Coupling from the past

How can we use the coalescence property to find an exact sample from the equilibrium distribution of the chain? The state of the system at the moment when complete coalescence occurs is not a valid sample from the equilibrium distribution; for example in figure 32.1b, final coalescence always occurs when the state is against one of the two walls, because trajectories merge only at the walls. So sampling forward in time until coalescence occurs is not a valid method.

The second key idea of exact sampling is that we can obtain exact samples by sampling *from a time T_0 in the past, up to the present*. If coalescence has occurred, the present sample is an unbiased sample from the equilibrium distribution; if not, we restart the simulation from a time T_0 further into the past, *reusing the same random numbers*. The simulation is repeated at a sequence of ever more distant times T_0, with a doubling of T_0 from one run to the next being a convenient choice. When coalescence occurs at a time before 'the present', we can record $x(0)$ as an *exact sample* from the equilibrium distribution of the Markov chain.

Figure 32.2 shows two exact samples produced in this way. In the leftmost panel of figure 32.2a, we start twenty-one chains in all possible initial conditions at $T_0 = -50$ and run them forward in time. Coalescence does not occur. We restart the simulation from all possible initial conditions at $T_0 = -100$, and reset the random number generator in such a way that the random numbers generated at each time t (in particular, from $t = -50$ to $t = 0$) will be identical to what they were in the first run. Notice that the trajectories produced from $t = -50$ to $t = 0$ by these runs that started from $T_0 = -100$ are identical to a *subset* of the trajectories in the first simulation with $T_0 = -50$. Coalescence still does not occur, so we double T_0 again to $T_0 = -200$. This time, all the trajectories coalesce and we obtain an exact sample, shown by the arrow. If we pick an earlier time such as $T_0 = -500$, all the trajectories must still end in the same point at $t = 0$, since every trajectory must pass through *some* state at $t = -200$, and *all* those states lead to the same final point. So if we ran the Markov chain for an infinite time in the past, from any initial condition, it would end in the same state. Figure 32.2b shows an exact sample produced in the same way with the Markov chains of figure 32.1b.

This method, called *coupling from the past*, is important because it allows us to obtain exact samples from the equilibrium distribution; but, as described here, it is of little practical use, since we are obliged to simulate chains starting in *all* initial states. In the examples shown, there are only twenty-one states, but in any realistic sampling problem there will be an utterly enormous number of states – think of the 2^{1000} states of a system of 1000 binary spins, for example. The whole point of introducing Monte Carlo methods was to try to avoid having to visit all the states of such a system!

Monotonicity

Having established that we can obtain valid samples by simulating forward from times in the past, starting in *all* possible states at those times, the third trick of Propp and Wilson, which makes the exact sampling method useful in practice, is the idea that, for some Markov chains, it may be possible to detect coalescence of all trajectories *without simulating all those trajectories*. This property holds, for example, in the chain of figure 32.1b, which has the property that *two trajectories never cross*. So if we simply track the two trajectories starting from the leftmost and rightmost states, we will know that

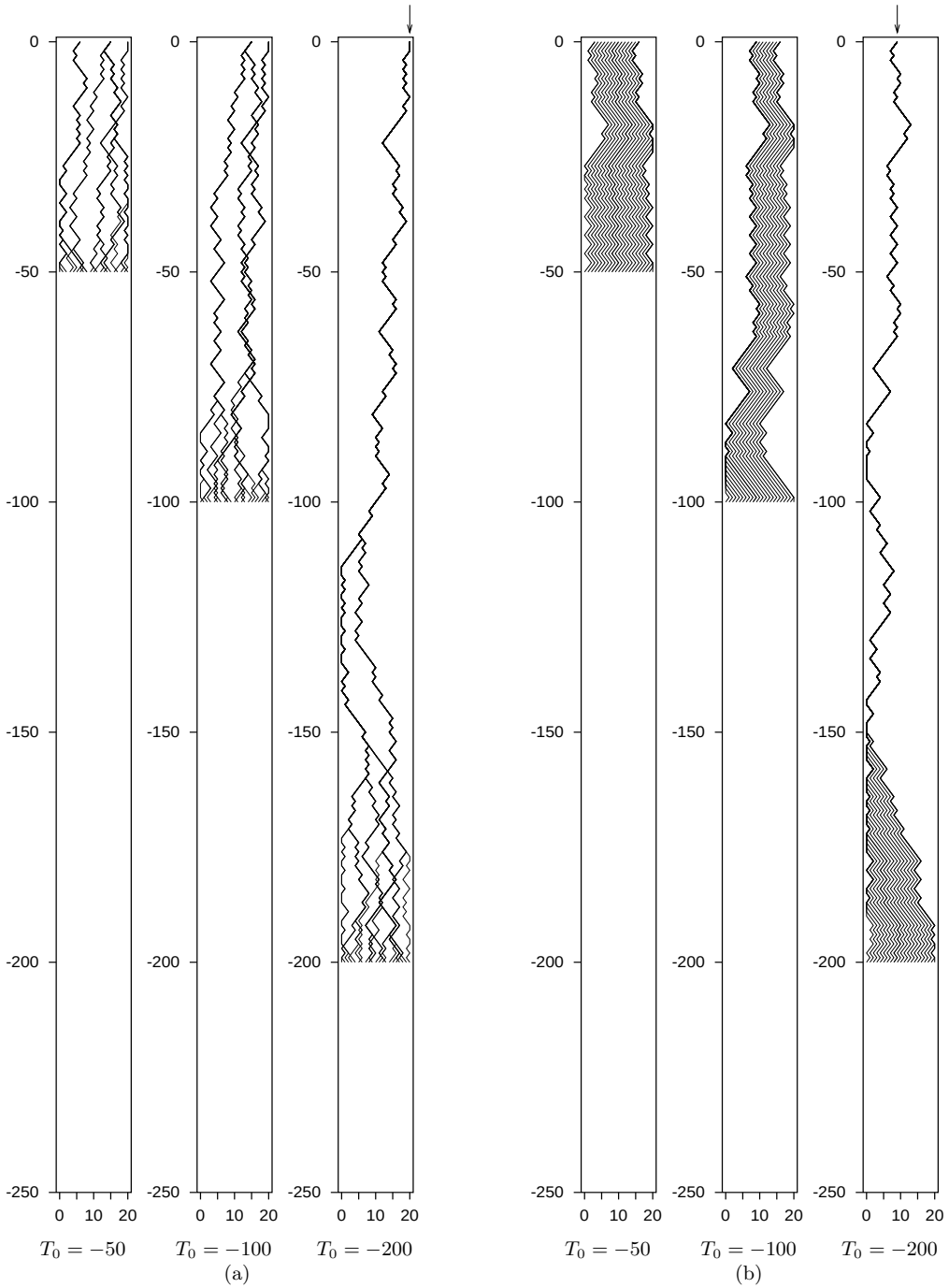


Figure 32.2. 'Coupling from the past', the second idea behind the exact sampling method.

