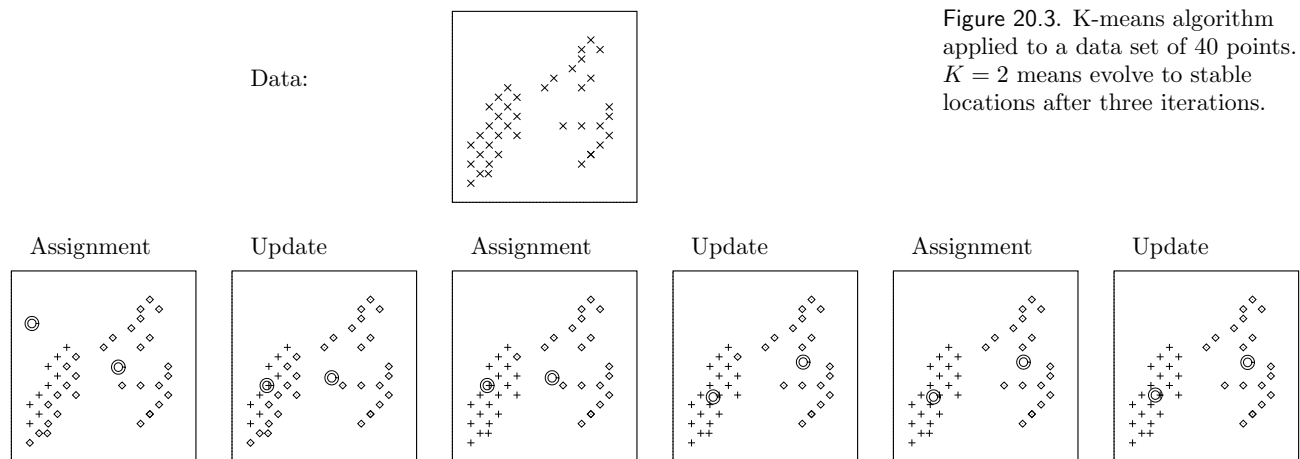


Figure 20.3. K-means algorithm applied to a data set of 40 points.  $K = 2$  means evolve to stable locations after three iterations.

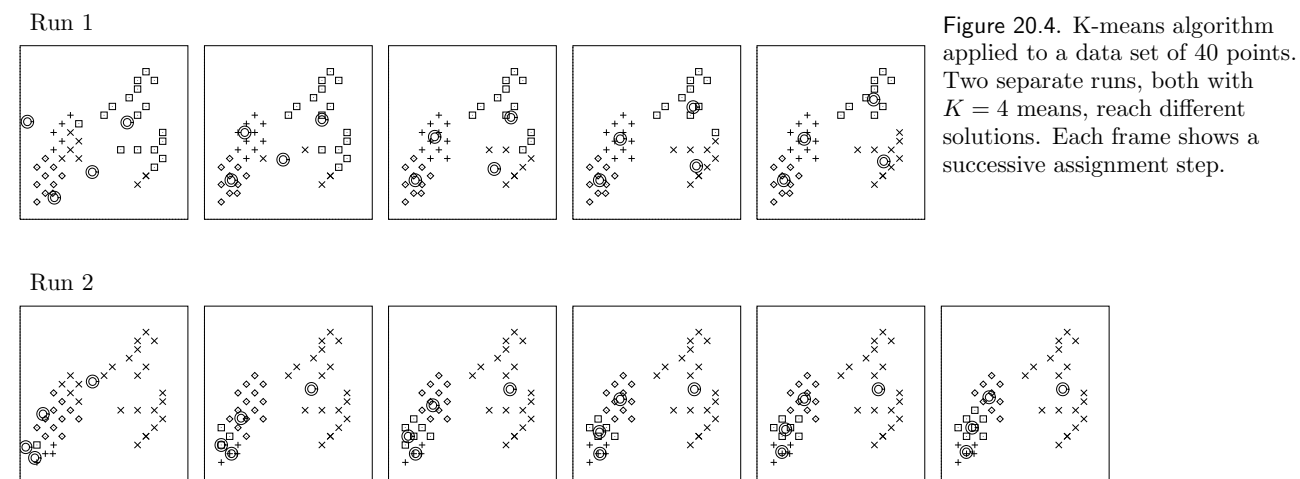


Figure 20.4. K-means algorithm applied to a data set of 40 points. Two separate runs, both with  $K = 4$  means, reach different solutions. Each frame shows a successive assignment step.

Exercise 20.1. [4, p.291] 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 20.4. 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

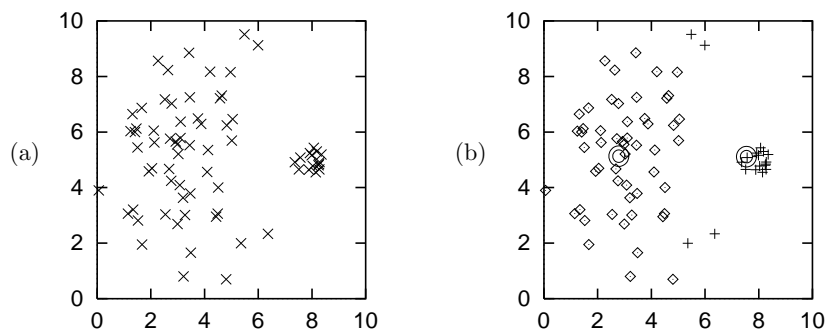


Figure 20.5. K-means algorithm for a case with two dissimilar clusters. (a) The “little ‘n’ large” data. (b) A stable set of assignments and means. Note that four points belonging to the broad cluster have been incorrectly assigned to the narrower cluster. (Points assigned to the right-hand cluster are shown by plus signs.)

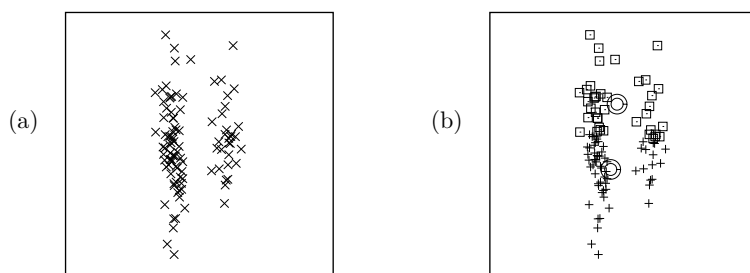


Figure 20.6. Two elongated clusters, and the stable solution found by the K-means algorithm.

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 20.5a 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 20.5b 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 displaced to the left of the true centres of the clusters. The K-means algorithm takes account only 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 that actually belong to the broad cluster are incorrectly assigned to the narrow cluster.

Figure 20.6 shows another case of K-means behaving badly. The data evidently fall into two elongated clusters. But the only stable state of the K-means algorithm is that shown in figure 20.6b: the two clusters have been sliced in half! These two examples show that there is something wrong with the distance  $d$  in the K-means 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.

## ► 20.2 Soft K-means clustering

These criticisms of K-means motivate the ‘soft K-means algorithm’, algorithm 20.7. 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  $n$ ).

$$r_k^{(n)} = \frac{\exp(-\beta d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)}))}{\sum_{k'} \exp(-\beta d(\mathbf{m}^{(k')}, \mathbf{x}^{(n)}))}. \quad (20.7)$$

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.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \quad (20.8)$$

where  $R^{(k)}$  is the total responsibility of mean  $k$ ,

$$R^{(k)} = \sum_n r_k^{(n)}. \quad (20.9)$$

Algorithm 20.7. Soft K-means algorithm, version 1.

Notice the similarity of this soft K-means algorithm to the hard K-means algorithm 20.2. The update step is identical; the only difference is that 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’ (20.7).

- Exercise 20.2.<sup>[2]</sup> 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 it. The soft K-means algorithm is demonstrated in figure 20.8. The lengthscale is shown by the radius of the circles surrounding the four means. Each panel shows the final fixed point reached for a different value of the lengthscale  $\sigma$ .

## ► 20.3 Conclusion

At this point, we may have fixed some of the problems with the original K-means algorithm by introducing an extra complexity-control parameter  $\beta$ . But how should we set  $\beta$ ? And what about the problem of the elongated clusters,



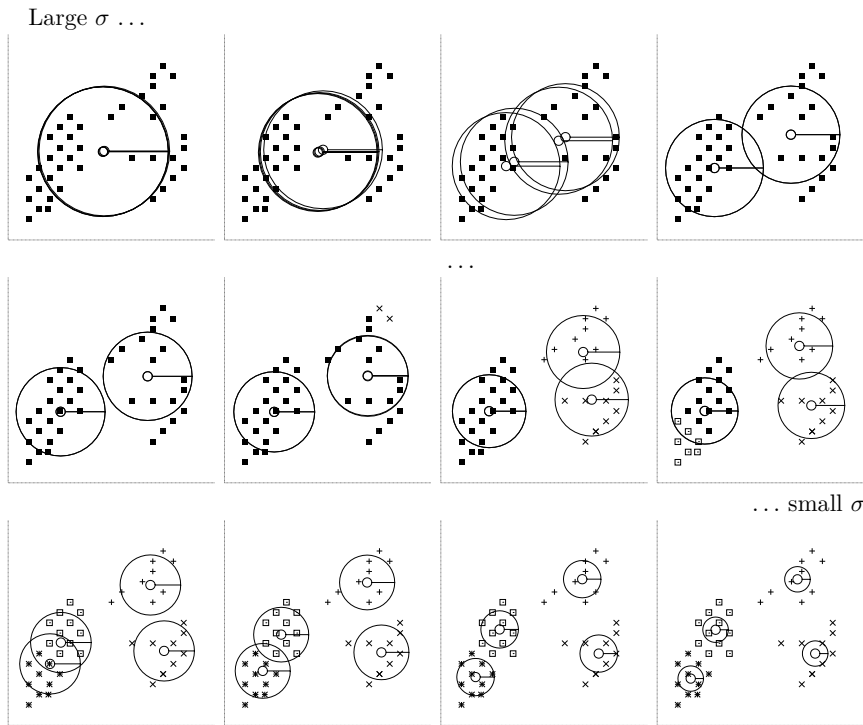


Figure 20.8. Soft K-means algorithm, version 1, applied to a data set of 40 points.  $K = 4$ . Implicit lengthscale parameter  $\sigma = 1/\beta^{1/2}$  varied from a large to a small value. Each picture shows the state of all four means, with the implicit lengthscale shown by the radius of the four 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.

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.

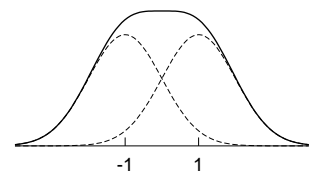
### Further reading

For a vector-quantization approach to clustering see (Luttrell, 1989; Luttrell, 1990).

## ► 20.4 Exercises

- ▷ Exercise 20.3.<sup>[3, p.291]</sup> Explore the properties of the soft K-means algorithm, version 1, assuming that the datapoints  $\{\mathbf{x}\}$  come from a *single* separable two-dimensional Gaussian distribution with mean zero and variances  $(\text{var}(x_1), \text{var}(x_2)) = (\sigma_1^2, \sigma_2^2)$ , 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 20.4.<sup>[3]</sup> 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.



## ► 20.5 Solutions

**Solution to exercise 20.1 (p.287).** 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 would be 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.

**Solution to exercise 20.3 (p.290).** If the means are initialized to  $\mathbf{m}^{(1)} = (m, 0)$  and  $\mathbf{m}^{(2)} = (-m, 0)$ , the assignment step for a point at location  $x_1, x_2$  gives

$$r_1(\mathbf{x}) = \frac{\exp(-\beta(x_1 - m)^2/2)}{\exp(-\beta(x_1 - m)^2/2) + \exp(-\beta(x_1 + m)^2/2)} \quad (20.10)$$

$$= \frac{1}{1 + \exp(-2\beta m x_1)}, \quad (20.11)$$

and the updated  $m$  is

$$m' = \frac{\int dx_1 P(x_1) x_1 r_1(\mathbf{x})}{\int dx_1 P(x_1) r_1(\mathbf{x})} \quad (20.12)$$

$$= 2 \int dx_1 P(x_1) x_1 \frac{1}{1 + \exp(-2\beta m x_1)}. \quad (20.13)$$

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

$$\frac{1}{1 + \exp(-2\beta m x_1)} \simeq \frac{1}{2} (1 + \beta m x_1) + \dots \quad (20.14)$$

so

$$m' \simeq \int dx_1 P(x_1) x_1 (1 + \beta m x_1) \quad (20.15)$$

$$= \sigma_1^2 \beta m. \quad (20.16)$$

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

$$\sigma_1^2 \leq 1/\beta \quad (20.17)$$

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 20.10 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 20.11 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.

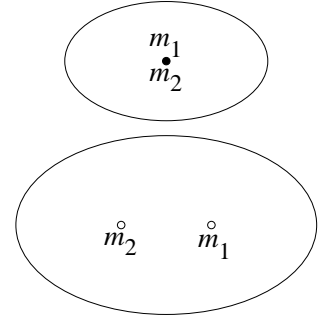


Figure 20.9. Schematic diagram of the bifurcation as the largest data variance  $\sigma_1$  increases from below  $1/\beta^{1/2}$  to above  $1/\beta^{1/2}$ . The data variance is indicated by the ellipse.

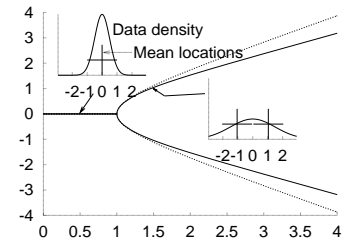


Figure 20.10. The stable mean locations as a function of  $\sigma_1$ , for constant  $\beta$ , found numerically (thick lines), and the approximation (20.22) (thin lines).

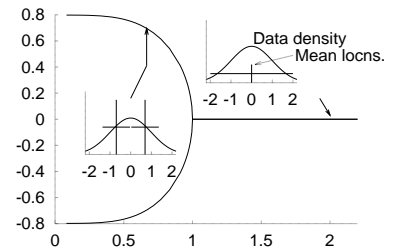


Figure 20.11. The stable mean locations as a function of  $1/\beta^{1/2}$ , for constant  $\sigma_1$ .

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

$$\frac{1}{1 + \exp(-2\beta m x_1)} \simeq \frac{1}{2}(1 + \beta m x_1 - \frac{1}{3}(\beta m x_1)^3) + \dots \quad (20.18)$$

then we can solve for the shape of the bifurcation to leading order, which depends on the fourth moment of the distribution:

$$m' \simeq \int dx_1 P(x_1) x_1 (1 + \beta m x_1 - \frac{1}{3}(\beta m x_1)^3) \quad (20.19)$$

$$= \sigma_1^2 \beta m - \frac{1}{3}(\beta m)^3 3\sigma_1^4. \quad (20.20)$$

[At (20.20) we use the fact that  $P(x_1)$  is Gaussian to find the fourth moment.] This map has a fixed point at  $m$  such that

$$\sigma_1^2 \beta (1 - (\beta m)^2 \sigma_1^2) = 1, \quad (20.21)$$

i.e.,

$$m = \pm \beta^{-1/2} \frac{(\sigma_1^2 \beta - 1)^{1/2}}{\sigma_1^2 \beta}. \quad (20.22)$$

The thin line in figure 20.10 shows this theoretical approximation. Figure 20.10 shows the bifurcation as a function of  $\sigma_1$  for fixed  $\beta$ ; figure 20.11 shows the bifurcation as a function of  $1/\beta^{1/2}$  for fixed  $\sigma_1$ .

- ▷ Exercise 20.5.<sup>[2, p.292]</sup> Why does the pitchfork in figure 20.11 tend to the values  $\sim \pm 0.8$  as  $1/\beta^{1/2} \rightarrow 0$ ? Give an analytic expression for this asymptote.

Solution to exercise 20.5 (p.292). The asymptote is the mean of the rectified Gaussian,

$$\frac{\int_0^\infty \text{Normal}(x, 1) x \, dx}{1/2} = \sqrt{2/\pi} \simeq 0.798. \quad (20.23)$$

## 21

### *Exact Inference by Complete Enumeration*

We open our toolbox of methods for handling probabilities by discussing a brute-force inference method: 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.

#### ► 21.1 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.

**Example 21.1.** Fred lives in Los Angeles and commutes 60 miles to work. Whilst at work, he receives a phone-call from his neighbour saying that Fred’s burglar alarm is ringing. What is the probability that there was a burglar in his house today? 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 was 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:

$$P(b, e, a, p, r) = P(b)P(e)P(a | b, e)P(p | a)P(r | e), \quad (21.1)$$

and plausible values for the probabilities are:

1. Burglar probability:

$$P(b=1) = \beta, \quad P(b=0) = 1 - \beta, \quad (21.2)$$

e.g.,  $\beta = 0.001$  gives a mean burglary rate of once every three years.

2. Earthquake probability:

$$P(e=1) = \epsilon, \quad P(e=0) = 1 - \epsilon, \quad (21.3)$$

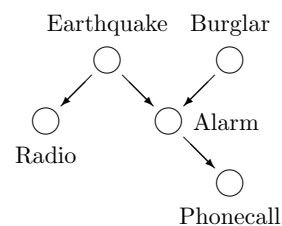


Figure 21.1. Belief network for the burglar alarm problem.

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. [This type of dependence of  $a$  on  $b$  and  $e$  is known as a 'noisy-or'.] The probabilities of  $a$  given  $b$  and  $e$  are then:

$$\begin{aligned} P(a=0 | b=0, e=0) &= (1-f), & P(a=1 | b=0, e=0) &= f \\ P(a=0 | b=1, e=0) &= (1-f)(1-\alpha_b), & P(a=1 | b=1, e=0) &= 1 - (1-f)(1-\alpha_b) \\ P(a=0 | b=0, e=1) &= (1-f)(1-\alpha_e), & P(a=1 | b=0, e=1) &= 1 - (1-f)(1-\alpha_e) \\ P(a=0 | b=1, e=1) &= (1-f)(1-\alpha_b)(1-\alpha_e), & P(a=1 | b=1, e=1) &= 1 - (1-f)(1-\alpha_b)(1-\alpha_e) \end{aligned}$$

or, in numbers,

$$\begin{aligned} P(a=0 | b=0, e=0) &= 0.999, & P(a=1 | b=0, e=0) &= 0.001 \\ P(a=0 | b=1, e=0) &= 0.00999, & P(a=1 | b=1, e=0) &= 0.99001 \\ P(a=0 | b=0, e=1) &= 0.98901, & P(a=1 | b=0, e=1) &= 0.01099 \\ P(a=0 | b=1, e=1) &= 0.0098901, & P(a=1 | b=1, e=1) &= 0.9901099. \end{aligned}$$

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:

$$P(b, e | a=1) = \frac{P(a=1 | b, e)P(b)P(e)}{P(a=1)}. \quad (21.4)$$

The numerator's four possible values are

$$\begin{aligned} P(a=1 | b=0, e=0) \times P(b=0) \times P(e=0) &= 0.001 \times 0.999 \times 0.999 = 0.000998 \\ P(a=1 | b=1, e=0) \times P(b=1) \times P(e=0) &= 0.99001 \times 0.001 \times 0.999 = 0.000989 \\ P(a=1 | b=0, e=1) \times P(b=0) \times P(e=1) &= 0.01099 \times 0.999 \times 0.001 = 0.000010979 \\ P(a=1 | b=1, e=1) \times P(b=1) \times P(e=1) &= 0.9901099 \times 0.001 \times 0.001 = 9.9 \times 10^{-7}. \end{aligned}$$

The normalizing constant is the sum of these four numbers,  $P(a=1) = 0.002$ , and the posterior probabilities are

$$\begin{aligned} P(b=0, e=0 | a=1) &= 0.4993 \\ P(b=1, e=0 | a=1) &= 0.4947 \\ P(b=0, e=1 | a=1) &= 0.0055 \\ P(b=1, e=1 | a=1) &= 0.0005. \end{aligned} \quad (21.5)$$

To answer the question, ‘what’s the probability a burglar was there?’ we *marginalize* over the earthquake variable  $e$ :

$$\begin{aligned} P(b=0|a=1) &= P(b=0, e=0|a=1) + P(b=0, e=1|a=1) = 0.505 \\ P(b=1|a=1) &= P(b=1, e=0|a=1) + P(b=1, e=1|a=1) = 0.495. \end{aligned} \quad (21.6)$$

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 (21.5) 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 (21.5), by their sum  $P(e=1|a=1) = 0.0060$ . The posterior probability of  $b$  is:

$$\begin{aligned} P(b=0|e=1, a=1) &= 0.92 \\ P(b=1|e=1, a=1) &= 0.08. \end{aligned} \quad (21.7)$$

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) reduces when Fred learns that an earthquake, an alternative explanation of the alarm, has happened.

### Explaining away

This phenomenon, that one of the possible causes ( $b=1$ ) of some data (the data in this case 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; 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  $b$ .

- ▷ Exercise 21.2.<sup>[2]</sup> After Fred receives the phone-call about the burglar alarm, but before he hears the radio report, what, from his point of view, is the probability that there was a small earthquake today?

## ► 21.2 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.48) 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. 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

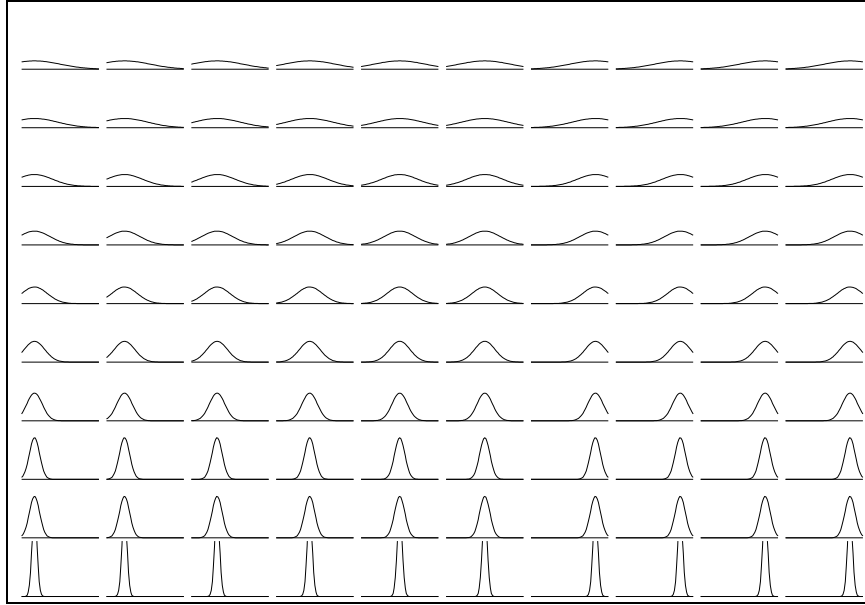


Figure 21.2. Enumeration of an entire (discretized) hypothesis space for one Gaussian with parameters  $\mu$  (horizontal axis) and  $\sigma$  (vertical).

function for the decay length as a function of  $\lambda$  by evaluating the likelihood at a finely-spaced series of points.

### A two-parameter model

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  $\sigma$ :

$$P(x | \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \equiv \text{Normal}(x; \mu, \sigma^2). \quad (21.8)$$

Figure 21.2 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 a picture showing the probability density that it puts on  $x$ . We now examine the inference of  $\mu$  and  $\sigma$  given data points  $x_n$ ,  $n = 1, \dots, N$ , assumed to be drawn independently from this density.

Imagine that we acquire data, for example the five points shown in figure 21.3. 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  $P(\{x_n\}_{n=1}^5 | \mu, \sigma)$ . The likelihood values are shown diagrammatically in figure 21.4 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 as a surface plot or contour plot as a function of  $\mu$  and  $\sigma$  (figure 21.5).

### A five-parameter mixture model

Eyeballing the data (figure 21.3), 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

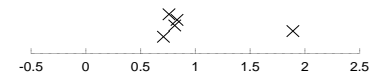


Figure 21.3. Five datapoints  $\{x_n\}_{n=1}^5$ . The horizontal coordinate is the value of the datum,  $x_n$ ; the vertical coordinate has no meaning.

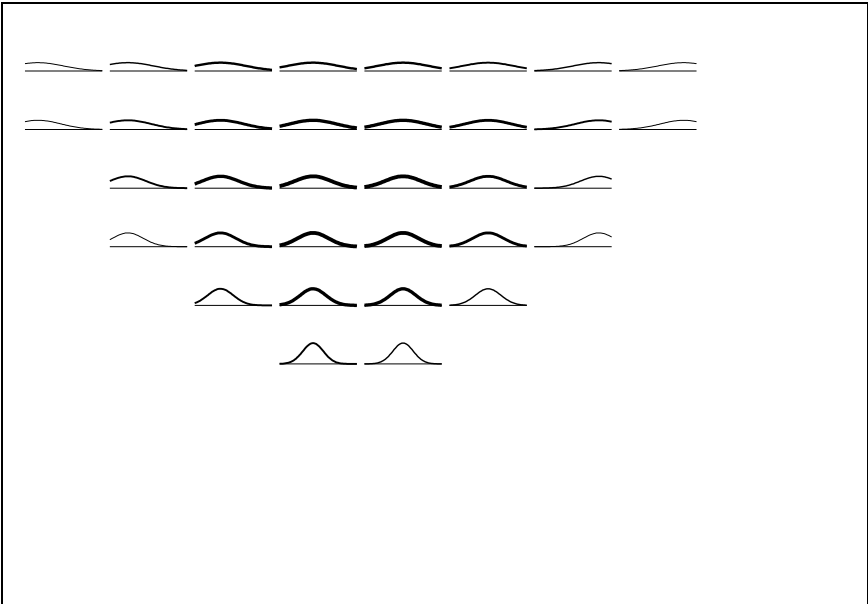


Figure 21.4. Likelihood function, given the data of figure 21.3, represented by line thickness. Subhypotheses having likelihood smaller than  $e^{-8}$  times the maximum likelihood are not shown.

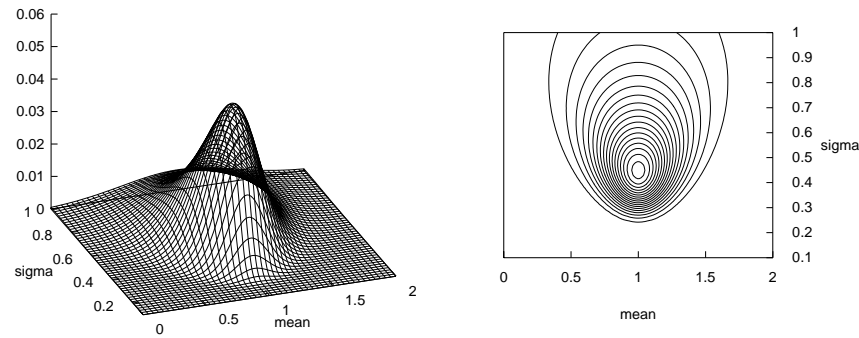


Figure 21.5. The likelihood function for the parameters of a Gaussian distribution. Surface plot and contour plot of the log likelihood as a function of  $\mu$  and  $\sigma$ . The data set of  $N = 5$  points had mean  $\bar{x} = 1.0$  and  $S = \sum (x - \bar{x})^2 = 1.0$ .



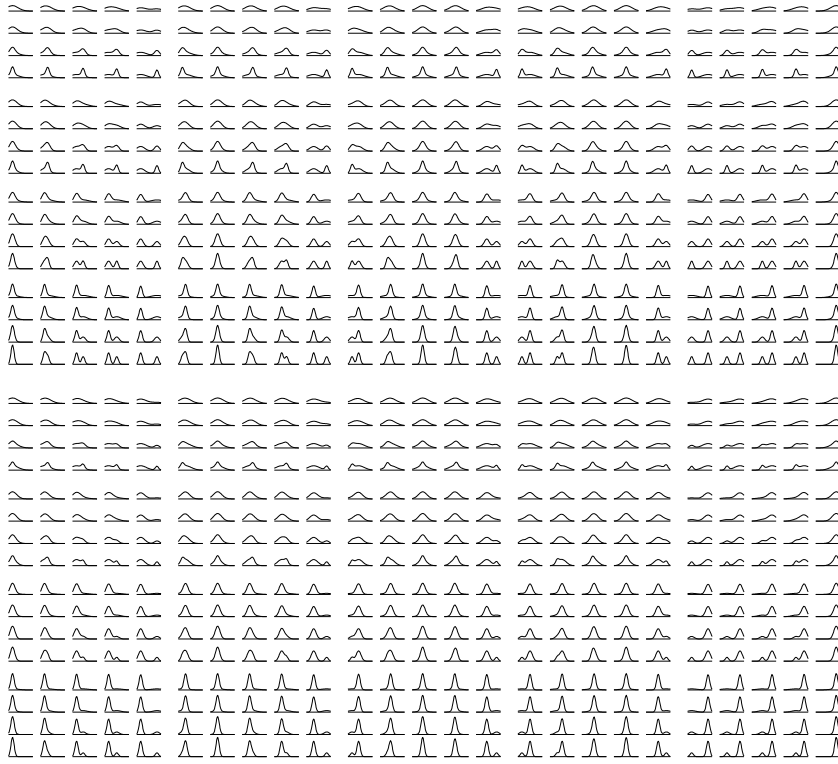


Figure 21.6. Enumeration of the entire (discretized) hypothesis space for a mixture of two Gaussians. Weight of the mixture components is  $\pi_1, \pi_2 = 0.6, 0.4$  in the top half and  $0.8, 0.2$  in the bottom half. Means  $\mu_1$  and  $\mu_2$  vary horizontally, and standard deviations  $\sigma_1$  and  $\sigma_2$  vary vertically.

coefficients  $\pi_1$  and  $\pi_2$ , satisfying  $\pi_1 + \pi_2 = 1$ ,  $\pi_i \geq 0$ .

$$P(x | \mu_1, \sigma_1, \pi_1, \mu_2, \sigma_2, \pi_2) = \frac{\pi_1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{\pi_2}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right)$$

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 21.6 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 21.7.

If we wish to compare the one-Gaussian model with the mixture-of-two model, we can find the models' 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  $\theta$ ,

$$P(\{x\} | \mathcal{H}) = \sum_{\theta} P(\theta) P(\{x\} | \theta, \mathcal{H}), \quad (21.9)$$

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 requires a grid of at least  $10^K$  points. This

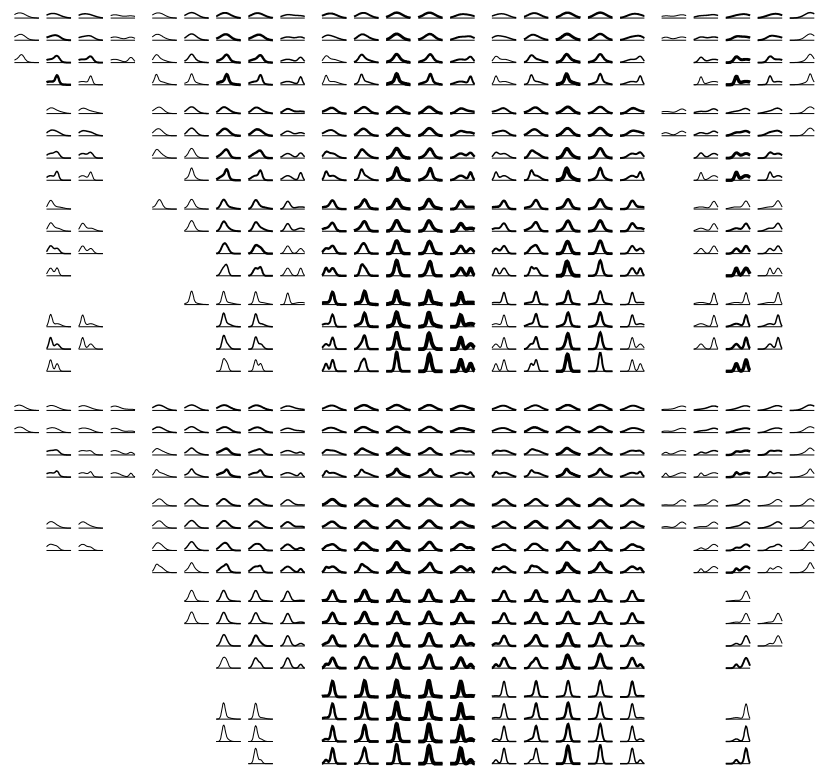


Figure 21.7. Inferring a mixture of two Gaussians. Likelihood function, given the data of figure 21.3, represented by line thickness. The hypothesis space is identical to that shown in figure 21.6. Subhypotheses having likelihood smaller than  $e^{-8}$  times the maximum likelihood are not shown, hence the blank regions, which correspond to hypotheses that the data have ruled out.



exponential growth of computation with model size is the reason why complete enumeration is rarely a feasible computational strategy.



**Exercise 21.3.**<sup>[1]</sup> Imagine fitting a mixture of ten Gaussians to data in a twenty-dimensional space. Estimate the computational cost of implementing inferences for this model by enumeration of a grid of parameter values.

## 22

---

### Maximum Likelihood and Clustering

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 that fits the data well. This is the philosophy behind the maximum likelihood method, which identifies the setting of the parameter vector  $\theta$  that 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. Likelihoods multiply; log likelihoods add.

#### ► 22.1 Maximum likelihood for one Gaussian

We return to the Gaussian for our first examples. Assume we have data  $\{x_n\}_{n=1}^N$ . The log likelihood is:

$$\ln P(\{x_n\}_{n=1}^N | \mu, \sigma) = -N \ln(\sqrt{2\pi}\sigma) - \sum_n (x_n - \mu)^2 / (2\sigma^2). \quad (22.1)$$

The likelihood can be expressed in terms of two functions of the data, the sample mean

$$\bar{x} \equiv \sum_{n=1}^N x_n / N, \quad (22.2)$$

and the sum of square deviations

$$S \equiv \sum_n (x_n - \bar{x})^2 : \quad (22.3)$$

$$\ln P(\{x_n\}_{n=1}^N | \mu, \sigma) = -N \ln(\sqrt{2\pi}\sigma) - [N(\mu - \bar{x})^2 + S] / (2\sigma^2). \quad (22.4)$$

Because the likelihood depends on the data only through  $\bar{x}$  and  $S$ , these two quantities are known as *sufficient statistics*.

**Example 22.1.** Differentiate the log likelihood with respect to  $\mu$  and show that, if the standard deviation is known to be  $\sigma$ , the maximum likelihood mean  $\mu$  of a Gaussian is equal to the sample mean  $\bar{x}$ , for any value of  $\sigma$ .

**Solution.**

$$\frac{\partial}{\partial \mu} \ln P = -\frac{N(\mu - \bar{x})}{\sigma^2} \quad (22.5)$$

$$= 0 \quad \text{when } \mu = \bar{x}. \quad \square \quad (22.6)$$

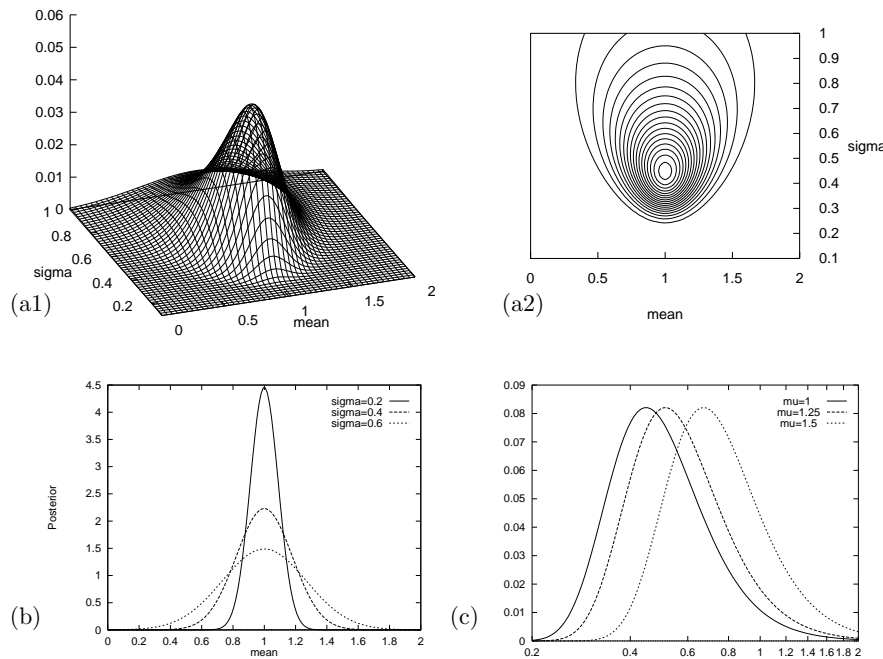


Figure 22.1. The likelihood function for the parameters of a Gaussian distribution. (a1, a2) Surface plot and contour plot of the log likelihood as a function of  $\mu$  and  $\sigma$ . The data set of  $N = 5$  points had mean  $\bar{x} = 1.0$  and  $S = \sum (x - \bar{x})^2 = 1.0$ . (b) The posterior probability of  $\mu$  for various values of  $\sigma$ . (c) The posterior probability of  $\sigma$  for various fixed values of  $\mu$  (shown as a density over  $\ln \sigma$ ).

If we Taylor-expand the log likelihood about the maximum, we can define approximate error bars on the maximum likelihood parameter: we 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  $e^{4/2}$ . In the special case of a likelihood that is a Gaussian function of the parameters, the quadratic approximation is exact.

**Example 22.2.** Find the second derivative of the log likelihood with respect to  $\mu$ , and find the error bars on  $\mu$ , given the data and  $\sigma$ .

**Solution.**

$$\frac{\partial^2}{\partial \mu^2} \ln P = -\frac{N}{\sigma^2}. \quad \square \quad (22.7)$$

Comparing this curvature with the curvature of the log of a Gaussian distribution over  $\mu$  of standard deviation  $\sigma_\mu$ ,  $\exp(-\mu^2/(2\sigma_\mu^2))$ , which is  $-1/\sigma_\mu^2$ , we can deduce that the error bars on  $\mu$  (derived from the likelihood function) are

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}}. \quad (22.8)$$

The error bars have this property: at the two points  $\mu = \bar{x} \pm \sigma_\mu$ , the likelihood is smaller than its maximum value by a factor of  $e^{1/2}$ .

**Example 22.3.** 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  $\ln \sigma$ , and error bars on  $\ln \sigma$ .

**Solution.** The likelihood's dependence on  $\sigma$  is

$$\ln P(\{x_n\}_{n=1}^N | \mu, \sigma) = -N \ln(\sqrt{2\pi}\sigma) - \frac{S_{\text{tot}}}{(2\sigma^2)}, \quad (22.9)$$

where  $S_{\text{tot}} = \sum_n (x_n - \mu)^2$ . To find the maximum of the likelihood, we can differentiate with respect to  $\ln \sigma$ . [It's often most hygienic to differentiate with

respect to  $\ln u$  rather than  $u$ , when  $u$  is a scale variable; we use  $du^n/d(\ln u) = nu^n$ .]

$$\frac{\partial \ln P(\{x_n\}_{n=1}^N | \mu, \sigma)}{\partial \ln \sigma} = -N + \frac{S_{\text{tot}}}{\sigma^2} \quad (22.10)$$

This derivative is zero when

$$\sigma^2 = \frac{S_{\text{tot}}}{N}, \quad (22.11)$$

i.e.,

$$\sigma = \sqrt{\frac{\sum_{n=1}^N (x_n - \mu)^2}{N}}. \quad (22.12)$$

The second derivative is

$$\frac{\partial^2 \ln P(\{x_n\}_{n=1}^N | \mu, \sigma)}{\partial (\ln \sigma)^2} = -2 \frac{S_{\text{tot}}}{\sigma^2}, \quad (22.13)$$

and at the maximum-likelihood value of  $\sigma^2$ , this equals  $-2N$ . So error bars on  $\ln \sigma$  are

$$\sigma_{\ln \sigma} = \frac{1}{\sqrt{2N}}. \quad \square \quad (22.14)$$

▷ Exercise 22.4.<sup>[1]</sup> Show that the values of  $\mu$  and  $\ln \sigma$  that jointly maximize the likelihood are:  $\{\mu, \sigma\}_{\text{ML}} = \{\bar{x}, \sigma_N = \sqrt{S/N}\}$ , where

$$\sigma_N \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N}}. \quad (22.15)$$

## ► 22.2 Maximum likelihood for a mixture of Gaussians

We now derive an algorithm for fitting a mixture of Gaussians to one-dimensional data. In fact, this algorithm is so important to understand that, *you*, gentle reader, get to derive the algorithm. Please work through the following exercise.



Exercise 22.5.<sup>[2, p.310]</sup> A random variable  $x$  is assumed to have a probability distribution that is a *mixture of two Gaussians*,

$$P(x | \mu_1, \mu_2, \sigma) = \left[ \sum_{k=1}^2 p_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu_k)^2}{2\sigma^2}\right) \right], \quad (22.16)$$

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

$$\begin{aligned} P(k_n = 1 | x_n, \theta) &= \frac{1}{1 + \exp[-(w_1 x_n + w_0)]} \\ P(k_n = 2 | x_n, \theta) &= \frac{1}{1 + \exp[+(w_1 x_n + w_0)]}, \end{aligned} \quad (22.17)$$

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  $\{\mu_k\}$  that maximize the likelihood,

$$P(\{x_n\}_{n=1}^N | \{\mu_k\}, \sigma) = \prod_n P(x_n | \{\mu_k\}, \sigma). \quad (22.18)$$

Let  $L$  denote the natural log of the likelihood. Show that the derivative of the log likelihood with respect to  $\mu_k$  is given by

$$\frac{\partial}{\partial \mu_k} L = \sum_n p_{k|n} \frac{(x_n - \mu_k)}{\sigma^2}, \quad (22.19)$$

where  $p_{k|n} \equiv P(k_n = k | x_n, \theta)$  appeared above at equation (22.17).

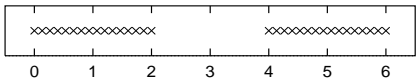
Show, neglecting terms in  $\frac{\partial}{\partial \mu_k} P(k_n = k | x_n, \theta)$ , that the second derivative is approximately given by

$$\frac{\partial^2}{\partial \mu_k^2} L = - \sum_n p_{k|n} \frac{1}{\sigma^2}. \quad (22.20)$$

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

$$\mu'_k = \frac{\sum_n p_{k|n} x_n}{\sum_n p_{k|n}}. \quad (22.21)$$

[The Newton–Raphson method for maximizing  $L(\mu)$  updates  $\mu$  to  $\mu' = \mu - \left[ \frac{\partial L}{\partial \mu} / \frac{\partial^2 L}{\partial \mu^2} \right]$ .



Assuming that  $\sigma = 1$ , sketch a contour plot of the likelihood function as a function of  $\mu_1$  and  $\mu_2$  for the data set shown above. The data set consists of 32 points. Describe the peaks in your sketch and indicate their widths.

Notice that the algorithm you have derived for maximizing the likelihood is identical to the soft K-means algorithm of section 20.4. Now that it is clear that clustering can be viewed as mixture-density-modelling, we are able to derive enhancements to the K-means algorithm, which rectify the problems we noted earlier.

► **22.3 Enhancements to soft K-means**

Algorithm 22.2 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 22.3 for two data sets that we've seen before. The second example shows

**Assignment step.** The responsibilities are

$$r_k^{(n)} = \frac{\pi_k \frac{1}{(\sqrt{2\pi}\sigma_k)^I} \exp\left(-\frac{1}{\sigma_k^2} d(\mathbf{m}^{(k)}, \mathbf{x}^{(n)})\right)}{\sum_{k'} \pi_k \frac{1}{(\sqrt{2\pi}\sigma_{k'})^I} \exp\left(-\frac{1}{\sigma_{k'}^2} d(\mathbf{m}^{(k')}, \mathbf{x}^{(n)})\right)} \quad (22.22)$$

where  $I$  is the dimensionality of  $\mathbf{x}$ .

**Update step.** Each cluster's parameters,  $\mathbf{m}^{(k)}$ , and  $\sigma_k^2$ , are adjusted to match the data points that it is responsible for.

$$\mathbf{m}^{(k)} = \frac{\sum_n r_k^{(n)} \mathbf{x}^{(n)}}{R^{(k)}} \quad (22.23)$$

$$\sigma_k^2 = \frac{\sum_n r_k^{(n)} (\mathbf{x}^{(n)} - \mathbf{m}^{(k)})^2}{IR^{(k)}} \quad (22.24)$$

$$\pi_k = \frac{R^{(k)}}{\sum_k R^{(k)}} \quad (22.25)$$

where  $R^{(k)}$  is the total responsibility of mean  $k$ ,

$$R^{(k)} = \sum_n r_k^{(n)}. \quad (22.26)$$

Algorithm 22.2. The soft K-means algorithm, version 2.

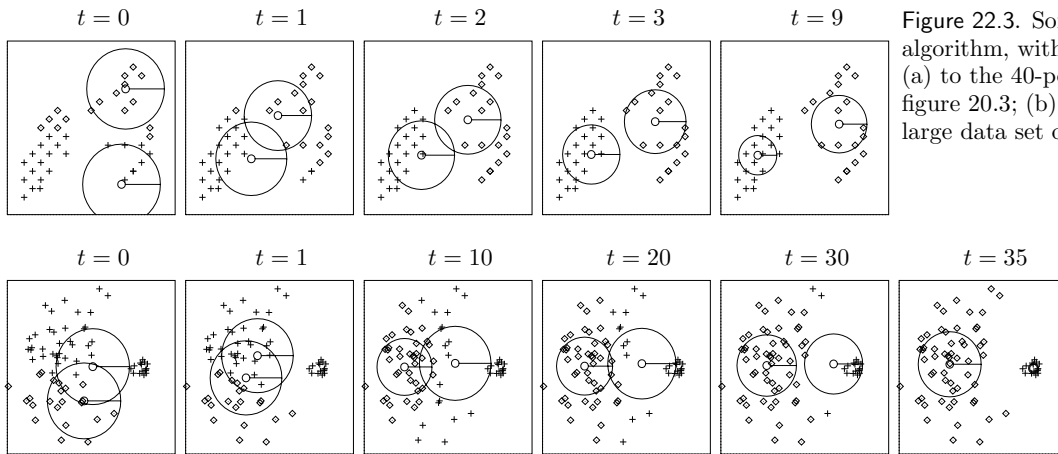


Figure 22.3. Soft K-means algorithm, with  $K = 2$ , applied (a) to the 40-point data set of figure 20.3; (b) to the little 'n' large data set of figure 20.5.

$$r_k^{(n)} = \frac{\pi_k \frac{1}{\prod_{i=1}^I \sqrt{2\pi}\sigma_i^{(k)}} \exp\left(-\sum_{i=1}^I (m_i^{(k)} - x_i^{(n)})^2 / 2(\sigma_i^{(k)})^2\right)}{\sum_{k'} \text{(numerator, with } k' \text{ in place of } k)} \quad (22.27)$$

$$\sigma_i^{2(k)} = \frac{\sum_n r_k^{(n)} (x_i^{(n)} - m_i^{(k)})^2}{R^{(k)}} \quad (22.28)$$

Algorithm 22.4. The soft K-means algorithm, version 3, which corresponds to a model of axis-aligned Gaussians.

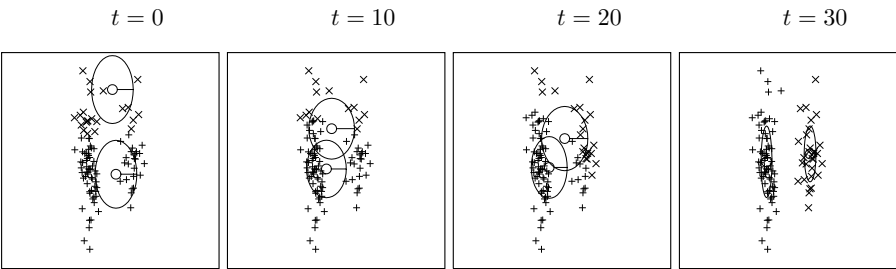


Figure 22.5. Soft K-means algorithm, version 3, applied to the data consisting of two cigar-shaped clusters.  $K = 2$  (cf. figure 20.6).

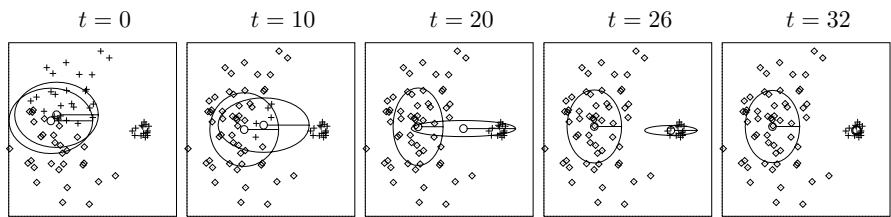


Figure 22.6. Soft K-means algorithm, version 3, applied to the little 'n' large data set.  $K = 2$ .

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 20.6. If we wish to model the clusters by axis-aligned Gaussians with possibly-unequal variances, we replace the assignment rule (22.22) and the variance update rule (22.24) by the rules (22.27) and (22.28) displayed in algorithm 22.4.

This third version of soft K-means is demonstrated in figure 22.5 on the ‘two cigars’ data set of figure 20.6. After 30 iterations, the algorithm correctly locates the two clusters. Figure 22.6 shows the same algorithm applied to the little ‘n’ large data set; again, the correct cluster locations are found.

A proof that the algorithm does indeed maximize the likelihood is deferred to section 33.7.

► 22.4 A fatal flaw of maximum likelihood

Finally, figure 22.7 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 22.6.<sup>[2]</sup> 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.

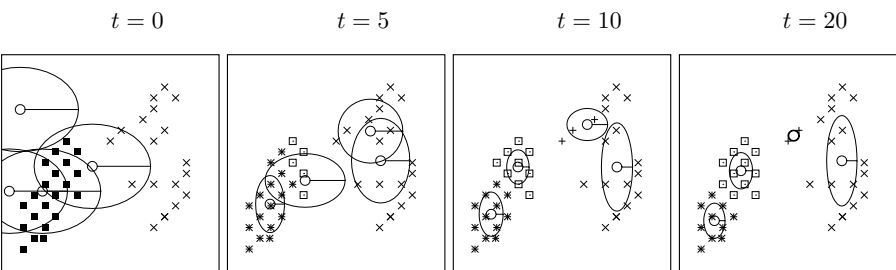


Figure 22.7. Soft K-means algorithm applied to a data set of 40 points.  $K = 4$ . Notice that at convergence, one very small cluster has formed between two data points.



### KABOOM!

Soft K-means can blow up. Put one cluster 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 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.

We conclude that maximum likelihood methods are not a satisfactory general solution to data-modelling problems: the likelihood may be infinitely large at certain parameter settings. Even if the likelihood does not have infinitely-large spikes, the maximum of the likelihood is often unrepresentative, in high-dimensional problems.

Even in low-dimensional problems, maximum likelihood solutions can be unrepresentative. As you may know from basic statistics, the maximum likelihood estimator (22.15) for a Gaussian's standard deviation,  $\sigma_N$ , is a *biased* estimator, a topic that we'll take up in Chapter 24.

### The maximum a posteriori (MAP) method

A popular replacement for maximizing the likelihood is maximizing the Bayesian posterior probability density of the parameters instead. However, multiplying the likelihood by a prior and maximizing the posterior does not make the above problems go away; the posterior density often also has infinitely-large spikes, and the maximum of the posterior probability density is often unrepresentative of the whole posterior distribution. Think back to the concept of typicality, which we encountered in Chapter 4: 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. Maxima are atypical.

A further reason for disliking the maximum *a posteriori* is that it is *basis-dependent*. If we make a nonlinear change of basis from the parameter  $\theta$  to the parameter  $u = f(\theta)$  then the probability density of  $\theta$  is transformed to

$$P(u) = P(\theta) \left| \frac{\partial \theta}{\partial u} \right|. \quad (22.29)$$

The maximum of the density  $P(u)$  will usually not coincide with the maximum of the density  $P(\theta)$ . (For figures illustrating such nonlinear changes of basis, see the next chapter.) It seems undesirable to use a method whose answers change when we change representation.

### 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).

## ► 22.5 Further exercises

### Exercises where maximum likelihood may be useful

Exercise 22.7.<sup>[3]</sup> Make a version of the K-means algorithm that models the data as a mixture of  $K$  arbitrary Gaussians, i.e., Gaussians that are not constrained to be axis-aligned.

- ▷ Exercise 22.8.<sup>[2]</sup> (a) A photon counter is pointed at a remote star for one minute, in order to infer the brightness, i.e., the rate of photons arriving at the counter per minute,  $\lambda$ . Assuming the number of photons collected  $r$  has a Poisson distribution with mean  $\lambda$ ,

$$P(r | \lambda) = \exp(-\lambda) \frac{\lambda^r}{r!}, \quad (22.30)$$

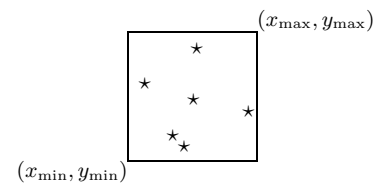
what is the maximum likelihood estimate for  $\lambda$ , given  $r = 9$ ? Find error bars on  $\ln \lambda$ .

- (b) Same situation, but now we assume that the counter detects not only photons from the star but also ‘background’ photons. The background rate of photons is known to be  $b = 13$  photons per minute. We assume the number of photons collected,  $r$ , has a Poisson distribution with mean  $\lambda + b$ . Now, given  $r = 9$  detected photons, what is the maximum likelihood estimate for  $\lambda$ ? Comment on this answer, discussing also the Bayesian posterior distribution, and the ‘unbiased estimator’ of sampling theory,  $\hat{\lambda} \equiv r - b$ .

Exercise 22.9.<sup>[2]</sup> A bent coin is tossed  $N$  times, giving  $N_a$  heads and  $N_b$  tails. Assume a beta distribution prior for the probability of heads,  $p$ , for example the uniform distribution. Find the maximum likelihood and maximum *a posteriori* values of  $p$ , then find the maximum likelihood and maximum *a posteriori* values of the logit  $a \equiv \ln[p/(1-p)]$ . Compare with the predictive distribution, i.e., the probability that the next toss will come up heads.

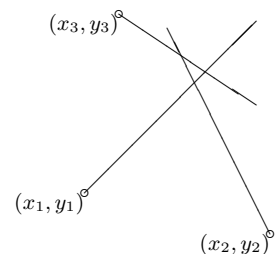
- ▷ Exercise 22.10.<sup>[2]</sup> Two men looked through prison bars; one saw stars, the other tried to infer where the window frame was.

From the other side of a room, you look through a window and see stars at locations  $\{(x_n, y_n)\}$ . You can’t see the window edges because it is totally dark apart from the stars. Assuming the window is rectangular and that the visible stars’ locations are independently randomly distributed, what are the inferred values of  $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ , according to maximum likelihood? Sketch the likelihood as a function of  $x_{\max}$ , for fixed  $x_{\min}$ ,  $y_{\min}$ , and  $y_{\max}$ .



- ▷ Exercise 22.11.<sup>[3]</sup> A sailor infers his location  $(x, y)$  by measuring the bearings of three buoys whose locations  $(x_n, y_n)$  are given on his chart. Let the true bearings of the buoys be  $\theta_n$ . Assuming that his measurement  $\tilde{\theta}_n$  of each bearing is subject to Gaussian noise of small standard deviation  $\sigma$ , what is his inferred location, by maximum likelihood?

The sailor’s rule of thumb says that the boat’s position can be taken to be the centre of the cocked hat, the triangle produced by the intersection of the three measured bearings (figure 22.8). Can you persuade him that the maximum likelihood answer is better?



- ▷ Exercise 22.12.<sup>[3, p.310]</sup> Maximum likelihood fitting of an exponential-family model.

Assume that a variable  $\mathbf{x}$  comes from a probability distribution of the form

$$P(\mathbf{x} | \mathbf{w}) = \frac{1}{Z(\mathbf{w})} \exp \left( \sum_k w_k f_k(\mathbf{x}) \right), \quad (22.31)$$

Figure 22.8. The standard way of drawing three slightly inconsistent bearings on a chart produces a triangle called a cocked hat. Where is the sailor?

where the functions  $f_k(\mathbf{x})$  are given, and the parameters  $\mathbf{w} = \{w_k\}$  are not known. A data set  $\{\mathbf{x}^{(n)}\}$  of  $N$  points is supplied.

Show by differentiating the log likelihood that the maximum-likelihood parameters  $\mathbf{w}_{\text{ML}}$  satisfy

$$\sum_{\mathbf{x}} P(\mathbf{x} | \mathbf{w}_{\text{ML}}) f_k(\mathbf{x}) = \frac{1}{N} \sum_n f_k(\mathbf{x}^{(n)}), \quad (22.32)$$

where the left-hand sum is over *all*  $\mathbf{x}$ , and the right-hand sum is over the data points. A shorthand for this result is that each function-average under the fitted model must equal the function-average found in the data:

$$\langle f_k \rangle_{P(\mathbf{x} | \mathbf{w}_{\text{ML}})} = \langle f_k \rangle_{\text{Data}}. \quad (22.33)$$

▷ Exercise 22.13.<sup>[3]</sup> ‘Maximum entropy’ fitting of models to constraints.

When confronted by a probability distribution  $P(\mathbf{x})$  about which only a few facts are known, the *maximum entropy principle* (maxent) offers a rule for *choosing* a distribution that satisfies those constraints. According to maxent, you should select the  $P(\mathbf{x})$  that maximizes the entropy

$$H = - \sum_{\mathbf{x}} P(\mathbf{x}) \log 1/P(\mathbf{x}), \quad (22.34)$$

subject to the constraints. Assuming the constraints assert that the *averages* of certain functions  $f_k(\mathbf{x})$  are known, i.e.,

$$\langle f_k \rangle_{P(\mathbf{x})} = F_k, \quad (22.35)$$

show, by introducing Lagrange multipliers (one for each constraint, including normalization), that the maximum-entropy distribution has the form

$$P(\mathbf{x})_{\text{Maxent}} = \frac{1}{Z} \exp \left( \sum_k w_k f_k(\mathbf{x}) \right), \quad (22.36)$$

where the parameters  $Z$  and  $\{w_k\}$  are set such that the constraints (22.35) are satisfied.

And hence the maximum entropy method gives identical results to maximum likelihood fitting of an exponential-family model (previous exercise).

The maximum entropy method has sometimes been recommended as a method for assigning prior distributions in Bayesian modelling. While the outcomes of the maximum entropy method are sometimes interesting and thought-provoking, I do not advocate maxent as *the* approach to assigning priors.

Maximum entropy is also sometimes proposed as a method for solving inference problems – for example, ‘given that the mean score of this unfair six-sided die is 2.5, what is its probability distribution  $(p_1, p_2, p_3, p_4, p_5, p_6)$ ?’ I think it is a bad idea to use maximum entropy in this way; it can give silly answers. The correct way to solve inference problems is to use Bayes’ theorem.

Exercises where maximum likelihood and MAP have difficulties

▷ Exercise 22.14.<sup>[2]</sup> This exercise explores the idea that maximizing a probability density is a poor way to find a point that is representative of the density. 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 proportional to  $r/\sqrt{k}$ . 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 that differ in radius  $\sigma_w$  by just 1%, and that 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.

▷ Exercise 22.15.<sup>[3]</sup> The seven scientists.  $N$  datapoints  $\{x_n\}$  are drawn from  $N$  distributions, all of which are Gaussian with a common mean  $\mu$  but with different unknown standard deviations  $\sigma_n$ . What are the maximum likelihood parameters  $\mu, \{\sigma_n\}$  given the data? For example, seven scientists (A, B, C, D, E, F, G) with wildly-differing experimental skills measure  $\mu$ . You expect some of them to do accurate work (i.e., to have small  $\sigma_n$ ), and some of them to turn in wildly inaccurate answers (i.e., to have enormous  $\sigma_n$ ). Figure 22.9 shows their seven results. What is  $\mu$ , and how reliable is each scientist?

I hope you agree that, intuitively, it looks pretty certain that A and B are both inept measurers, that D–G are better, and that the true value of  $\mu$  is somewhere close to 10. But what does maximizing the likelihood tell you?

Exercise 22.16.<sup>[3]</sup> Problems with MAP method. A collection of widgets  $i = 1, \dots, k$  have a property called ‘wodge’,  $w_i$ , which we measure, widget by widget, in noisy experiments with a known noise level  $\sigma_v = 1.0$ . Our model for these quantities is that they come from a Gaussian prior  $P(w_i | \alpha) = \text{Normal}(0, 1/\alpha)$ , where  $\alpha = 1/\sigma_w^2$  is not known. Our prior for this variance is flat over  $\log \sigma_w$  from  $\sigma_w = 0.1$  to  $\sigma_w = 10$ .

Scenario 1. Suppose four widgets have been measured and give the following data:  $\{d_1, d_2, d_3, d_4\} = \{2.2, -2.2, 2.8, -2.8\}$ . We are interested in inferring the wodges of these four widgets.

- (a) Find the values of  $\mathbf{w}$  and  $\alpha$  that maximize the posterior probability  $P(\mathbf{w}, \log \alpha | \mathbf{d})$ .
- (b) Marginalize over  $\alpha$  and find the posterior probability density of  $\mathbf{w}$  given the data. [Integration skills required. See MacKay (1999a) for solution.] Find maxima of  $P(\mathbf{w} | \mathbf{d})$ . [Answer: two maxima – one at  $\mathbf{w}_{\text{MP}} = \{1.8, -1.8, 2.2, -2.2\}$ , with error bars on all four parameters

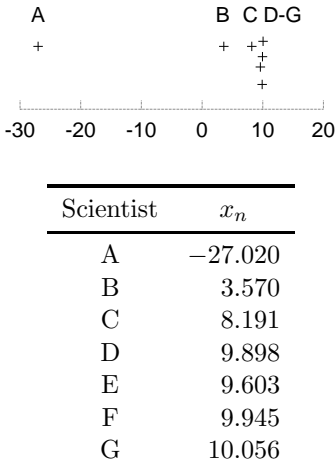


Figure 22.9. Seven measurements  $\{x_n\}$  of a parameter  $\mu$  by seven scientists each having his own noise-level  $\sigma_n$ .

(obtained from Gaussian approximation to the posterior)  $\pm 0.9$ ; and one at  $\mathbf{w}'_{\text{MP}} = \{0.03, -0.03, 0.04, -0.04\}$  with error bars  $\pm 0.1$ .]

**Scenario 2.** Suppose in addition to the four measurements above we are now informed that there are four more widgets that have been measured with a much less accurate instrument, having  $\sigma'_\nu = 100.0$ . Thus we now have both well-determined and ill-determined parameters, as in a typical ill-posed problem. The data from these measurements were a string of uninformative values,  $\{d_5, d_6, d_7, d_8\} = \{100, -100, 100, -100\}$ .

We are again asked to infer the woggles of the widgets. Intuitively, our inferences about the well-measured widgets should be negligibly affected by this vacuous information about the poorly-measured widgets. But what happens to the MAP method?

- Find the values of  $\mathbf{w}$  and  $\alpha$  that maximize the posterior probability  $P(\mathbf{w}, \log \alpha | \mathbf{d})$ .
- Find maxima of  $P(\mathbf{w} | \mathbf{d})$ . [Answer: only one maximum,  $\mathbf{w}_{\text{MP}} = \{0.03, -0.03, 0.03, -0.03, 0.0001, -0.0001, 0.0001, -0.0001\}$ , with error bars on all eight parameters  $\pm 0.11$ .]

## ► 22.6 Solutions

**Solution to exercise 22.5 (p.302).** Figure 22.10 shows a contour plot of the likelihood function for the 32 data points. 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.

**Solution to exercise 22.12 (p.307).** The log likelihood is:

$$\ln P(\{\mathbf{x}^{(n)}\} | \mathbf{w}) = -N \ln Z(\mathbf{w}) + \sum_n \sum_k w_k f_k(\mathbf{x}^{(n)}). \quad (22.37)$$

$$\frac{\partial}{\partial w_k} \ln P(\{\mathbf{x}^{(n)}\} | \mathbf{w}) = -N \frac{\partial}{\partial w_k} \ln Z(\mathbf{w}) + \sum_n f_k(\mathbf{x}^{(n)}). \quad (22.38)$$

Now, the fun part is what happens when we differentiate the log of the normalizing constant:

$$\begin{aligned} \frac{\partial}{\partial w_k} \ln Z(\mathbf{w}) &= \frac{1}{Z(\mathbf{w})} \sum_{\mathbf{x}} \frac{\partial}{\partial w_k} \exp \left( \sum_{k'} w_{k'} f_{k'}(\mathbf{x}) \right) \\ &= \frac{1}{Z(\mathbf{w})} \sum_{\mathbf{x}} \exp \left( \sum_{k'} w_{k'} f_{k'}(\mathbf{x}) \right) f_k(\mathbf{x}) = \sum_{\mathbf{x}} P(\mathbf{x} | \mathbf{w}) f_k(\mathbf{x}), \end{aligned} \quad (22.39)$$

so

$$\frac{\partial}{\partial w_k} \ln P(\{\mathbf{x}^{(n)}\} | \mathbf{w}) = -N \sum_{\mathbf{x}} P(\mathbf{x} | \mathbf{w}) f_k(\mathbf{x}) + \sum_n f_k(\mathbf{x}^{(n)}), \quad (22.40)$$

and at the maximum of the likelihood,

$$\sum_{\mathbf{x}} P(\mathbf{x} | \mathbf{w}_{\text{ML}}) f_k(\mathbf{x}) = \frac{1}{N} \sum_n f_k(\mathbf{x}^{(n)}). \quad (22.41)$$

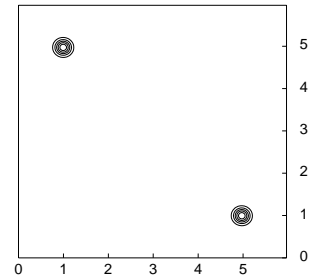


Figure 22.10. The likelihood as a function of  $\mu_1$  and  $\mu_2$ .

## 23

### Useful Probability Distributions

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.

#### ► 23.1 Distributions over integers

Binomial, Poisson, exponential

We already encountered the binomial distribution and the Poisson distribution on page 2.

The *binomial distribution* for an integer  $r$  with parameters  $f$  (the bias,  $f \in [0, 1]$ ) and  $N$  (the number of trials) is:

$$P(r | f, N) = \binom{N}{r} f^r (1 - f)^{N-r} \quad r \in \{0, 1, 2, \dots, N\}. \quad (23.1)$$

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:

$$P(r | \lambda) = e^{-\lambda} \frac{\lambda^r}{r!} \quad r \in \{0, 1, 2, \dots\}. \quad (23.2)$$

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  $\lambda$ .

The *exponential distribution on integers*,

$$P(r | f) = f^r (1 - f) \quad r \in (0, 1, 2, \dots, \infty), \quad (23.3)$$

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

$$P(r | f) = (1 - f) e^{-\lambda r} \quad r \in (0, 1, 2, \dots, \infty), \quad (23.4)$$

where  $\lambda = \ln(1/f)$ .

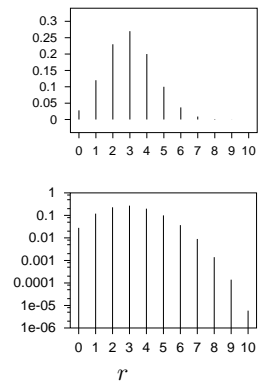


Figure 23.1. The binomial distribution  $P(r | f = 0.3, N = 10)$ , on a linear scale (top) and a logarithmic scale (bottom).

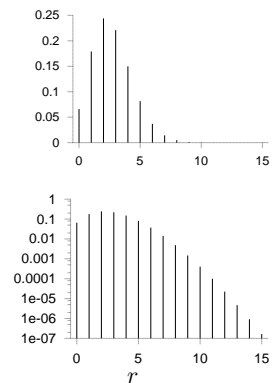


Figure 23.2. The Poisson distribution  $P(r | \lambda = 2.7)$ , on a linear scale (top) and a logarithmic scale (bottom).

## ► 23.2 Distributions over unbounded real numbers

Gaussian, Student, Cauchy, biexponential, inverse-cosh.

The *Gaussian distribution* or normal distribution with mean  $\mu$  and standard deviation  $\sigma$  is

$$P(x | \mu, \sigma) = \frac{1}{Z} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \quad x \in (-\infty, \infty), \quad (23.5)$$

where

$$Z = \sqrt{2\pi\sigma^2}. \quad (23.6)$$

It is sometimes useful to work with the quantity  $\tau \equiv 1/\sigma^2$ , which is called the *precision* parameter of the Gaussian.

A sample  $z$  from a standard univariate Gaussian can be generated by computing

$$z = \cos(2\pi u_1) \sqrt{2 \ln(1/u_2)}, \quad (23.7)$$

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.

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,  $6 \times 10^{-5}$ , and  $6 \times 10^{-7}$ . In my experience, deviations from a mean four or five times greater than the typical deviation may be rare, but not as rare as  $6 \times 10^{-5}$ ! 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.

- **Exercise 23.1.**<sup>[1]</sup> 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 log scale 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  $\pi_1 + \pi_2 = 1$ ,  $\pi_i \geq 0$ .

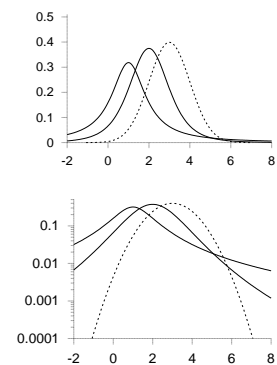
$$P(x | \mu_1, \sigma_1, \pi_1, \mu_2, \sigma_2, \pi_2) = \frac{\pi_1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) + \frac{\pi_2}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right).$$

If we take an appropriately weighted mixture of an infinite number of Gaussians, all having mean  $\mu$ , we obtain a *Student- $t$  distribution*,

$$P(x | \mu, s, n) = \frac{1}{Z} \frac{1}{(1 + (x - \mu)^2/(ns^2))^{(n+1)/2}}, \quad (23.8)$$

where

$$Z = \sqrt{\pi ns^2} \frac{\Gamma(n/2)}{\Gamma((n+1)/2)} \quad (23.9)$$



**Figure 23.3.** Three unimodal distributions. 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  $\sigma = 3$  (dashed line), 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’.

and  $n$  is called the number of degrees of freedom and  $\Gamma$  is the gamma function. If  $n > 1$  then the Student distribution (23.8) 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  $n \rightarrow \infty$ , 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*,

$$P(x|\mu, s) = \frac{1}{Z} \exp\left(-\frac{|x-\mu|}{s}\right) \quad x \in (-\infty, \infty) \quad (23.10)$$

where

$$Z = 2s. \quad (23.11)$$

The *inverse-cosh distribution*

$$P(x|\beta) \propto \frac{1}{[\cosh(\beta x)]^{1/\beta}} \quad (23.12)$$

is a popular model in independent component analysis. In the limit of large  $\beta$ , the probability distribution  $P(x|\beta)$  becomes a biexponential distribution. In the limit  $\beta \rightarrow 0$   $P(x|\beta)$  approaches a Gaussian with mean zero and variance  $1/\beta$ .

### ► 23.3 Distributions over positive real numbers

Exponential, gamma, inverse-gamma, and log-normal.

The *exponential distribution*,

$$P(x|s) = \frac{1}{Z} \exp\left(-\frac{x}{s}\right) \quad x \in (0, \infty), \quad (23.13)$$

where

$$Z = s, \quad (23.14)$$

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  $\infty$ . Just as the Gaussian distribution has two parameters  $\mu$  and  $\sigma$  which control the mean and width of the distribution, the gamma distribution has two parameters. It is the product of the one-parameter exponential distribution (23.13) with a polynomial,  $x^{c-1}$ . The exponent  $c$  in the polynomial is the second parameter.

$$P(x|s, c) = \Gamma(x; s, c) = \frac{1}{Z} \left(\frac{x}{s}\right)^{c-1} \exp\left(-\frac{x}{s}\right), \quad 0 \leq x < \infty \quad (23.15)$$

where

$$Z = \Gamma(c)s. \quad (23.16)$$



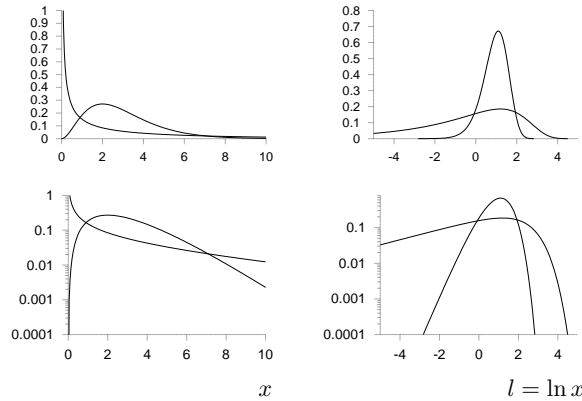


Figure 23.4. Two gamma distributions, with parameters  $(s, c) = (1, 3)$  (heavy lines) and  $10, 0.3$  (light lines), shown on linear vertical scales (top) and logarithmic vertical scales (bottom); and shown as a function of  $x$  on the left (23.15) and  $l = \ln x$  on the right (23.18).

This is a simple peaked distribution with mean  $sc$  and variance  $s^2c$ .

It is often natural to represent a positive real variable  $x$  in terms of its logarithm  $l = \ln x$ . The probability density of  $l$  is

$$P(l) = P(x(l)) \left| \frac{\partial x}{\partial l} \right| = P(x(l))x(l) \quad (23.17)$$

$$= \frac{1}{Z_l} \left( \frac{x(l)}{s} \right)^c \exp \left( -\frac{x(l)}{s} \right), \quad (23.18)$$

where

$$Z_l = \Gamma(c). \quad (23.19)$$

[The gamma distribution is named after its normalizing constant – an odd convention, it seems to me!]