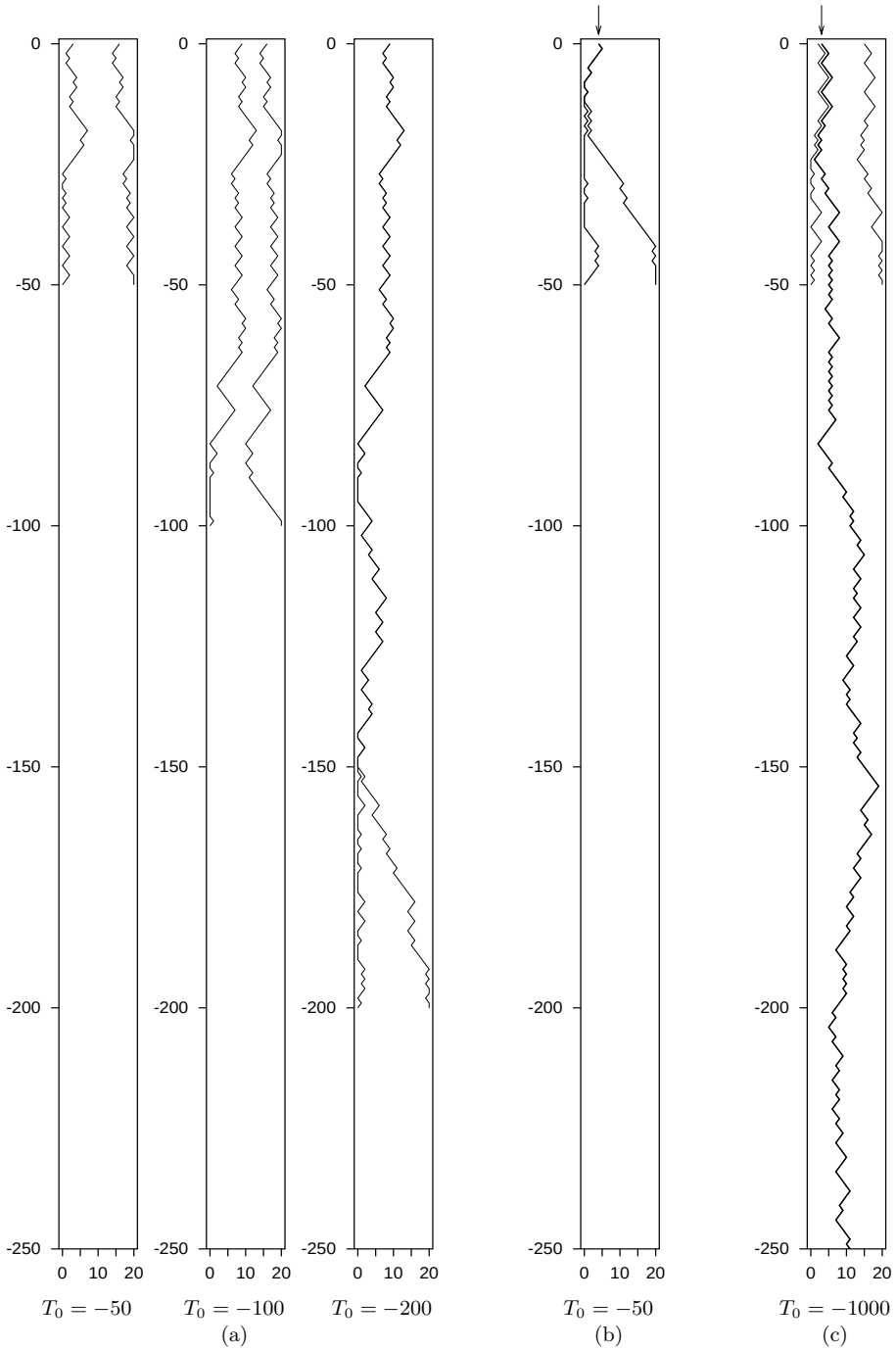


Figure 32.3. (a) Ordering of states, the third idea behind the exact sampling method. The trajectories shown here are the left-most and right-most trajectories of figure 32.2b. In order to establish what the state at time zero is, we only need to run simulations from $T_0 = -50$, $T_0 = -100$, and $T_0 = -200$, after which point coalescence occurs.

(b,c) Two more exact samples from the target density, generated by this method, and different random number seeds. The initial times required were $T_0 = -50$ and $T_0 = -1000$, respectively.

coalescence of *all* trajectories has occurred when *those two* trajectories coalesce. Figure 32.3a illustrates this idea by showing only the left-most and right-most trajectories of figure 32.2b. Figure 32.3(b,c) shows two more exact samples from the same equilibrium distribution generated by running the 'coupling from the past' method starting from the two end-states alone. In (b), two runs coalesced starting from $T_0 = -50$; in (c), it was necessary to try times up to $T_0 = -1000$ to achieve coalescence.

▶ 32.3 Exact sampling from interesting distributions

In the toy problem we studied, the states could be put in a one-dimensional order such that no two trajectories crossed. The states of many interesting state spaces can also be put into a *partial order* and coupled Markov chains can be found that respect this partial order. [An example of a partial order on the four possible states of two spins is this: $(+,+) > (+,-) > (-,-)$; and $(+,+) > (-,+) > (-,-)$; and the states $(+,-)$ and $(-,+)$ are not ordered.] For such systems, we can show that coalescence has occurred merely by verifying that coalescence has occurred for all the histories whose initial states were 'maximal' and 'minimal' states of the state space.

As an example, consider the Gibbs sampling method applied to a ferromagnetic Ising spin system, with the partial ordering of states being defined thus: state $\mathbf{x}$ is 'greater than or equal to' state $\mathbf{y}$ if $x_i \geq y_i$ for all spins i. The maximal and minimal states are the the all-up and all-down states. The Markov chains are coupled together as shown in algorithm 32.4. Propp and Wilson (1996) show that exact samples can be generated for this system, although the time to find exact samples is large if the Ising model is below its critical temperature, since the Gibbs sampling method itself is slowly-mixing under these conditions. Propp and Wilson have improved on this method for the Ising model by using a Markov chain called the single-bond heat bath algorithm to sample from a related model called the random cluster model; they show that exact samples from the random cluster model can be obtained rapidly and can be converted into exact samples from the Ising model. Their ground-breaking paper includes an exact sample from a 16-million-spin Ising model at its critical temperature. A sample for a smaller Ising model is shown in figure 32.5.

Compute $a_i := \sum_j J_{ij} x_j$
Draw u from Uniform$(0, 1)$
If $u < 1/(1 + e^{-2a_i})$
  $x_i := +1$
Else
  $x_i := -1$

Algorithm 32.4. Gibbs sampling coupling method. The Markov chains are coupled together by having all chains update the same spin i at each time step and having all chains share a common sequence of random numbers u.

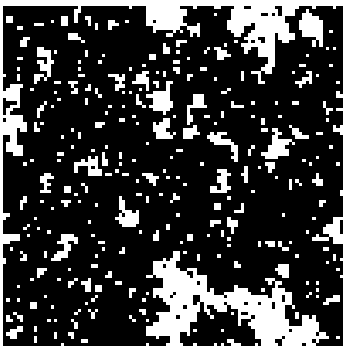


Figure 32.5. An exact sample from the Ising model at its critical temperature, produced by D.B. Wilson. Such samples can be produced within seconds on an ordinary computer by exact sampling.

A generalization of the exact sampling method for 'non-attractive' distributions

The method of Propp and Wilson for the Ising model, sketched above, can be applied only to probability distributions that are, as they call them, 'attractive'. Rather than define this term, let's say what it means, for practical purposes: the method can be applied to spin systems in which all the couplings are positive (e.g., the ferromagnet), and to a few special spin systems with negative couplings (e.g., as we already observed in Chapter 31, the rectangular ferromagnet and antiferromagnet are equivalent); but it cannot be applied to general spin systems in which some couplings are negative, because in such systems the trajectories followed by the all-up and all-down states are not guaranteed to be upper and lower bounds for the set of all trajectories. Fortunately, however, we do not need to be so strict. It is possible to re-express the Propp and Wilson algorithm in a way that generalizes to the case of spin systems with negative couplings. The idea of the *summary state* version of exact sampling is still that we keep track of bounds on the set of

all trajectories, and detect when these bounds are equal, so as to find exact samples. But the bounds will not themselves be actual trajectories, and they will not necessarily be *tight* bounds.

Instead of simulating two trajectories, each of which moves in a state space $\{-1,+1\}^N$, we simulate one *trajectory envelope* in an augmented state space $\{-1,+1,?\}^N$, where the symbol `?` denotes 'either -1 or $+1$'. We call the state of this augmented system the 'summary state'. An example summary state of a six-spin system is `++-?+?`. This summary state is shorthand for the set of states

`++-+++`, `++-++-`, `++--++`, `++--+-` .

The update rule at each step of the Markov chain takes a single spin, enumerates all possible states of the neighbouring spins that are compatible with the current summary state, and, for each of these local scenarios, computes the new value (`+` or `-`) of the spin using Gibbs sampling (coupled to a random number u as in algorithm 32.4). If all these new values agree, then the new value of the updated spin in the summary state is set to the unanimous value (`+` or `-`). Otherwise, the new value of the spin in the summary state is '`?`'. The initial condition, at time T_0, is given by setting all the spins in the summary state to '`?`', which corresponds to considering all possible start configurations.

In the case of a spin system with positive couplings, this summary state simulation will be identical to the simulation of the uppermost state and lowermost states, in the style of Propp and Wilson, with coalescence occuring when all the '`?`' symbols have disappeared. The summary state method can be applied to general spin systems with any couplings. The only shortcoming of this method is that the envelope may describe an unnecessarily large set of states, so there is no guarantee that the summary state algorithm will converge; the time for coalescence to be *detected* may be considerably larger than the actual time taken for the underlying Markov chain to coalesce.

The summary state scheme has been applied to exact sampling in belief networks by Harvey and Neal (2000), and to the triangular antiferromagnetic Ising model by Childs *et al.* (2001). Summary state methods were first introduced by Huber (1998); they also go by the names sandwiching methods and bounding chains.

Further reading

For further reading, impressive pictures of exact samples from other distributions, and generalizations of the exact sampling method, browse the perfectly-random sampling website.[1]

For beautiful exact-sampling demonstrations running live in your web-browser, see Jim Propp's website.[2]

Other uses for coupling

The idea of coupling together Markov chains by having them share a random number generator has other applications beyond exact sampling. Pinto and Neal (2001) have shown that the accuracy of estimates obtained from a Markov chain Monte Carlo simulation (the second problem discussed in section 29.1, p.357), using the estimator

$$\hat{\Phi}_P \equiv \frac{1}{T} \sum_t \phi(\mathbf{x}^{(t)}), \tag{32.1}$$

[1] `http://www.dbwilson.com/exact/`

[2] `http://www.math.wisc.edu/~propp/tiling/www/applets/`

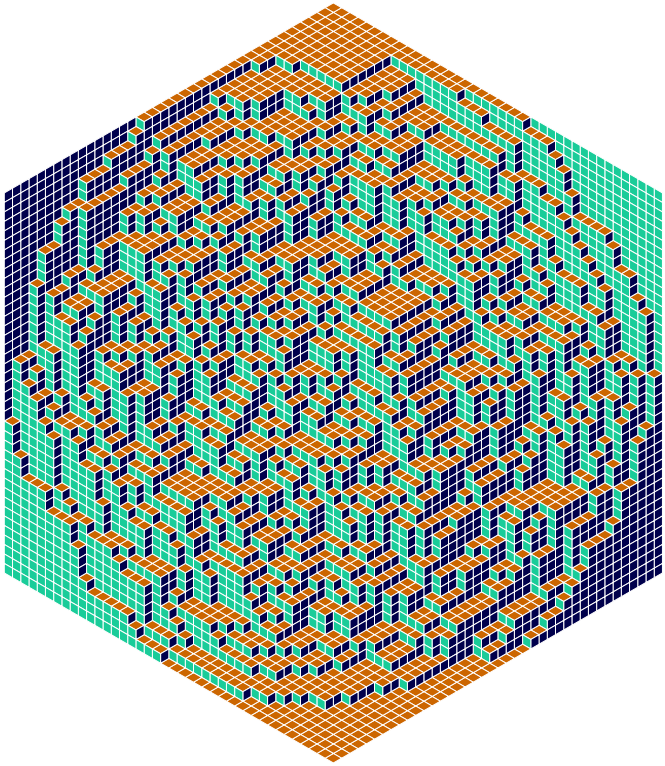


Figure 32.6. A perfectly random tiling of a hexagon by lozenges, provided by J.G. Propp and D.B. Wilson.

can be improved by coupling the chain of interest, which converges to P, to a second chain, which generates samples from a second, simpler distribution, Q. The coupling must be set up in such a way that the states of the two chains are strongly correlated. The idea is that we first estimate the expectations of a function of interest, ϕ, under P and under Q in the normal way (32.1) and compare the estimate under Q, $\hat{\Phi}_Q$, with the true value of the expectation under Q, Φ_Q which we assume can be evaluated exactly. If $\hat{\Phi}_Q$ is an overestimate then it is likely that $\hat{\Phi}_P$ will be an overestimate too. The difference $(\hat{\Phi}_Q - \Phi_Q)$ can thus be used to correct $\hat{\Phi}_P$.