Figure 23.4 shows a couple of gamma distributions as a function of  $x$  and of  $l$ . Notice that where the original gamma distribution (23.15) 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  $sc = 1, c \rightarrow 0$ , 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  $l = \ln x$  we find the latter density is uniform.

- ▷ Exercise 23.2.<sup>[1]</sup> 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  $l = \ln x$ , 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*,

$$P(v | s, c) = \frac{1}{Z_v} \left( \frac{1}{sv} \right)^{c+1} \exp \left( -\frac{1}{sv} \right), \quad 0 \leq v < \infty \quad (23.20)$$

where

$$Z_v = \Gamma(c)/s. \quad (23.21)$$

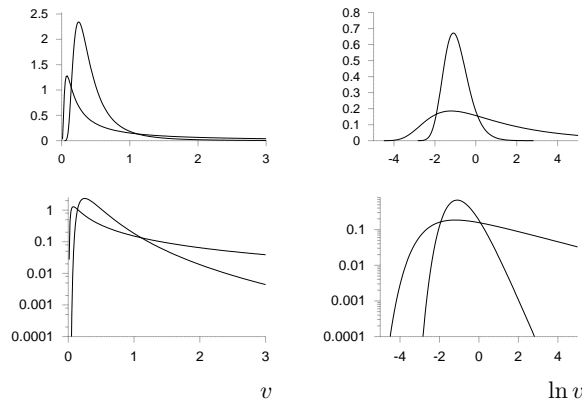


Figure 23.5. Two inverse gamma distributions, with parameters  $(s, c) = (1, 3)$  (heavy lines) and  $10, 0.3$  (light lines), shown on linear vertical scales (top) and logarithmic vertical scales (bottom); and shown as a function of  $x$  on the left and  $l = \ln x$  on the right.

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.

Gamma distributions also arise naturally in the distributions of waiting times between Poisson-distributed events. Given a Poisson process with rate  $\lambda$ , the probability density of the arrival time  $x$  of the  $m$ th event is

$$\frac{\lambda(\lambda x)^{m-1}}{(m-1)!} e^{-\lambda x}. \quad (23.22)$$

#### Log-normal distribution

Another distribution over a positive real number  $x$  is the *log-normal* distribution, which is the distribution that results when  $l = \ln x$  has a normal distribution. We define  $m$  to be the median value of  $x$ , and  $s$  to be the standard deviation of  $\ln x$ .

$$P(l | m, s) = \frac{1}{Z} \exp\left(-\frac{(l - \ln m)^2}{2s^2}\right) \quad l \in (-\infty, \infty), \quad (23.23)$$

where

$$Z = \sqrt{2\pi s^2}, \quad (23.24)$$

implies

$$P(x | m, s) = \frac{1}{x Z} \exp\left(-\frac{(\ln x - \ln m)^2}{2s^2}\right) \quad x \in (0, \infty). \quad (23.25)$$

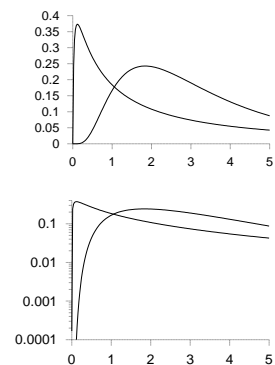


Figure 23.6. Two log-normal distributions, with parameters  $(m, s) = (3, 1.8)$  (heavy line) and  $(3, 0.7)$  (light line), shown on linear vertical scales (top) and logarithmic vertical scales (bottom). [Yes, they really do have the same value of the median,  $m = 3$ .]

### ► 23.4 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*:

$$P(\theta | \mu, \beta) = \frac{1}{Z} \exp(\beta \cos(\theta - \mu)) \quad \theta \in (0, 2\pi). \quad (23.26)$$

The normalizing constant is  $Z = 2\pi I_0(\beta)$ , where  $I_0(x)$  is a modified Bessel function.

A distribution that arises from Brownian diffusion around the circle is the wrapped Gaussian distribution,

$$P(\theta | \mu, \sigma) = \sum_{n=-\infty}^{\infty} \text{Normal}(\theta; (\mu + 2\pi n), \sigma^2) \quad \theta \in (0, 2\pi). \quad (23.27)$$

### ► 23.5 Distributions over probabilities

Beta distribution, Dirichlet distribution, entropic distribution

The *beta distribution* is a probability density over a variable  $p$  that is a probability,  $p \in (0, 1)$ :

$$P(p | u_1, u_2) = \frac{1}{Z(u_1, u_2)} p^{u_1-1} (1-p)^{u_2-1}. \quad (23.28)$$

The parameters  $u_1, u_2$  may take any positive value. The normalizing constant is the beta function,

$$Z(u_1, u_2) = \frac{\Gamma(u_1)\Gamma(u_2)}{\Gamma(u_1 + u_2)}. \quad (23.29)$$

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  $l \equiv \ln p / (1 - p)$ , we find it is always a pleasant bell-shaped density over  $l$ , while the density over  $p$  may have singularities at  $p = 0$  and  $p = 1$  (figure 23.7).

#### 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:

$$P(\mathbf{p} | \alpha \mathbf{m}) = \frac{1}{Z(\alpha \mathbf{m})} \prod_{i=1}^I p_i^{\alpha m_i - 1} \delta(\sum_i p_i - 1) \equiv \text{Dirichlet}^{(I)}(\mathbf{p} | \alpha \mathbf{m}). \quad (23.30)$$

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:

$$Z(\alpha \mathbf{m}) = \prod_i \Gamma(\alpha m_i) / \Gamma(\alpha). \quad (23.31)$$

The vector  $\mathbf{m}$  is the mean of the probability distribution:

$$\int \text{Dirichlet}^{(I)}(\mathbf{p} | \alpha \mathbf{m}) \mathbf{p} d^I \mathbf{p} = \mathbf{m}. \quad (23.32)$$

When working with a probability vector  $\mathbf{p}$ , it is often helpful to work in the ‘softmax basis’, in which, for example, a three-dimensional probability  $\mathbf{p} = (p_1, p_2, p_3)$  is represented by three numbers  $a_1, a_2, a_3$  satisfying  $a_1 + a_2 + a_3 = 0$  and

$$p_i = \frac{1}{Z} e^{a_i}, \quad \text{where } Z = \sum_i e^{a_i}. \quad (23.33)$$

This nonlinear transformation is analogous to the  $\sigma \rightarrow \ln \sigma$  transformation for a scale variable and the logit transformation for a single probability,  $p \rightarrow$

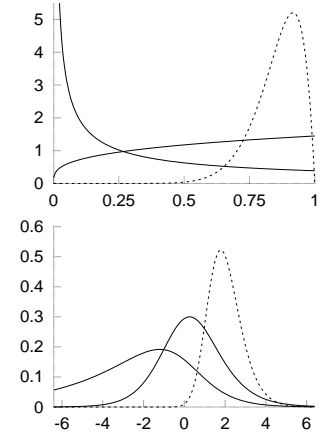


Figure 23.7. Three beta distributions, with  $(u_1, u_2) = (0.3, 1)$ ,  $(1.3, 1)$ , and  $(12, 2)$ . The upper figure shows  $P(p | u_1, u_2)$  as a function of  $p$ ; the lower shows the corresponding density over the *logit*,

$$\ln \frac{p}{1-p}.$$

Notice how well-behaved the densities are as a function of the logit.

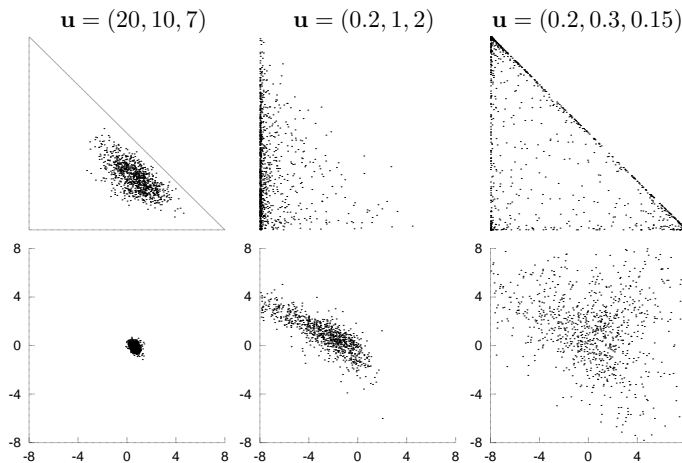


Figure 23.8. Three Dirichlet distributions over a three-dimensional probability vector  $(p_1, p_2, p_3)$ . The upper figures show 1000 random draws from each distribution, showing the values of  $p_1$  and  $p_2$  on the two axes.  $p_3 = 1 - (p_1 + p_2)$ . The triangle in the first figure is the simplex of legal probability distributions. The lower figures show the same points in the ‘softmax’ basis (equation (23.33)). The two axes show  $a_1$  and  $a_2$ .  $a_3 = -a_1 - a_2$ .

$\ln \frac{p}{1-p}$ . In the softmax basis, the ugly minus-ones in the exponents in the Dirichlet distribution (23.30) disappear, and the density is given by:

$$P(\mathbf{a} | \alpha \mathbf{m}) \propto \frac{1}{Z(\alpha \mathbf{m})} \prod_{i=1}^I p_i^{\alpha m_i} \delta(\sum_i a_i). \quad (23.34)$$

The role of the parameter  $\alpha$  can be characterized in two ways. First,  $\alpha$  measures the sharpness of the distribution (figure 23.8); it measures how different we expect typical samples  $\mathbf{p}$  from the distribution to be from the mean  $\mathbf{m}$ , just as the precision  $\tau = 1/\sigma^2$  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  $\alpha$  in higher-dimensional situations can be visualized by drawing a typical sample from the distribution  $\text{Dirichlet}^{(I)}(\mathbf{p} | \alpha \mathbf{m})$ , 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 23.9 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 small 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  $\mathbf{F} = (F_1, F_2, \dots, F_I)$  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.

**Exercise 23.3.**<sup>[3]</sup> The Dirichlet distribution satisfies a nice additivity property.

Imagine that a biased six-sided die has two red faces and four blue faces. The die is rolled  $N$  times and two Bayesians examine the outcomes in order to infer the bias of the die and make predictions. One Bayesian has access to the red/blue colour outcomes only, and he infers a two-component probability vector  $(p_R, p_B)$ . The other Bayesian has access to each full outcome: he can see which of the six faces came up, and he infers a six-component probability vector  $(p_1, p_2, p_3, p_4, p_5, p_6)$ , where

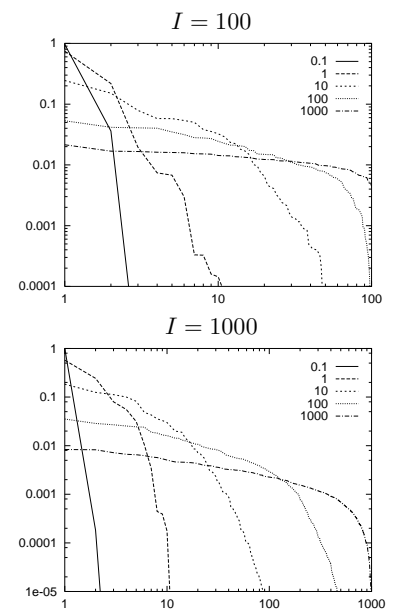


Figure 23.9. Zipf plots for random samples from Dirichlet distributions with various values of  $\alpha = 0.1 \dots 1000$ . For each value of  $I = 100$  or  $1000$  and each  $\alpha$ , one sample  $\mathbf{p}$  from the Dirichlet distribution was generated. The Zipf plot shows the probabilities  $p_i$ , ranked by magnitude, versus their rank.

$p_R = p_1 + p_2$  and  $p_B = p_3 + p_4 + p_5 + p_6$ . Assuming that the second Bayesian assigns a Dirichlet distribution to  $(p_1, p_2, p_3, p_4, p_5, p_6)$  with hyperparameters  $(u_1, u_2, u_3, u_4, u_5, u_6)$ , show that, in order for the first Bayesian's inferences to be consistent with those of the second Bayesian, the first Bayesian's prior should be a Dirichlet distribution with hyperparameters  $((u_1 + u_2), (u_3 + u_4 + u_5 + u_6))$ .

**Hint:** a brute-force approach is to compute the integral  $P(p_R, p_B) = \int d^6 \mathbf{p} P(\mathbf{p} | \mathbf{u}) \delta(p_R - (p_1 + p_2)) \delta(p_B - (p_3 + p_4 + p_5 + p_6))$ . A cheaper approach is to compute the predictive distributions, given arbitrary data  $(F_1, F_2, F_3, F_4, F_5, F_6)$ , and find the condition for the two predictive distributions to match for all data.

The *entropic distribution* for a probability vector  $\mathbf{p}$  is sometimes used in the 'maximum entropy' image reconstruction community.

$$P(\mathbf{p} | \alpha, \mathbf{m}) = \frac{1}{Z(\alpha, \mathbf{m})} \exp[-\alpha D_{\text{KL}}(\mathbf{p} || \mathbf{m})] \delta(\sum_i p_i - 1), \quad (23.35)$$

where  $\mathbf{m}$ , the measure, is a positive vector, and  $D_{\text{KL}}(\mathbf{p} || \mathbf{m}) = \sum_i p_i \log p_i / m_i$ .

### Further reading

See (MacKay and Peto, 1995) for fun with Dirichlets.

### ► 23.6 Further exercises

Exercise 23.4.<sup>[2]</sup>  $N$  datapoints  $\{x_n\}$  are drawn from a gamma distribution  $P(x | s, c) = \Gamma(x; s, c)$  with unknown parameters  $s$  and  $c$ . What are the maximum likelihood parameters  $s$  and  $c$ ?

## 24

---

### Exact Marginalization

How can we avoid the exponentially large 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  $\infty$ .

#### ► 24.1 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  $\sigma$ :

$$P(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \equiv \text{Normal}(x; \mu, \sigma^2). \quad (24.1)$$

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 that embodies our supposed ignorance. In section 21.2, 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  $P(\mu|\mu_0, \sigma_\mu) = \text{Normal}(\mu; \mu_0, \sigma_\mu^2)$ . In the limit  $\mu_0=0$ ,  $\sigma_\mu \rightarrow \infty$ , 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 (the *precision* parameter)  $\beta = 1/\sigma^2$ :

$$P(\beta) = \Gamma(\beta; b_\beta, c_\beta) = \frac{1}{\Gamma(c_\beta)} \frac{\beta^{c_\beta-1}}{b_\beta^{c_\beta}} \exp\left(-\frac{\beta}{b_\beta}\right), \quad 0 \leq \beta < \infty. \quad (24.2)$$

This is a simple peaked distribution with mean  $b_\beta c_\beta$  and variance  $b_\beta^2 c_\beta$ . In the limit  $b_\beta c_\beta = 1, c_\beta \rightarrow 0$ , 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  $\ln \sigma$ , or  $\ln \beta$ , 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.

Reminder: when we change variables from  $\sigma$  to  $l(\sigma)$ , a one-to-one function of  $\sigma$ , the probability density transforms from  $P_\sigma(\sigma)$  to

$$P_l(l) = P_\sigma(\sigma) \left| \frac{\partial \sigma}{\partial l} \right|.$$

Here, the Jacobian is

$$\left| \frac{\partial \sigma}{\partial \ln \sigma} \right| = \sigma.$$

*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

$$\bar{x} \equiv \sum_{n=1}^N x_n / N, \quad (24.3)$$

and two estimators of  $\sigma$  are:

$$\sigma_N \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N}} \quad \text{and} \quad \sigma_{N-1} \equiv \sqrt{\frac{\sum_{n=1}^N (x_n - \bar{x})^2}{N-1}}. \quad (24.4)$$

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 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.

### 24.1: Inferring the mean and variance of a Gaussian distribution

321

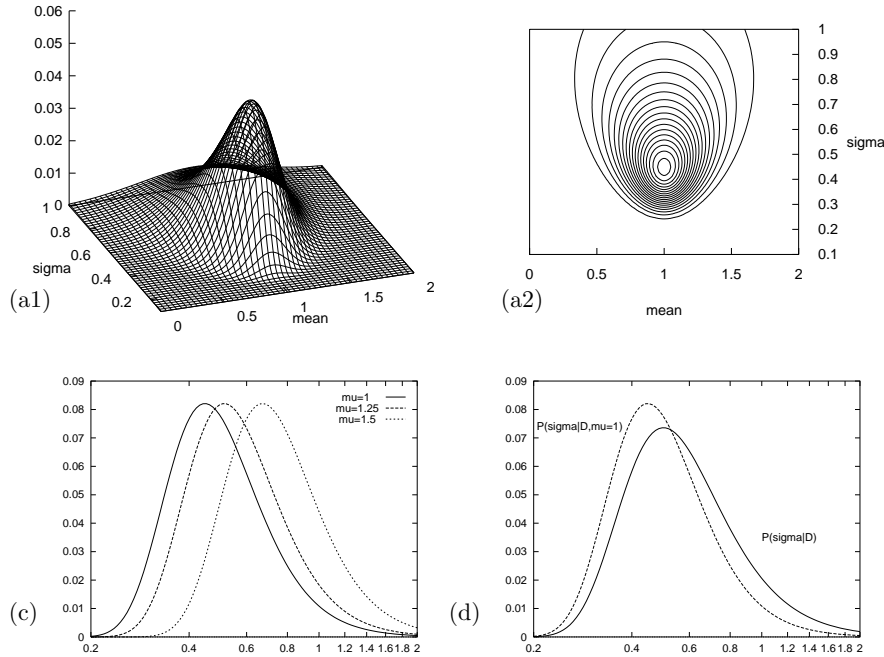


Figure 24.1. The likelihood function for the parameters of a Gaussian distribution, repeated from figure 21.5. (a1, a2) Surface plot and contour plot of the log likelihood as a function of  $\mu$  and  $\sigma$ . The data set of  $N = 5$  points had mean  $\bar{x} = 1.0$  and  $S = \sum (x - \bar{x})^2 = 1.0$ . 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  $\mu$  (shown as a density over  $\ln \sigma$ ). (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  $P(\sigma | D, \mu = \bar{x})$  is at  $\sigma_N$ . (Both probabilities are shown as densities over  $\ln \sigma$ .)

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 24.1.** [2, p.323] 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  $\sigma_{N-1}^2$  that 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 24.1a. The log likelihood is:

$$\ln P(\{x_n\}_{n=1}^N | \mu, \sigma) = -N \ln(\sqrt{2\pi}\sigma) - \sum_n (x_n - \mu)^2 / (2\sigma^2), \quad (24.5)$$

$$= -N \ln(\sqrt{2\pi}\sigma) - [N(\mu - \bar{x})^2 + S] / (2\sigma^2), \quad (24.6)$$

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:

$$P(\mu, \sigma | \{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu, \sigma)}{P(\{x_n\}_{n=1}^N)} \quad (24.7)$$

$$= \frac{\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N(\mu - \bar{x})^2 + S}{2\sigma^2}\right) \frac{1}{\sigma_\mu} \frac{1}{\sigma}}{P(\{x_n\}_{n=1}^N)}. \quad (24.8)$$



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  $(\mu, \ln \sigma)$ , in which our prior is flat, so that the posterior probability maximum coincides with the maximum of the likelihood. As we saw in exercise 22.4 (p.302), the maximum likelihood solution for  $\mu$  and  $\ln \sigma$  is  $\{\mu, \sigma\}_{\text{ML}} = \{\bar{x}, \sigma_N = \sqrt{S/N}\}$ .

There is more to the posterior distribution than just its mode. As can be seen in figure 24.1a, the likelihood has a skew peak. As we increase  $\sigma$ , the width of the conditional distribution of  $\mu$  increases (figure 22.1b). 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 24.1c).

The posterior probability of  $\mu$  given  $\sigma$  is

$$P(\mu | \{x_n\}_{n=1}^N, \sigma) = \frac{P(\{x_n\}_{n=1}^N | \mu, \sigma) P(\mu)}{P(\{x_n\}_{n=1}^N | \sigma)} \quad (24.9)$$

$$\propto \exp(-N(\mu - \bar{x})^2 / (2\sigma^2)) \quad (24.10)$$

$$= \text{Normal}(\mu; \bar{x}, \sigma^2/N). \quad (24.11)$$

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:

$$P(\sigma | \{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N | \sigma) P(\sigma)}{P(\{x_n\}_{n=1}^N)}. \quad (24.12)$$

The data-dependent term  $P(\{x_n\}_{n=1}^N | \sigma)$  appeared earlier as the normalizing constant in equation (24.9); 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  $P(\mu) = \text{constant}$  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:

$$\ln P(\{x_n\}_{n=1}^N | \sigma) = -N \ln(\sqrt{2\pi}\sigma) - \frac{S}{2\sigma^2} + \ln \frac{\sqrt{2\pi}\sigma/\sqrt{N}}{\sigma_\mu}. \quad (24.13)$$

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 Chapter 28.) When we differentiate the log evidence with respect to  $\ln \sigma$ , to find the most probable  $\sigma$ , the additional volume factor  $(\sigma/\sqrt{N})$  shifts the maximum from  $\sigma_N$  to

$$\sigma_{N-1} = \sqrt{S/(N-1)}. \quad (24.14)$$

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. In the terminology of classical

statistics, the Bayesian's best guess for  $\sigma$  sets  $\chi^2$  (the measure of deviance defined by  $\chi^2 \equiv \sum_n (x_n - \hat{\mu})^2 / \hat{\sigma}^2$ ) equal to the number of degrees of freedom,  $N - 1$ .

Figure 24.1d 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 24.1c and d.

The final inference we might wish to make is 'given the data, what is  $\mu$ ?'

- ▷ Exercise 24.2.<sup>[3]</sup> Marginalize over  $\sigma$  and obtain the posterior marginal distribution of  $\mu$ , which is a Student- $t$  distribution:

$$P(\mu | D) \propto 1 / (N(\mu - \bar{x})^2 + S)^{N/2}. \quad (24.15)$$

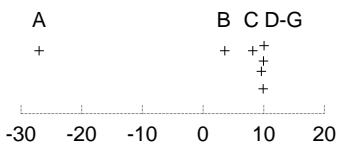
**Further reading**

A bible of exact marginalization is Bretthorst's (1988) book on Bayesian spectrum analysis and parameter estimation.

► **24.2 Exercises**

- ▷ Exercise 24.3.<sup>[3]</sup> [This exercise requires macho integration capabilities.] Give a Bayesian solution to exercise 22.15 (p.309), where seven scientists of varying capabilities have measured  $\mu$  with personal noise levels  $\sigma_n$ , and we are interested in inferring  $\mu$ . Let the prior on each  $\sigma_n$  be a broad prior, for example a gamma distribution with parameters  $(s, c) = (10, 0.1)$ . Find the posterior distribution of  $\mu$ . Plot it, and explore its properties for a variety of data sets such as the one given, and the data set  $\{x_n\} = \{13.01, 7.39\}$ .

[Hint: first find the posterior distribution of  $\sigma_n$  given  $\mu$  and  $x_n$ ,  $P(\sigma_n | x_n, \mu)$ . Note that the normalizing constant for this inference is  $P(x_n | \mu)$ . Marginalize over  $\sigma_n$  to find this normalizing constant, then use Bayes' theorem a second time to find  $P(\mu | \{x_n\})$ .]



► **24.3 Solutions**

Solution to exercise 24.1 (p.321). 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  $\mu$ . Any other value of  $\mu$  (in particular, the true value of  $\mu$ ) will have a larger value of the sum-squared deviation that  $\mu = \bar{x}$ .

So the expected mean squared deviation from the sample mean is necessarily smaller than the mean squared deviation  $\sigma^2$  about the true mean.

## 25

---

### *Exact Marginalization in Trellises*

In this chapter we will discuss a few exact methods that are used in probabilistic modelling. As an example we will discuss the task of decoding a linear error-correcting code. 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 16).

#### ► 25.1 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, take the  $(7, 4)$  Hamming code. In Chapter 1, we discussed the codeword decoding problem for that code, assuming a binary symmetric channel. 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

$$P(\mathbf{t} | \mathbf{y}) = \frac{P(\mathbf{y} | \mathbf{t})P(\mathbf{t})}{P(\mathbf{y})}. \quad (25.1)$$

**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,

$$P(\mathbf{y} | \mathbf{t}) = \prod_{n=1}^N P(y_n | t_n). \quad (25.2)$$

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

$$P(y_n | t_n = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - x)^2}{2\sigma^2}\right) \quad (25.3)$$

$$P(y_n | t_n = 0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n + x)^2}{2\sigma^2}\right). \quad (25.4)$$

From the point of view of decoding, all that matters is the *likelihood ratio*, which for the case of the Gaussian channel is

$$\frac{P(y_n | t_n = 1)}{P(y_n | t_n = 0)} = \exp\left(\frac{2xy_n}{\sigma^2}\right). \quad (25.5)$$



**Exercise 25.1.**<sup>[2]</sup> 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 usually assumed to be uniform over all valid codewords.

The denominator in (25.1) is the normalizing constant

$$P(\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{y} | \mathbf{t}) P(\mathbf{t}). \quad (25.6)$$

The complete solution to the codeword decoding problem is a list of all codewords and their probabilities as given by equation (25.1). 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  $P(\mathbf{y} | \mathbf{t})$ .

**Example:** In Chapter 1, for the (7, 4) Hamming code and a binary symmetric channel we discussed a method for deducing the most probable codeword from the syndrome of the received signal, thus solving the MAP codeword decoding problem for that case. We would like a more general solution.

The MAP codeword decoding problem can 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 25.3, we will discuss an exact method known as the *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 MAP codeword decoding for a *general* linear code is known to be NP-complete (which means in layman's terms that MAP codeword decoding has a complexity that scales exponentially with the blocklength, 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 (25.1) by *marginalizing* over the other bits.

$$P(t_n | \mathbf{y}) = \sum_{\{t_{n'}: n' \neq n\}} P(\mathbf{t} | \mathbf{y}). \quad (25.7)$$

We can also write this marginal with the aid of a truth function  $\mathbb{1}[S]$  that is one if the proposition  $S$  is true and zero otherwise.

$$P(t_n = 1 | \mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t} | \mathbf{y}) \mathbb{1}[t_n = 1] \quad (25.8)$$

$$P(t_n = 0 | \mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t} | \mathbf{y}) \mathbb{1}[t_n = 0]. \quad (25.9)$$

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*. We will describe this algorithm, which is an example of the *sum-product 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.

## ► 25.2 Codes and trellises

In Chapters 1 and 11, we represented linear  $(N, K)$  codes in terms of their generator matrices and their parity-check matrices. In the case of a *systematic* block code, the first  $K$  transmitted bits in each block of size  $N$  are the source bits, and the remaining  $M = N - K$  bits are the parity-check bits. This means that the generator matrix of the code can be written

$$\mathbf{G}^T = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P} \end{bmatrix}, \quad (25.10)$$

and the parity-check matrix can be written

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_M \end{bmatrix}, \quad (25.11)$$

where  $\mathbf{P}$  is an  $M \times K$  matrix.

In this section we will 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.

### Definition of a trellis

Our definition will be quite narrow. For a more comprehensive view of trellises, the reader should consult Kschischang and Sorkine (1995).

**A trellis** is a *graph* consisting of *nodes* (also known as states or vertices) and *edges*. The nodes are grouped into vertical slices called *times*, and the times are ordered such that each edge connects a node in one time to a node in a neighbouring time. 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.

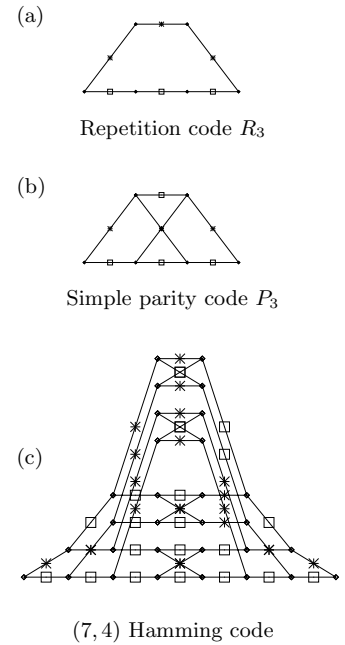


Figure 25.1. Examples of trellises. Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

A trellis with  $N+1$  times defines a code of blocklength  $N$  as follows: a codeword is obtained by taking a path that crosses 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 time ‘time 0’ and the rightmost ‘time  $N$ ’. We will number the leftmost state ‘state 0’ and the rightmost ‘state  $I$ ’, where  $I$  is the total number of states (vertices) in the trellis. The  $n$ th bit of the codeword is emitted as we move from time  $n-1$  to time  $n$ .

The *width* of the trellis at a given time is the number of nodes in that time. 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. 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 25.1(a–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 25.2.<sup>[2]</sup> Confirm that the sixteen codewords listed in table 1.14 are generated by the trellis shown in figure 25.1c.

### 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 time 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 25.4.

Notice that for the linear trellises in figure 25.1, 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 25.4. But we now know enough to discuss the decoding problem.

## ► 25.3 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. 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, (a) we want to identify the most probable path in order to solve the codeword decoding problem; and (b) we want to find the probability that the transmitted symbol at time  $n$  was a zero or a one, to solve the bitwise decoding problem.

**Example 25.3.** Consider the case of a single transmission from the Hamming  $(7, 4)$  trellis shown in figure 25.1c.


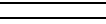
<b>t</b>	Likelihood	Posterior probability	
0000000	0.0275562	0.25	
0001011	0.0001458	0.0013	
0010111	0.0013122	0.012	
0011100	0.0030618	0.027	
0100110	0.0002268	0.0020	
0101101	0.0000972	0.0009	
0110001	0.0708588	0.63	
0111010	0.0020412	0.018	
1000101	0.0001458	0.0013	
1001110	0.0000042	0.0000	
1010010	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 25.2. 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).

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

$$\frac{P(y_1 | x_1 = 1)}{P(y_1 | x_1 = 0)} = \frac{0.1}{0.9}, \quad \frac{P(y_2 | x_2 = 1)}{P(y_2 | x_2 = 0)} = \frac{0.4}{0.6}, \quad \text{etc.} \quad (25.12)$$

How should this received signal be decoded?

1. 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.

2. We can find the posterior probability over codewords by explicit enumeration of all sixteen codewords. This posterior distribution is shown in figure 25.2. Of course, we aren't really interested in such brute-force solutions, and the aim of this chapter 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 25.2, we can also compute the posterior marginal distributions of each of the bits. The result is shown in figure 25.3. 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.  $\square$

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.

$n$	Likelihood		Posterior marginals			
	$P(y_n   t_n = 1)$	$P(y_n   t_n = 0)$	$P(t_n = 1   \mathbf{y})$		$P(t_n = 0   \mathbf{y})$	
1	0.1	0.9	0.061		0.939	
2	0.4	0.6	0.674		0.326	
3	0.9	0.1	0.746		0.254	
4	0.1	0.9	0.061		0.939	
5	0.1	0.9	0.061		0.939	
6	0.1	0.9	0.061		0.939	
7	0.3	0.7	0.659		0.341	

Figure 25.3. Marginal posterior probabilities for the 7 bits under the posterior distribution of figure 25.2.



**Exercise 25.4.** [2, p.333] 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.]

We now discuss how to use message passing on a code’s trellis to solve the decoding problems.

*The min-sum algorithm*

The MAP codeword decoding problem can be solved using the min-sum algorithm that was introduced in section 16.3. Each codeword of the code corresponds to a path across the trellis. Just as the cost of a journey is the sum of the costs of its constituent steps, the log likelihood of a codeword is the sum of the bitwise log likelihoods. By convention, we flip the sign of the log likelihood (which we would like to maximize) and talk in terms of a cost, which we would like to minimize.

We associate with each edge a cost  $-\log P(y_n | t_n)$ , where  $t_n$  is the transmitted bit associated with that edge, and  $y_n$  is the received symbol. The min-sum algorithm presented in section 16.3 can then identify the most probable codeword in a number of computer operations equal to the number of edges in the trellis. This algorithm is also known as the Viterbi algorithm (Viterbi, 1967).

*The sum-product algorithm*

To solve the bitwise decoding problem, we can make a small modification to the min-sum algorithm, so that the messages passed through the trellis define ‘the probability of the data up to the current point’ instead of ‘the cost of the best route to this point’. We replace the costs on the edges,  $-\log P(y_n | t_n)$ , by the likelihoods themselves,  $P(y_n | t_n)$ . We replace the min and sum operations of the min-sum algorithm by a sum and product respectively.

Let  $i$  run over nodes/states,  $i = 0$  be the label for the start state,  $\mathcal{P}(i)$  denote the set of states that are parents of state  $i$ , and  $w_{ij}$  be the likelihood associated with the edge from node  $j$  to node  $i$ . We define the forward-pass messages  $\alpha_i$  by

$$\alpha_0 = 1$$



$$\alpha_i = \sum_{j \in \mathcal{P}(i)} w_{ij} \alpha_j. \quad (25.13)$$

These messages can be computed sequentially from left to right.

- ▷ Exercise 25.5.<sup>[2]</sup> Show that for a node  $i$  whose time-coordinate is  $n$ ,  $\alpha_i$  is proportional to the joint probability that the codeword's path passed through node  $i$  and that the first  $n$  received symbols were  $y_1, \dots, y_n$ .

The message  $\alpha_I$  computed at the end node of the trellis is proportional to the marginal probability of the data.

- ▷ Exercise 25.6.<sup>[2]</sup> What is the constant of proportionality? [Answer:  $2^K$ ]

We define a second set of backward-pass messages  $\beta_i$  in a similar manner. Let node  $I$  be the end node.

$$\begin{aligned} \beta_I &= 1 \\ \beta_j &= \sum_{i: j \in \mathcal{P}(i)} w_{ij} \beta_i. \end{aligned} \quad (25.14)$$

These messages can be computed sequentially in a backward pass from right to left.

- ▷ Exercise 25.7.<sup>[2]</sup> Show that for a node  $i$  whose time-coordinate is  $n$ ,  $\beta_i$  is proportional to the conditional probability, *given* that the codeword's path passed through node  $i$ , that the subsequent received symbols were  $y_{n+1} \dots y_N$ .

Finally, to find the probability that the  $n$ th bit was a 1 or 0, we do two summations of products of the forward and backward messages. Let  $i$  run over nodes at time  $n$  and  $j$  run over nodes at time  $n-1$ , and let  $t_{ij}$  be the value of  $t_n$  associated with the trellis edge from node  $j$  to node  $i$ . For each value of  $t = 0/1$ , we compute

$$r_n^{(t)} = \sum_{i, j: j \in \mathcal{P}(i), t_{ij}=t} \alpha_j w_{ij} \beta_i. \quad (25.15)$$

Then the posterior probability that  $t_n$  was  $t = 0/1$  is

$$P(t_n = t | \mathbf{y}) = \frac{1}{Z} r_n^{(t)}, \quad (25.16)$$

where the normalizing constant  $Z = r_n^{(0)} + r_n^{(1)}$  should be identical to the final forward message  $\alpha_I$  that was computed earlier.

- Exercise 25.8.<sup>[2]</sup> Confirm that the above sum-product algorithm does compute  $P(t_n = t | \mathbf{y})$ .

Other names for the sum-product algorithm presented here are 'the forward-backward algorithm', 'the BCJR algorithm', and 'belief propagation'.

- ▷ Exercise 25.9.<sup>[2, p.333]</sup> A codeword of the simple parity code  $P_3$  is transmitted, and the received signal  $\mathbf{y}$  has associated likelihoods shown in table 25.4. Use the min-sum algorithm and the sum-product algorithm in the trellis (figure 25.1) to solve the MAP codeword decoding problem and the bitwise decoding problem. Confirm your answers by enumeration of all codewords (000, 011, 110, 101). [Hint: use logs to base 2 and do the min-sum computations by hand. When working the sum-product algorithm by hand, you may find it helpful to use three colours of pen, one for the  $\alpha$ s, one for the  $w$ s, and one for the  $\beta$ s.]

$n$	$P(y_n   t_n)$	
	$t_n = 0$	$t_n = 1$
1	1/4	1/2
2	1/2	1/4
3	1/8	1/2

Table 25.4. Bitwise likelihoods for a codeword of  $P_3$ .

## ► 25.4 More on trellises

We now 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 table 25.5.

Codeword	0000000	0001011	0100110	1100011	0101101
Span	0000000	0001111	0111110	1111111	0111111

Table 25.5. 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

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad (25.17)$$

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:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \quad (25.18)$$

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 25.6.