32.4 Exercises

▷ Exercise 32.1.[2, p.421] Is there any relationship between the probability distribution of the time taken for all trajectories to coalesce, and the equilibration time of a Markov chain? Prove that there is a relationship, or find a single chain that can be realized in two different ways that have different coalescence times.

▷ Exercise 32.2.[2] Imagine that Fred ignores the requirement that the random bits used at some time t, in every run from increasingly distant times T_0, must be identical, and makes a coupled-Markov-chain simulator that uses fresh random numbers every time T_0 is changed. Describe what happens if Fred applies his method to the Markov chain that is intended to sample from the uniform distribution over the states 0, 1, and 2, using the Metropolis method, driven by a random bit source as in figure 32.1b.

Exercise 32.3.[5] Investigate the application of perfect sampling to linear regression in Holmes and Mallick (1998) or Holmes and Denison (2002) and try to generalize it.

Exercise 32.4.[3] The concept of coalescence has many applications. Some surnames are more frequent than others, and some die out altogether. Make

a model of this process; how long will it take until everyone has the same surname?

Similarly, variability in any particular portion of the human genome (which forms the basis of forensic DNA fingerprinting) is inherited like a surname. A DNA fingerprint is like a string of surnames. Should the fact that these surnames are subject to coalescences, so that some surnames are by chance more prevalent than others, affect the way in which DNA fingerprint evidence is used in court?

▷ Exercise 32.5.[2] How can you use a coin to create a random ranking of 3 people? Construct a solution that uses exact sampling. For example, you could apply exact sampling to a Markov chain in which the coin is repeatedly used alternately to decide whether to switch first and second, then whether to switch second and third.

Exercise 32.6.[5] Finding the partition function Z of a probability distribution is a difficult problem. Many Markov chain Monte Carlo methods produce valid samples from a distribution without ever finding out what Z is.

Is there any probability distribution and Markov chain such that either the time taken to produce a perfect sample or the number of random bits used to create a perfect sample are related to the value of Z? Are there some situations in which the time to coalescence conveys information about Z?

▶ 32.5 Solutions

Solution to exercise 32.1 (p.420). It is perhaps surprising that there is no direct relationship between the equilibration time and the time to coalescence. We can prove this using the example of the uniform distribution over the integers $\mathcal{A} = \{0, 1, 2, \ldots, 20\}$. A Markov chain that converges to this distribution in exactly one iteration is the chain for which the probability of state s_{t+1} given s_t is the uniform distribution, for all s_t. Such a chain can be coupled to a random number generator in two ways: (a) we could draw a random integer $u \in \mathcal{A}$, and set s_{t+1} equal to u regardless of s_t; or (b) we could draw a random integer $u \in \mathcal{A}$, and set s_{t+1} equal to $(s_t + u) \bmod 21$. Method (b) would produce a cohort of trajectories locked together, similar to the trajectories in figure 32.1, except that no coalescence ever occurs. Thus, while the equilibration times of methods (a) and (b) are both one, the coalescence times are respectively one and infinity.

It seems plausible on the other hand that coalescence time provides some sort of upper bound on equilibration time.

33

Variational Methods

Variational methods are an important technique for the approximation of complicated probability distributions, having applications in statistical physics, data modelling and neural networks.

▶ 33.1 Variational free energy minimization

One method for approximating a complex distribution in a physical system is *mean field theory*. Mean field theory is a special case of a general *variational free energy* approach of Feynman and Bogoliubov which we will now study. The key piece of mathematics needed to understand this method is Gibbs' inequality, which we repeat here.

Gibbs' inequality first appeared in equation (1.24); see also exercise 2.26 (p.37).

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

$$D_{\mathrm{KL}}(Q||P) = \sum_x Q(x) \log \frac{Q(x)}{P(x)}. \tag{33.1}$$

The relative entropy satisfies $D_{\mathrm{KL}}(Q||P) \geq 0$ (Gibbs' inequality) with equality only if $Q = P$. In general $D_{\mathrm{KL}}(Q||P) \neq D_{\mathrm{KL}}(P||Q)$.

In this chapter we will replace the log by ln, and measure the divergence in nats.

Probability distributions in statistical physics

In statistical physics one often encounters probability distributions of the form

$$P(\mathbf{x} \,|\, \beta, \mathbf{J}) = \frac{1}{Z(\beta, \mathbf{J})} \exp[-\beta E(\mathbf{x}; \mathbf{J})] \,, \tag{33.2}$$

where for example the state vector is $\mathbf{x} \in \{-1, +1\}^N$, and $E(\mathbf{x}; \mathbf{J})$ is some energy function such as

$$E(\mathbf{x}; \mathbf{J}) = -\frac{1}{2} \sum_{m,n} J_{mn} x_m x_n - \sum_n h_n x_n. \tag{33.3}$$

The partition function (normalizing constant) is

$$Z(\beta, \mathbf{J}) \equiv \sum_{\mathbf{x}} \exp[-\beta E(\mathbf{x}; \mathbf{J})] \,. \tag{33.4}$$

The probability distribution of equation (33.2) is complex. Not unbearably complex – we can, after all, evaluate $E(\mathbf{x}; \mathbf{J})$ for any particular $\mathbf{x}$ in a time

polynomial in the number of spins. But evaluating the normalizing constant $Z(\beta, \mathbf{J})$ is difficult, as we saw in Chapter 29, and describing the properties of the probability distribution is also hard. Knowing the value of $E(\mathbf{x}; \mathbf{J})$ at a few arbitrary points $\mathbf{x}$, for example, gives no useful information about what the average properties of the system are.

An evaluation of $Z(\beta, \mathbf{J})$ would be particularly desirable because from Z we can derive all the thermodynamic properties of the system.

Variational free energy minimization is a method for *approximating* the complex distribution $P(\mathbf{x})$ by a simpler ensemble $Q(\mathbf{x}; \boldsymbol{\theta})$ that is parameterized by adjustable parameters $\boldsymbol{\theta}$. We adjust these parameters so as to get Q to best approximate P, in some sense. A by-product of this approximation is a lower bound on $Z(\beta, \mathbf{J})$.

The variational free energy

The objective function chosen to measure the quality of the approximation is the *variational free energy*

$$\beta\tilde{F}(\boldsymbol{\theta}) = \sum_{\mathbf{x}} Q(\mathbf{x}; \boldsymbol{\theta}) \ln \frac{Q(\mathbf{x}; \boldsymbol{\theta})}{\exp[-\beta E(\mathbf{x}; \mathbf{J})]}. \tag{33.5}$$

This expression can be manipulated into a couple of interesting forms: first,

$$\beta\tilde{F}(\boldsymbol{\theta}) = \beta \sum_{\mathbf{x}} Q(\mathbf{x}; \boldsymbol{\theta}) E(\mathbf{x}; \mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x}; \boldsymbol{\theta}) \ln \frac{1}{Q(\mathbf{x}; \boldsymbol{\theta})} \tag{33.6}$$

$$\equiv \beta \left\langle E(\mathbf{x}; \mathbf{J}) \right\rangle_Q - S_Q, \tag{33.7}$$

where $\left\langle E(\mathbf{x}; \mathbf{J}) \right\rangle_Q$ is the average of the energy function under the distribution $Q(\mathbf{x}; \boldsymbol{\theta})$, and S_Q is the entropy of the distribution $Q(\mathbf{x}; \boldsymbol{\theta})$ (we set k_{B} to one in the definition of S so that it is identical to the definition of the entropy H in Part I).

Second, we can use the definition of $P(\mathbf{x} \,|\, \beta, \mathbf{J})$ to write:

$$\beta\tilde{F}(\boldsymbol{\theta}) = \sum_{\mathbf{x}} Q(\mathbf{x}; \boldsymbol{\theta}) \ln \frac{Q(\mathbf{x}; \boldsymbol{\theta})}{P(\mathbf{x} \,|\, \beta, \mathbf{J})} - \ln Z(\beta, \mathbf{J}) \tag{33.8}$$

$$= D_{\mathrm{KL}}(Q||P) + \beta F, \tag{33.9}$$

where F is the true free energy, defined by

$$\beta F \equiv -\ln Z(\beta, \mathbf{J}), \tag{33.10}$$

and $D_{\mathrm{KL}}(Q||P)$ is the relative entropy between the approximating distribution $Q(\mathbf{x}; \boldsymbol{\theta})$ and the true distribution $P(\mathbf{x} \,|\, \beta, \mathbf{J})$. Thus by Gibbs' inequality, the variational free energy $\tilde{F}(\boldsymbol{\theta})$ is bounded below by F and attains this value only for $Q(\mathbf{x}; \boldsymbol{\theta}) = P(\mathbf{x} \,|\, \beta, \mathbf{J})$.

Our strategy is thus to vary $\boldsymbol{\theta}$ in such a way that $\beta\tilde{F}(\boldsymbol{\theta})$ is minimized. The approximating distribution then gives a simplified approximation to the true distribution that may be useful, and the value of $\beta\tilde{F}(\boldsymbol{\theta})$ will be an upper bound for βF. Equivalently, $\tilde{Z} \equiv e^{-\beta\tilde{F}(\boldsymbol{\theta})}$ is a lower bound for Z.

Can the objective function $\beta\tilde{F}$ be evaluated?

We have already agreed that the evaluation of various interesting sums over $\mathbf{x}$ is intractable. For example, the partition function

$$Z = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x}; \mathbf{J})), \tag{33.11}$$

the energy

$$\langle E \rangle_P = \frac{1}{Z} \sum_{\mathbf{x}} E(\mathbf{x}; \mathbf{J}) \exp(-\beta E(\mathbf{x}; \mathbf{J})) \,, \tag{33.12}$$

and the entropy

$$S \equiv \sum_{\mathbf{x}} P(\mathbf{x} \,|\, \beta, \mathbf{J}) \ln \frac{1}{P(\mathbf{x} \,|\, \beta, \mathbf{J})} \tag{33.13}$$

are all presumed to be impossible to evaluate. So why should we suppose that this objective function $\beta \tilde{F}(\boldsymbol{\theta})$, which is also defined in terms of a sum over all $\mathbf{x}$ (33.5), should be a convenient quantity to deal with? Well, for a range of interesting energy functions, and for sufficiently simple approximating distributions, the variational free energy *can* be efficiently evaluated.

▶ 33.2 Variational free energy minimization for spin systems

An example of a tractable variational free energy is given by the spin system whose energy function was given in equation (33.3), which we can approximate with a *separable* approximating distribution,

$$Q(\mathbf{x}; \mathbf{a}) = \frac{1}{Z_Q} \exp\left(\sum_n a_n x_n \right). \tag{33.14}$$

The variational parameters $\boldsymbol{\theta}$ of the variational free energy (33.5) are the components of the vector $\mathbf{a}$. To evaluate the variational free energy we need the entropy of this distribution,

$$S_Q = \sum_{\mathbf{x}} Q(\mathbf{x}; \mathbf{a}) \ln \frac{1}{Q(\mathbf{x}; \mathbf{a})}, \tag{33.15}$$

and the mean of the energy,

$$\langle E(\mathbf{x}; \mathbf{J}) \rangle_Q = \sum_{\mathbf{x}} Q(\mathbf{x}; \mathbf{a}) E(\mathbf{x}; \mathbf{J}). \tag{33.16}$$

The entropy of the separable approximating distribution is simply the sum of the entropies of the individual spins (exercise 4.2, p.68),

$$S_Q = \sum_n H_2^{(e)}(q_n), \tag{33.17}$$

where q_n is the probability that spin n is $+1$,

$$q_n = \frac{e^{a_n}}{e^{a_n} + e^{-a_n}} = \frac{1}{1 + \exp(-2a_n)}, \tag{33.18}$$

and

$$H_2^{(e)}(q) = q \ln \frac{1}{q} + (1 - q) \ln \frac{1}{(1 - q)}. \tag{33.19}$$

The mean energy under Q is easy to obtain because $\sum_{m,n} J_{mn} x_m x_n$ is a sum of terms each involving the product of two *independent* random variables. (There are no self-couplings, so $J_{mn} = 0$ when $m = n$.) If we define the mean value of x_n to be $\bar{x}_n$, which is given by

$$\bar{x}_n = \frac{e^{a_n} - e^{-a_n}}{e^{a_n} + e^{-a_n}} = \tanh(a_n) = 2q_n - 1, \tag{33.20}$$

we obtain

$$\langle E(\mathbf{x};\mathbf{J})\rangle_Q = \sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a}) \left[-\frac{1}{2} \sum_{m,n} J_{mn} x_m x_n - \sum_n h_n x_n \right] \quad (33.21)$$

$$= -\frac{1}{2} \sum_{m,n} J_{mn} \bar{x}_m \bar{x}_n - \sum_n h_n \bar{x}_n. \quad (33.22)$$

So the variational free energy is given by

$$\beta \tilde{F}(\mathbf{a}) = \beta \langle E(\mathbf{x};\mathbf{J})\rangle_Q - S_Q = \beta \left(-\frac{1}{2} \sum_{m,n} J_{mn} \bar{x}_m \bar{x}_n - \sum_n h_n \bar{x}_n \right) - \sum_n H_2^{(e)}(q_n). \quad (33.23)$$

We now consider minimizing this function with respect to the variational parameters $\mathbf{a}$. If $q = 1/(1+e^{-2a})$, the derivative of the entropy is

$$\frac{\partial}{\partial q} H_2^e(q) = \ln \frac{1-q}{q} = -2a. \quad (33.24)$$

So we obtain

$$\frac{\partial}{\partial a_m} \beta \tilde{F}(\mathbf{a}) = \beta \left[-\sum_n J_{mn} \bar{x}_n - h_m \right] \left(2 \frac{\partial q_m}{\partial a_m} \right) - \ln \left(\frac{1-q_m}{q_m} \right) \left(\frac{\partial q_m}{\partial a_m} \right)$$

$$= 2 \left(\frac{\partial q_m}{\partial a_m} \right) \left[-\beta \left(\sum_n J_{mn} \bar{x}_n + h_m \right) + a_m \right]. \quad (33.25)$$

This derivative is equal to zero when

$$a_m = \beta \left(\sum_n J_{mn} \bar{x}_n + h_m \right). \quad (33.26)$$

So $\tilde{F}(\mathbf{a})$ is extremized at any point that satisfies equation (33.26) and

$$\bar{x}_n = \tanh(a_n). \quad (33.27)$$

The variational free energy $\tilde{F}(\mathbf{a})$ may be a multimodal function, in which case each stationary point (maximum, minimum or saddle) will satisfy equations (33.26) and (33.27). One way of using these equations, in the case of a system with an arbitrary coupling matrix $\mathbf{J}$, is to update each parameter a_m and the corresponding value of $\bar{x}_m$ using equation (33.26), one at a time. This *asynchronous updating of the parameters* is guaranteed to decrease $\beta \tilde{F}(\mathbf{a})$.

Equations (33.26) and (33.27) may be recognized as the *mean field* equations for a spin system. The variational parameter a_n may be thought of as the strength of a fictitious field applied to an isolated spin n. Equation (33.27) describes the mean response of spin n, and equation (33.26) describes how the field a_m is set in response to the mean state of all the other spins.

The variational free energy derivation is a helpful viewpoint for mean field theory for two reasons.

1. This approach associates an objective function $\beta \tilde{F}$ with the mean field equations; such an objective function is useful because it can help identify alternative dynamical systems that minimize the same function.

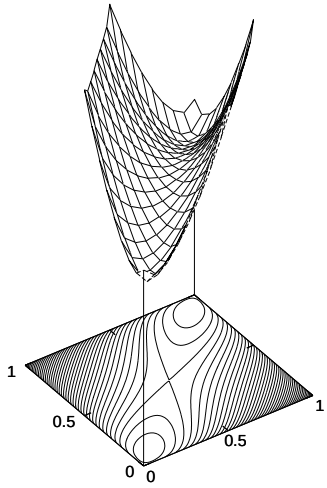


Figure 33.1. The variational free energy of the two-spin system whose energy is $E(\mathbf{x}) = -x_1 x_2$, as a function of the two variational parameters q_1 and q_2. The inverse-temperature is $\beta = 1.44$. The function plotted is

$$\beta \tilde{F} = -\beta \bar{x}_1 \bar{x}_2 - H_2^{(e)}(q_1) - H_2^{(e)}(q_2),$$

where $\bar{x}_n = 2q_n - 1$. Notice that for fixed q_2 the function is convex $\smile$ with respect to q_1, and for fixed q_1 it is convex $\smile$ with respect to q_2.

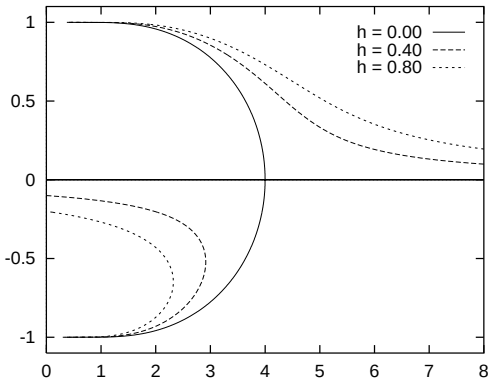


Figure 33.2. Solutions of the variational free energy extremization problem for the Ising model, for three different applied fields h. Horizontal axis: temperature $T = 1/\beta$. Vertical axis: magnetization $\bar{x}$. The critical temperature found by mean field theory is $T_c^{\rm mft} = 4$.

2. The theory is readily generalized to other approximating distributions. We can imagine introducing a more complex approximation $Q(\mathbf{x}; \boldsymbol{\theta})$ that might for example capture correlations among the spins instead of modelling the spins as independent. One could then evaluate the variational free energy and optimize the parameters $\boldsymbol{\theta}$ of this more complex approximation. The more degrees of freedom the approximating distribution has, the tighter the bound on the free energy becomes. However, if the complexity of an approximation is increased, the evaluation of either the mean energy or the entropy typically becomes more challenging.