We build the trellis incrementally as shown in figure 25.6. We start with the trellis corresponding to the subcode given by the first row of the generator matrix. Then we add in one subcode at a time. The vertices within the span of the new subcode are all duplicated. The edge symbols in the original trellis are left unchanged and the edge symbols in the second part of the trellis are flipped wherever the new subcode has a 1 and otherwise left alone.

Another (7, 4) Hamming code can be generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (25.19)$$

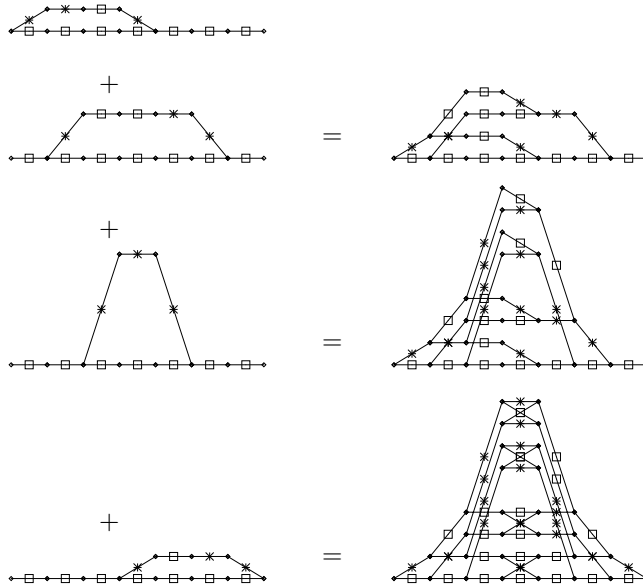


Figure 25.6. Trellises for four subcodes of the (7, 4) Hamming code (left column), and the sequence of trellises that are made when constructing the trellis for the (7, 4) Hamming code (right column). Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

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:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (25.20)$$

The trellis obtained from the permuted matrix  $\mathbf{G}$  given in equation (25.19) is shown in figure 25.7a. 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 25.1c. We thus observe that *rearranging the order of the codeword bits can sometimes lead to smaller, simpler trellises*.

#### 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  $\mathbf{H}\mathbf{r}$ , 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)

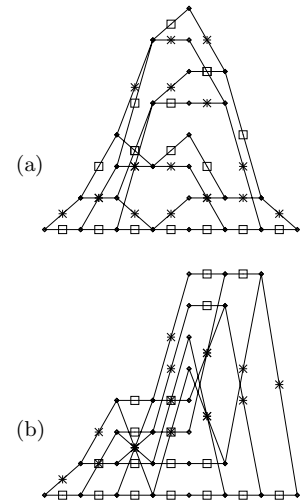


Figure 25.7. Trellises for the permuted (7, 4) Hamming code generated from (a) the generator matrix by the method of figure 25.6; (b) the parity-check matrix by the method on page 332. Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

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 25.7b shows the trellis corresponding to the parity-check matrix of equation (25.20).

► 25.5 Solutions

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

Table 25.8. The posterior probability over codewords for exercise 25.4.

Solution to exercise 25.4 (p.329). The posterior probability over codewords is shown in table 25.8. The most probable codeword is 0000000. The marginal posterior probabilities of all seven bits are:

n	Likelihood		Posterior marginals			
	$P(y_n   t_n = 1)$	$P(y_n   t_n = 0)$	$P(t_n = 1   \mathbf{y})$		$P(t_n = 0   \mathbf{y})$	
1	0.2	0.8	0.266		0.734	
2	0.2	0.8	0.266		0.734	
3	0.9	0.1	0.677		0.323	
4	0.2	0.8	0.266		0.734	
5	0.2	0.8	0.266		0.734	
6	0.2	0.8	0.266		0.734	
7	0.2	0.8	0.266		0.734	

So the bitwise decoding is 0010000, which is not actually a codeword.

Solution to exercise 25.9 (p.330). The MAP codeword is 101, and its likelihood is  $1/8$ . The normalizing constant of the sum-product algorithm is  $Z = \alpha_I = 3/16$ . The intermediate  $\alpha_i$  are (from left to right)  $1/2, 1/4, 5/16, 4/16$ ; the intermediate  $\beta_i$  are (from right to left),  $1/2, 1/8, 9/32, 3/16$ . The bitwise decoding is:  $P(t_1 = 1 | \mathbf{y}) = 3/4$ ;  $P(t_1 = 1 | \mathbf{y}) = 1/4$ ;  $P(t_1 = 1 | \mathbf{y}) = 5/6$ . The codewords' probabilities are  $1/12, 2/12, 1/12, 8/12$  for 000, 011, 110, 101.

## 26

---

### *Exact Marginalization in Graphs*

We now take a more general view of the tasks of inference and marginalization. Before reading this chapter, you should read about message passing in Chapter 16.

#### ► 26.1 The general problem

Assume that a function  $P^*$  of a set of  $N$  variables  $\mathbf{x} \equiv \{x_n\}_{n=1}^N$  is defined as a product of  $M$  factors as follows:

$$P^*(\mathbf{x}) = \prod_{m=1}^M f_m(\mathbf{x}_m). \quad (26.1)$$

Each of the factors  $f_m(\mathbf{x}_m)$  is a function of a subset  $\mathbf{x}_m$  of the variables that make up  $\mathbf{x}$ . If  $P^*$  is a positive function then we may be interested in a second normalized function,

$$P(\mathbf{x}) \equiv \frac{1}{Z} P^*(\mathbf{x}) = \frac{1}{Z} \prod_{m=1}^M f_m(\mathbf{x}_m), \quad (26.2)$$

where the normalizing constant  $Z$  is defined by

$$Z = \sum_{\mathbf{x}} \prod_{m=1}^M f_m(\mathbf{x}_m). \quad (26.3)$$

As an example of the notation we've just introduced, here's a function of three binary variables  $x_1, x_2, x_3$  defined by the five factors:

$$\begin{aligned} f_1(x_1) &= \begin{cases} 0.1 & x_1 = 0 \\ 0.9 & x_1 = 1 \end{cases} \\ f_2(x_2) &= \begin{cases} 0.1 & x_2 = 0 \\ 0.9 & x_2 = 1 \end{cases} \\ f_3(x_3) &= \begin{cases} 0.9 & x_3 = 0 \\ 0.1 & x_3 = 1 \end{cases} \\ f_4(x_1, x_2) &= \begin{cases} 1 & (x_1, x_2) = (0, 0) \text{ or } (1, 1) \\ 0 & (x_1, x_2) = (1, 0) \text{ or } (0, 1) \end{cases} \\ f_5(x_2, x_3) &= \begin{cases} 1 & (x_2, x_3) = (0, 0) \text{ or } (1, 1) \\ 0 & (x_2, x_3) = (1, 0) \text{ or } (0, 1) \end{cases} \\ P^*(\mathbf{x}) &= f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_1, x_2)f_5(x_2, x_3) \\ P(\mathbf{x}) &= \frac{1}{Z} f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_1, x_2)f_5(x_2, x_3). \end{aligned} \quad (26.4)$$

The five subsets of  $\{x_1, x_2, x_3\}$  denoted by  $\mathbf{x}_m$  in the general function (26.1) are here  $\mathbf{x}_1 = \{x_1\}$ ,  $\mathbf{x}_2 = \{x_2\}$ ,  $\mathbf{x}_3 = \{x_3\}$ ,  $\mathbf{x}_4 = \{x_1, x_2\}$ , and  $\mathbf{x}_5 = \{x_2, x_3\}$ .

The function  $P(\mathbf{x})$ , by the way, may be recognized as the posterior probability distribution of the three transmitted bits in a repetition code (section 1.2) when the received signal is  $\mathbf{r} = (1, 1, 0)$  and the channel is a binary symmetric channel with flip probability 0.1. The factors  $f_4$  and  $f_5$  respectively enforce the constraints that  $x_1$  and  $x_2$  must be identical and that  $x_2$  and  $x_3$  must be identical. The factors  $f_1, f_2, f_3$  are the likelihood functions contributed by each component of  $\mathbf{r}$ .

A function of the factored form (26.1) can be depicted by a *factor graph*, in which the variables are depicted by circular nodes and the factors are depicted by square nodes. An edge is put between variable node  $n$  and factor node  $m$  if the function  $f_m(\mathbf{x}_m)$  has any dependence on variable  $x_n$ . The factor graph for the example function (26.4) is shown in figure 26.1.

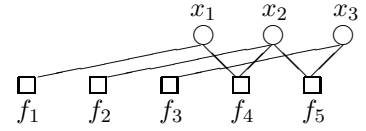


Figure 26.1. The factor graph associated with the function  $P^*(\mathbf{x})$  (26.4).

### The normalization problem

The first task to be solved is to compute the normalizing constant  $Z$ .

### The marginalization problems

The second task to be solved is to compute the marginal function of any variable  $x_n$ , defined by

$$Z_n(x_n) = \sum_{\{x_{n'}\}, n' \neq n} P^*(\mathbf{x}). \quad (26.5)$$

For example, if  $f$  is a function of three variables then the marginal for  $n = 1$  is defined by

$$Z_1(x_1) = \sum_{x_2, x_3} f(x_1, x_2, x_3). \quad (26.6)$$

This type of summation, over ‘all the  $x_{n'}$  except for  $x_n$ ’ is so important that it can be useful to have a special notation for it – the ‘not-sum’ or ‘summary’.

The third task to be solved is to compute the normalized marginal of any variable  $x_n$ , defined by

$$P_n(x_n) \equiv \sum_{\{x_{n'}\}, n' \neq n} P(\mathbf{x}). \quad (26.7)$$

[We include the suffix ‘ $n$ ’ in  $P_n(x_n)$ , departing from our normal practice in the rest of the book, where we would omit it.]

▷ Exercise 26.1.<sup>[1]</sup> Show that the normalized marginal is related to the marginal  $Z_n(x_n)$  by

$$P_n(x_n) = \frac{Z_n(x_n)}{Z}. \quad (26.8)$$

We might also be interested in marginals over a subset of the variables, such as

$$Z_{12}(x_1, x_2) \equiv \sum_{x_3} P^*(x_1, x_2, x_3). \quad (26.9)$$

All these tasks are intractable in general. Even if every factor is a function of only three variables, the cost of computing exact solutions for  $Z$  and for the marginals is believed in general to grow exponentially with the number of variables  $N$ .

For certain functions  $P^*$ , however, the marginals can be computed efficiently by exploiting the factorization of  $P^*$ . The idea of how this efficiency

arises is well illustrated by the message-passing examples of Chapter 16. The sum-product algorithm that we now review is a generalization of message-passing rule-set B (p.242). As was the case there, the sum-product algorithm is only valid if the graph is tree-like.

## ► 26.2 The sum-product algorithm

### Notation

We identify the set of variables that the  $m$ th factor depends on,  $\mathbf{x}_m$ , by the set of their indices  $\mathcal{N}(m)$ . For our example function (26.4), the sets are  $\mathcal{N}(1) = \{1\}$  (since  $f_1$  is a function of  $x_1$  alone),  $\mathcal{N}(2) = \{2\}$ ,  $\mathcal{N}(3) = \{3\}$ ,  $\mathcal{N}(4) = \{1, 2\}$ , and  $\mathcal{N}(5) = \{2, 3\}$ . Similarly we define the set of factors in which variable  $n$  participates, by  $\mathcal{M}(n)$ . We denote a set  $\mathcal{N}(m)$  with variable  $n$  excluded by  $\mathcal{N}(m) \setminus n$ . We introduce the shorthand  $\mathbf{x}_m \setminus n$  or  $\mathbf{x}_{m \setminus n}$  to denote the set of variables in  $\mathbf{x}_m$  with  $x_n$  excluded, i.e.,

$$\mathbf{x}_m \setminus n \equiv \{x_{n'} : n' \in \mathcal{N}(m) \setminus n\}. \quad (26.10)$$

The sum-product algorithm will involve messages of two types passing along the edges in the factor graph: messages  $q_{n \rightarrow m}$  from variable nodes to factor nodes, and messages  $r_{m \rightarrow n}$  from factor nodes to variable nodes. A message (of either type,  $q$  or  $r$ ) that is sent along an edge connecting factor  $f_m$  to variable  $x_n$  is always a function of the variable  $x_n$ .

Here are the two rules for the updating of the two sets of messages.

#### From variable to factor:

$$q_{n \rightarrow m}(x_n) = \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m' \rightarrow n}(x_n). \quad (26.11)$$

#### From factor to variable:

$$r_{m \rightarrow n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left( f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m) \setminus n} q_{n' \rightarrow m}(x_{n'}) \right). \quad (26.12)$$

### How these rules apply to leaves in the factor graph

A node that has only one edge connecting it to another node is called a leaf node.

Some factor nodes in the graph may be connected to only one variable node, in which case the set  $\mathcal{N}(m) \setminus n$  of variables appearing in the factor message update (26.12) is an empty set, and the product of functions  $\prod_{n' \in \mathcal{N}(m) \setminus n} q_{n' \rightarrow m}(x_{n'})$  is the empty product, whose value is 1. Such a factor node therefore always broadcasts to its one neighbour  $x_n$  the message  $r_{m \rightarrow n}(x_n) = f_m(x_n)$ .

Similarly, there may be variable nodes that are connected to only one factor node, so the set  $\mathcal{M}(n) \setminus m$  in (26.11) is empty. These nodes perpetually broadcast the message  $q_{n \rightarrow m}(x_n) = 1$ .

### Starting and finishing, method 1

The algorithm can be initialized in two ways. If the graph is tree-like then it must have nodes that are leaves. These leaf nodes can broadcast their

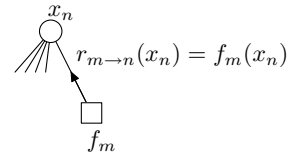


Figure 26.2. A factor node that is a leaf node perpetually sends the message  $r_{m \rightarrow n}(x_n) = f_m(x_n)$  to its one neighbour  $x_n$ .

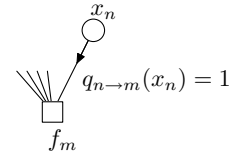


Figure 26.3. A variable node that is a leaf node perpetually sends the message  $q_{n \rightarrow m}(x_n) = 1$ .

messages to their respective neighbours from the start.

$$\text{For all leaf variable nodes } n: \quad q_{n \rightarrow m}(x_n) = 1 \quad (26.13)$$

$$\text{For all leaf factor nodes } m: \quad r_{m \rightarrow n}(x_n) = f_m(x_n). \quad (26.14)$$

We can then adopt the procedure used in Chapter 16's message-passing rule-set B (p.242): a message is created in accordance with the rules (26.11, 26.12) only if all the messages on which it depends are present. For example, in figure 26.4, the message from  $x_1$  to  $f_1$  will be sent only when the message from  $f_4$  to  $x_1$  has been received; and the message from  $x_2$  to  $f_2$ ,  $q_{2 \rightarrow 2}$ , can be sent only when the messages  $r_{4 \rightarrow 2}$  and  $r_{5 \rightarrow 2}$  have both been received.

Messages will thus flow through the tree, one in each direction along every edge, and after a number of steps equal to the diameter of the graph, every message will have been created.

The answers we require can then be read out. The marginal function of  $x_n$  is obtained by multiplying all the incoming messages at that node.

$$Z_n(x_n) = \prod_{m \in \mathcal{M}(n)} r_{m \rightarrow n}(x_n). \quad (26.15)$$

The normalizing constant  $Z$  can be obtained by summing any marginal function,  $Z = \sum_{x_n} Z_n(x_n)$ , and the normalized marginals obtained from

$$P_n(x_n) = \frac{Z_n(x_n)}{Z}. \quad (26.16)$$

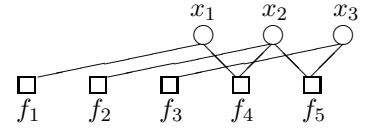


Figure 26.4. Our model factor graph for the function  $P^*(\mathbf{x})$  (26.4).

▷ Exercise 26.2.<sup>[2]</sup> Apply the sum-product algorithm to the function defined in equation (26.4) and figure 26.1. Check that the normalized marginals are consistent with what you know about the repetition code  $R_3$ .

Exercise 26.3.<sup>[3]</sup> Prove that the sum-product algorithm correctly computes the marginal functions  $Z_n(x_n)$  if the graph is tree-like.

Exercise 26.4.<sup>[3]</sup> Describe how to use the messages computed by the sum-product algorithm to obtain more complicated marginal functions in a tree-like graph, for example  $Z_{1,2}(x_1, x_2)$ , for two variables  $x_1$  and  $x_2$  that are connected to one common factor node.

### Starting and finishing, method 2

Alternatively, the algorithm can be initialized by setting *all* the initial messages from variables to 1:

$$\text{for all } n, m: \quad q_{n \rightarrow m}(x_n) = 1, \quad (26.17)$$

then proceeding with the factor message update rule (26.12), alternating with the variable message update rule (26.11). Compared with method 1, this lazy initialization method leads to a load of wasted computations, whose results are gradually flushed out by the correct answers computed by method 1.

After a number of iterations equal to the diameter of the factor graph, the algorithm will converge to a set of messages satisfying the sum-product relationships (26.11, 26.12).

Exercise 26.5.<sup>[2]</sup> Apply this second version of the sum-product algorithm to the function defined in equation (26.4) and figure 26.1.



The reason for introducing this lazy method is that (unlike method 1) it can be applied to graphs that are not tree-like. When the sum-product algorithm is run on a graph with cycles, the algorithm does not necessarily converge, and certainly does not in general compute the correct marginal functions; but it is nevertheless an algorithm of great practical importance, especially in the decoding of sparse-graph codes.

*Sum-product algorithm with on-the-fly normalization*

If we are interested in only the *normalized* marginals, then another version of the sum-product algorithm may be useful. The factor-to-variable messages  $r_{m \rightarrow n}$  are computed in just the same way (26.12), but the variable-to-factor messages are normalized thus:

$$q_{n \rightarrow m}(x_n) = \alpha_{nm} \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m' \rightarrow n}(x_n) \quad (26.18)$$

where  $\alpha_{nm}$  is a scalar chosen such that

$$\sum_{x_n} q_{n \rightarrow m}(x_n) = 1. \quad (26.19)$$

**Exercise 26.6.**<sup>[2]</sup> Apply this normalized version of the sum-product algorithm to the function defined in equation (26.4) and figure 26.1.

*A factorization view of the sum-product algorithm*

One way to view the sum-product algorithm is that it reexpresses the original factored function, the product of  $M$  factors  $P^*(\mathbf{x}) = \prod_{m=1}^M f_m(\mathbf{x}_m)$ , as another factored function which is the product of  $M + N$  factors,

$$P^*(\mathbf{x}) = \prod_{m=1}^M \phi_m(\mathbf{x}_m) \prod_{n=1}^N \psi_n(x_n). \quad (26.20)$$

Each factor  $\phi_m$  is associated with a factor node  $m$ , and each factor  $\psi_n(x_n)$  is associated with a variable node. Initially  $\phi_m(\mathbf{x}_m) = f_m(\mathbf{x}_m)$  and  $\psi_n(x_n) = 1$ .

Each time a factor-to-variable message  $r_{m \rightarrow n}(x_n)$  is sent, the factorization is updated thus:

$$\psi_n(x_n) = \prod_{m \in \mathcal{M}(n)} r_{m \rightarrow n}(x_n) \quad (26.21)$$

$$\phi_m(\mathbf{x}_m) = \frac{f(\mathbf{x}_m)}{\prod_{n \in \mathcal{N}(m)} r_{m \rightarrow n}(x_n)}. \quad (26.22)$$

And each message can be computed in terms of  $\phi$  and  $\psi$  using

$$r_{m \rightarrow n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left( \phi_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)} \psi_{n'}(x_{n'}) \right) \quad (26.23)$$

which differs from the assignment (26.12) in that the product is over all  $n' \in \mathcal{N}(m)$ .

**Exercise 26.7.**<sup>[2]</sup> Confirm that the update rules (26.21–26.23) are equivalent to the sum-product rules (26.11–26.12). So  $\psi_n(x_n)$  eventually becomes the marginal  $Z_n(x_n)$ .

This factorization viewpoint applies whether or not the graph is tree-like.

### Computational tricks

On-the-fly normalization is a good idea from a computational point of view because if  $P^*$  is a product of many factors, its values are likely to be very large or very small.

Another useful computational trick involves passing the logarithms of the messages  $q$  and  $r$  instead of  $q$  and  $r$  themselves; the computations of the products in the algorithm (26.11, 26.12) are then replaced by simpler additions. The summations in (26.12) of course become more difficult: to carry them out and return the logarithm, we need to compute softmax functions like

$$l = \ln(e^{l_1} + e^{l_2} + e^{l_3}). \quad (26.24)$$

But this computation can be done efficiently using look-up tables along with the observation that the value of the answer  $l$  is typically just a little larger than  $\max_i l_i$ . If we store in look-up tables values of the function

$$\ln(1 + e^\delta) \quad (26.25)$$

(for negative  $\delta$ ) then  $l$  can be computed exactly in a number of look-ups and additions scaling as the number of terms in the sum. If look-ups and sorting operations are cheaper than `exp()` then this approach costs less than the direct evaluation (26.24). The number of operations can be further reduced by omitting negligible contributions from the smallest of the  $\{l_i\}$ .

A third computational trick applicable to certain error-correcting codes is to pass not the messages but the Fourier transforms of the messages. This again makes the computations of the factor-to-variable messages quicker. A simple example of this Fourier transform trick is given in Chapter 47 at equation (47.9).

## ► 26.3 The min-sum algorithm

The sum-product algorithm solves the problem of finding the marginal function of a given product  $P^*(\mathbf{x})$ . This is analogous to solving the bitwise decoding problem of section 25.1. And just as there were other decoding problems (for example, the codeword decoding problem), we can define other tasks involving  $P^*(\mathbf{x})$  that can be solved by modifications of the sum-product algorithm. For example, consider this task, analogous to the codeword decoding problem:

**The maximization problem.** Find the setting of  $\mathbf{x}$  that maximizes the product  $P^*(\mathbf{x})$ .

This problem can be solved by replacing the two operations **add** and **multiply** everywhere they appear in the sum-product algorithm by another pair of operations that satisfy the distributive law, namely **max** and **multiply**. If we replace summation  $(+, \sum)$  by maximization, we notice that the quantity formerly known as the normalizing constant,

$$Z = \sum_{\mathbf{x}} P^*(\mathbf{x}), \quad (26.26)$$

becomes  $\max_{\mathbf{x}} P^*(\mathbf{x})$ .

Thus the sum-product algorithm can be turned into a max-product algorithm that computes  $\max_{\mathbf{x}} P^*(\mathbf{x})$ , and from which the solution of the maximization problem can be deduced. Each ‘marginal’  $Z_n(x_n)$  then lists the maximum value that  $P^*(\mathbf{x})$  can attain for each value of  $x_n$ .

In practice, the max-product algorithm is most often carried out in the negative log likelihood domain, where **max** and **product** become **min** and **sum**. The min-sum algorithm is also known as the Viterbi algorithm.

## ► 26.4 The junction tree algorithm

What should one do when the factor graph one is interested in is not a tree?

There are several options, and they divide into exact methods and approximate methods. The most widely used exact method for handling marginalization on graphs with cycles is called the junction tree algorithm. This algorithm works by agglomerating variables together until the agglomerated graph has no cycles. You can probably figure out the details for yourself; the complexity of the marginalization grows exponentially with the number of agglomerated variables. Read more about the junction tree algorithm in (Lauritzen, 1996; Jordan, 1998).

There are many approximate methods, and we'll visit some of them over the next few chapters – Monte Carlo methods and variational methods, to name a couple. However, the most amusing way of handling factor graphs to which the sum-product algorithm may not be applied is, as we already mentioned, to apply the sum-product algorithm! We simply compute the messages for each node in the graph, as if the graph were a tree, iterate, and cross our fingers. This so-called 'loopy' message passing has great importance in the decoding of error-correcting codes, and we'll come back to it in section 33.8 and Part VI.

### Further reading

For further reading about factor graphs and the sum-product algorithm, see Kschischang *et al.* (2001), Yedidia *et al.* (2000), Yedidia *et al.* (2001a), Yedidia *et al.* (2002), Wainwright *et al.* (2003), and Forney (2001).

See also Pearl (1988). A good reference for the fundamental theory of graphical models is Lauritzen (1996). A readable introduction to Bayesian networks is given by Jensen (1996).

Interesting message-passing algorithms that have different capabilities from the sum-product algorithm include *expectation propagation* (Minka, 2001) and *survey propagation* (Braunstein *et al.*, 2003). See also section 33.8.

## ► 26.5 Exercises

- ▷ Exercise 26.8.<sup>[2]</sup> Express the joint probability distribution from the burglar alarm and earthquake problem (example 21.1 (p.293)) as a factor graph, and find the marginal probabilities of all the variables as each piece of information comes to Fred's attention, using the sum-product algorithm with on-the-fly normalization.

## 27

### Laplace's Method

The idea behind the Laplace approximation is simple. We assume that an unnormalized probability density  $P^*(x)$ , whose normalizing constant

$$Z_P \equiv \int P^*(x) dx \quad (27.1)$$

is of interest, has a peak at a point  $x_0$ . We Taylor-expand the logarithm of  $P^*(x)$  around this peak:

$$\ln P^*(x) \simeq \ln P^*(x_0) - \frac{c}{2}(x - x_0)^2 + \dots, \quad (27.2)$$

where

$$c = - \left. \frac{\partial^2}{\partial x^2} \ln P^*(x) \right|_{x=x_0}. \quad (27.3)$$

We then approximate  $P^*(x)$  by an unnormalized Gaussian,

$$Q^*(x) \equiv P^*(x_0) \exp \left[ -\frac{c}{2}(x - x_0)^2 \right], \quad (27.4)$$

and we approximate the normalizing constant  $Z_P$  by the normalizing constant of this Gaussian,

$$Z_Q = P^*(x_0) \sqrt{\frac{2\pi}{c}}. \quad (27.5)$$

We can generalize this integral to approximate  $Z_P$  for a density  $P^*(\mathbf{x})$  over a  $K$ -dimensional space  $\mathbf{x}$ . If the matrix of second derivatives of  $-\ln P^*(\mathbf{x})$  at the maximum  $\mathbf{x}_0$  is  $\mathbf{A}$ , defined by:

$$A_{ij} = - \left. \frac{\partial^2}{\partial x_i \partial x_j} \ln P^*(\mathbf{x}) \right|_{\mathbf{x}=\mathbf{x}_0}, \quad (27.6)$$

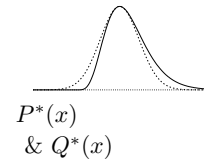
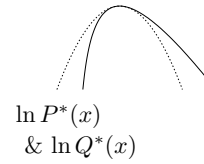
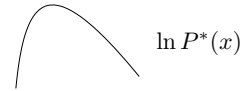
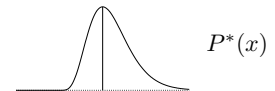
so that the expansion (27.2) is generalized to

$$\ln P^*(\mathbf{x}) \simeq \ln P^*(\mathbf{x}_0) - \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \mathbf{A} (\mathbf{x} - \mathbf{x}_0) + \dots, \quad (27.7)$$

then the normalizing constant can be approximated by:

$$Z_P \simeq Z_Q = P^*(\mathbf{x}_0) \frac{1}{\sqrt{\det \frac{1}{2\pi} \mathbf{A}}} = P^*(\mathbf{x}_0) \sqrt{\frac{(2\pi)^K}{\det \mathbf{A}}}. \quad (27.8)$$

Predictions can be made using the approximation  $Q$ . Physicists also call this widely-used approximation the *saddle-point approximation*.



The fact that the normalizing constant of a Gaussian is given by

$$\int d^K \mathbf{x} \exp \left[ -\frac{1}{2} \mathbf{x}^\top \mathbf{A} \mathbf{x} \right] = \sqrt{\frac{(2\pi)^K}{\det \mathbf{A}}} \quad (27.9)$$

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

$$\int du_i \exp \left[ -\frac{1}{2} \lambda_i u_i^2 \right] = \sqrt{\frac{2\pi}{\lambda_i}}. \quad (27.10)$$

The product of the eigenvalues  $\lambda_i$  is the determinant of  $\mathbf{A}$ .

The Laplace approximation is basis-dependent: if  $x$  is transformed to a nonlinear function  $u(x)$  and the density is transformed to  $P(u) = P(x) |dx/du|$  then in general the approximate normalizing constants  $Z_Q$  will be different. This can be viewed as a defect – since the true value  $Z_P$  is basis-independent – or an opportunity – because we can hunt for a choice of basis in which the Laplace approximation is most accurate.

## ► 27.1 Exercises



**Exercise 27.1.**<sup>[2]</sup> (See also exercise 22.8 (p.307).) A photon counter is pointed at a remote star for one minute, in order to infer the rate of photons arriving at the counter per minute,  $\lambda$ . Assuming the number of photons collected  $r$  has a Poisson distribution with mean  $\lambda$ ,

$$P(r | \lambda) = \exp(-\lambda) \frac{\lambda^r}{r!}, \quad (27.11)$$

and assuming the improper prior  $P(\lambda) = 1/\lambda$ , make Laplace approximations to the posterior distribution

- (a) over  $\lambda$
- (b) over  $\log \lambda$ . [Note the improper prior transforms to  $P(\log \lambda) = \text{constant}$ .]

► **Exercise 27.2.**<sup>[2]</sup> Use Laplace's method to approximate the integral

$$Z(u_1, u_2) = \int_{-\infty}^{\infty} da f(a)^{u_1} (1 - f(a))^{u_2}, \quad (27.12)$$

where  $f(a) = 1/(1 + e^{-a})$  and  $u_1, u_2$  are positive. Check the accuracy of the approximation against the exact answer (23.29, p.316) for  $(u_1, u_2) = (1/2, 1/2)$  and  $(u_1, u_2) = (1, 1)$ . Measure the error  $(\log Z_P - \log Z_Q)$  in bits.

► **Exercise 27.3.**<sup>[3]</sup> **Linear regression.**  $N$  datapoints  $\{(x^{(n)}, t^{(n)})\}$  are generated by the experimenter choosing each  $x^{(n)}$ , then the world delivering a noisy version of the linear function

$$y(x) = w_0 + w_1 x, \quad (27.13)$$

$$t^{(n)} \sim \text{Normal}(y(x^{(n)}), \sigma_\nu^2). \quad (27.14)$$

Assuming Gaussian priors on  $w_0$  and  $w_1$ , make the Laplace approximation to the posterior distribution of  $w_0$  and  $w_1$  (which is exact, in fact) and obtain the predictive distribution for the next datapoint  $t^{(N+1)}$ , given  $x^{(N+1)}$ .

(See MacKay (1992a) for further reading.)

# 28

## Model Comparison and Occam's Razor

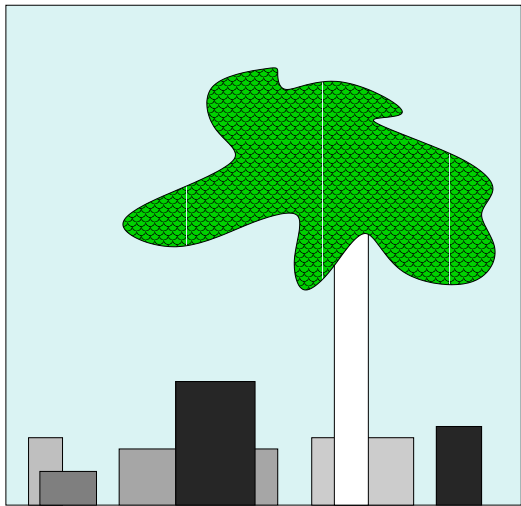


Figure 28.1. A picture to be interpreted. It contains a tree and some boxes.

### ► 28.1 Occam's razor

How many boxes are in the picture (figure 28.1)? In particular, how many boxes are in the vicinity of the tree? If we looked with x-ray spectacles, would we see one or two boxes behind the trunk (figure 28.2)? (Or even more?) Occam's razor is the principle that states a preference for simple theories. 'Accept the simplest explanation that fits the data'. Thus according to Occam's razor, we should deduce that there is only one box behind the tree. Is this an ad hoc rule of thumb? Or is there a convincing reason for believing there is most likely one box? Perhaps your intuition likes the argument 'well, it would be a remarkable *coincidence* for the two boxes to be just the same height and colour as each other'. If we wish to make artificial intelligences that interpret data correctly, we must translate this intuitive feeling into a concrete theory.

#### *Motivations for Occam's razor*

If several explanations are compatible with a set of observations, Occam's razor advises us to buy the simplest. 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')

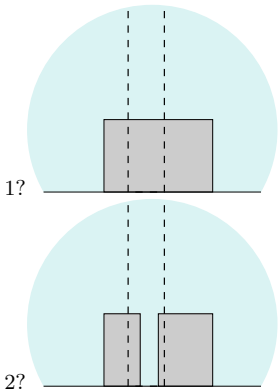
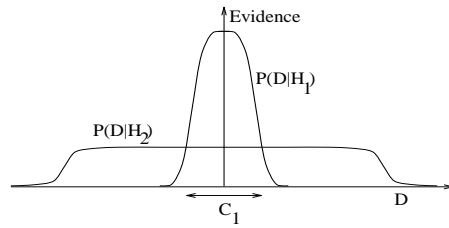


Figure 28.2. How many boxes are behind the tree?



(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:

Coherent inference (as embodied by Bayesian probability) automatically embodies Occam's razor, quantitatively.

It is indeed *more probable* that there's one box behind the tree, and we can compute how much more probable one is than two.

### 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  $\mathcal{H}_2$ :

$$\frac{P(\mathcal{H}_1 | D)}{P(\mathcal{H}_2 | D)} = \frac{P(\mathcal{H}_1) P(D | \mathcal{H}_1)}{P(\mathcal{H}_2) P(D | \mathcal{H}_2)}. \quad (28.1)$$

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 28.3). 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 the next two numbers, and infer the underlying process 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

Figure 28.3. 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  $C_1$  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  $C_1$ , the *less powerful* model  $\mathcal{H}_1$  will be the *more probable* model.

$-x^3/11 + 9/11x^2 + 23/11$ ? 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 (28.1). 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:

$$P(D|\mathcal{H}_a) = \frac{1}{101} \frac{1}{101} = 0.00010. \quad (28.2)$$

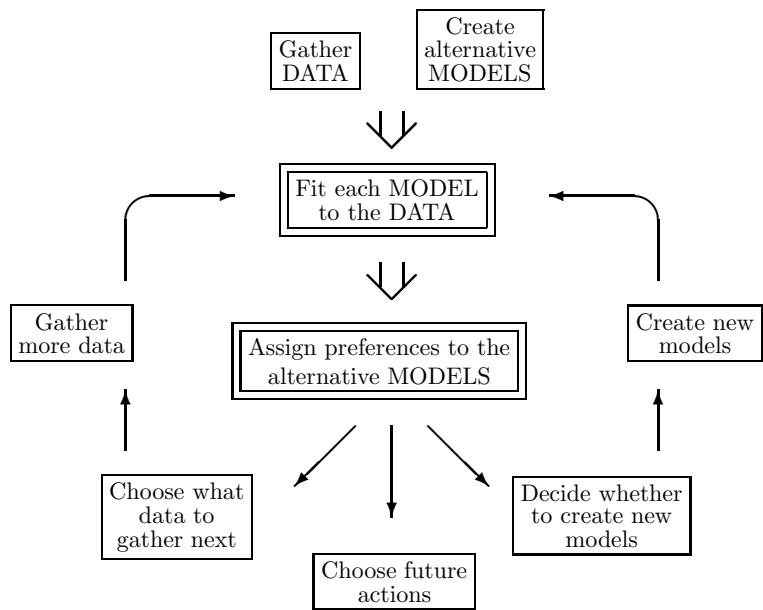
To evaluate  $P(D|\mathcal{H}_c)$ , we must similarly say what values the fractions  $c, d$  and  $e$  might take on. [I choose to represent these numbers as fractions rather than real numbers because if we used real numbers, the model would assign, relative to  $\mathcal{H}_a$ , an infinitesimal probability 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:

$$\begin{aligned} P(D|\mathcal{H}_c) &= \left(\frac{1}{101}\right) \left(\frac{4}{101} \frac{1}{50}\right) \left(\frac{4}{101} \frac{1}{50}\right) \left(\frac{2}{101} \frac{1}{50}\right) \\ &= 0.0000000000025 = 2.5 \times 10^{-12}. \end{aligned} \quad (28.3)$$

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.  $\square$

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, and so can predict a greater variety of data sets (figure 28.3). 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 with the sun at the centre. 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* on the basis of these probabilities so as to minimize the

Figure 28.4. Where Bayesian inference fits into the data modelling process. This figure illustrates an abstraction of the part of the scientific process in which data are 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' 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. Bayesian methods may be used to find the most probable parameter values, and error bars on those parameters. The result of applying Bayesian methods 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 Bayesian methods are in a class of their own. This second inference problem requires a quantitative Occam's razor to penalize over-complex models. Bayesian methods can assign objective preferences to the alternative models in a way that automatically embodies Occam's razor.

expectation of a 'loss function'. This chapter concerns inference alone and no loss functions are involved. When we discuss model comparison, this should not be construed as implying model *choice*. Ideal Bayesian predictions do not involve choice between models; rather, predictions are made by summing over all the alternative models, weighted by their probabilities.