▶ 33.3 Example: mean field theory for the ferromagnetic Ising model

In the simple Ising model studied in Chapter 31, every coupling J_{mn} is equal to J if m and n are neighbours and zero otherwise. There is an applied field $h_n = h$ that is the same for all spins. A very simple approximating distribution is one with just a single variational parameter a, which defines a separable distribution

$$Q(\mathbf{x}; a) = \frac{1}{Z_Q} \exp\left(\sum_n a x_n \right) \tag{33.28}$$

in which all spins are independent and have the same probability

$$q_n = \frac{1}{1 + \exp(-2a)} \tag{33.29}$$

of being up. The mean magnetization is

$$\bar{x} = \tanh(a) \tag{33.30}$$

and the equation (33.26) which defines the minimum of the variational free energy becomes

$$a = \beta \left(CJ\bar{x} + h \right), \tag{33.31}$$

where C is the number of couplings that a spin is involved in – $C = 4$ in the case of a rectangular two-dimensional Ising model. We can solve equations (33.30) and (33.31) for $\bar{x}$ numerically – in fact, it is easiest to vary $\bar{x}$ and solve for β – and obtain graphs of the free energy minima and maxima as a function of temperature as shown in figure 33.2. The solid line shows $\bar{x}$ versus $T = 1/\beta$ for the case $C = 4, J = 1$.

When $h = 0$, there is a pitchfork bifurcation at a critical temperature $T_c^{\rm mft}$. [A pitchfork bifurcation is a transition like the one shown by the solid lines in

figure 33.2, from a system with one minimum as a function of a (on the right) to a system (on the left) with two minima and one maximum; the maximum is the middle one of the three lines. The solid lines look like a pitchfork.] Above this temperature, there is only one minimum in the variational free energy, at $a = 0$ and $\bar{x} = 0$; this minimum corresponds to an approximating distribution that is uniform over all states. Below the critical temperature, there are two minima corresponding to approximating distributions that are symmetry-broken, with all spins more likely to be up, or all spins more likely to be down. The state $\bar{x} = 0$ persists as a stationary point of the variational free energy, but now it is a local *maximum* of the variational free energy.

When $h > 0$, there is a global variational free energy minimum at any temperature for a positive value of $\bar{x}$, shown by the upper dotted curves in figure 33.2. As long as $h < JC$, there is also a second local minimum in the free energy, if the temperature is sufficiently small. This second minimum corresponds to a self-preserving state of magnetization in the opposite direction to the applied field. The temperature at which the second minimum appears is smaller than $T_c^{\rm mft}$, and when it appears, it is accompanied by a saddle point located between the two minima. A name given to this type of bifurcation is a saddle-node bifurcation.

The variational free energy per spin is given by

$$\beta\tilde{F} = \beta\left(-\frac{C}{2}J\bar{x}^2 - h\bar{x}\right) - H_2^{(e)}\left(\frac{\bar{x}+1}{2}\right). \tag{33.32}$$

Exercise 33.1.[2] Sketch the variational free energy as a function of its one parameter $\bar{x}$ for a variety of values of the temperature T and the applied field h.

Figure 33.2 reproduces the key properties of the real Ising system – that, for $h = 0$, there is a critical temperature below which the system has long-range order, and that it can adopt one of two macroscopic states. However, by probing a little more we can reveal some inadequacies of the variational approximation. To start with, the critical temperature $T_c^{\rm mft}$ is 4, which is nearly a factor of 2 greater than the true critical temperature $T_c = 2.27$. Also, the variational model has equivalent properties in any number of dimensions, including $d = 1$, where the true system does not have a phase transition. So the bifurcation at $T_c^{\rm mft}$ should not be described as a phase transition.

For the case $h = 0$ we can follow the trajectory of the global minimum as a function of β and find the entropy, heat capacity and fluctuations of the approximating distribution and compare them with those of a real 8×8 fragment using the matrix method of Chapter 31. As shown in figure 33.3, one of the biggest differences is in the fluctuations in energy. The real system has large fluctuations near the critical temperature, whereas the approximating distribution has no correlations among its spins and thus has an energy-variance which scales simply linearly with the number of spins.

▶ 33.4 Variational methods in inference and data modelling

In statistical data modelling we are interested in the posterior probability distribution of a parameter vector $\mathbf{w}$ given data D and model assumptions $\mathcal{H}$, $P(\mathbf{w} \mid D, \mathcal{H})$.

$$P(\mathbf{w} \mid D, \mathcal{H}) = \frac{P(D \mid \mathbf{w}, \mathcal{H})P(\mathbf{w} \mid \mathcal{H})}{P(D \mid \mathcal{H})}. \tag{33.33}$$

In traditional approaches to model fitting, a single parameter vector $\mathbf{w}$ is optimized to find the mode of this distribution. What is really of interest is

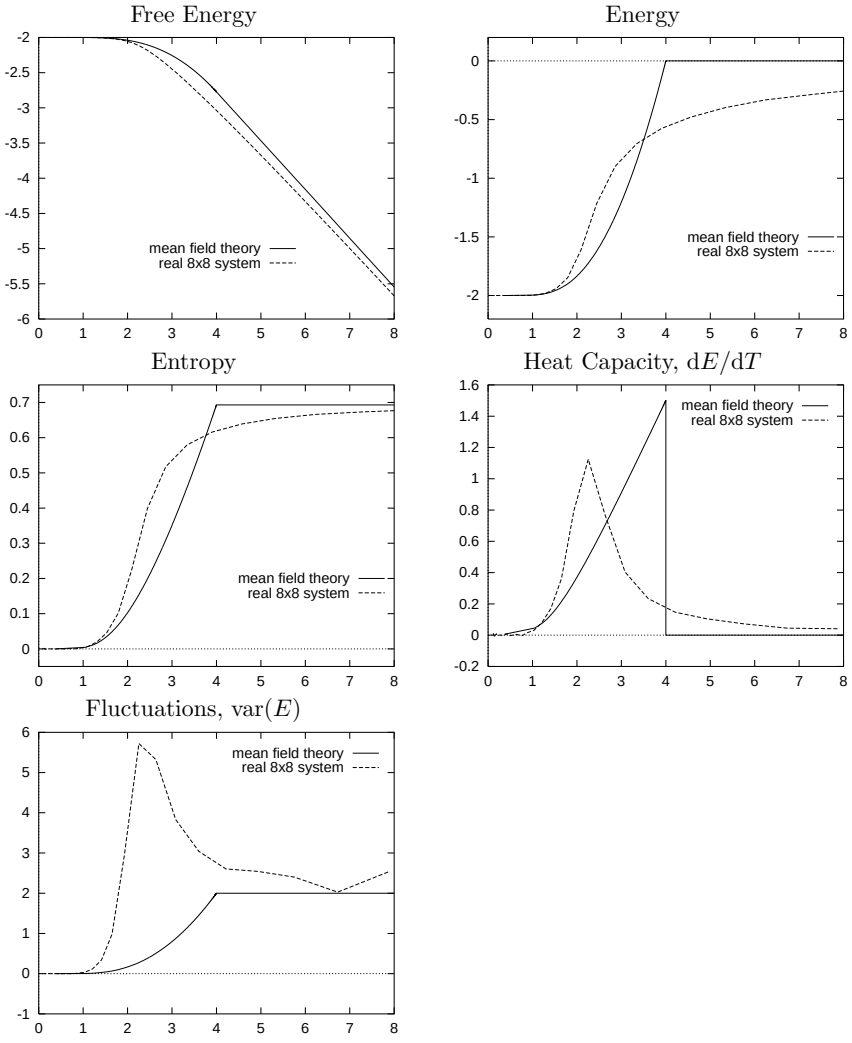


Figure 33.3. Comparison of approximating distribution's properties with those of a real 8×8 fragment. Notice that the variational free energy of the approximating distribution is indeed an upper bound on the free energy of the real system. All quantities are shown 'per spin'.

the whole distribution. We may also be interested in its normalizing constant $P(D\,|\,\mathcal{H})$ if we wish to do model comparison. The probability distribution $P(\mathbf{w}\,|\,D,\mathcal{H})$ is often a complex distribution. In a variational approach to inference, we introduce an approximating probability distribution over the parameters, $Q(\mathbf{w};\boldsymbol{\theta})$, and optimize this distribution (by varying its own parameters $\boldsymbol{\theta}$) so that it approximates the posterior distribution of the parameters $P(\mathbf{w}\,|\,D,\mathcal{H})$ well.

One objective function we may choose to measure the quality of the approximation is the variational free energy

$$\tilde{F}(\boldsymbol{\theta}) = \int \mathrm{d}^k\mathbf{w}\; Q(\mathbf{w};\boldsymbol{\theta}) \ln \frac{Q(\mathbf{w};\boldsymbol{\theta})}{P(D\,|\,\mathbf{w},\mathcal{H})P(\mathbf{w}\,|\,\mathcal{H})}. \tag{33.34}$$

The denominator $P(D\,|\,\mathbf{w},\mathcal{H})P(\mathbf{w}\,|\,\mathcal{H})$ is, within a multiplicative constant, the posterior probability $P(\mathbf{w}\,|\,D,\mathcal{H}) = P(D\,|\,\mathbf{w},\mathcal{H})P(\mathbf{w}\,|\,\mathcal{H})/P(D\,|\,\mathcal{H})$. So the variational free energy $\tilde{F}(\boldsymbol{\theta})$ can be viewed as the sum of $-\ln P(D\,|\,\mathcal{H})$ and the relative entropy between $Q(\mathbf{w};\boldsymbol{\theta})$ and $P(\mathbf{w}\,|\,D,\mathcal{H})$. $\tilde{F}(\boldsymbol{\theta})$ is bounded below by $-\ln P(D\,|\,\mathcal{H})$ and only attains this value for $Q(\mathbf{w};\boldsymbol{\theta}) = P(\mathbf{w}\,|\,D,\mathcal{H})$. For certain models and certain approximating distributions, this free energy, and its derivatives with respect to the approximating distribution's parameters, can be evaluated.