Bayesian methods are able consistently and quantitatively to solve both the inference tasks. There is a popular myth that states that Bayesian methods differ from orthodox statistical methods only by the inclusion of subjective priors, which are difficult to assign, and which 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 chapter 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:

$$P(\mathbf{w} | D, \mathcal{H}_i) = \frac{P(D | \mathbf{w}, \mathcal{H}_i)P(\mathbf{w} | \mathcal{H}_i)}{P(D | \mathcal{H}_i)}, \quad (28.4)$$

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  $\mathbf{w}_{\text{MP}}$ ,  $\mathbf{A} = -\nabla\nabla \ln P(\mathbf{w} | D, \mathcal{H}_i)|_{\mathbf{w}_{\text{MP}}}$ , and Taylor-expanding the log posterior probability with  $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_{\text{MP}}$ :

$$P(\mathbf{w} | D, \mathcal{H}_i) \simeq P(\mathbf{w}_{\text{MP}} | D, \mathcal{H}_i) \exp \left( -1/2 \Delta\mathbf{w}^T \mathbf{A} \Delta\mathbf{w} \right), \quad (28.5)$$

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

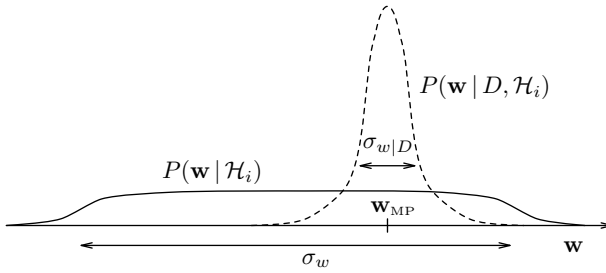


Figure 28.5. The Occam factor. This figure shows the quantities that determine the Occam factor for a hypothesis  $\mathcal{H}_i$  having a single parameter  $\mathbf{w}$ . The prior distribution (solid line) for the parameter has width  $\sigma_w$ . The posterior distribution (dashed line) has a single peak at  $\mathbf{w}_{\text{MP}}$  with characteristic width  $\sigma_{w|D}$ . The Occam factor is

$$\sigma_{w|D} P(\mathbf{w}_{\text{MP}} | \mathcal{H}_i) = \frac{\sigma_w | D}{\sigma_w}.$$

no fundamental status in Bayesian inference – they both change under nonlinear reparameterizations. Maximization of a posterior probability is useful only if an approximation like equation (28.5) gives a good summary of the distribution.]

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:

$$P(\mathcal{H}_i | D) \propto P(D | \mathcal{H}_i) P(\mathcal{H}_i). \quad (28.6)$$

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 (28.4). 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 (28.6) 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 repeat the key idea: 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 (28.4):

$$P(D | \mathcal{H}_i) = \int P(D | \mathbf{w}, \mathcal{H}_i) P(\mathbf{w} | \mathcal{H}_i) d\mathbf{w}. \quad (28.7)$$

For many problems the posterior  $P(\mathbf{w} | D, \mathcal{H}_i) \propto P(D | \mathbf{w}, \mathcal{H}_i) P(\mathbf{w} | \mathcal{H}_i)$  has a strong peak at the most probable parameters  $\mathbf{w}_{\text{MP}}$  (figure 28.5). 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,  $\sigma_{w|D}$ :

$$P(D | \mathcal{H}_i) \simeq \underbrace{P(D | \mathbf{w}_{\text{MP}}, \mathcal{H}_i)}_{\text{Best fit likelihood}} \times \underbrace{P(\mathbf{w}_{\text{MP}} | \mathcal{H}_i) \sigma_{w|D}}_{\text{Occam factor}}. \quad (28.8)$$

Evidence  $\simeq$  Best fit likelihood  $\times$  Occam factor

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 28.5). Then  $P(\mathbf{w}_{\text{MP}} | \mathcal{H}_i) = 1/\sigma_w$ , and

$$\text{Occam factor} = \frac{\sigma_{w|D}}{\sigma_w}, \quad (28.9)$$

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 stronger Occam factor than a simpler model. The Occam factor also penalizes models that 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 28.6 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  $P(\mathbf{w} | \mathcal{H}_3)$  (cf. figure 28.5). [In the case of model  $\mathcal{H}_1$  which is very poorly

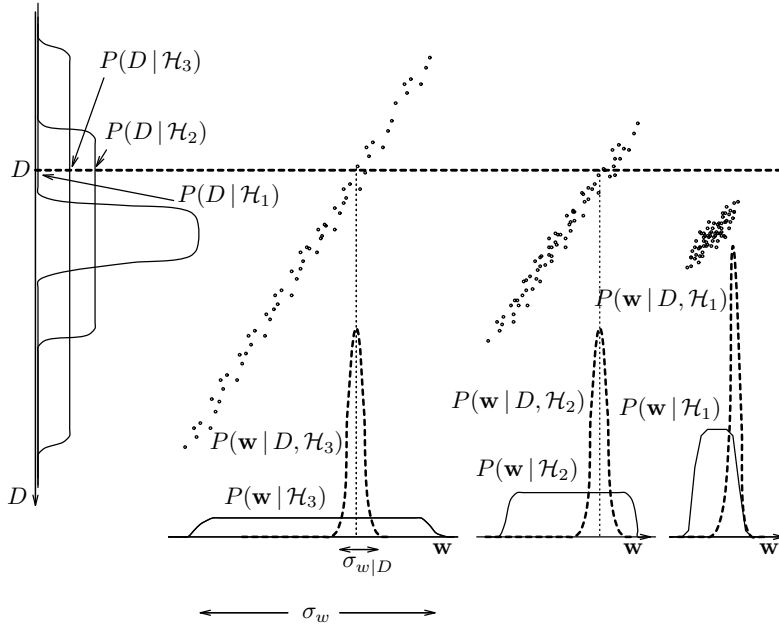


Figure 28.6. A hypothesis space consisting of three exclusive models, each having one parameter  $\mathbf{w}$ , and a one-dimensional data set  $D$ . The ‘data set’ is a single measured value which differs from the parameter  $\mathbf{w}$  by a small amount of additive noise. Typical samples from the joint distribution  $P(D, \mathbf{w}, \mathcal{H})$  are shown by dots. (N.B., 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 (cf. figure 28.3). The evidence for the different models is obtained by marginalizing onto the  $D$  axis at the left-hand side (cf. figure 28.5).

matched to the data, the shape of the posterior distribution will depend on 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 28.3 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  $\mathcal{H}_2$ . 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 (cf. equation (28.8) and Chapter 27):

$$P(D | \mathcal{H}_i) \simeq \underbrace{P(D | \mathbf{w}_{\text{MP}}, \mathcal{H}_i)}_{\text{Evidence}} \times \underbrace{P(\mathbf{w}_{\text{MP}} | \mathcal{H}_i) \det^{-\frac{1}{2}}(\mathbf{A}/2\pi)}_{\text{Occam factor}}, \quad (28.10)$$

where  $\mathbf{A} = -\nabla\nabla \ln P(\mathbf{w} | D, \mathcal{H}_i)$ , the Hessian which we evaluated when we calculated the error bars on  $\mathbf{w}_{\text{MP}}$  (equation 28.5 and Chapter 27). As the amount of data collected 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 computationally demanding than the task of finding for each model the best-fit parameters and their error bars.

## ► 28.2 Example

Let's return to the example that opened this chapter. Are there one or two boxes behind the tree in figure 28.1? Why do coincidences make us suspicious?

Let's assume the image of the area round the trunk and box has a size of 50 pixels, that the trunk is 10 pixels wide, and that 16 different colours of boxes can be distinguished. The theory  $\mathcal{H}_1$  that says there is one box near the trunk has four free parameters: three coordinates defining the top three edges of the box, and one parameter giving the box's colour. (If boxes could levitate, there would be five free parameters.)

The theory  $\mathcal{H}_2$  that says there are two boxes near the trunk has eight free parameters (twice four), plus a ninth, a binary variable that indicates which of the two boxes is the closest to the viewer.

What is the evidence for each model? We'll do  $\mathcal{H}_1$  first. We need a prior on the parameters to evaluate the evidence. For convenience, let's work in pixels. Let's assign a separable prior to the horizontal location of the box, its width, its height, and its colour. The height could have any of, say, 20 distinguishable values, so could the width, and so could the location. The colour could have any of 16 values. We'll put uniform priors over these variables. We'll ignore all the parameters associated with other objects in the image, since they don't come into the model comparison between  $\mathcal{H}_1$  and  $\mathcal{H}_2$ . The evidence is

$$P(D|\mathcal{H}_1) = \frac{1}{20} \frac{1}{20} \frac{1}{20} \frac{1}{16} \quad (28.11)$$

since only one setting of the parameters fits the data, and it predicts the data perfectly.

As for model  $\mathcal{H}_2$ , six of its nine parameters are well-determined, and three of them are partly-constrained by the data. If the left-hand box is furthest away, for example, then its width is at least 8 pixels and at most 30; if it's the closer of the two boxes, then its width is between 8 and 18 pixels. (I'm assuming here that the visible portion of the left-hand box is about 8 pixels wide.) To get the evidence we need to sum up the prior probabilities of all viable hypotheses. To do an exact calculation, we need to be more specific about the data and the priors, but let's just get the ballpark answer, assuming that the two unconstrained real variables have half their values available, and that the binary variable is completely undetermined. (As an exercise, you can make an explicit model and work out the exact answer.)

$$P(D|\mathcal{H}_2) \simeq \frac{1}{20} \frac{1}{20} \frac{10}{20} \frac{1}{16} \frac{1}{20} \frac{1}{20} \frac{10}{20} \frac{1}{16} \frac{2}{2}. \quad (28.12)$$

Thus the posterior probability ratio is (assuming equal prior probability):

$$\frac{P(D|\mathcal{H}_1)P(\mathcal{H}_1)}{P(D|\mathcal{H}_2)P(\mathcal{H}_2)} = \frac{1}{\frac{1}{20} \frac{10}{20} \frac{10}{20} \frac{1}{16}} \quad (28.13)$$

$$= 20 \times 2 \times 2 \times 16 \simeq 1000/1. \quad (28.14)$$

So the data are roughly 1000 to 1 in favour of the simpler hypothesis. The four factors in (28.13) can be interpreted in terms of Occam factors. The more complex model has four extra parameters for sizes and colours – three for sizes, and one for colour. It has to pay two big Occam factors ( $1/20$  and  $1/16$ ) for the highly suspicious coincidences that the two box heights match exactly and the two colours match exactly; and it also pays two lesser Occam factors for the two lesser coincidences that both boxes happened to have one of their edges conveniently hidden behind a tree or behind each other.

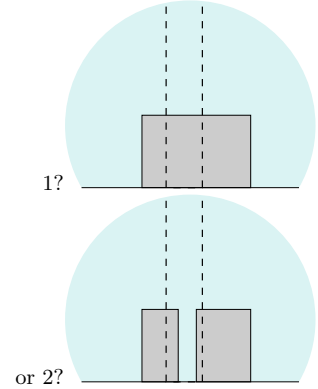


Figure 28.7. How many boxes are behind the tree?

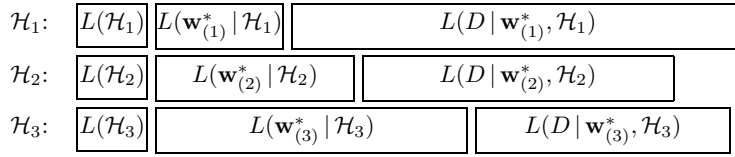


Figure 28.8. A popular view of model comparison by minimum description length. Each model  $\mathcal{H}_i$  communicates the data  $D$  by sending the identity of the model, sending the best-fit parameters of the model  $\mathbf{w}^*$ , then sending the data relative to those parameters. As we proceed to more complex models the length of the parameter message increases. On the other hand, the length of the data message decreases, because a complex model is able to fit the data better, making the residuals smaller. In this example the intermediate model  $\mathcal{H}_2$  achieves the optimum trade-off between these two trends.

### ► 28.3 Minimum description length (MDL)

A complementary view of Bayesian model comparison is obtained by replacing probabilities of events by the lengths in bits of messages that communicate the events without loss to a receiver. Message lengths  $L(\mathbf{x})$  correspond to a probabilistic model over events  $\mathbf{x}$  via the relations:

$$P(\mathbf{x}) = 2^{-L(\mathbf{x})}, \quad L(\mathbf{x}) = -\log_2 P(\mathbf{x}). \quad (28.15)$$

The MDL principle (Wallace and Boulton, 1968) states that one should prefer models that can communicate the data in the smallest number of bits. Consider a two-part message that states which model,  $\mathcal{H}$ , is to be used, and then communicates the data  $D$  within that model, to some pre-arranged precision  $\delta D$ . This produces a message of length  $L(D, \mathcal{H}) = L(\mathcal{H}) + L(D | \mathcal{H})$ . The lengths  $L(\mathcal{H})$  for different  $\mathcal{H}$  define an implicit prior  $P(\mathcal{H})$  over the alternative models. Similarly  $L(D | \mathcal{H})$  corresponds to a density  $P(D | \mathcal{H})$ . Thus, a procedure for assigning message lengths can be mapped onto posterior probabilities:

$$L(D, \mathcal{H}) = -\log P(\mathcal{H}) - \log (P(D | \mathcal{H}) \delta D) \quad (28.16)$$

$$= -\log P(\mathcal{H} | D) + \text{const.} \quad (28.17)$$

In principle, then, MDL can always be interpreted as Bayesian model comparison and *vice versa*. However, this simple discussion has not addressed how one would actually evaluate the key data-dependent term  $L(D | \mathcal{H})$ , which corresponds to the evidence for  $\mathcal{H}$ . Often, this message is imagined as being subdivided into a parameter block and a data block (figure 28.8). Models with a small number of parameters have only a short parameter block but do not fit the data well, and so the data message (a list of large residuals) is long. As the number of parameters increases, the parameter block lengthens, and the data message becomes shorter. There is an optimum model complexity ( $\mathcal{H}_2$  in the figure) for which the sum is minimized.

This picture glosses over some subtle issues. We have not specified the precision to which the parameters  $\mathbf{w}$  should be sent. This precision has an important effect (unlike the precision  $\delta D$  to which real-valued data  $D$  are sent, which, assuming  $\delta D$  is small relative to the noise level, just introduces an additive constant). As we decrease the precision to which  $\mathbf{w}$  is sent, the parameter message shortens, but the data message typically lengthens because the truncated parameters do not match the data so well. There is a non-trivial optimal precision. In simple Gaussian cases it is possible to solve for this optimal precision (Wallace and Freeman, 1987), and it is closely related to the posterior error bars on the parameters,  $\mathbf{A}^{-1}$ , where  $\mathbf{A} = -\nabla \nabla \ln P(\mathbf{w} | D, \mathcal{H})$ . It turns out that the optimal parameter message length is virtually identical to the log of the Occam factor in equation (28.10). (The random element involved in parameter truncation means that the encoding is slightly sub-optimal.)

With care, therefore, one can replicate Bayesian results in MDL terms. Although some of the earliest work on complex model comparison involved the MDL framework (Patrick and Wallace, 1982), MDL has no apparent advantages over the direct probabilistic approach.

MDL does have its uses as a pedagogical tool. The description length concept is useful for motivating prior probability distributions. Also, different ways of breaking down the task of communicating data using a model can give helpful insights into the modelling process, as will now be illustrated.

*On-line learning and cross-validation.*

In cases where the data consist of a sequence of points  $D = \mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \dots, \mathbf{t}^{(N)}$ , the log evidence can be decomposed as a sum of ‘on-line’ predictive performances:

$$\begin{aligned} \log P(D | \mathcal{H}) &= \log P(\mathbf{t}^{(1)} | \mathcal{H}) + \log P(\mathbf{t}^{(2)} | \mathbf{t}^{(1)}, \mathcal{H}) \\ &+ \log P(\mathbf{t}^{(3)} | \mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \mathcal{H}) + \dots + \log P(\mathbf{t}^{(N)} | \mathbf{t}^{(1)} \dots \mathbf{t}^{(N-1)}, \mathcal{H}). \end{aligned} \quad (28.18)$$

This decomposition can be used to explain the difference between the evidence and ‘leave-one-out cross-validation’ as measures of predictive ability. Cross-validation examines the average value of just the last term,  $\log P(\mathbf{t}^{(N)} | \mathbf{t}^{(1)} \dots \mathbf{t}^{(N-1)}, \mathcal{H})$ , under random re-orderings of the data. The evidence, on the other hand, sums up how well the model predicted all the data, starting from scratch.

*The ‘bits-back’ encoding method.*

Another MDL thought experiment (Hinton and van Camp, 1993) involves incorporating random bits into our message. The data are communicated using a parameter block and a data block. The parameter vector sent is a random sample from the posterior,  $P(\mathbf{w} | D, \mathcal{H}) = P(D | \mathbf{w}, \mathcal{H})P(\mathbf{w} | \mathcal{H})/P(D | \mathcal{H})$ . This sample  $\mathbf{w}$  is sent to an arbitrary small granularity  $\delta\mathbf{w}$  using a message length  $L(\mathbf{w} | \mathcal{H}) = -\log[P(\mathbf{w} | \mathcal{H})\delta\mathbf{w}]$ . The data are encoded relative to  $\mathbf{w}$  with a message of length  $L(D | \mathbf{w}, \mathcal{H}) = -\log[P(D | \mathbf{w}, \mathcal{H})\delta D]$ . Once the data message has been received, the random bits used to generate the sample  $\mathbf{w}$  from the posterior can be deduced by the receiver. The number of bits so recovered is  $-\log[P(\mathbf{w} | D, \mathcal{H})\delta\mathbf{w}]$ . These recovered bits need not count towards the message length, since we might use some other optimally-encoded message as a random bit string, thereby communicating that message at the same time. The net description cost is therefore:

$$\begin{aligned} L(\mathbf{w} | \mathcal{H}) + L(D | \mathbf{w}, \mathcal{H}) - \text{‘Bits back’} &= -\log \frac{P(\mathbf{w} | \mathcal{H}) P(D | \mathbf{w}, \mathcal{H}) \delta D}{P(\mathbf{w} | D, \mathcal{H})} \\ &= -\log P(D | \mathcal{H}) - \log \delta D. \end{aligned} \quad (28.19)$$

Thus this thought experiment has yielded the optimal description length. Bits-back encoding has been turned into a practical compression method for data modelled with latent variable models by Frey (1998).

## Further reading

Bayesian methods are introduced and contrasted with sampling-theory statistics in (Jaynes, 1983; Gull, 1988; Lored, 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).

One debate worth understanding is the question of whether it’s permissible to use improper priors in Bayesian inference (Dawid *et al.*, 1996). If we want to do model comparison (as discussed in this chapter), it is essential to use proper priors – otherwise the evidences and the Occam factors are



meaningless. Only when one has no intention to do model comparison may it be safe to use improper priors, and even in such cases there are pitfalls, as Dawid *et al.* explain. I would agree with their advice to *always use proper priors*, tempered by an encouragement to be smart when making calculations, recognizing opportunities for approximation.

► 28.4 Exercises

Exercise 28.1.<sup>[3]</sup> Random variables  $x$  come independently from a probability distribution  $P(x)$ . According to model  $\mathcal{H}_0$ ,  $P(x)$  is a uniform distribution

$$P(x|\mathcal{H}_0) = \frac{1}{2} \quad x \in (-1, 1). \tag{28.20}$$

According to model  $\mathcal{H}_1$ ,  $P(x)$  is a nonuniform distribution with an unknown parameter  $m \in (-1, 1)$ :

$$P(x|m, \mathcal{H}_1) = \frac{1}{2}(1 + mx) \quad x \in (-1, 1). \tag{28.21}$$

Given the data  $D = \{0.3, 0.5, 0.7, 0.8, 0.9\}$ , what is the evidence for  $\mathcal{H}_0$  and  $\mathcal{H}_1$ ?

Exercise 28.2.<sup>[3]</sup> Datapoints  $(x, t)$  are believed to come from a straight line. The experimenter chooses  $x$ , and  $t$  is Gaussian-distributed about

$$y = w_0 + w_1 x \tag{28.22}$$

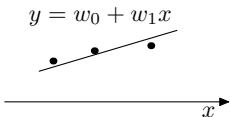
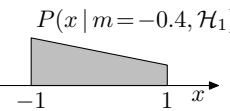
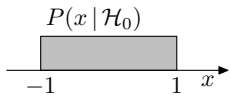
with variance  $\sigma_\nu^2$ . According to model  $\mathcal{H}_1$ , the straight line is horizontal, so  $w_1 = 0$ . According to model  $\mathcal{H}_2$ ,  $w_1$  is a parameter with prior distribution Normal(0, 1). Both models assign a prior distribution Normal(0, 1) to  $w_0$ . Given the data set  $D = \{(-8, 8), (-2, 10), (6, 11)\}$ , and assuming the noise level is  $\sigma_\nu = 1$ , what is the evidence for each model?

Exercise 28.3.<sup>[3]</sup> A six-sided die is rolled 30 times and the numbers of times each face came up were  $\mathbf{F} = \{3, 3, 2, 2, 9, 11\}$ . What is the probability that the die is a perfectly fair die ( $\mathcal{H}_0$ ), assuming the alternative hypothesis  $\mathcal{H}_1$  says that the die has a biased distribution  $\mathbf{p}$ , and the prior density for  $\mathbf{p}$  is uniform over the simplex  $p_i \geq 0, \sum_i p_i = 1$ ?

Solve this problem two ways: exactly, using the helpful Dirichlet formulae (23.30, 23.31), and approximately, using Laplace's method. Notice that your choice of basis for the Laplace approximation is important. See MacKay (1998a) for discussion of this exercise.

Exercise 28.4.<sup>[3]</sup> The influence of race on the imposition of the death penalty for murder in America has been much studied. The following three-way table classifies 326 cases in which the defendant was convicted of murder. The three variables are the defendant's race, the victim's race, and whether the defendant was sentenced to death. (Data from M. Radelet, 'Racial characteristics and imposition of the death penalty,' *American Sociological Review*, **46** (1981), pp.918-927.)

White defendant			Black defendant		
	Death penalty			Death penalty	
	Yes	No		Yes	No
White victim	19	132	White victim	11	52
Black victim	0	9	Black victim	6	97



It seems that the death penalty was applied much more often when the victim was white than when the victim was black. When the victim was white 14% of defendants got the death penalty, but when the victim was black 6% of defendants got the death penalty. [Incidentally, these data provide an example of a phenomenon known as *Simpson's paradox*: a higher fraction of white defendants are sentenced to death overall, but in cases involving black victims a higher fraction of black defendants are sentenced to death and in cases involving white victims a higher fraction of black defendants are sentenced to death.]

Quantify the evidence for the four alternative hypotheses shown in figure 28.9. I should mention that I don't believe any of these models is adequate: several additional variables are important in murder cases, such as whether the victim and murderer knew each other, whether the murder was premeditated, and whether the defendant had a prior criminal record; none of these variables is included in the table. So this is an academic exercise in model comparison rather than a serious study of racial bias in the state of Florida.

The hypotheses are shown as graphical models, with arrows showing dependencies between the variables  $v$  (victim race),  $m$  (murderer race), and  $d$  (whether death penalty given). Model  $\mathcal{H}_{00}$  has only one free parameter, the probability of receiving the death penalty; model  $\mathcal{H}_{11}$  has four such parameters, one for each state of the variables  $v$  and  $m$ . Assign uniform priors to these variables. How sensitive are the conclusions to the choice of prior?

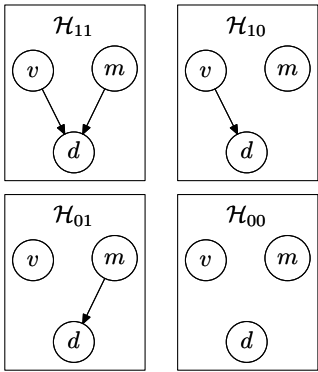


Figure 28.9. Four hypotheses concerning the dependence of the imposition of the death penalty  $d$  on the race of the victim  $v$  and the race of the convicted murderer  $m$ .  $\mathcal{H}_{01}$ , for example, asserts that the probability of receiving the death penalty does depend on the murderer's race, but not on the victim's.

---

## About Chapter 29

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 dependencies and independencies 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 discusses 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 (1993b), 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

$$\mathbf{u} = \mathbf{M}\mathbf{v} \quad (29.1)$$

to

$$\mathbf{u}^\top = \mathbf{v}^\top \mathbf{M}^\top. \quad (29.2)$$

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  $T(z; y)T(y; x)\pi(x)$ .

## 29

---

### Monte Carlo Methods

#### ► 29.1 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

$$\Phi = \langle \phi(\mathbf{x}) \rangle \equiv \int d^N \mathbf{x} P(\mathbf{x}) \phi(\mathbf{x}). \quad (29.3)$$

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 functions  $\phi(\mathbf{x})$  whose expectations we might be interested in include the first and second moments of quantities that we wish to predict, from which we can compute means and variances; 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  $\phi_2(\mathbf{x}) = (t(\mathbf{x}))^2$ ,

$$\Phi_1 \equiv \mathcal{E}[\phi_1(\mathbf{x})] \quad \text{and} \quad \Phi_2 \equiv \mathcal{E}[\phi_2(\mathbf{x})], \quad (29.4)$$

then using

$$\bar{t} = \Phi_1 \quad \text{and} \quad \text{var}(t) = \Phi_2 - \Phi_1^2. \quad (29.5)$$

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

$$\hat{\Phi} \equiv \frac{1}{R} \sum_r \phi(\mathbf{x}^{(r)}). \quad (29.6)$$

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  $\sigma^2/R$ , where  $\sigma^2$  is the variance of  $\phi$ ,

$$\sigma^2 = \int d^N \mathbf{x} P(\mathbf{x}) (\phi(\mathbf{x}) - \Phi)^2. \quad (29.7)$$

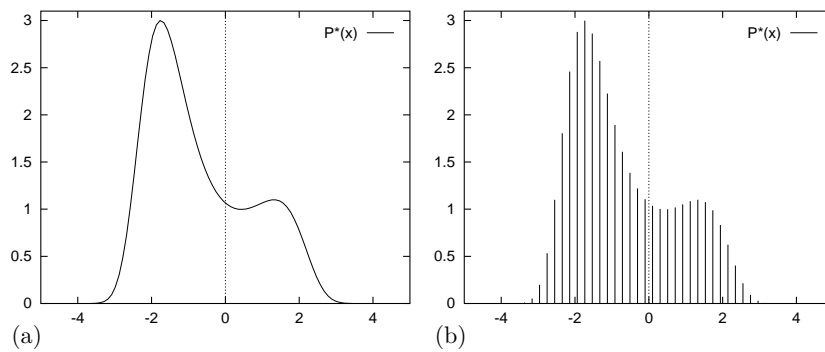


Figure 29.1. (a) The function  $P^*(x) = \exp[0.4(x - 0.4)^2 - 0.08x^4]$ . How to draw samples from this density? (b) The function  $P^*(x)$  evaluated at a discrete set of uniformly spaced points  $\{x_i\}$ . How to draw samples from this discrete distribution?

This is one of the important properties of Monte Carlo methods.

The accuracy of the Monte Carlo estimate (29.6) 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  $\sigma^2/R$ . 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

$$P(\mathbf{x}) = P^*(\mathbf{x})/Z. \quad (29.8)$$

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

$$Z = \int d^N \mathbf{x} P^*(\mathbf{x}). \quad (29.9)$$

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.

Let us start with a simple one-dimensional example. Imagine that we wish to draw samples from the density  $P(x) = P^*(x)/Z$  where

$$P^*(x) = \exp[0.4(x - 0.4)^2 - 0.08x^4], \quad x \in (-\infty, \infty). \quad (29.10)$$

We can plot this function (figure 29.1a). 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 29.1b). How could we solve this problem? If we evaluate  $p_i^* = P^*(x_i)$  at each point  $x_i$ , we can compute

$$Z = \sum_i p_i^* \quad (29.11)$$

and

$$p_i = p_i^*/Z \quad (29.12)$$

and we can then sample from the probability distribution  $\{p_i\}$  using various methods based on a source of random bits (see section 6.3). 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  $Z$  (29.11). To compute  $Z$  we have to visit every point in the space. In figure 29.1b 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 are 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

$$P^*(\mathbf{x}) = \exp[-\beta E(\mathbf{x})] \quad (29.13)$$

where  $x_n \in \{\pm 1\}$  and

$$E(\mathbf{x}) = - \left[ \frac{1}{2} \sum_{m,n} J_{mn} x_m x_n + \sum_n H_n x_n \right]. \quad (29.14)$$

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), as we'll describe in Chapter 32.

### A useful analogy

Imagine the tasks of drawing random water samples from a lake and finding the average plankton concentration (figure 29.2). The depth of the lake at  $\mathbf{x} = (x, y)$  is  $P^*(\mathbf{x})$ , and we assert (in order to make the analogy work) that the plankton concentration is a function of  $\mathbf{x}$ ,  $\phi(\mathbf{x})$ . The required average concentration is an integral like (29.3), namely

$$\Phi = \langle \phi(\mathbf{x}) \rangle \equiv \frac{1}{Z} \int d^N \mathbf{x} P^*(\mathbf{x}) \phi(\mathbf{x}), \quad (29.15)$$

\* Translation for American readers: 'such systems are a dime a dozen'; incidentally, this equivalence (10c = 6p) shows that the correct exchange rate between our currencies is £1.00 = \$1.67.

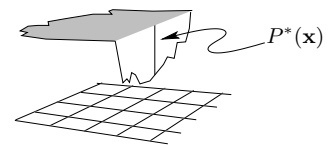


Figure 29.2. A lake whose depth at  $\mathbf{x} = (x, y)$  is  $P^*(\mathbf{x})$ .

where  $Z = \int dx dy P^*(\mathbf{x})$  is the volume of the lake. You are provided with a boat, a satellite navigation system, and a plumbline. Using the navigator, you can take your boat to any desired location  $\mathbf{x}$  on the map; using the plumbline you can measure  $P^*(\mathbf{x})$  at that point. You can also measure the plankton concentration there.

Problem 1 is to draw  $1 \text{ cm}^3$  water samples at random from the lake, in such a way that each sample is equally likely to come from any point within the lake. Problem 2 is to find the average plankton concentration.

These are difficult problems to solve because at the outset we know nothing about the depth  $P^*(\mathbf{x})$ . Perhaps much of the volume of the lake is contained in narrow, deep underwater canyons (figure 29.3), in which case, to correctly sample from the lake and correctly estimate  $\Phi$  our method must implicitly discover the canyons and find their volume relative to the rest of the lake. Difficult problems, yes; nevertheless, we'll see that clever Monte Carlo methods can solve them.

### Uniform sampling

Having accepted that we cannot exhaustively 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

$$Z_R = \sum_{r=1}^R P^*(\mathbf{x}^{(r)}), \quad (29.16)$$

and estimate  $\Phi = \int d^N \mathbf{x} \phi(\mathbf{x}) P(\mathbf{x})$  by

$$\hat{\Phi} = \sum_{r=1}^R \phi(\mathbf{x}^{(r)}) \frac{P^*(\mathbf{x}^{(r)})}{Z_R}. \quad (29.17)$$

Is anything wrong with this strategy? Well, it depends on the functions  $\phi(\mathbf{x})$  and  $P^*(\mathbf{x})$ . Let us assume that  $\phi(\mathbf{x})$  is a benign, smoothly varying function and concentrate on the nature of  $P^*(\mathbf{x})$ . As we learnt in Chapter 4, a high-dimensional distribution is often concentrated in a small region of the state space known as its typical set  $T$ , whose volume is given by  $|T| \simeq 2^{H(\mathbf{X})}$ , where  $H(\mathbf{X})$  is the entropy of the probability distribution  $P(\mathbf{x})$ . 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 4; 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.

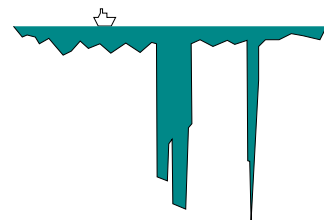


Figure 29.3. A slice through a lake that includes some canyons.

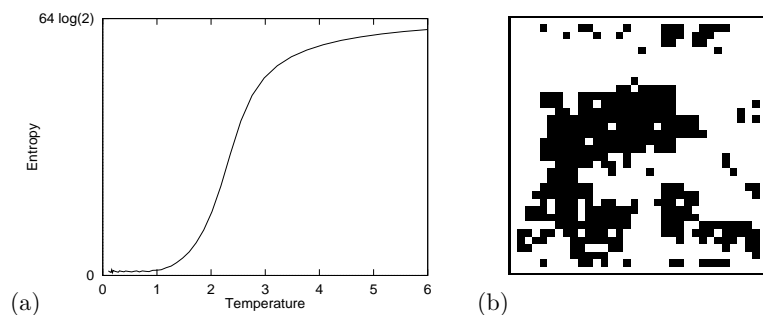


Figure 29.4. (a) Entropy of a 64-spin Ising model as a function of temperature. (b) One state of a 1024-spin Ising model.

The number of samples required to hit the typical set once is thus of order

$$R_{\min} \simeq 2^{N-H}. \quad (29.18)$$

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 29.4). For this probability distribution the number of samples required simply to hit the typical set once is of order

$$R_{\min} \simeq 2^{N-N/2} = 2^{N/2}, \quad (29.19)$$

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*.

## ► 29.2 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

$$P(x) = P^*(x)/Z. \quad (29.20)$$



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 29.5. 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 (29.6). 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*

$$w_r \equiv \frac{P^*(x^{(r)})}{Q^*(x^{(r)})} \quad (29.21)$$

which we use to adjust the ‘importance’ of each point in our estimator thus:

$$\hat{\Phi} \equiv \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r}. \quad (29.22)$$

- ▷ **Exercise 29.1.** [2, p.384] 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 (29.22). 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.

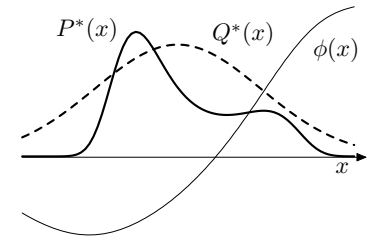
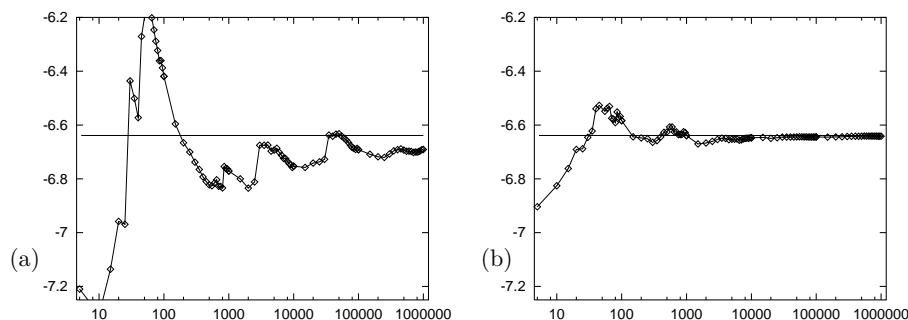


Figure 29.5. Functions involved in importance sampling. We wish to estimate the expectation of  $\phi(x)$  under  $P(x) \propto P^*(x)$ . We can generate samples from the simpler distribution  $Q(x) \propto Q^*(x)$ . We can evaluate  $Q^*$  and  $P^*$  at any point.

Figure 29.6. 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 29.6. 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 29.2.** [2, p.385] 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  $R$ .

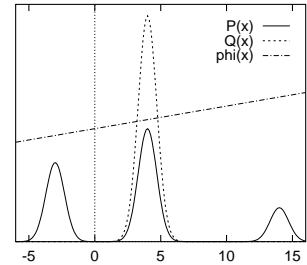


Figure 29.7. A multimodal distribution  $P^*(x)$  and a unimodal sampler  $Q(x)$ .