The approximation of posterior probability distributions using variational free energy minimization provides a useful approach to approximating Bayesian inference in a number of fields ranging from neural networks to the decoding of error-correcting codes (Hinton and van Camp, 1993; Hinton and Zemel, 1994; Dayan *et al.*, 1995; Neal and Hinton, 1998; MacKay, 1995a). The method is sometimes called *ensemble learning* to contrast it with traditional learning processes in which a single parameter vector is optimized. Another name for it is *variational Bayes*. Let us examine how ensemble learning works in the simple case of a Gaussian distribution.

▶ 33.5 The case of an unknown Gaussian: approximating the posterior distribution of μ and σ

We will fit an approximating ensemble $Q(\mu,\sigma)$ to the posterior distribution that we studied in Chapter 24,

$$P(\mu,\sigma\,|\,\{x_n\}_{n=1}^N) = \frac{P(\{x_n\}_{n=1}^N\,|\,\mu,\sigma)P(\mu,\sigma)}{P(\{x_n\}_{n=1}^N)} \tag{33.35}$$

$$= \frac{\frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{N(\mu-\bar{x})^2+S}{2\sigma^2}\right) \frac{1}{\sigma_\mu}\frac{1}{\sigma}}{P(\{x_n\}_{n=1}^N)}. \tag{33.36}$$

We make the single assumption that the approximating ensemble is separable in the form $Q(\mu,\sigma) = Q_\mu(\mu)Q_\sigma(\sigma)$. No restrictions on the functional form of $Q_\mu(\mu)$ and $Q_\sigma(\sigma)$ are made.

We write down a variational free energy,

$$\tilde{F}(Q) = \int \mathrm{d}\mu\,\mathrm{d}\sigma\; Q_\mu(\mu)Q_\sigma(\sigma) \ln \frac{Q_\mu(\mu)Q_\sigma(\sigma)}{P(D\,|\,\mu,\sigma)P(\mu,\sigma)}. \tag{33.37}$$

We can find the optimal separable distribution Q by considering separately the optimization of $\tilde{F}$ over $Q_\mu(\mu)$ for fixed $Q_\sigma(\sigma)$, and then the optimization of $Q_\sigma(\sigma)$ for fixed $Q_\mu(\mu)$.

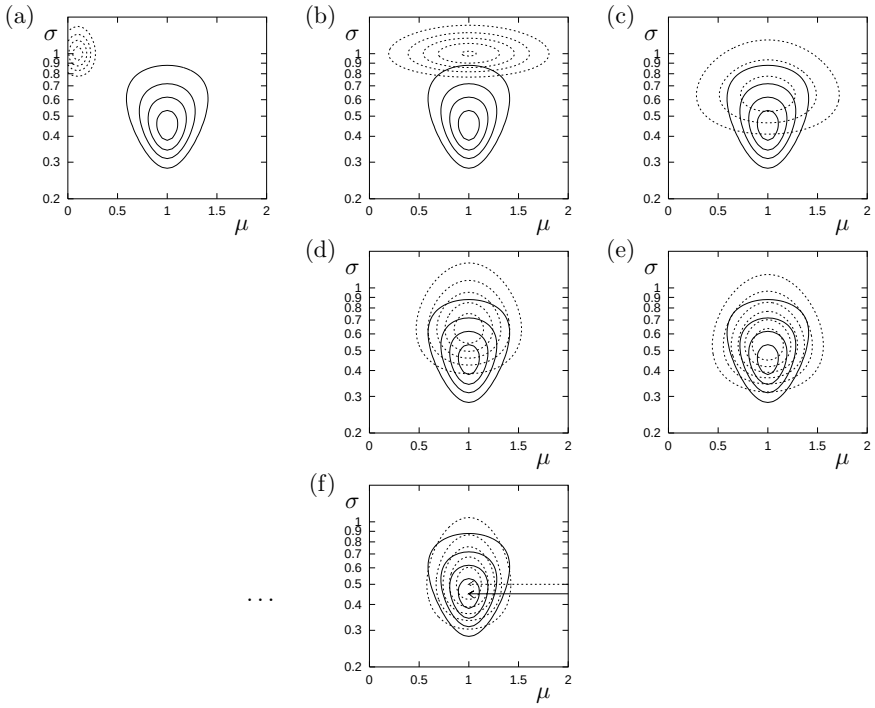


Figure 33.4. Optimization of an approximating distribution. The posterior distribution $P(\mu, \sigma \mid \{x_n\})$, which is the same as that in figure 24.1, is shown by solid contours. (a) Initial condition. The approximating distribution $Q(\mu, \sigma)$ (dotted contours) is an arbitrary separable distribution. (b) Q_μ has been updated, using equation (33.41). (c) Q_σ has been updated, using equation (33.44). (d) Q_μ updated again. (e) Q_σ updated again. (f) Converged approximation (after 15 iterations). The arrows point to the peaks of the two distributions, which are at $\sigma_N = 0.45$ (for P) and $\sigma_{N-1} = 0.5$ (for Q).

Optimization of $Q_\mu(\mu)$

As a functional of $Q_\mu(\mu)$, $\tilde{F}$ is:

$$\tilde{F} = -\int \mathrm{d}\mu\, Q_\mu(\mu) \left[\int \mathrm{d}\sigma\, Q_\sigma(\sigma) \ln P(D \mid \mu, \sigma) + \ln[P(\mu)/Q_\mu(\mu)] \right] + \kappa \quad (33.38)$$

$$= \int \mathrm{d}\mu\, Q_\mu(\mu) \left[\int \mathrm{d}\sigma\, Q_\sigma(\sigma) N\beta \frac{1}{2}(\mu - \bar{x})^2 + \ln Q_\mu(\mu) \right] + \kappa', \quad (33.39)$$

where $\beta \equiv 1/\sigma^2$ and κ denote constants that do not depend on $Q_\mu(\mu)$. The dependence on Q_σ thus collapses down to a simple dependence on the mean

$$\bar{\beta} \equiv \int \mathrm{d}\sigma\, Q_\sigma(\sigma) 1/\sigma^2. \quad (33.40)$$

Now we can recognize the function $-N\bar{\beta}\frac{1}{2}(\mu - \bar{x})^2$ as the logarithm of a Gaussian identical to the posterior distribution for a particular value of $\beta = \bar{\beta}$. Since a relative entropy $\int Q \ln(Q/P)$ is minimized by setting $Q = P$, we can immediately write down the distribution $Q_\mu^{\mathrm{opt}}(\mu)$ that minimizes $\tilde{F}$ for fixed Q_σ:

$$Q_\mu^{\mathrm{opt}}(\mu) = P(\mu \mid D, \bar{\beta}, \mathcal{H}) = \mathrm{Normal}(\mu; \bar{x}, \sigma^2_{\mu|D}). \quad (33.41)$$

where $\sigma^2_{\mu|D} = 1/(N\bar{\beta})$.

Optimization of $Q_\sigma(\sigma)$

We represent $Q_\sigma(\sigma)$ using the density over β, $Q_\sigma(\beta) \equiv Q_\sigma(\sigma)\,|\mathrm{d}\sigma/\mathrm{d}\beta|$. As a functional of $Q_\sigma(\beta)$, $\tilde{F}$ is (neglecting additive constants):

The prior $P(\sigma) \propto 1/\sigma$ transforms to $P(\beta) \propto 1/\beta$.

$$\tilde{F} = -\int \mathrm{d}\beta\, Q_\sigma(\beta) \left[\int \mathrm{d}\mu\, Q_\mu(\mu) \ln P(D \mid \mu, \sigma) + \ln[P(\beta)/Q_\sigma(\beta)] \right] \quad (33.42)$$

$$= \int \mathrm{d}\beta\, Q_\sigma(\beta) \left[(N\sigma^2_{\mu|D} + S)\beta/2 - \left(\tfrac{N}{2} - 1\right) \ln \beta + \ln Q_\sigma(\beta) \right], \quad (33.43)$$

where the integral over μ is performed assuming $Q_\mu(\mu) = Q_\mu^{\rm opt}(\mu)$. Here, the β-dependent expression in square brackets can be recognized as the logarithm of a gamma distribution over β – see equation (23.15) – giving as the distribution that minimizes $\tilde{F}$ for fixed Q_μ:

$$Q_\sigma^{\rm opt}(\beta) = \Gamma(\beta; b', c'), \tag{33.44}$$

with

$$\frac{1}{b'} = \frac{1}{2}(N\sigma_{\mu|D}^2 + S) \quad \text{and} \quad c' = \frac{N}{2}. \tag{33.45}$$

In figure 33.4, these two update rules (33.41, 33.44) are applied alternately, starting from an arbitrary initial condition. The algorithm converges to the optimal approximating ensemble in a few iterations.

Direct solution for the joint optimum $Q_\mu(\mu)Q_\sigma(\sigma)$

In this problem, we do not need to resort to iterative computation to find the optimal approximating ensemble. Equations (33.41) and (33.44) define the optimum implicitly. We must simultaneously have $\sigma_{\mu|D}^2 = 1/(N\bar{\beta})$, and $\bar{\beta} = b'c'$. The solution is:

$$1/\bar{\beta} = S/(N-1). \tag{33.46}$$

This is similar to the true posterior distribution of σ, which is a gamma distribution with $c' = \frac{N-1}{2}$ and $1/b' = S/2$ (see equation 24.13). This true posterior also has a mean value of β satisfying $1/\bar{\beta} = S/(N-1)$; the only difference is that the approximating distribution's parameter c' is too large by $1/2$.

> The approximations given by variational free energy minimization always tend to be more compact than the true distribution.

In conclusion, ensemble learning gives an approximation to the posterior that agrees nicely with the conventional estimators. The approximate posterior distribution over β is a gamma distribution with mean $\bar{\beta}$ corresponding to a variance of $\sigma^2 = S/(N-1) = \sigma_{N-1}^2$. And the approximate posterior distribution over μ is a Gaussian with mean $\bar{x}$ and standard deviation $\sigma_{N-1}/\sqrt{N}$.

The variational free energy minimization approach has the nice property that it is parameterization-independent; it avoids the problem of basis-dependence from which MAP methods and Laplace's method suffer.

A convenient software package for automatic implementation of variational inference in graphical models is `VIBES` (Bishop *et al.*, 2002). It plays the same role for variational inference as `BUGS` plays for Monte Carlo inference.

▶ 33.6 Interlude

One of my students asked:

> How do you ever come up with a useful approximating distribution, given that the true distribution is so complex you can't compute it directly?

Let's answer this question in the context of Bayesian data modelling. Let the 'true' distribution of interest be the posterior probability distribution over a set of parameters $\mathbf{x}$, $P(\mathbf{x} \mid D)$. A standard data modelling practice is to find a single, 'best-fit' setting of the parameters, $\mathbf{x}^*$, for example, by finding the maximum of the likelihood function $P(D \mid \mathbf{x})$, or of the posterior distribution.

One interpretation of this standard practice is that the full description of our knowledge about $\mathbf{x}$, $P(\mathbf{x} \mid D)$, is being approximated by a delta-function, a probability distribution concentrated on $\mathbf{x}^*$. From this perspective, *any* approximating distribution $Q(\mathbf{x}; \boldsymbol{\theta})$, no matter how crummy it is, *has* to be an improvement on the spike produced by the standard method! So even if we use only a simple Gaussian approximation, we are doing well.

We now study an application of the variational approach to a realistic example – data clustering.

▶ 33.7 K-means clustering and the expectation–maximization algorithm as a variational method

In Chapter 20, we introduced the soft K-means clustering algorithm, version 1. In Chapter 22, we introduced versions 2 and 3 of this algorithm, and motivated the algorithm as a maximum likelihood algorithm.

K-means clustering is an example of an 'expectation–maximization' (EM) algorithm, with the two steps, which we called 'assignment' and 'update', being known as the 'E-step' and the 'M-step' respectively.

We now give a more general view of K-means clustering, due to Neal and Hinton (1998), in which the algorithm is shown to optimize a variational objective function. Neal and Hinton's derivation applies to any EM algorithm.

The probability of everything

Let the parameters of the mixture model – the means, standard deviations, and weights – be denoted by $\boldsymbol{\theta}$. For each data point, there is a missing variable (also known as a latent variable), the class label k_n for that point. The probability of everything, given our assumed model $\mathcal{H}$, is

$$P(\{\mathbf{x}^{(n)}, k_n\}_{n=1}^N, \boldsymbol{\theta} \mid \mathcal{H}) \;=\; P(\boldsymbol{\theta} \mid \mathcal{H}) \prod_{n=1}^{N} \Big[P(\mathbf{x}^{(n)} \mid k_n, \boldsymbol{\theta}) P(k_n \mid \boldsymbol{\theta}) \Big]. \quad (33.47)$$

The posterior probability of everything, given the data, is proportional to the probability of everything:

$$P(\{k_n\}_{n=1}^N, \boldsymbol{\theta} \mid \{\mathbf{x}^{(n)}\}_{n=1}^N, \mathcal{H}) \;=\; \frac{P(\{\mathbf{x}^{(n)}, k_n\}_{n=1}^N, \boldsymbol{\theta} \mid \mathcal{H})}{P(\{\mathbf{x}^{(n)}\}_{n=1}^N \mid \mathcal{H})}. \quad (33.48)$$

We now approximate this posterior distribution by a separable distribution

$$Q_k(\{k_n\}_{n=1}^N)\, Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}), \quad (33.49)$$

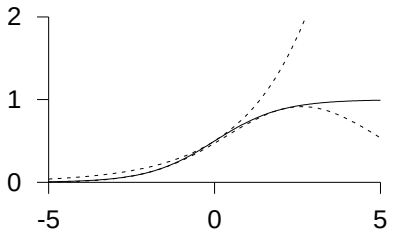
and define a variational free energy in the usual way:

$$\tilde{F}(Q_k, Q_{\boldsymbol{\theta}}) = \sum_{\{k_n\}} \int \mathrm{d}^D \boldsymbol{\theta}\; Q_k(\{k_n\}_{n=1}^N)\, Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln \frac{Q_k(\{k_n\}_{n=1}^N)\, Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})}{P(\{\mathbf{x}^{(n)}, k_n\}_{n=1}^N, \boldsymbol{\theta} \mid \mathcal{H})}. \quad (33.50)$$

$\tilde{F}$ is bounded below by minus the evidence, $\ln P(\{\mathbf{x}^{(n)}\}_{n=1}^N \mid \mathcal{H})$. We can now make an iterative algorithm with an 'assignment' step and an 'update' step. In the assignment step, $Q_k(\{k_n\}_{n=1}^N)$ is adjusted to reduce $\tilde{F}$, for fixed $Q_{\boldsymbol{\theta}}$; in the update step, $Q_{\boldsymbol{\theta}}$ is adjusted to reduce $\tilde{F}$, for fixed Q_k.

If we wish to obtain exactly the soft K-means algorithm, we impose a further constraint on our approximating distribution: $Q_{\boldsymbol{\theta}}$ is constrained to be a delta function centred on a point estimate of $\boldsymbol{\theta}$, $\boldsymbol{\theta} = \boldsymbol{\theta}^*$:

$$Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*). \quad (33.51)$$



Upper bound

$$\frac{1}{1+e^{-a}} \leq \exp(\mu a - H_2^e(\mu)) \qquad \mu \in [0,1]$$

Lower bound

$$\frac{1}{1+e^{-a}} \geq g(\nu)\exp\left[(a-\nu)/2 - \lambda(\nu)(a^2-\nu^2)\right]$$

where $\lambda(\nu) = [g(\nu) - 1/2]/2\nu$.

Figure 33.5. Illustration of the Jaakkola–Jordan variational method. Upper and lower bounds on the logistic function (solid line)

$$g(a) \equiv \frac{1}{1+e^{-a}}.$$

These upper and lower bounds are exponential or Gaussian functions of a, and so easier to integrate over. The graph shows the sigmoid function and upper and lower bounds with $\mu = 0.505$ and $\nu = -2.015$.

Unfortunately, this distribution contributes to the variational free energy an infinitely large integral $\int \mathrm{d}^D\boldsymbol{\theta}\, Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln Q_{\boldsymbol{\theta}}(\boldsymbol{\theta})$, so we'd better leave that term out of $\tilde{F}$, treating it as an additive constant. [Using a delta function $Q_{\boldsymbol{\theta}}$ is not a good idea if our aim is to minimize $\tilde{F}$!] Moving on, our aim is to derive the soft K-means algorithm.

▷ Exercise 33.2.[2] Show that, given $Q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$, the optimal Q_k, in the sense of minimizing $\tilde{F}$, is a separable distribution in which the probability that $k_n = k$ is given by the responsibility $r_k^{(n)}$.

▷ Exercise 33.3.[3] Show that, given a separable Q_k as described above, the optimal $\boldsymbol{\theta}^*$, in the sense of minimizing $\tilde{F}$, is obtained by the update step of the soft K-means algorithm. (Assume a uniform prior on $\boldsymbol{\theta}$.)

Exercise 33.4.[4] We can instantly improve on the infinitely large value of $\tilde{F}$ achieved by soft K-means clustering by allowing $Q_{\boldsymbol{\theta}}$ to be a more general distribution than a delta-function. Derive an update step in which $Q_{\boldsymbol{\theta}}$ is allowed to be a separable distribution, a product of $Q_\mu(\{\mu\})$, $Q_\sigma(\{\sigma\})$, and $Q_\pi(\pi)$. Discuss whether this generalized algorithm still suffers from soft K-means's 'kaboom' problem, where the algorithm glues an ever-shrinking Gaussian to one data point.

Sadly, while it sounds like a promising generalization of the algorithm to allow $Q_{\boldsymbol{\theta}}$ to be a non-delta-function, and the 'kaboom' problem goes away, other artefacts can arise in this approximate inference method, involving local minima of $\tilde{F}$. For further reading, see (MacKay, 1997a; MacKay, 2001).

▶ 33.8 Variational methods other than free energy minimization

There are other strategies for approximating a complicated distribution $P(\mathbf{x})$, in addition to those based on minimizing the relative entropy between an approximating distribution, Q, and P. One approach pioneered by Jaakkola and Jordan is to create adjustable upper and lower bounds Q^U and Q^L to P, as illustrated in figure 33.5. These bounds (which are unnormalized densities) are parameterized by variational parameters which are adjusted in order to obtain the tightest possible fit. The lower bound can be adjusted to *maximize*

$$\sum_{\mathbf{x}} Q^L(\mathbf{x}), \tag{33.52}$$

and the upper bound can be adjusted to *minimize*

$$\sum_{\mathbf{x}} Q^U(\mathbf{x}). \tag{33.53}$$

Using the normalized versions of the optimized bounds we then compute approximations to the predictive distributions. Further reading on such methods can be found in the references (Jaakkola and Jordan, 2000a; Jaakkola and Jordan, 2000b; Jaakkola and Jordan, 1996; Gibbs and MacKay, 2000).