### 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,

$$P^*(\mathbf{x}) = \begin{cases} 1 & 0 \leq \rho(\mathbf{x}) \leq R_P \\ 0 & \rho(\mathbf{x}) > R_P, \end{cases} \quad (29.23)$$

where  $\rho(\mathbf{x}) \equiv (\sum_i x_i^2)^{1/2}$ , and the proposal density is a Gaussian centred on the origin,

$$Q(\mathbf{x}) = \prod_i \text{Normal}(x_i; 0, \sigma^2). \quad (29.24)$$

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 I – see exercise 6.14 (p.124), 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:

$$\rho^2 \sim N\sigma^2 \pm \sqrt{2N}\sigma^2. \quad (29.25)$$

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 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

$$\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N}{2} \pm \frac{\sqrt{2N}}{2}\right). \quad (29.26)$$

Thus the weights  $w_r = P^*/Q$  will typically have values in the range

$$(2\pi\sigma^2)^{N/2} \exp\left(\frac{N}{2} \pm \frac{\sqrt{2N}}{2}\right). \quad (29.27)$$

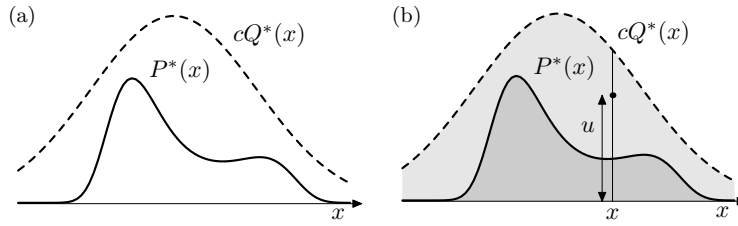


Figure 29.8. Rejection sampling. (a) The functions involved in rejection sampling. We desire samples from  $P(x) \propto P^*(x)$ . We are able to draw samples from  $Q(x) \propto Q^*(x)$ , and we know a value  $c$  such that  $cQ^*(x) > P^*(x)$  for all  $x$ . (b) A point  $(x, u)$  is generated at random in the lightly shaded area under the curve  $cQ^*(x)$ . If this point also lies below  $P^*(x)$  then it is accepted.

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 (29.27). The largest weight and the median weight will typically be in the ratio:

$$\frac{w_r^{\max}}{w_r^{\text{med}}} = \exp(\sqrt{2N}). \quad (29.28)$$

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 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  $\exp(\sqrt{N})$ , so the weights will too, unless  $Q$  is a near-perfect approximation to  $P$ .

### ► 29.3 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

$$cQ^*(x) > P^*(x), \text{ for all } x. \quad (29.29)$$

A schematic picture of the two functions is shown in figure 29.8a.

We generate two random numbers. The first,  $x$ , is generated from the proposal density  $Q(x)$ . We then evaluate  $cQ^*(x)$  and generate a uniformly distributed random variable  $u$  from the interval  $[0, cQ^*(x)]$ . These two random numbers can be viewed as selecting a point in the two-dimensional plane as shown in figure 29.8b.

We now evaluate  $P^*(x)$  and accept or reject the sample  $x$  by comparing the value of  $u$  with the value of  $P^*(x)$ . If  $u > P^*(x)$  then  $x$  is rejected; otherwise it is accepted, which means that we add  $x$  to our set of samples  $\{x^{(r)}\}$ . The value of  $u$  is discarded.

Why does this procedure generate samples from  $P(x)$ ? The proposed point  $(x, u)$  comes with uniform probability from the lightly shaded area underneath the curve  $cQ^*(x)$  as shown in figure 29.8b. The rejection rule rejects all the points that lie above the curve  $P^*(x)$ . So the points  $(x, u)$  that are accepted are uniformly distributed in the heavily shaded area under  $P^*(x)$ . This implies

that the probability density of the  $x$ -coordinates of the accepted points must be proportional to  $P^*(x)$ , so the samples must be independent samples from  $P(x)$ .

Rejection sampling will work best if  $Q$  is a good approximation to  $P$ . If  $Q$  is very different from  $P$  then, for  $cQ$  to exceed  $P$  everywhere,  $c$  will necessarily have to be large and the frequency of rejection will be large.

#### Rejection sampling in many dimensions

In a high-dimensional problem it is very likely that the requirement that  $cQ^*$  be an upper bound for  $P^*$  will force  $c$  to be so huge that acceptances will be very rare indeed. Finding such a value of  $c$  may be difficult too, since in many problems we know neither where the modes of  $P^*$  are located nor how high they are.

As a case study, consider a pair of  $N$ -dimensional Gaussian distributions with mean zero (figure 29.9). 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 1% 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  $cQ$  exceeds  $P$  for all  $\mathbf{x}$ .] So, what value of  $c$  is required if the dimensionality is  $N = 1000$ ? The density of  $Q(\mathbf{x})$  at the origin is  $1/(2\pi\sigma_Q^2)^{N/2}$ , so for  $cQ$  to exceed  $P$  we need to set

$$c = \frac{(2\pi\sigma_Q^2)^{N/2}}{(2\pi\sigma_P^2)^{N/2}} = \exp\left(N \ln \frac{\sigma_Q}{\sigma_P}\right). \quad (29.30)$$

With  $N = 1000$  and  $\frac{\sigma_Q}{\sigma_P} = 1.01$ , we find  $c = \exp(10) \simeq 20,000$ . What will the acceptance rate be for this value of  $c$ ? The answer is immediate: since the acceptance rate is the ratio of the volume under the curve  $P(\mathbf{x})$  to the volume under  $cQ(\mathbf{x})$ , the fact that  $P$  and  $Q$  are both normalized here implies that the acceptance rate will be  $1/c$ , for example,  $1/20,000$ . 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  $P(\mathbf{x})$ .

### ► 29.4 The Metropolis–Hastings method

Importance sampling and rejection sampling work well only 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–Hastings 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 29.10; this figure shows the density  $Q(x'; x^{(t)})$  for two different states  $x^{(1)}$  and  $x^{(2)}$ .

As before, we assume that we can evaluate  $P^*(x)$  for any  $x$ . A tentative new state  $x'$  is generated from the proposal density  $Q(x'; x^{(t)})$ . To decide

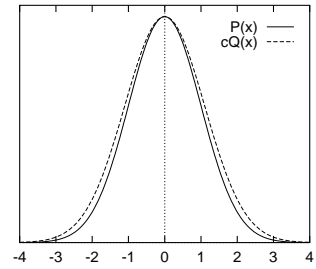


Figure 29.9. A Gaussian  $P(x)$  and a slightly broader Gaussian  $Q(x)$  scaled up by a factor  $c$  such that  $cQ(x) \geq P(x)$ .

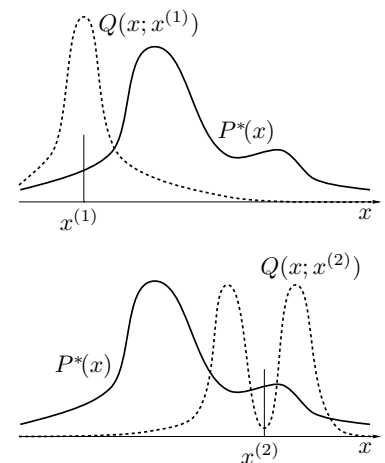


Figure 29.10. Metropolis–Hastings method in one dimension. The proposal distribution  $Q(x'; x)$  is here shown as having a shape that changes as  $x$  changes, though this is not typical of the proposal densities used in practice.

whether to accept the new state, we compute the quantity

$$a = \frac{P^*(x')}{P^*(x^{(t)})} \frac{Q(x^{(t)}; x')}{Q(x'; x^{(t)})}. \quad (29.31)$$

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 again onto the list.

**Notation.** I have used the superscript  $r = 1, \dots, R$  to label points that are *independent* samples from a distribution, and the superscript  $t = 1, \dots, T$  to label the sequence of states in a Markov chain. It is important to note that a Metropolis–Hastings simulation of  $T$  iterations does not produce  $T$  *independent* samples from the target distribution  $P$ . The samples are dependent.

To compute the acceptance probability (29.31) 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 method’ 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  $t \rightarrow \infty$ , the probability distribution of  $x^{(t)}$  tends to  $P(x) = P^*(x)/Z$ . [This statement should not be seen as implying that  $Q$  *has* to assign positive probability to every point  $x'$  – we will discuss examples later where  $Q(x'; x) = 0$  for some  $x, x'$ ; notice also that we have said nothing about how rapidly the convergence to  $P(x)$  takes place.]

The Metropolis method is an example of a *Markov chain Monte Carlo* method (abbreviated MCMC). In contrast to rejection sampling, where the accepted points  $\{x^{(r)}\}$  are *independent* samples from the desired distribution, Markov chain Monte Carlo methods involve a Markov process in which a sequence of states  $\{x^{(t)}\}$  is generated, each sample  $x^{(t)}$  having a probability distribution that depends on the previous value,  $x^{(t-1)}$ . Since successive samples are dependent, 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

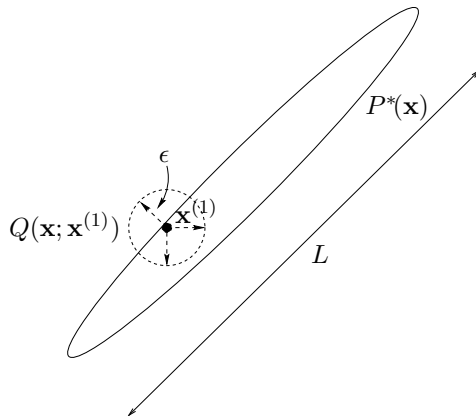


Figure 29.11. Metropolis method in two dimensions, showing a traditional proposal density that has a sufficiently small step size  $\epsilon$  that the acceptance frequency will be about 0.5.

with a length scale  $\epsilon$  that is short relative to the longest length scale  $L$  of the probable region (figure 29.11). 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 if large 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 29.3.**<sup>[1]</sup> 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 likely to have moved only 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

$$T \simeq (L/\epsilon)^2 \quad (29.32)$$

iterations to obtain an independent sample.

This rule of thumb gives only 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.

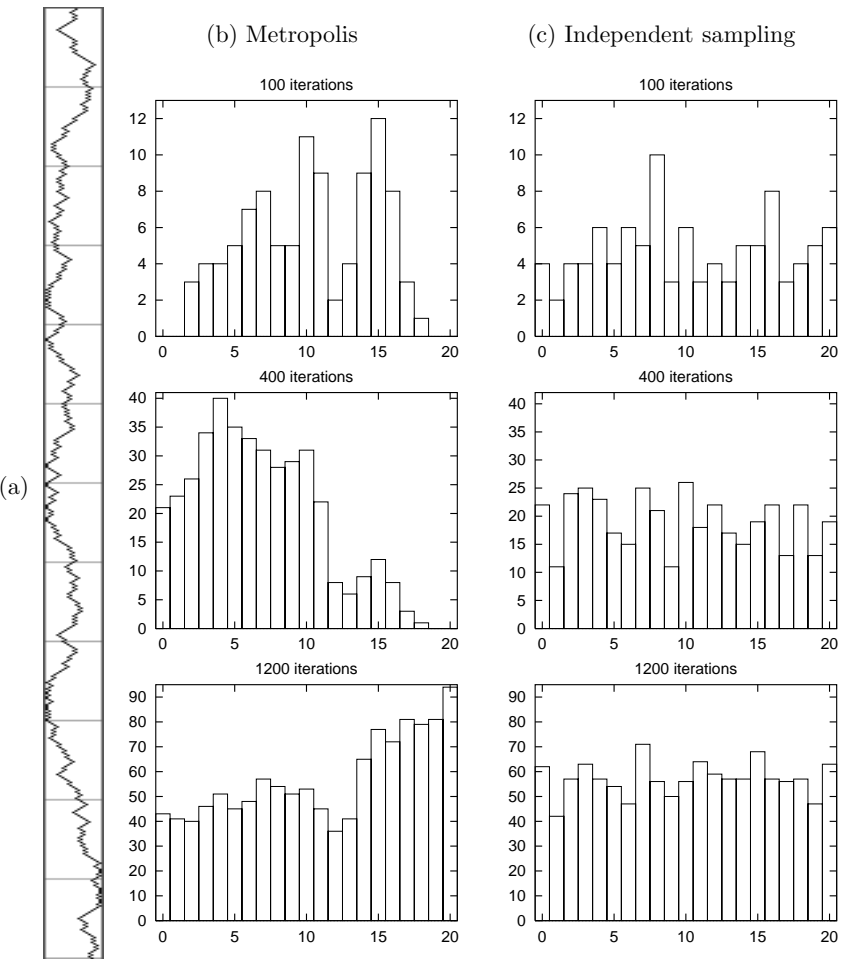


Figure 29.12. Metropolis method for a toy problem. (a) The state sequence for  $t = 1, \dots, 600$ . Horizontal direction = states from 0 to 20; vertical direction = time from 1 to 600; the cross bars mark time intervals of duration 50. (b) Histogram of occupancy of the states after 100, 400, and 1200 iterations. (c) For comparison, histograms resulting when successive points are drawn *independently* from the target distribution.

To illustrate how slowly a random walk explores a state space, figure 29.12 shows a simulation of a Metropolis algorithm for generating samples from the distribution:

$$P(x) = \begin{cases} 1/21 & x \in \{0, 1, 2, \dots, 20\} \\ 0 & \text{otherwise.} \end{cases} \quad (29.33)$$

The proposal distribution is

$$Q(x'; x) = \begin{cases} 1/2 & x' = x \pm 1 \\ 0 & \text{otherwise.} \end{cases} \quad (29.34)$$

Because the target distribution  $P(x)$  is uniform, rejections 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 29.12a. 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 (29.32) 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 generated only 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  $\{0, 1, 2, \dots, 20\}$  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 next chapter.

### *Metropolis method in high dimensions*

The rule of thumb (29.32), 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  $\epsilon$  in each direction. 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 frequency is close to 1. Under this assumption, each variable  $x_n$  evolves independently of all the others, executing a random walk with step size 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.



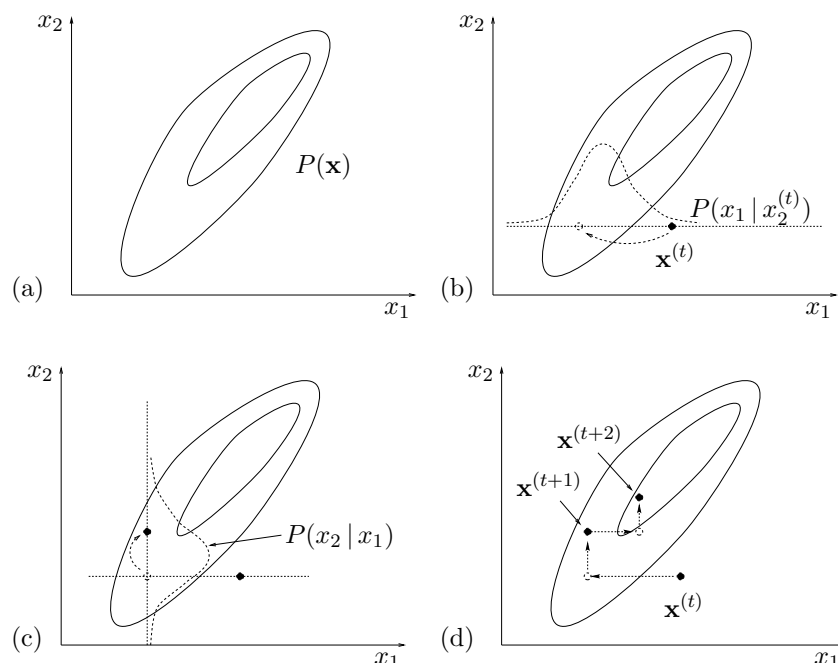


Figure 29.13. Gibbs sampling. (a) The joint density  $P(\mathbf{x})$  from which samples are required. (b) Starting from a state  $\mathbf{x}^{(t)}$ ,  $x_1$  is sampled from the conditional density  $P(x_1 | x_2^{(t)})$ . (c) A sample is then made from the conditional density  $P(x_2 | x_1)$ . (d) A couple of iterations of Gibbs sampling.

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 useful 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.

## ► 29.5 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* or ‘Glauber dynamics’, is a method for sampling from distributions over at least two dimensions. Gibbs sampling can be viewed as a Metropolis method in which a sequence of proposal distributions  $Q$  are defined in terms of the *conditional* distributions of the joint distribution  $P(\mathbf{x})$ . It is assumed that, whilst  $P(\mathbf{x})$  is too complex to draw samples from directly, its conditional distributions  $P(x_i | \{x_j\}_{j \neq i})$  are tractable to work with. For many graphical models (but not all) these one-dimensional conditional distributions are straightforward to sample from. For example, if a Gaussian distribution for some variables  $\mathbf{d}$  has an unknown mean  $\mathbf{m}$ , and the prior distribution of  $\mathbf{m}$  is Gaussian, then the conditional distribution of  $\mathbf{m}$  given  $\mathbf{d}$  is also Gaussian. Conditional distributions that are not of standard form may still be sampled from by *adaptive rejection sampling* if the conditional distribution satisfies certain convexity properties (Gilks and Wild, 1992).

Gibbs sampling is illustrated for a case with two variables  $(x_1, x_2) = \mathbf{x}$  in figure 29.13. 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:

$$x_1^{(t+1)} \sim P(x_1 | x_2^{(t)}, x_3^{(t)}, \dots, x_K^{(t)}) \quad (29.35)$$

$$x_2^{(t+1)} \sim P(x_2 | x_1^{(t+1)}, x_3^{(t)}, \dots, x_K^{(t)}) \quad (29.36)$$

$$x_3^{(t+1)} \sim P(x_3 | x_1^{(t+1)}, x_2^{(t+1)}, \dots, x_K^{(t)}), \text{ etc.} \quad (29.37)$$

### Convergence of Gibbs sampling to the target density

- ▷ Exercise 29.4.<sup>[2]</sup> Show that a single variable-update of 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  $t \rightarrow \infty$ , as long as  $P(\mathbf{x})$  does not have pathological properties.

- ▷ Exercise 29.5.<sup>[2, p.385]</sup> 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  $\mathbf{n} = (n_1, \dots, n_n, \dots, n_N)$ ,

$$P(\mathbf{n} | \mathbf{f}, \mathbf{z}) = \frac{1}{Z} \prod_{n=1}^N f_n^{n_n} (1 - f_n)^{(1-n_n)} \mathbb{1}[\mathbf{H}\mathbf{n} = \mathbf{z}], \quad (29.38)$$

where  $f_n$  is the normalized likelihood for the  $n$ th transmitted bit and  $\mathbf{z}$  is the observed syndrome. The factor  $\mathbb{1}[\mathbf{H}\mathbf{n} = \mathbf{z}]$  is 1 if  $\mathbf{n}$  has the correct syndrome  $\mathbf{z}$  and 0 otherwise.

What about the syndrome decoding problem for any linear error-correcting 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 that 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 30.3, p.390, 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).<sup>1</sup>

<sup>1</sup><http://www.mrc-bsu.cam.ac.uk/bugs/>



2. The chain must also be *ergodic*, that is,

$$p^{(t)}(\mathbf{x}) \rightarrow \pi(\mathbf{x}) \text{ as } t \rightarrow \infty, \text{ for any } p^{(0)}(\mathbf{x}). \quad (29.42)$$

A couple of reasons why a chain might not be ergodic are:

- (a) Its matrix might be *reducible*, which means that the state space contains two or more subsets of states that can never be reached from each other. Such a chain has many invariant distributions; which one  $p^{(t)}(\mathbf{x})$  would tend to as  $t \rightarrow \infty$  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* set, which means that, for some initial conditions,  $p^{(t)}(\mathbf{x})$  doesn't tend to an invariant distribution, but instead tends to a periodic limit-cycle.

A simple Markov chain with this property is the random walk on the  $N$ -dimensional hypercube. The chain  $T$  takes the state from one corner to a randomly chosen adjacent corner. The unique invariant distribution of this chain is the uniform distribution over all  $2^N$  states, but the chain is not ergodic; it is periodic with period two: if we divide the states into states with 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  $-1$ .

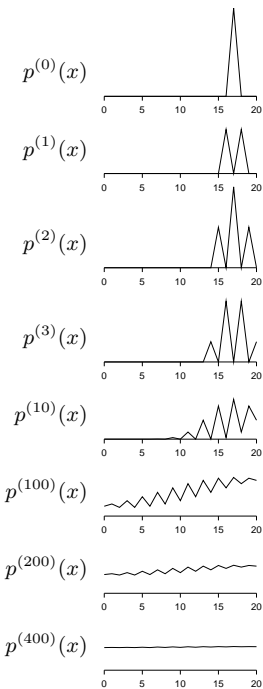


Figure 29.15. The probability distribution of the state of the Markov chain for initial condition  $x_0 = 17$  (example 29.6 (p.372)).

### 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

$$P(\mathbf{x}') = \int d^N \mathbf{x} B(\mathbf{x}'; \mathbf{x}) P(\mathbf{x}), \quad (29.43)$$

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.,

$$T(\mathbf{x}', \mathbf{x}) = \sum_b p_b B_b(\mathbf{x}', \mathbf{x}), \quad (29.44)$$

where  $\{p_b\}$  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  $\mathbf{x}''$  to  $\mathbf{x}'$  using  $B_2$ .

$$T(\mathbf{x}', \mathbf{x}) = \int d^N \mathbf{x}'' B_2(\mathbf{x}', \mathbf{x}'') B_1(\mathbf{x}'', \mathbf{x}). \quad (29.45)$$

### Detailed balance

Many useful transition probabilities satisfy the *detailed balance* property:

$$T(\mathbf{x}_a; \mathbf{x}_b)P(\mathbf{x}_b) = T(\mathbf{x}_b; \mathbf{x}_a)P(\mathbf{x}_a), \text{ for all } \mathbf{x}_b \text{ and } \mathbf{x}_a. \quad (29.46)$$

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 29.7.<sup>[2]</sup> 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 satisfies 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 29.8.<sup>[2]</sup> Show that, if we concatenate two base transitions  $B_1$  and  $B_2$  that satisfy detailed balance, it is not necessarily the case that the  $T$  thus defined (29.45) satisfies detailed balance.

Exercise 29.9.<sup>[2]</sup> Does Gibbs sampling, with several variables all updated in a deterministic sequence, satisfy detailed balance?

## ► 29.7 Slice sampling

Slice sampling (Neal, 1997a; Neal, 2003) is a Markov chain Monte Carlo method that 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. The simplest version of slice sampling is similar to Gibbs sampling in that it consists of one-dimensional transitions 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 29.3).

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  $u' \sim \text{Uniform}(0, P^*(x))$ 
3: create a horizontal interval  $(x_l, x_r)$  enclosing  $x$ 
4: loop {
5:     draw  $x' \sim \text{Uniform}(x_l, x_r)$ 
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 the overall method 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 step out in steps of length  $w$  until we find endpoints  $x_l$  and  $x_r$  at which  $P^*$  is smaller than  $u$ . The algorithm is shown in figure 29.16.

```

3a: draw  $r \sim \text{Uniform}(0, 1)$ 
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 in the Metropolis method, the choice of the step size is

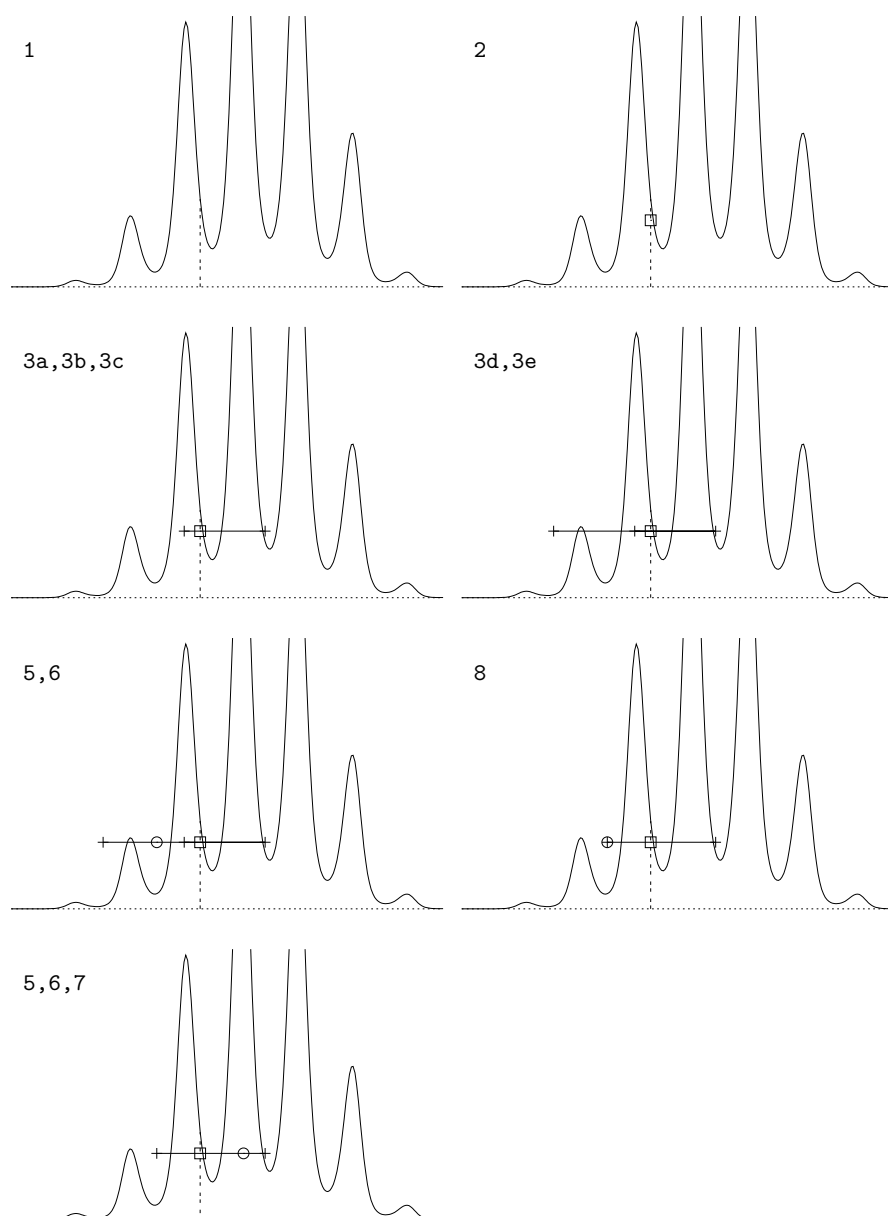


Figure 29.16. Slice sampling. Each panel is labelled by the steps of the algorithm that are executed in it. At step 1,  $P^*(x)$  is evaluated at the current point  $x$ . At step 2, a vertical coordinate is selected giving the point  $(x, u')$  shown by the box; At steps 3a–c, an interval of size  $w$  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 larger than  $u'$ , so a step to the left of size  $w$  is made. At step 3e,  $P^*$  is evaluated at the right end of the interval and is found to be smaller than  $u'$ , so no stepping out to the right is needed. When step 3d is repeated,  $P^*$  is found to be smaller than  $u'$ , so the stepping out halts. At step 5 a point is drawn from the interval, shown by a  $\circ$ . 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.

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, since the interval typically shrinks by a factor in the ballpark of 0.6 each time a point is rejected. 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.

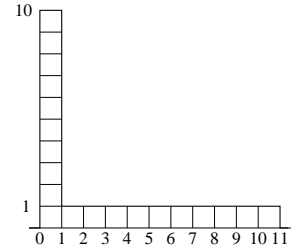


Figure 29.17.  $P^*(x)$ .

- ▷ Exercise 29.10.<sup>[2]</sup> Investigate the properties of slice sampling applied to the density shown in figure 29.17.  $x$  is a real variable between 0.0 and 11.0. 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 probabilities of these transitions do yield an asymptotic probability density that is correct.

### How slice sampling is used in real problems

An  $N$ -dimensional density  $P(\mathbf{x}) \propto P^*(\mathbf{x})$  may be sampled with the help of the one-dimensional slice sampling method presented above by picking a sequence of directions  $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots$  and defining  $\mathbf{x} = \mathbf{x}^{(t)} + x\mathbf{y}^{(t)}$ . The function  $P^*(x)$  above is replaced by  $P^*(\mathbf{x}) = P^*(\mathbf{x}^{(t)} + x\mathbf{y}^{(t)})$ . The directions may be chosen in various ways; for example, as in Gibbs sampling, the directions could be the coordinate axes; alternatively, the directions  $\mathbf{y}^{(t)}$  may be selected at random in any manner such that the overall procedure satisfies detailed balance.

### 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, 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,  $0, 1, 2, \dots, B-1$ , many or all of which correspond to valid values of  $x$ . Using an integer grid eliminates any errors in detailed balance that might 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,  $P^{**}(X) \propto P^*(x)|dx/dX|$ .

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 $N$ .
$N := \text{randbits}(l)$	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: $U := \text{randbits}(b)$	Define a random translation $U$ of the binary coordinate system.
2: set $l$ to a value $l \leq b$	Set initial $l$ -bit sampling range.
3: do {	
4: $N := \text{randbits}(l)$	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 011111111 and 100000000 would otherwise be permanently in different sectors, making it difficult for  $X$  to move from one to the other.

The sequence of intervals from which the new candidate points are drawn is illustrated in figure 29.18. 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: $N := \text{randbits}(l)$	
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.

**Exercise 29.11.**<sup>[4]</sup> In the shrinking phase, after an unacceptable  $X'$  has been produced, the choice of  $\Delta l$  is allowed to depend on the difference between the slice's height  $Y$  and the value of  $P^*(X')$ , without spoiling the algorithm's validity. (Prove this.) It might be a good idea to choose a larger value of  $\Delta l$  when  $Y - P^*(X')$  is large. Investigate this idea theoretically or empirically.

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.

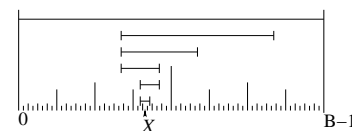


Figure 29.18. The sequence of intervals from which the new candidate points are drawn.

## ► 29.8 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. The exact sampling methods of Chapter 32 offer a solution to this problem for certain Markov chains.

**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 Hamiltonian Monte Carlo method, overrelaxation, and simulated annealing.

## ► 29.9 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 (1993b)) and annealed importance sampling (Neal, 1998).
2. ‘Thermodynamic integration’ during simulated annealing, the ‘acceptance ratio’ method, and ‘umbrella sampling’ (reviewed by Neal (1993b)).
3. ‘Reversible jump Markov chain Monte Carlo’ (Green, 1995).

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 hyperparameters which play a role similar to that played by the distinct models (Neal, 1996). In noting the possibility of not computing  $Z$ , I am not endorsing this approach. The normalizing constant  $Z$  is often the single most important number in the problem, and I think every effort should be devoted to calculating it.

*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 29.12. [2, p.385] Explain why the rate of movement through the state space will be greater when  $B$  proposals  $Q^{(1)}, \dots, Q^{(B)}$  are considered *individually* in sequence, compared with the case of a single proposal  $Q^*$  defined by the concatenation of  $Q^{(1)}, \dots, Q^{(B)}$ . 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 that 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 (29.35–29.37). For big problems it may be more efficient to sample *groups* of variables jointly, that is to use several proposal distributions:

$$\begin{aligned} x_1^{(t+1)}, \dots, x_a^{(t+1)} &\sim P(x_1, \dots, x_a | x_{a+1}^{(t)}, \dots, x_K^{(t)}) \\ x_{a+1}^{(t+1)}, \dots, x_b^{(t+1)} &\sim P(x_{a+1}, \dots, x_b | x_1^{(t+1)}, \dots, x_a^{(t+1)}, x_{b+1}^{(t)}, \dots, x_K^{(t)}), \text{ etc.} \end{aligned} \quad (29.47)$$

### *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

$$\sigma^2 = \int d^N \mathbf{x} P(\mathbf{x}) (\phi(\mathbf{x}) - \Phi)^2. \quad (29.48)$$

We have now discussed a variety of methods for generating samples from  $P(\mathbf{x})$ . How many independent samples  $R$  should we aim for?

In many problems, we really only need about twelve independent samples from  $P(\mathbf{x})$ . Imagine that  $\mathbf{x}$  is an unknown vector such as the amount of corrosion present in each of 10 000 underground pipelines around Cambridge, and  $\phi(\mathbf{x})$  is the total cost of repairing those pipelines. The distribution  $P(\mathbf{x})$  describes the probability of a state  $\mathbf{x}$  given the tests that have been carried out on some pipelines and the assumptions about the physics of corrosion. The quantity  $\Phi$  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.

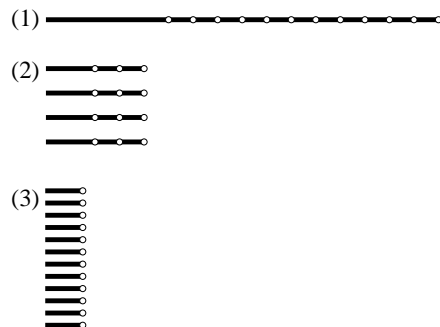


Figure 29.19. Three possible Markov chain Monte Carlo strategies for obtaining twelve samples in a fixed amount of computer time. Time is represented by horizontal lines; samples by white circles. (1) A single run consisting of one long ‘burn in’ period followed by a sampling period. (2) Four medium-length runs with different initial conditions and a medium-length burn in period. (3) Twelve short runs.

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 29.19):

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.

Finally, I should emphasize that there is no need to make the points in the estimate nearly-independent. Averaging over dependent points is fine – it won’t lead to any bias in the estimates. For example, when you use strategy 1 or 2, you may, if you wish, include all the points between the first and last sample in each run. Of course, estimating the accuracy of the estimate is harder when the points are dependent.

## ► 29.10 Summary

- Monte Carlo methods are a powerful tool that allow one to sample from 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

$$\int \phi(\mathbf{x}) P(\mathbf{x}) \simeq \frac{1}{R} \sum_r \phi(\mathbf{x}^{(r)}). \quad (29.49)$$

- In high-dimensional problems the only satisfactory methods are those based on Markov chains, such as 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.

## ► 29.11 Exercises



**Exercise 29.13.** [2C, p.386] A study of importance sampling. We already established in section 29.2 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, even with friendly Gaussian distributions.

Imagine that we want to know the expectation of a function  $\phi(x)$  under a distribution  $P(x)$ ,

$$\Phi = \int dx P(x) \phi(x), \quad (29.50)$$

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

$$Z = \int dx P^*(x) = \int dx Q(x) \frac{P^*(x)}{Q(x)} = \left\langle \frac{P^*(x)}{Q(x)} \right\rangle_{x \sim Q}. \quad (29.51)$$

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  $\sigma_q^2 \rightarrow \sigma_p^2/2$ ?

Check your theory by simulating this importance-sampling problem on a computer.



**Exercise 29.14.** [2] Consider the Metropolis algorithm for the one-dimensional toy problem of section 29.4, sampling from  $\{0, 1, \dots, 20\}$ . Whenever the current state is one of the end states, the proposal density given in equation (29.34) 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 29.15.<sup>[3C]</sup> Implement Gibbs sampling for the inference of a single one-dimensional Gaussian, which we studied using maximum likelihood in section 22.1. Assign a broad Gaussian prior to  $\mu$  and a broad gamma prior (24.2) to the precision parameter  $\beta = 1/\sigma^2$ . Each update of  $\mu$  will involve a sample from a Gaussian distribution, and each update of  $\sigma$  requires a sample from a gamma distribution.



Exercise 29.16.<sup>[3C]</sup> Gibbs sampling for clustering. Implement Gibbs sampling for the inference of a mixture of  $K$  one-dimensional Gaussians, which we studied using maximum likelihood in section 22.2. Allow the clusters to have different standard deviations  $\sigma_k$ . Assign priors to the means and standard deviations in the same way as the previous exercise. Either fix the prior probabilities of the classes  $\{\pi_k\}$  to be equal or put a uniform prior over the parameters  $\pi$  and include them in the Gibbs sampling.

Notice the similarity of Gibbs sampling to the soft K-means clustering algorithm (algorithm 22.2). We can alternately *assign* the class labels  $\{k_n\}$  given the parameters  $\{\mu_k, \sigma_k\}$ , then *update* the parameters given the class labels. The assignment step involves sampling from the probability distributions defined by the responsibilities (22.22), and the update step updates the means and variances using probability distributions centred on the K-means algorithm's values (22.23, 22.24).

Do your experiments confirm that Monte Carlo methods bypass the overfitting difficulties of maximum likelihood discussed in section 22.4?

A solution to this exercise and the previous one, written in `octave`, is available.<sup>2</sup>

- ▷ Exercise 29.17.<sup>[3C]</sup> Implement Gibbs sampling for the seven scientists inference problem, which we encountered in exercise 22.15 (p.309), and which you may have solved by exact marginalization (exercise 24.3 (p.323)) [it's not essential to have done the latter].
- ▷ Exercise 29.18.<sup>[2]</sup> A Metropolis method is used to explore a distribution  $P(\mathbf{x})$  that is actually a 1000-dimensional spherical Gaussian distribution of standard deviation 1 in all dimensions. The proposal density  $Q$  is a 1000-dimensional spherical Gaussian distribution of standard deviation  $\epsilon$ . Roughly what is the step size  $\epsilon$  if the acceptance rate is 0.5? Assuming this value of  $\epsilon$ ,
- roughly how long would the method take to traverse the distribution and generate a sample independent of the initial condition?
  - By how much does  $\ln P(\mathbf{x})$  change in a typical step? By how much should  $\ln P(\mathbf{x})$  vary when  $\mathbf{x}$  is drawn from  $P(\mathbf{x})$ ?
  - What happens if, rather than using a Metropolis method that tries to change all components at once, one instead uses a concatenation of Metropolis updates changing one component at a time?

<sup>2</sup><http://www.inference.phy.cam.ac.uk/mackay/itila/>

- ▷ Exercise 29.19.<sup>[2]</sup> 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  $\ln P^*(\mathbf{x})$ . 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  $-\ln P(\mathbf{x})$  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 that satisfies  $Q(\mathbf{x}; \mathbf{x}') = Q(\mathbf{x}'; \mathbf{x})$ . Assuming that accepted jumps either increase  $\ln P^*(\mathbf{x})$  by some amount or decrease it by a *small* amount, e.g.  $\ln e = 1$  (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.

Exercise 29.20.<sup>[3]</sup> Markov chain Monte Carlo methods do not compute partition functions  $Z$ , yet they allow ratios of quantities like  $Z$  to be estimated. For example, consider a random-walk Metropolis algorithm in a state space where the energy is zero in a connected accessible region, and infinitely large everywhere else; and imagine that the accessible space can be chopped into two regions connected by one or more corridor states. The fraction of times spent in each region at equilibrium is proportional to the volume of the region. How does the Monte Carlo method manage to do this without measuring the volumes?

Exercise 29.21.<sup>[5]</sup> Philosophy.

One curious defect of these Monte Carlo methods – which are widely used by Bayesian statisticians – is that they are all non-Bayesian (O’Hagan, 1987). They involve computer experiments from which *estimators* of quantities of interest are derived. These estimators depend on the proposal distributions that were used to generate the samples and on the random numbers that happened to come out of our random number generator. In contrast, an alternative Bayesian approach to the problem would use the results of our computer experiments to infer the properties of the target function  $P(\mathbf{x})$  and generate predictive distributions for quantities of interest such as  $\Phi$ . This approach would give answers that would depend only on the computed values of  $P^*(\mathbf{x}^{(r)})$  at the points  $\{\mathbf{x}^{(r)}\}$ ; the answers would not depend on how those points were chosen. Can you make a Bayesian Monte Carlo method? (See Rasmussen and Ghahramani (2003) for a practical attempt.)

## ► 29.12 Solutions

Solution to exercise 29.1 (p.362). We wish to show that

$$\hat{\Phi} \equiv \frac{\sum_r w_r \phi(x^{(r)})}{\sum_r w_r} \quad (29.52)$$

converges to the expectation of  $\Phi$  under  $P$ . We consider the numerator and the denominator separately. First, the denominator. Consider a single importance weight

$$w_r \equiv \frac{P^*(x^{(r)})}{Q^*(x^{(r)})}. \quad (29.53)$$

What is its expectation, averaged under the distribution  $Q = Q^*/Z_Q$  of the point  $x^{(r)}$ ?

$$\langle w_r \rangle = \int dx Q(x) \frac{P^*(x)}{Q^*(x)} = \int dx \frac{1}{Z_Q} P^*(x) = \frac{Z_P}{Z_Q}. \quad (29.54)$$

So the expectation of the denominator is

$$\left\langle \sum_r w_r \right\rangle = R \frac{Z_P}{Z_Q}. \quad (29.55)$$

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. [In fact, the estimate converges to the right answer even if this variance is infinite, as long as the expectation is well-defined.] Similarly, the expectation of one term in the numerator is

$$\langle w_r \phi(x) \rangle = \int dx Q(x) \frac{P^*(x)}{Q^*(x)} \phi(x) = \int dx \frac{1}{Z_Q} P^*(x) \phi(x) = \frac{Z_P}{Z_Q} \Phi, \quad (29.56)$$

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  $\hat{\Phi}$  converges 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  $R$ .

**Solution to exercise 29.2 (p.363).** 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 a 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 if no points in the other modes have been seen.

**Solution to exercise 29.5 (p.371).** The posterior distribution for the syndrome decoding problem is a pathological distribution from the point of view of Gibbs sampling. The factor  $\mathbb{1}[\mathbf{H}\mathbf{n} = \mathbf{z}]$  is 1 only 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.

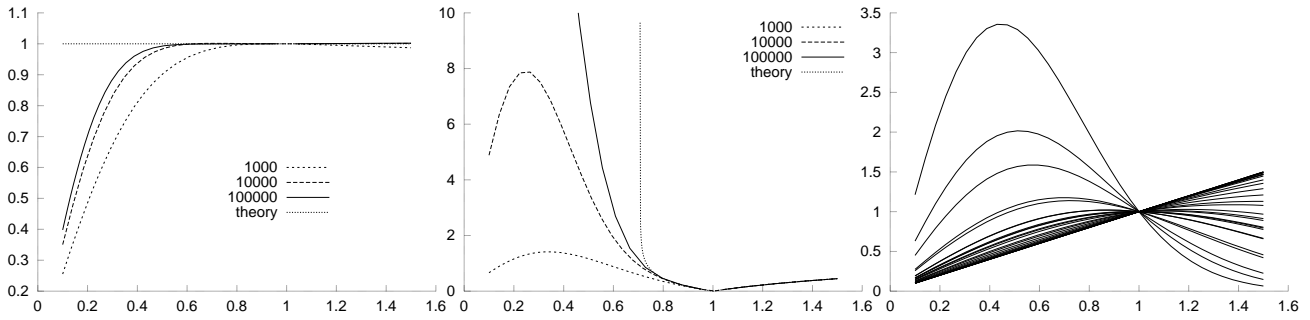
A general code has exactly the same problem. The points corresponding to valid codewords are relatively few in number and they are not adjacent (at least for any useful code). So Gibbs sampling is no use for syndrome decoding for two reasons. First, finding *any* reasonably good hypothesis is difficult, 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.

One could attempt to perform Gibbs sampling using the bits of the original message  $\mathbf{s}$  as the variables. This approach would not get locked up in the way just described, but, for a good code, any single bit flip would substantially alter the reconstructed codeword, so if one had found a state with reasonably large likelihood, Gibbs sampling would take an impractically large time to escape from it.

**Solution to exercise 29.12 (p.380).** 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  $f B \epsilon^2$ .



**Solution to exercise 29.13 (p.382).** The weights are  $w = P(x)/Q(x)$  and  $x$  is drawn from  $Q$ . The mean weight is

$$\int dx Q(x) [P(x)/Q(x)] = \int dx P(x) = 1, \quad (29.57)$$

assuming the integral converges. The variance is

$$\text{var}(w) = \int dx Q(x) \left[ \frac{P(x)}{Q(x)} - 1 \right]^2 \quad (29.58)$$

$$= \int dx \frac{P(x)^2}{Q(x)} - 2P(x) + Q(x) \quad (29.59)$$

$$= \left[ \int dx \frac{Z_Q}{Z_P^2} \exp \left( -\frac{x^2}{2} \left( \frac{2}{\sigma_p^2} - \frac{1}{\sigma_q^2} \right) \right) \right] - 1, \quad (29.60)$$

where  $Z_Q/Z_P^2 = \sigma_q/(\sqrt{2\pi}\sigma_p^2)$ . The integral in (29.60) is finite only if the coefficient of  $x^2$  in the exponent is positive, i.e., if

$$\sigma_q^2 > \frac{1}{2}\sigma_p^2. \quad (29.61)$$

If this condition is satisfied, the variance is

$$\text{var}(w) = \frac{\sigma_q}{\sqrt{2\pi}\sigma_p^2} \sqrt{2\pi} \left( \frac{2}{\sigma_p^2} - \frac{1}{\sigma_q^2} \right)^{-\frac{1}{2}} - 1 = \frac{\sigma_q^2}{\sigma_p (2\sigma_q^2 - \sigma_p^2)^{1/2}} - 1. \quad (29.62)$$

As  $\sigma_q$  approaches the critical value – about  $0.7\sigma_p$  – the variance becomes infinite. Figure 29.20 illustrates these phenomena for  $\sigma_p = 1$  with  $\sigma_q$  varying from 0.1 to 1.5. *The same random number seed was used for all runs*, so the weights and estimates follow smooth curves. 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 nevertheless infinite.

**Figure 29.20.** 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.

## 30

---

### *Efficient Monte Carlo Methods*

This chapter discusses several methods for reducing random walk behaviour in Metropolis methods. The aim is to reduce the time required to obtain effectively independent samples. For brevity, we will say ‘independent samples’ when we mean ‘effectively independent samples’.

#### ► 30.1 Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo method is a Metropolis method, applicable to continuous state spaces, that makes use of gradient information to reduce random walk behaviour. [The Hamiltonian Monte Carlo method was originally called hybrid Monte Carlo, for historical reasons.]

For many systems whose probability  $P(\mathbf{x})$  can be written in the form

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z}, \quad (30.1)$$

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 that have higher probability!

#### *Overview of Hamiltonian Monte Carlo*

In the Hamiltonian 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

$$H(\mathbf{x}, \mathbf{p}) = E(\mathbf{x}) + K(\mathbf{p}), \quad (30.2)$$

where  $K(\mathbf{p})$  is a ‘kinetic energy’ such as  $K(\mathbf{p}) = \mathbf{p}^T \mathbf{p} / 2$ . These two proposals are used to create (asymptotically) samples from the joint density

$$P_H(\mathbf{x}, \mathbf{p}) = \frac{1}{Z_H} \exp[-H(\mathbf{x}, \mathbf{p})] = \frac{1}{Z_H} \exp[-E(\mathbf{x})] \exp[-K(\mathbf{p})]. \quad (30.3)$$

This density is separable, so the marginal distribution of  $\mathbf{x}$  is the desired distribution  $\exp[-E(\mathbf{x})]/Z$ . So, simply discarding the momentum variables, we obtain a sequence of samples  $\{\mathbf{x}^{(t)}\}$  that 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
    
```

Algorithm 30.1. Octave source code for the Hamiltonian Monte Carlo method.

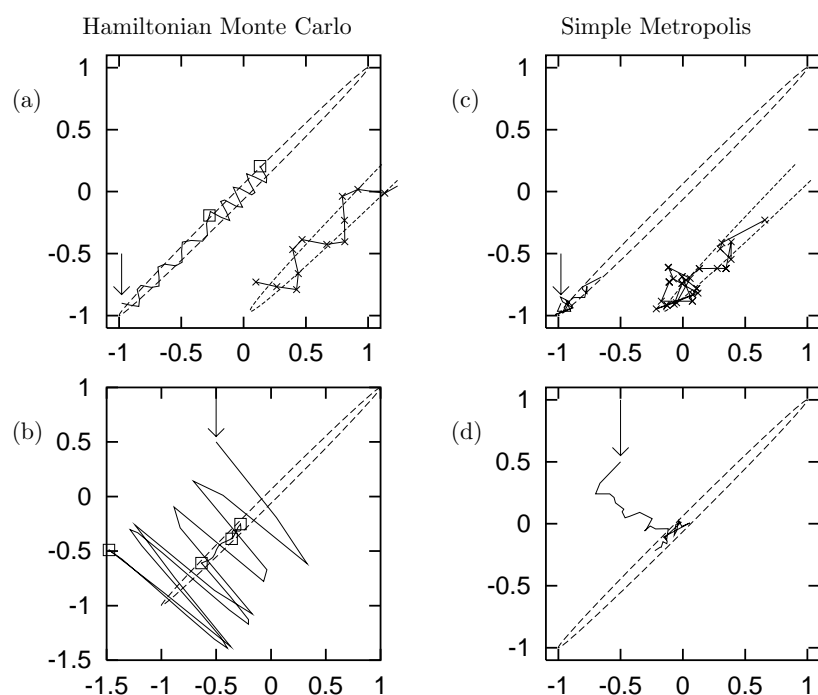


Figure 30.2. (a,b) Hamiltonian Monte Carlo used to generate samples from a bivariate Gaussian with correlation  $\rho = 0.998$ . (c,d) For comparison, a simple random-walk Metropolis method, given equal computer time.

### Details of Hamiltonian Monte Carlo

The first proposal, which can be viewed as a Gibbs sampling update, draws a new momentum from the Gaussian density  $\exp[-K(\mathbf{p})]/Z_K$ . This proposal is always accepted. 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

$$\dot{\mathbf{x}} = \mathbf{p} \quad (30.4)$$

$$\dot{\mathbf{p}} = -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}}. \quad (30.5)$$

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.

The second proposal is accepted in accordance with the Metropolis rule. 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 (29.31) 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 30.1 describes a Hamiltonian 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 30.2 shows this algorithm generating samples from a bivariate Gaussian whose energy function is  $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x}$  with

$$\mathbf{A} = \begin{bmatrix} 250.25 & -249.75 \\ -249.75 & 250.25 \end{bmatrix}, \quad (30.6)$$

corresponding to a variance–covariance matrix of

$$\begin{bmatrix} 1 & 0.998 \\ 0.998 & 1 \end{bmatrix}. \quad (30.7)$$

In figure 30.2a, 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 30.2b 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 kinetic energy becomes much smaller. After the fourth

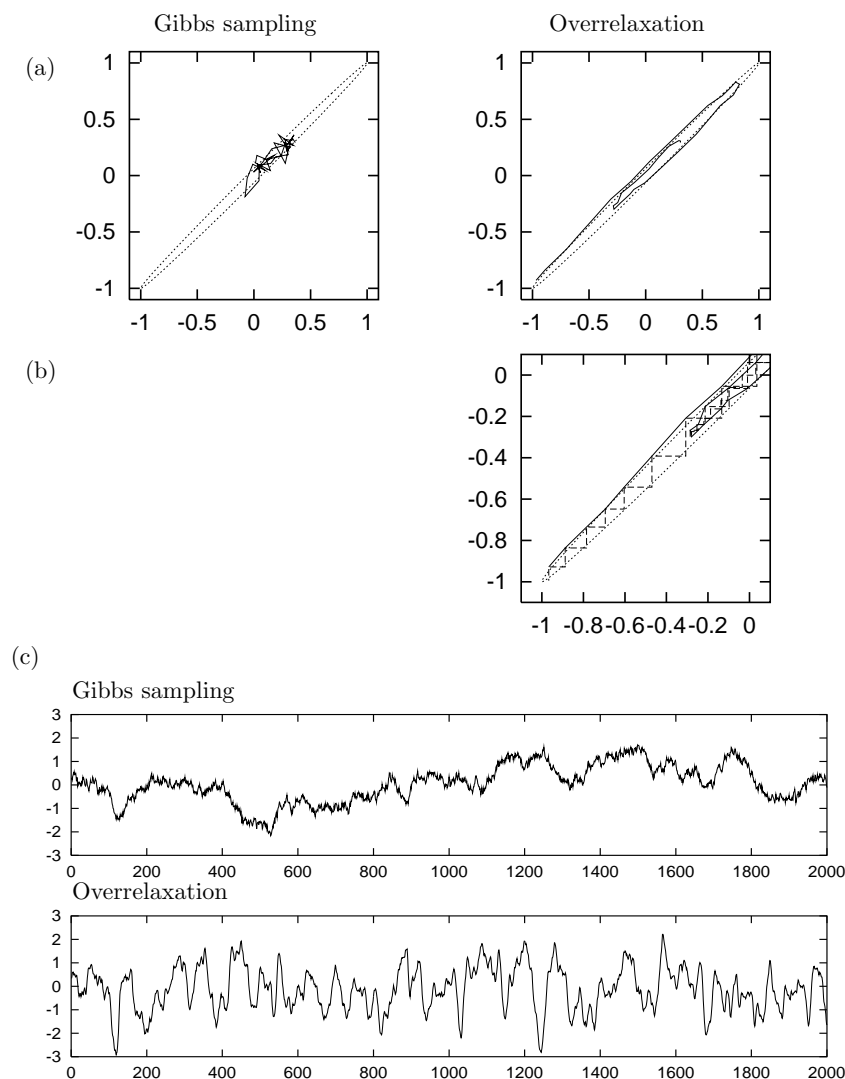


Figure 30.3. Overrelaxation contrasted with Gibbs sampling for a bivariate Gaussian with correlation  $\rho = 0.998$ . (a) The state sequence for 40 iterations, each iteration involving one update of both variables. The overrelaxation method had  $\alpha = -0.98$ . (This excessively large value is chosen to make it easy to see how the overrelaxation method reduces random walk behaviour.) The dotted line shows the contour  $\mathbf{x}^T \Sigma^{-1} \mathbf{x} = 1$ . (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).)

trajectory has been simulated, the state appears to have become typical of the target density.

Figures 30.2(c) and (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 step size was 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.

## ► 30.2 Overrelaxation

The method of *overrelaxation* is a method for reducing random walk behaviour in Gibbs sampling. Overrelaxation was originally introduced for systems in which all the conditional distributions are Gaussian.

An example of a joint distribution that is *not* Gaussian but whose conditional distributions *are* all Gaussian is  $P(x, y) = \exp(-x^2 y^2 - x^2 - y^2) / Z$ .

### 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  $x_i^{(t)}$ . The state makes lengthy random walks in cases where the variables are strongly correlated, as illustrated in the left-hand panel of figure 30.3. 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. 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

$$x_i^{(t+1)} = \mu + \alpha(x_i^{(t)} - \mu) + (1 - \alpha^2)^{1/2}\sigma\nu, \quad (30.8)$$

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 30.1.**<sup>[2]</sup> 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 (30.8). 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. Each individual transition for one coordinate just described *does* 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 30.3. This may significantly reduce the time required to obtain independent samples.

**Exercise 30.2.**<sup>[3]</sup> 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. If the updates were in a *random order*, then  $T$  would be symmetric. Investigate, for the toy two-dimensional Gaussian distribution, the assertion that the advantages of overrelaxation are lost if the overrelaxed updates are made in a random order.

### 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  $x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(K)}$ , 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  $0, 1, 2, \dots, K$ . 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. For practical purposes Neal estimates that ordered overrelaxation may speed up a simulation by a factor of ten or twenty.

### ► 30.3 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 is set to a large value and gradually reduced to 1. This procedure is supposed to reduce the chance that the simulation gets stuck in an unrepresentative probability island.

We assume that we wish to sample from a distribution of the form

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \quad (30.9)$$

where  $E(\mathbf{x})$  can be evaluated. In the simplest simulated annealing method, we instead sample from the distribution

$$P_T(\mathbf{x}) = \frac{1}{Z(T)} e^{-\frac{E(\mathbf{x})}{T}} \quad (30.10)$$

and decrease  $T$  gradually to 1.

Often the energy function can be separated into two terms,

$$E(\mathbf{x}) = E_0(\mathbf{x}) + E_1(\mathbf{x}), \quad (30.11)$$

of which the first term is ‘nice’ (for example, a separable function of  $\mathbf{x}$ ) and the second is ‘nasty’. In these cases, a better simulated annealing method might make use of the distribution

$$P'_T(\mathbf{x}) = \frac{1}{Z'(T)} e^{-E_0(\mathbf{x}) - E_1(\mathbf{x})/T} \quad (30.12)$$

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’ method (Marinari and Parisi, 1992) corrects 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.

### ► 30.4 Skilling’s multi-state leapfrog method

A fourth method for speeding up Monte Carlo simulations, due to John Skilling, has a similar spirit to overrelaxation, 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  $S$

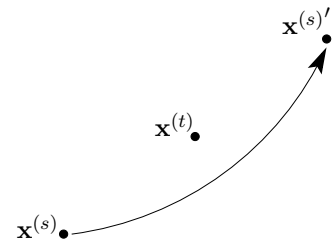
might be six or twelve. The aim is that all  $S$  of these vectors will represent independent samples from the same distribution  $P(\mathbf{x})$ .

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  $\mathbf{x}^{(t)}$ :

$$\mathbf{x}^{(s)'} = \mathbf{x}^{(t)} + (\mathbf{x}^{(t)} - \mathbf{x}^{(s)}) = 2\mathbf{x}^{(t)} - \mathbf{x}^{(s)}. \quad (30.13)$$

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 simplest 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,  $\mathbf{x}^{(s)'}$ .



#### 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 emphasized, 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.

- ▷ **Exercise 30.3.** [2, p.398] 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.



### ► 30.5 Monte Carlo algorithms as communication channels

It may be a helpful perspective, when thinking about speeding up Monte Carlo methods, to think about the information that is being communicated. Two communications take place when a sample from  $P(\mathbf{x})$  is being generated.

First, the selection of a particular  $\mathbf{x}$  from  $P(\mathbf{x})$  necessarily requires that at least  $\log 1/P(\mathbf{x})$  random bits be consumed. [Recall the use of inverse arithmetic coding as a method for generating samples from given distributions (section 6.3).]

Second, the generation of a sample conveys information about  $P(\mathbf{x})$  from the subroutine that is able to evaluate  $P^*(\mathbf{x})$  (and from any other subroutines that have access to properties of  $P^*(\mathbf{x})$ ).

Consider a dumb Metropolis method, for example. In a dumb Metropolis method, the proposals  $Q(\mathbf{x}'; \mathbf{x})$  have nothing to do with  $P(\mathbf{x})$ . Properties of  $P(\mathbf{x})$  are only involved in the algorithm at the acceptance step, when the ratio  $P^*(\mathbf{x}')/P^*(\mathbf{x})$  is computed. The channel from the true distribution  $P(\mathbf{x})$  to the user who is interested in computing properties of  $P(\mathbf{x})$  thus passes through a bottleneck: all the information about  $P$  is conveyed by the string of acceptances and rejections. If  $P(\mathbf{x})$  were replaced by a different distribution  $P_2(\mathbf{x})$ , the only way in which this change would have an influence is that the string of acceptances and rejections would be changed. I am not aware of much use being made of this information-theoretic view of Monte Carlo algorithms, but I think it is an instructive viewpoint: if the aim is to obtain information about properties of  $P(\mathbf{x})$  then presumably it is helpful to identify the channel through which this information flows, and maximize the rate of information transfer.

**Example 30.4.** The information-theoretic viewpoint offers a simple justification for the widely-adopted rule of thumb, which states that the parameters of a dumb Metropolis method should be adjusted such that the acceptance rate is about one half. Let's call the acceptance history, that is, the binary string of accept or reject decisions,  $\mathbf{a}$ . The information learned about  $P(\mathbf{x})$  after the algorithm has run for  $T$  steps is less than or equal to the information content of  $\mathbf{a}$ , since all information about  $P$  is mediated by  $\mathbf{a}$ . And the information content of  $\mathbf{a}$  is upper-bounded by  $TH_2(f)$ , where  $f$  is the acceptance rate. This bound on information acquired about  $P$  is maximized by setting  $f = 1/2$ .

Another helpful analogy for a dumb Metropolis method is an evolutionary one. Each proposal generates a progeny  $\mathbf{x}'$  from the current state  $\mathbf{x}$ . These two individuals then compete with each other, and the Metropolis method uses a noisy survival-of-the-fittest rule. If the progeny  $\mathbf{x}'$  is fitter than the parent (i.e.,  $P^*(\mathbf{x}') > P^*(\mathbf{x})$ , assuming the  $Q/Q$  factor is unity) then the progeny replaces the parent. The survival rule also allows less-fit progeny to replace the parent, sometimes. Insights about the rate of evolution can thus be applied to Monte Carlo methods.

**Exercise 30.5.**<sup>[3]</sup> Let  $\mathbf{x} \in \{0, 1\}^G$  and let  $P(\mathbf{x})$  be a separable distribution,

$$P(\mathbf{x}) = \prod_g p(x_g), \quad (30.14)$$

with  $p(0) = p_0$  and  $p(1) = p_1$ , for example  $p_1 = 0.1$ . Let the proposal density of a dumb Metropolis algorithm  $Q$  involve flipping a fraction  $m$  of the  $G$  bits in the state  $\mathbf{x}$ . Analyze how long it takes for the chain to

converge to the target density as a function of  $m$ . Find the optimal  $m$  and deduce how long the Metropolis method must run for.

Compare the result with the results for an evolving population under natural selection found in Chapter 19.

The insight that the fastest progress that a standard Metropolis method can make, in information terms, is about one bit per iteration, gives a strong motivation for speeding up the algorithm. This chapter has already reviewed several methods for reducing random-walk behaviour. Do these methods also speed up the rate at which information is acquired?

**Exercise 30.6.**<sup>[4]</sup> Does Gibbs sampling, which is a smart Metropolis method whose proposal distributions do depend on  $P(\mathbf{x})$ , allow information about  $P(\mathbf{x})$  to leak out at a rate faster than one bit per iteration? Find toy examples in which this question can be precisely investigated.

**Exercise 30.7.**<sup>[4]</sup> Hamiltonian Monte Carlo is another smart Metropolis method in which the proposal distributions depend on  $P(\mathbf{x})$ . Can Hamiltonian Monte Carlo extract information about  $P(\mathbf{x})$  at a rate faster than one bit per iteration?

**Exercise 30.8.**<sup>[5]</sup> In importance sampling, the weight  $w_r = P^*(\mathbf{x}^{(r)})/Q^*(\mathbf{x}^{(r)})$ , a floating-point number, is computed and retained until the end of the computation. In contrast, in the dumb Metropolis method, the ratio  $a = P^*(\mathbf{x}')/P^*(\mathbf{x})$  is reduced to a single bit ('is  $a$  bigger than or smaller than the random number  $u$ ?'). Thus in principle importance sampling preserves more information about  $P^*$  than does dumb Metropolis. Can you find a toy example in which this extra information does indeed lead to faster convergence of importance sampling than Metropolis? Can you design a Markov chain Monte Carlo algorithm that moves around adaptively, like a Metropolis method, and that retains more useful information about the value of  $P^*$ , like importance sampling?

In Chapter 19 we noticed that an evolving population of  $N$  individuals can make faster evolutionary progress if the individuals engage in sexual reproduction. This observation motivates looking at Monte Carlo algorithms in which multiple parameter vectors  $\mathbf{x}$  are evolved and interact.

## ► 30.6 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 either that eventually all the vectors  $\mathbf{x}$  should be samples from  $P(\mathbf{x})$  (as illustrated by Skilling's leapfrog method), or that information associated with the final vectors  $\mathbf{x}$  should allow us to approximate expectations under  $P(\mathbf{x})$ , as in importance sampling.

### *Genetic methods*

Genetic algorithms are not often described by their proponents as Monte Carlo algorithms, but I think this is the correct categorization, and an ideal genetic algorithm would be one that can be proved to be a valid Monte Carlo algorithm that converges to a specified density.

I'll use  $R$  to denote the number of vectors in the population. We aim to have  $P^*(\{\mathbf{x}^{(r)}\}_1^R) = \prod P^*(\mathbf{x}^{(r)})$ . A genetic algorithm involves moves of two or three types.

First, individual moves in which one state vector is perturbed,  $\mathbf{x}^{(r)} \rightarrow \mathbf{x}^{(r)'}$ , which could be performed using any of the Monte Carlo methods we have mentioned so far.

Second, we allow crossover moves of the form  $\mathbf{x}, \mathbf{y} \rightarrow \mathbf{x}', \mathbf{y}'$ ; in a typical crossover move, the progeny  $\mathbf{x}'$  receives half his state vector from one parent,  $\mathbf{x}$ , and half from the other,  $\mathbf{y}$ ; the secret of success in a genetic algorithm is that the parameter  $\mathbf{x}$  must be encoded in such a way that the crossover of two independent states  $\mathbf{x}$  and  $\mathbf{y}$ , both of which have good fitness  $P^*$ , should have a reasonably good chance of producing progeny who are equally fit. This constraint is a hard one to satisfy in many problems, which is why genetic algorithms are mainly talked about and hyped up, and rarely used by serious experts. Having introduced a crossover move  $\mathbf{x}, \mathbf{y} \rightarrow \mathbf{x}', \mathbf{y}'$ , we need to choose an acceptance rule. One easy way to obtain a valid algorithm is to accept or reject the crossover proposal using the Metropolis rule with  $P^*(\{\mathbf{x}^{(r)}\}_1^R)$  as the target density – this involves comparing the fitnesses before and after the crossover using the ratio

$$\frac{P^*(\mathbf{x}')P^*(\mathbf{y}')}{P^*(\mathbf{x})P^*(\mathbf{y})}. \quad (30.15)$$

If the crossover operator is reversible then we have an easy proof that this procedure satisfies detailed balance and so is a valid component in a chain converging to  $P^*(\{\mathbf{x}^{(r)}\}_1^R)$ .

- ▷ Exercise 30.9.<sup>[3]</sup> Discuss whether the above two operators, individual variation and crossover with the Metropolis acceptance rule, will give a more efficient Monte Carlo method than a standard method with only one state vector and no crossover.

The reason why the sexual community could acquire information faster than the asexual community in Chapter 19 was because the crossover operation produced diversity with standard deviation  $\sqrt{G}$ , then the Blind Watchmaker was able to convey lots of information about the fitness function by *killing off* the less fit offspring. The above two operators do *not* offer a speed-up of  $\sqrt{G}$  compared with standard Monte Carlo methods because there is no killing. What's required, in order to obtain a speed-up, is two things: multiplication and death; and at least one of these must operate *selectively*. Either we must kill off the less-fit state vectors, or we must allow the more-fit state vectors to give rise to more offspring. While it's easy to sketch these ideas, it is hard to define a valid method for doing it.

Exercise 30.10.<sup>[5]</sup> Design a birth rule and a death rule such that the chain converges to  $P^*(\{\mathbf{x}^{(r)}\}_1^R)$ .

I believe this is still an open research problem.

### Particle filters

Particle filters, which are particularly popular in inference problems involving temporal tracking, are multistate methods that mix the ideas of importance sampling and Markov chain Monte Carlo. See Isard and Blake (1996), Isard and Blake (1998), Berzuini *et al.* (1997), Berzuini and Gilks (2001), Doucet *et al.* (2001).

### ► 30.7 Methods that do not necessarily help

It is common practice to use *many* initial conditions for a particular Markov chain (figure 29.19). 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 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.

#### Further reading

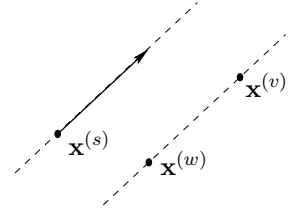
The Hamiltonian Monte Carlo method (Duane *et al.*, 1987) is reviewed in Neal (1993b). This excellent tome also reviews a huge range of other Monte Carlo methods, including the related topics of simulated annealing and free energy estimation.

### ► 30.8 Further exercises

Exercise 30.11.<sup>[4]</sup> An important detail of the Hamiltonian Monte Carlo method is that the simulation of the Hamiltonian dynamics, while it may be inaccurate, must be perfectly reversible, in the sense that if the initial condition  $(\mathbf{x}, \mathbf{p})$  goes to  $(\mathbf{x}', \mathbf{p}')$ , then the same simulator must take  $(\mathbf{x}', -\mathbf{p}')$  to  $(\mathbf{x}, -\mathbf{p})$ , and the inaccurate dynamics must conserve state-space volume. [The leapfrog method in algorithm 30.1 satisfies these rules.]

Explain why these rules must be satisfied and create an example illustrating the problems that arise if they are not.

**Exercise 30.12.**<sup>[4]</sup> A multi-state idea for slice sampling. Investigate the following multi-state method for slice sampling. As in Skilling's multi-state leapfrog method (section 30.4), maintain a set of  $S$  state vectors  $\{\mathbf{x}^{(s)}\}$ . Update one state vector  $\mathbf{x}^{(s)}$  by one-dimensional slice sampling in a direction  $\mathbf{y}$  determined by picking two other state vectors  $\mathbf{x}^{(v)}$  and  $\mathbf{x}^{(w)}$  at random and setting  $\mathbf{y} = \mathbf{x}^{(v)} - \mathbf{x}^{(w)}$ . Investigate this method on toy problems such as a highly-correlated multivariate Gaussian distribution. Bear in mind that if  $S - 1$  is smaller than the number of dimensions  $N$  then this method will not be ergodic by itself, so it may need to be mixed with other methods. Are there classes of problems that are better solved by this slice-sampling method than by the standard methods for picking  $\mathbf{y}$  such as cycling through the coordinate axes or picking  $\mathbf{u}$  at random from a Gaussian distribution?



### ► 30.9 Solutions

**Solution to exercise 30.3 (p. 393).** 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

$$(x_n^{(1)})' = 2x_n^{(2)} - x_n^{(1)}. \quad (30.16)$$

Assuming that  $x_n^{(1)}$  and  $x_n^{(2)}$  are Gaussian random variables from  $\text{Normal}(0, 1)$ ,  $(x_n^{(1)})'$  is Gaussian from  $\text{Normal}(0, \sigma^2)$ , where  $\sigma^2 = 2^2 + (-1)^2 = 5$ . The change in energy contributed by this one dimension will be

$$\frac{1}{2} \left[ (2x_n^{(2)} - x_n^{(1)})^2 - (x_n^{(1)})^2 \right] = 2(x_n^{(2)})^2 - 2x_n^{(2)}x_n^{(1)} \quad (30.17)$$

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

$$e^{-2N}. \quad (30.18)$$

This implies that Skilling's method, as described, is not effective in very high-dimensional problems – at least, not once convergence has occurred. 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 Hamiltonian Monte Carlo and overrelaxation methods offer.

---

## *About Chapter 31*

Some of the neural network models that we will encounter are related to Ising models, which are idealized 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 25 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 (cf. Chapter 17): a two-dimensional bar-code in which a black dot may not be completely surrounded by black dots, and a white dot may not be completely surrounded by white dots, is similar to an antiferromagnetic Ising model at low temperature. Evaluating the entropy of this Ising model is equivalent to evaluating the capacity of the constrained channel for conveying bits.

If you would like to jog your memory on statistical physics and thermodynamics, you might find Appendix B helpful. I also recommend the book by Reif (1965).

# 31

---

## *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

$$E(\mathbf{x}; J, H) = - \left[ \frac{1}{2} \sum_{m,n} J_{mn} x_m x_n + \sum_n H x_n \right], \quad (31.1)$$

where  $H$  is the applied field. If  $J > 0$  then the model is ferromagnetic, and if  $J < 0$  it is antiferromagnetic. 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

$$P(\mathbf{x} | \beta, J, H) = \frac{1}{Z(\beta, J, H)} \exp[-\beta E(\mathbf{x}; J, H)], \quad (31.2)$$

where  $\beta = 1/k_B T$ ,  $k_B$  is Boltzmann's constant, and

$$Z(\beta, J, H) \equiv \sum_{\mathbf{x}} \exp[-\beta E(\mathbf{x}; J, H)]. \quad (31.3)$$

### *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

$$E(\mathbf{x}; \mathbf{J}, \mathbf{h}) = - \left[ \frac{1}{2} \sum_{m,n} J_{mn} x_m x_n + \sum_n h_n x_n \right], \quad (31.4)$$

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

$$C \equiv \frac{\partial}{\partial T} \bar{E}, \quad (31.5)$$

where

$$\bar{E} = \frac{1}{Z} \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})) E(\mathbf{x}). \quad (31.6)$$

To work out the heat capacity of a system, we might naively guess that we have to increase the temperature and measure the energy change. Heat capacity, however, is intimately related to energy *fluctuations* at constant temperature. Let's start from the partition function,

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})). \quad (31.7)$$

The mean energy is obtained by differentiation with respect to  $\beta$ :

$$\frac{\partial \ln Z}{\partial \beta} = \frac{1}{Z} \sum_{\mathbf{x}} -E(\mathbf{x}) \exp(-\beta E(\mathbf{x})) = -\bar{E}. \quad (31.8)$$

A further differentiation spits out the variance of the energy:

$$\frac{\partial^2 \ln Z}{\partial \beta^2} = \frac{1}{Z} \sum_{\mathbf{x}} E(\mathbf{x})^2 \exp(-\beta E(\mathbf{x})) - \bar{E}^2 = \langle E^2 \rangle - \bar{E}^2 = \text{var}(E). \quad (31.9)$$

But the heat capacity is also the derivative of  $\bar{E}$  with respect to temperature:

$$\frac{\partial \bar{E}}{\partial T} = -\frac{\partial}{\partial T} \frac{\partial \ln Z}{\partial \beta} = -\frac{\partial^2 \ln Z}{\partial \beta^2} \frac{\partial \beta}{\partial T} = -\text{var}(E)(-1/k_B T^2). \quad (31.10)$$

So for any system at temperature  $T$ ,

$$C = \frac{\text{var}(E)}{k_B T^2} = k_B \beta^2 \text{var}(E). \quad (31.11)$$

Thus if we can observe the variance of the energy of a system at equilibrium, we can 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  $\text{var}(E)$  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

$$C^{(\beta)} \equiv \frac{\partial \bar{E}}{\partial \beta} = \text{var}(E), \quad (31.12)$$

and heat capacity and fluctuations would be identical quantities.

▷ **Exercise 31.1.**<sup>[2]</sup> [We will call the entropy of a physical system  $S$  rather than  $H$ , while we are in a statistical physics chapter; we set  $k_B = 1$ .]

The entropy of a system whose states are  $\mathbf{x}$ , at temperature  $T = 1/\beta$ , is

$$S = -\sum p(\mathbf{x}) [\ln 1/p(\mathbf{x})] \quad (31.13)$$

where

$$p(\mathbf{x}) = \frac{1}{Z(\beta)} \exp[-\beta E(\mathbf{x})]. \quad (31.14)$$

(a) Show that

$$S = \ln Z(\beta) + \beta \bar{E}(\beta) \quad (31.15)$$

where  $\bar{E}(\beta)$  is the mean energy of the system.

(b) Show that

$$S = -\frac{\partial F}{\partial T}, \quad (31.16)$$

where the free energy  $F = -kT \ln Z$  and  $kT = 1/\beta$ .

### ► 31.1 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,

$$P(+1 | b_n) = \frac{1}{1 + \exp(-2\beta b_n)}, \quad (31.17)$$

where  $\beta = 1/k_B T$  and  $b_n$  is the local field

$$b_n = \sum_{m: (m,n) \in \mathcal{N}} J x_m + H. \quad (31.18)$$

[The factor of 2 appears in equation (31.17) 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 (31.2). An alternative to the Gibbs sampling formula (31.17) is the Metropolis algorithm, in which we consider the change in energy that results from flipping the chosen spin from its current state  $x_n$ ,

$$\Delta E = 2x_n b_n, \quad (31.19)$$

and adopt this change in configuration with probability

$$P(\text{accept}; \Delta E, \beta) = \begin{cases} 1 & \Delta E \leq 0 \\ \exp(-\beta \Delta E) & \Delta E > 0. \end{cases} \quad (31.20)$$

### 31.1: Ising models – Monte Carlo simulation

403

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 merits of Gibbs sampling and the Metropolis algorithm may be subtle.

#### Rectangular geometry

I first simulated an Ising model with the rectangular geometry shown in figure 31.1, and with periodic boundary conditions. A line between two spins indicates that they are neighbours. I set the external field  $H = 0$  and considered the two cases  $J = \pm 1$ , which are a ferromagnet and antiferromagnet respectively.

I started 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  $m$ ,

$$m = \frac{1}{N} \sum_n x_n. \quad (31.21)$$

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 difficult to detect ‘equilibrium’ – or even to give a clear definition of a system’s being ‘at equilibrium’! [But in Chapter 32 we will see a solution to this problem.] 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.

#### Results for small $N$ with $J = 1$ .

I simulated an  $l \times l$  grid for  $l = 4, 5, \dots, 10, 40, 64$ . Let’s have a quick think about what results we expect. At low temperatures the system is expected to be in a ground state. The rectangular Ising model with  $J = 1$  has two ground states, the all  $+1$  state and the all  $-1$  state. The energy per spin of either ground state is  $-2$ . At high temperatures, 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 31.3, top): the energy rises monotonically. As we increase the number of spins to 100 (figure 31.3, 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 31.5 shows details of the graphs for  $N = 100$  and  $N = 4096$ . Figure 31.2 shows a sequence of typical states from the simulation of  $N = 4096$  spins at a sequence of decreasing temperatures.

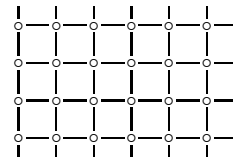


Figure 31.1. Rectangular Ising model.

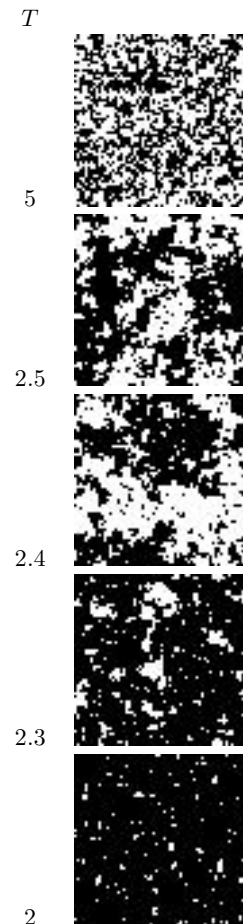


Figure 31.2. Sample states of rectangular Ising models with  $J = 1$  at a sequence of temperatures  $T$ .

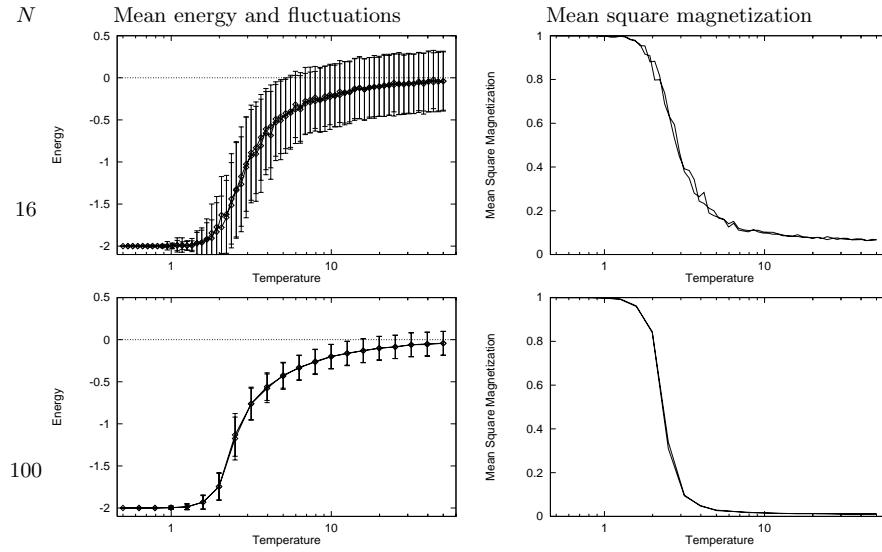


Figure 31.3. Monte Carlo simulations of rectangular Ising models with  $J = 1$ . Mean energy and fluctuations in energy as a function of temperature (left). Mean square magnetization as a function of temperature (right). In the top row,  $N = 16$ , and the bottom,  $N = 100$ . For even larger  $N$ , see later figures.

### 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; a peak 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 (such as a particle in a box) 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 31.4).

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

$$E(\beta) = \epsilon \frac{\exp(-\beta\epsilon)}{1 + \exp(-\beta\epsilon)} = \epsilon \frac{1}{1 + \exp(\beta\epsilon)} \quad (31.22)$$

and the derivative with respect to  $\beta$  is

$$dE/d\beta = -\epsilon^2 \frac{\exp(\beta\epsilon)}{[1 + \exp(\beta\epsilon)]^2}. \quad (31.23)$$

So the heat capacity is

$$C = dE/dT = -\frac{dE}{d\beta} \frac{1}{k_B T^2} = \frac{\epsilon^2}{k_B T^2} \frac{\exp(\beta\epsilon)}{[1 + \exp(\beta\epsilon)]^2} \quad (31.24)$$

and the fluctuations in energy are given by  $\text{var}(E) = C k_B T^2 = -dE/d\beta$ , which was evaluated in (31.23). The heat capacity and fluctuations are plotted in figure 31.6. 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* that is interesting, rather than a peak in the heat capacity. The Ising model has such a peak in its fluctuations, as can be seen in the second row of figure 31.5.

### Rectangular Ising model with $J = -1$

What do we expect to happen in the case  $J = -1$ ? The ground states of an infinite system are the two checkerboard patterns (figure 31.7), and they have

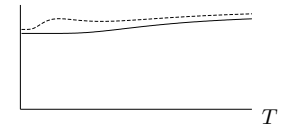


Figure 31.4. Schematic diagram to explain the meaning of a Schottky anomaly. The curve shows the heat capacity of two gases as a function of temperature. The lower curve shows a normal gas whose heat capacity is an increasing function of temperature. The upper curve has a small peak in the heat capacity, 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.

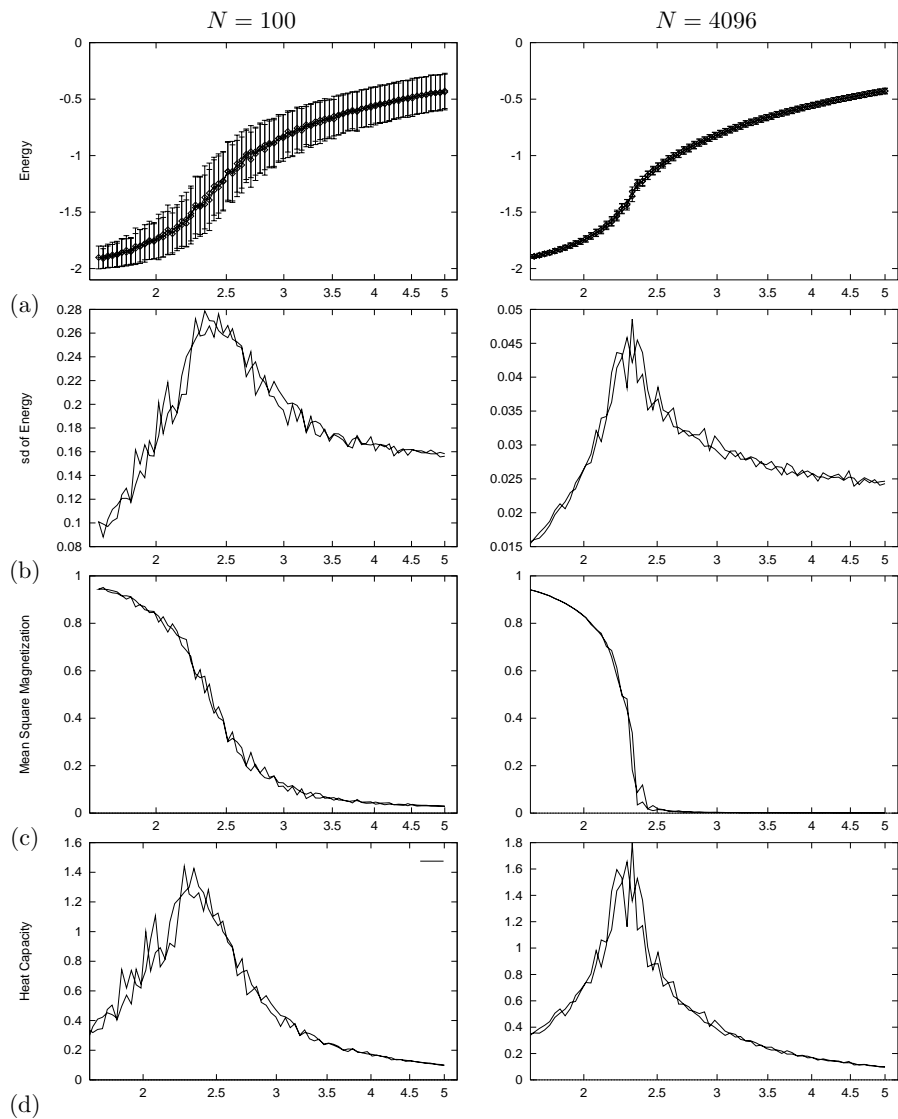


Figure 31.5. Detail of Monte Carlo simulations of rectangular Ising models with  $J = 1$ . (a) Mean energy and fluctuations in energy as a function of temperature. (b) Fluctuations in energy (standard deviation). (c) Mean square magnetization. (d) Heat capacity.

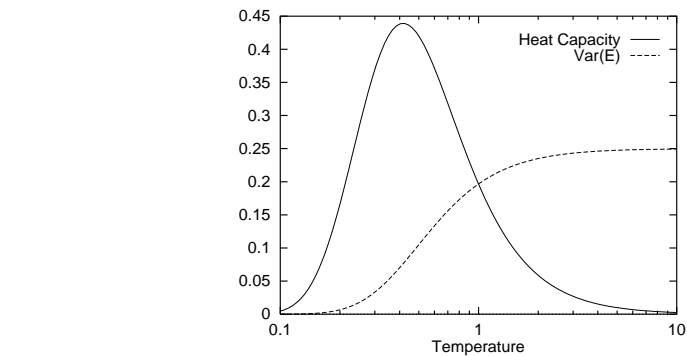


Figure 31.6. Schottky anomaly – Heat capacity and fluctuations in energy as a function of temperature for a two-level system with separation  $\epsilon = 1$  and  $k_B = 1$ .

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 31.8), 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, 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 31.9 compares the energies of the ferromagnetic and antiferromagnetic models with  $N = 25$ . Here, the difference is striking.

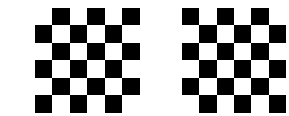
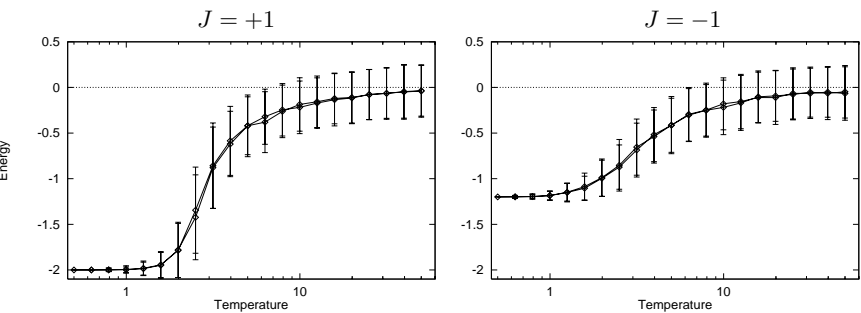


Figure 31.7. The two ground states of a rectangular Ising model with  $J = -1$ .

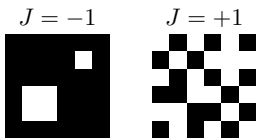


Figure 31.8. Two states of rectangular Ising models with  $J = \pm 1$  that have identical energy.

Figure 31.9. 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.

### 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 31.10. (Solid lines show ‘happy’ couplings which contribute  $-|J|$  to the energy; dashed lines show ‘unhappy’ couplings which contribute  $|J|$ .) 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. Sample states are shown in figure 31.12, and figure 31.11 shows the energy, fluctuations, and heat capacity for  $N = 4096$ .

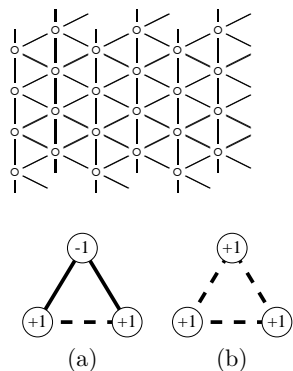


Figure 31.10. In an antiferromagnetic triangular Ising model, any three neighbouring spins are frustrated. Of the eight possible configurations of three spins, six have energy  $-|J|$  (a), and two have energy  $3|J|$  (b).

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.

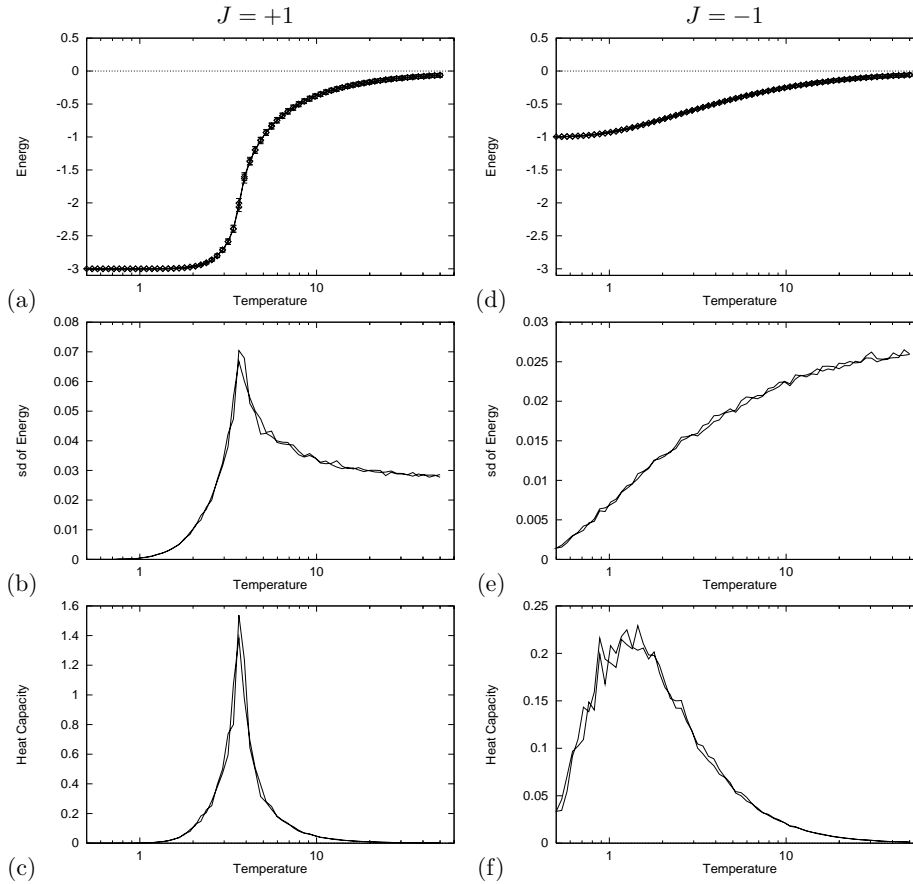


Figure 31.11. Monte Carlo simulations of triangular Ising models with  $J = \pm 1$  and  $N = 4096$ . (a–c)  $J = 1$ . (d–f)  $J = -1$ . (a, d) Mean energy and fluctuations in energy as a function of temperature. (b, e) Fluctuations in energy (standard deviation). (c, f) Heat capacity.

### ► 31.2 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 that obtains physical properties of the model from the partition function

$$Z(\beta, \mathbf{J}, \mathbf{b}) \equiv \sum_{\mathbf{x}} \exp[-\beta E(\mathbf{x}; \mathbf{J}, \mathbf{b})], \quad (31.25)$$

where the summation is over all states  $\mathbf{x}$ , and the inverse temperature is  $\beta = 1/T$ . [As usual, Let  $k_B = 1$ .] The free energy is given by  $F = -\frac{1}{\beta} \ln Z$ . 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 trick similar to the sum-product algorithm discussed in Chapter 25. We concentrate on models that have the form of a long thin strip of width  $W$  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,  $W$ . The final clever trick

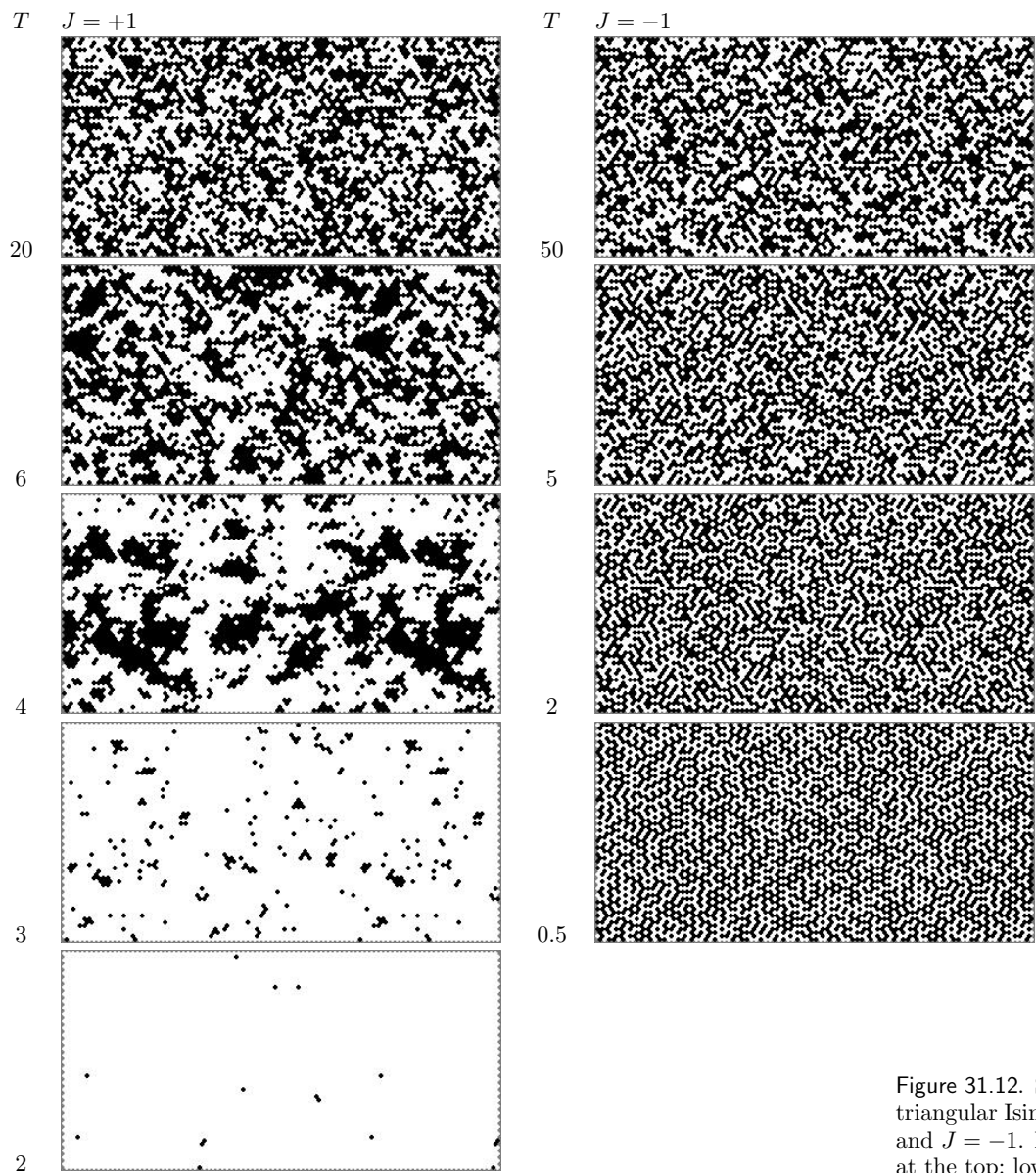


Figure 31.12. Sample states of triangular Ising models with  $J = 1$  and  $J = -1$ . High temperatures at the top; low at the bottom.

### 31.2: Direct computation of partition function of Ising models

409

is to note that if the system is translation-invariant along its length then we need to do only *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 the  $C$  columns of the thin strip  $s_1, s_2, \dots, s_C$ , 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

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})) \quad (31.26)$$

$$= \sum_{s_1} \sum_{s_2} \cdots \sum_{s_C} \exp\left(-\beta \sum_{c=1}^C \mathcal{E}(s_c, s_{c+1})\right), \quad (31.27)$$

where  $\mathcal{E}(s_c, s_{c+1})$  is an appropriately defined energy, and, if we want periodic boundary conditions,  $s_{C+1}$  is defined to be  $s_1$ . One definition for  $\mathcal{E}$  is:

$$\mathcal{E}(s_c, s_{c+1}) = \sum_{\substack{(m,n) \in \mathcal{N}: \\ m \in c, n \in c+1}} J x_m x_n + \frac{1}{4} \sum_{\substack{(m,n) \in \mathcal{N}: \\ m \in c, n \in c}} J x_m x_n + \frac{1}{4} \sum_{\substack{(m,n) \in \mathcal{N}: \\ m \in c+1, n \in c+1}} J x_m x_n. \quad (31.28)$$

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

$$M_{ss'} = \exp(-\beta \mathcal{E}(s, s')). \quad (31.29)$$

Then continuing from equation (31.27),

$$Z = \sum_{s_1} \sum_{s_2} \cdots \sum_{s_C} \left[ \prod_{c=1}^C M_{s_c, s_{c+1}} \right] \quad (31.30)$$

$$= \text{Trace} [\mathbf{M}^C] \quad (31.31)$$

$$= \sum_a \mu_a^C, \quad (31.32)$$

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  $\mu_{\max}$ :

$$Z \rightarrow \mu_{\max}^C. \quad (31.33)$$

So the free energy per spin in the limit of an infinite thin strip is given by:

$$f = -kT \ln Z / (WC) = -kTC \ln \mu_{\max} / (WC) = -kT \ln \mu_{\max} / W. \quad (31.34)$$

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  $\mathbf{M}$ !

#### Computations

I computed the partition functions of *long-thin-strip* Ising models with the geometries shown in figure 31.14.

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  $H = 0$ .

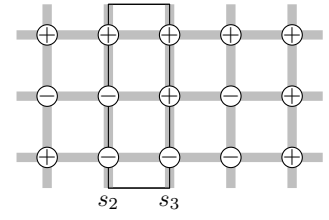
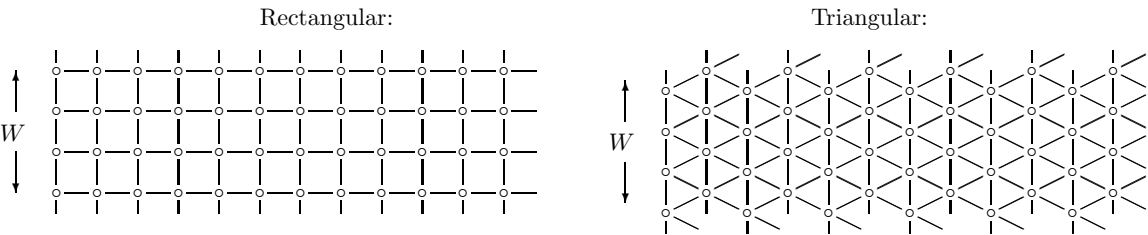


Figure 31.13. Illustration to help explain the definition (31.28).  $\mathcal{E}(s_2, s_3)$  counts all the contributions to the energy in the rectangle. The total energy is given by stepping the rectangle along. Each horizontal bond inside the rectangle is counted once; each vertical bond is half-inside the rectangle (and will be half-inside an adjacent rectangle) so half its energy is included in  $\mathcal{E}(s_2, s_3)$ ; the factor of  $1/4$  appears in the second term because  $m$  and  $n$  both run over all nodes in column  $c$ , so each bond is visited twice.

For the state shown here,  $s_2 = (100)_2$ ,  $s_3 = (110)_2$ , the horizontal bonds contribute  $+J$  to  $\mathcal{E}(s_2, s_3)$ , and the vertical bonds contribute  $-J/2$  on the left and  $-J/2$  on the right, assuming periodic boundary conditions between top and bottom. So  $\mathcal{E}(s_2, s_3) = 0$ .





*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  $(1, 1, \dots, 1)$ . 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 (31.32) – as well as infinite ones.

Figure 31.14. Two long-thin-strip Ising models. A line between two spins indicates that they are neighbours. The strips have width  $W$  and infinite length.

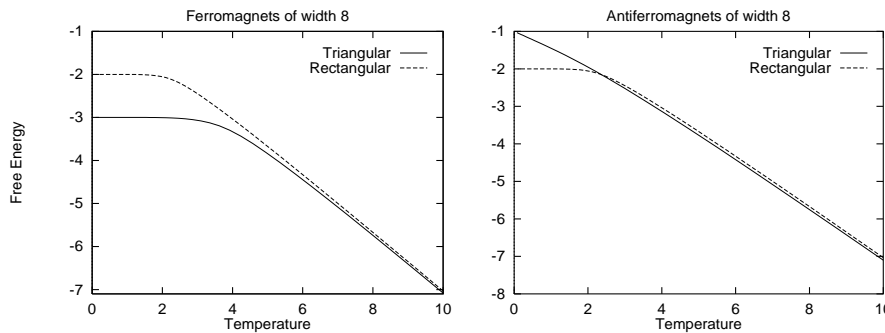


Figure 31.15. Free energy per spin of long-thin-strip Ising models. Note the non-zero gradient at  $T = 0$  in the case of the triangular antiferromagnet.

*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  $\ln(2)$ . The mean energy per spin goes to zero. The free energy per spin should tend to  $-\ln(2)/\beta$ . The free energies are shown in figure 31.15.

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

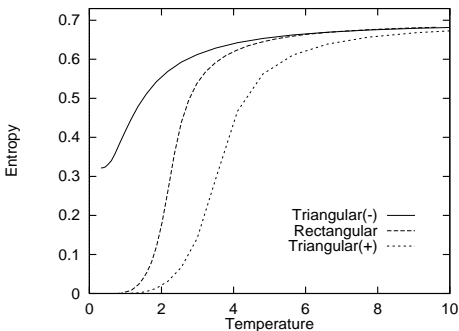


Figure 31.16. Entropies (in nats) of width 8 Ising systems as a function of temperature, obtained by differentiating the free energy curves in figure 31.15. The rectangular ferromagnet and antiferromagnet have identical thermal properties. For the triangular systems, the upper curve (–) denotes the antiferromagnet and the lower curve (+) the ferromagnet.

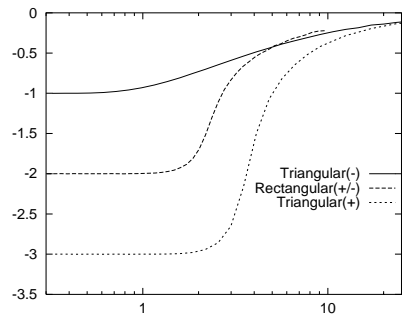


Figure 31.17. Mean energy versus temperature of long thin strip Ising models with width 8. Compare with figure 31.3.

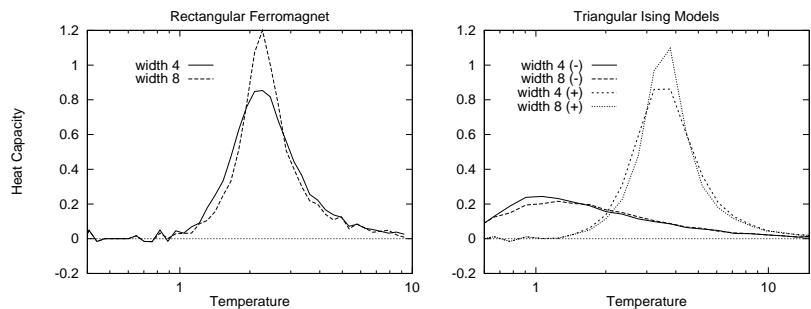


Figure 31.18. Heat capacities of (a) rectangular model; (b) triangular models with different widths, (+) and (-) denoting ferromagnet and antiferromagnet. Compare with figure 31.11.

energy) then the entropy as  $T \rightarrow 0$  is non-zero. We can find the entropy from the free energy using  $S = -\partial F/\partial T$ .

The entropy of the triangular antiferromagnet at absolute zero appears to be about 0.3, that is, about half its high temperature value (figure 31.16). The mean energy as a function of temperature is plotted in figure 31.17. It is evaluated using the identity  $\langle E \rangle = -\partial \ln Z/\partial \beta$ .

Figure 31.18 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 31.19 shows the fluctuations in energy as a function of temperature. All of these figures should show smooth graphs; the roughness of the curves is due to inaccurate numerics. The nature of any phase transition is not obvious, but the graphs seem compatible with the assertion that the ferromagnet shows, and the antiferromagnet does not show a phase transition.

The pictures of the free energy in figure 31.15 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.

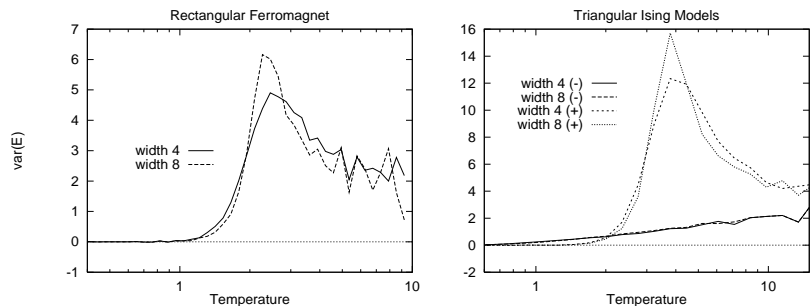


Figure 31.19. Energy variances, per spin, of (a) rectangular model; (b) triangular models with different widths, (+) and (-) denoting ferromagnet and antiferromagnet. Compare with figure 31.11.

### *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 quite identical. One could a more accurate comparison by finding all eigenvalues for the strip of width  $W$  and computing  $\sum \lambda^W$  to get the partition function of a  $W \times W$  patch.

## ► 31.3 Exercises

- ▷ Exercise 31.2.<sup>[4]</sup> 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 31.3.<sup>[3]</sup> 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  $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \dots, \mathbf{x}_{(S)}$ . 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.

## 32

---

### *Exact Monte Carlo Sampling*

#### ► 32.1 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 29, 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.

#### ► 32.2 Exact sampling concepts

Propp and Wilson’s *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 32.1a-i shows twenty-one Markov chains identical to the one described in section 29.4, which samples from  $\{0, 1, \dots, 20\}$  using the Metropolis algorithm (figure 29.12, p.368); each of the chains has a different initial condition but they are all driven by a single random number generator; the chains coalesce after about 80 steps. Figure 32.1a-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 32.1b shows similar Markov chains, each of which has identical proposal density to those in section 29.4 and figure 32.1a; but in figure 32.1b, 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 merge only when a rejection occurs, and rejections occur only at the walls (for this particular Markov chain), coalescence will occur only when the chains are all in the leftmost state or all in the rightmost state.

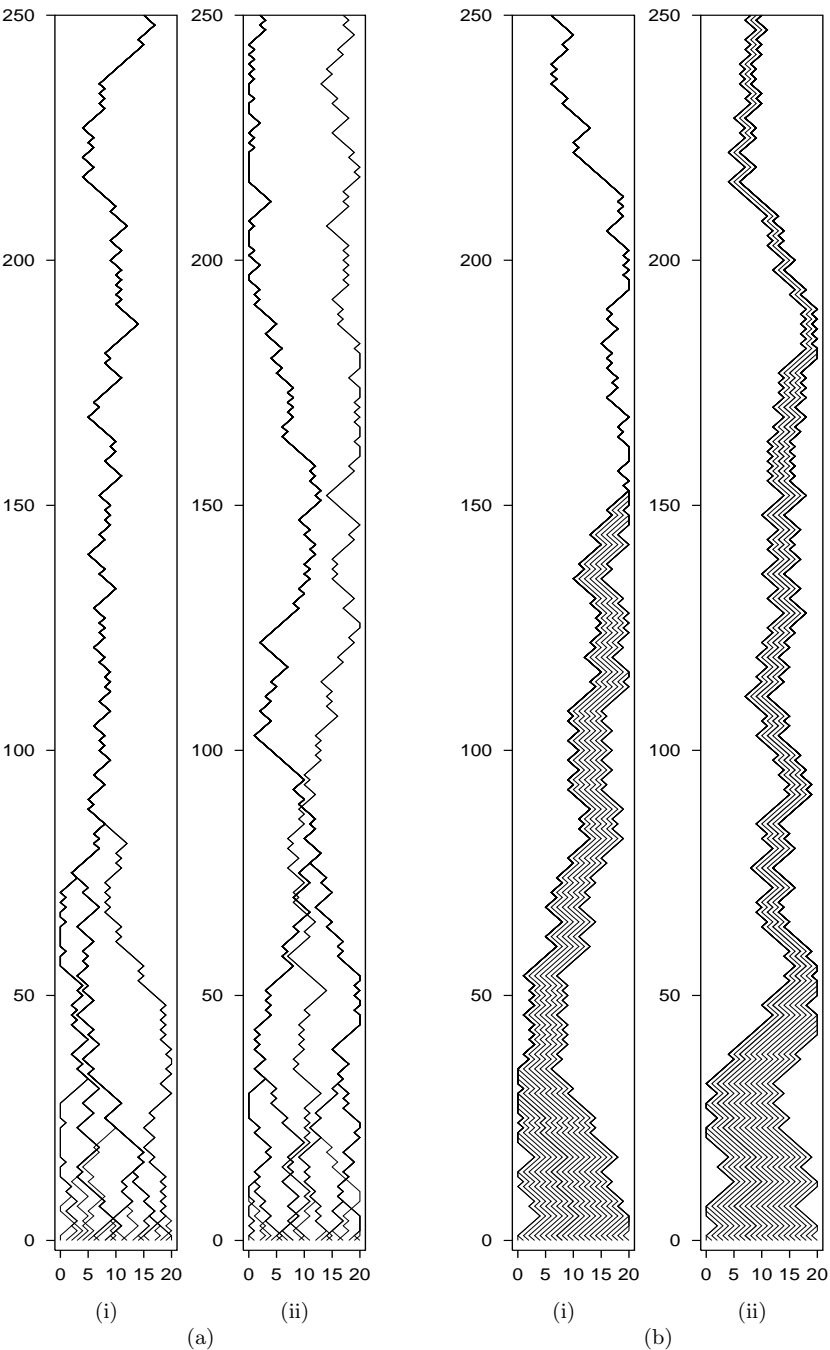


Figure 32.1. Coalescence, the first idea behind the exact sampling method. Time runs from bottom to top. In the leftmost panel, coalescence occurred within 100 steps. Different coalescence properties are obtained depending on the way each state uses the random numbers it is supplied with. (a) Two runs 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.