Further reading

The Bethe and Kikuchi free energies

In Chapter 26 we discussed the sum–product algorithm for functions of the factor-graph form (26.1). If the factor graph is tree-like, the sum–product algorithm converges and correctly computes the marginal function of any variable x_n and can also yield the joint marginal function of subsets of variables that appear in a common factor, such as $\mathbf{x}_m$.

The sum–product algorithm may also be applied to factor graphs that are not tree-like. If the algorithm converges to a fixed point, it has been shown that that fixed point is a stationary point (usually a minimum) of a function of the messages called the Kikuchi free energy. In the special case where all factors in factor graph are functions of one or two variables, the Kikuchi free energy is called the Bethe free energy.

For articles on this idea, and new approximate inference algorithms motivated by it, see Yedidia (2000); Yedidia *et al.* (2000); Welling and Teh (2001); Yuille (2001); Yedidia *et al.* (2001b); Yedidia *et al.* (2001a).

▶ 33.9 Further exercises

Exercise 33.5.[2, p.435] This exercise explores the assertion, made above, that the approximations given by variational free energy minimization always tend to be more compact than the true distribution. Consider a two dimensional Gaussian distribution $P(\mathbf{x})$ with axes aligned with the directions $\mathbf{e}^{(1)} = (1, 1)$ and $\mathbf{e}^{(2)} = (1, -1)$. Let the variances in these two directions be σ_1^2 and σ_2^2. What is the optimal variance if this distribution is approximated by a *spherical* Gaussian with variance σ_Q^2, optimized by variational free energy minimization? If we instead optimized the objective function

$$G = \int \mathrm{d}\mathbf{x}\, P(\mathbf{x}) \ln \frac{P(\mathbf{x})}{Q(\mathbf{x}; \sigma^2)}, \tag{33.54}$$

what would be the optimal value of σ^2? Sketch a contour of the true distribution $P(\mathbf{x})$ and the two approximating distributions in the case $\sigma_1/\sigma_2 = 10$.

[Note that in general it is not possible to evaluate the objective function G, because integrals under the true distribution $P(\mathbf{x})$ are usually intractable.]

Exercise 33.6.[2, p.436] What do you think of the idea of using a variational method to optimize an approximating distribution Q which we then use as a proposal density for importance sampling?

Exercise 33.7.[2] Define the *relative entropy* or *Kullback–Leibler divergence* between two probability distributions P and Q, and state Gibbs' inequality.

Consider the problem of approximating a joint distribution $P(x, y)$ by a separable distribution $Q(x, y) = Q_X(x)Q_Y(y)$. Show that if the objec-

tive function for this approximation is

$$G(Q_X, Q_Y) = \sum_{x,y} P(x,y) \log_2 \frac{P(x,y)}{Q_X(x) Q_Y(y)}$$

that the minimal value of G is achieved when Q_X and Q_Y are equal to the marginal distributions over x and y.

Now consider the alternative objective function

$$F(Q_X, Q_Y) = \sum_{x,y} Q_X(x) Q_Y(y) \log_2 \frac{Q_X(x) Q_Y(y)}{P(x,y)};$$

the probability distribution $P(x,y)$ shown in the margin is to be approximated by a separable distribution $Q(x,y) = Q_X(x)Q_Y(y)$. State the value of $F(Q_X, Q_Y)$ if Q_X and Q_Y are set to the marginal distributions over x and y.

$P(x,y)$		x			
		1	2	3	4
y	1	1/8	1/8	0	0
	2	1/8	1/8	0	0
	3	0	0	1/4	0
	4	0	0	0	1/4

Show that $F(Q_X, Q_Y)$ has three distinct minima, identify those minima, and evaluate F at each of them.

▶ 33.10 Solutions

Solution to exercise 33.5 (p.434). We need to know the relative entropy between two one-dimensional Gaussian distributions:

$$\begin{aligned}
&\int \mathrm{d}x \, \mathrm{Normal}(x; 0, \sigma_Q) \ln \frac{\mathrm{Normal}(x; 0, \sigma_Q)}{\mathrm{Normal}(x; 0, \sigma_P)} \\
&= \int \mathrm{d}x \, \mathrm{Normal}(x; 0, \sigma_Q) \left[\ln \frac{\sigma_P}{\sigma_Q} - \frac{1}{2} x^2 \left(\frac{1}{\sigma_Q^2} - \frac{1}{\sigma_P^2} \right) \right] \qquad (33.55) \\
&= \frac{1}{2} \left(\ln \frac{\sigma_P^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_P^2} \right). \qquad (33.56)
\end{aligned}$$

So, if we approximate P, whose variances are σ_1^2 and σ_2^2, by Q, whose variances are both σ_Q^2, we find

$$F(\sigma_Q^2) = \frac{1}{2} \left(\ln \frac{\sigma_1^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_1^2} + \ln \frac{\sigma_2^2}{\sigma_Q^2} - 1 + \frac{\sigma_Q^2}{\sigma_2^2} \right); \qquad (33.57)$$

differentiating,

$$\frac{\mathrm{d}}{\mathrm{d} \ln(\sigma_Q^2)} F = \frac{1}{2} \left[-2 + \left(\frac{\sigma_Q^2}{\sigma_1^2} + \frac{\sigma_Q^2}{\sigma_2^2} \right) \right], \qquad (33.58)$$

which is zero when

$$\frac{1}{\sigma_Q^2} = \frac{1}{2} \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right). \qquad (33.59)$$

Thus we set the approximating distribution's inverse variance to the mean inverse variance of the target distribution P.

In the case $\sigma_1 = 10$ and $\sigma_2 = 1$, we obtain $\sigma_Q \simeq \sqrt{2}$, which is just a factor of $\sqrt{2}$ larger than σ_2, pretty much *independent* of the value of the larger standard deviation σ_1. *Variational free energy minimization typically leads to approximating distributions whose length scales match the shortest length scale of the target distribution.* The approximating distribution might be viewed as *too compact*.

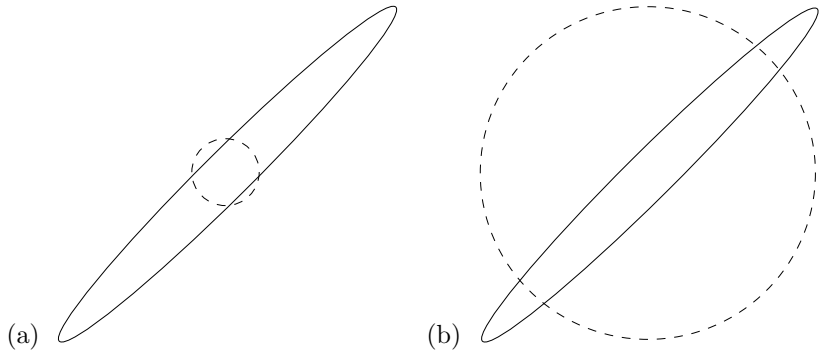


Figure 33.6. Two separable Gaussian approximations (dotted lines) to a bivariate Gaussian distribution (solid line). (a) The approximation that minimizes the variational free energy. (b) The approximation that minimizes the objective function G. In each figure, the lines show the contours at which $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} = 1$, where $\mathbf{A}$ is the inverse covariance matrix of the Gaussian.

In contrast, if we use the objective function G then we find:

$$G(\sigma_Q^2) = \frac{1}{2}\left(\ln \sigma_Q^2 + \frac{\sigma_1^2}{\sigma_Q^2} + \ln \sigma_Q^2 + \frac{\sigma_2^2}{\sigma_Q^2}\right) + \text{constant}, \tag{33.60}$$

where the constant depends on σ_1 and σ_2 only. Differentiating,

$$\frac{\mathrm{d}}{\mathrm{d}\ln \sigma_Q^2} G = \frac{1}{2}\left[2 - \left(\frac{\sigma_1^2}{\sigma_Q^2} + \frac{\sigma_2^2}{\sigma_Q^2}\right)\right], \tag{33.61}$$

which is zero when

$$\sigma_Q^2 = \frac{1}{2}\left(\sigma_1^2 + \sigma_2^2\right). \tag{33.62}$$

Thus we set the approximating distribution's variance to the mean variance of the target distribution P.

In the case $\sigma_1 = 10$ and $\sigma_2 = 1$, we obtain $\sigma_Q \simeq 10/\sqrt{2}$, which is just a factor of $\sqrt{2}$ smaller than σ_1, independent of the value of σ_2.

The two approximations are shown to scale in figure 33.6.

Solution to exercise 33.6 (p.434). The best possible variational approximation is of course the target distribution P. Assuming that this is not possible, a good variational approximation is *more compact* than the true distribution. In contrast, a good sampler is *more heavy tailed* than the true distribution. An over-compact distribution would be a lousy sampler with a large variance.

34

Independent Component Analysis and Latent Variable Modelling

▶ 34.1 Latent variable models

Many statistical models are generative models (that is, models that specify a full probability density over all variables in the situation) that make use of *latent variables* to describe a probability distribution over observables.

Examples of latent variable models include Chapter 22's mixture models, which model the observables as coming from a superposed mixture of simple probability distributions (the latent variables are the unknown class labels of the examples); hidden Markov models (Rabiner and Juang, 1986; Durbin *et al.*, 1998); and factor analysis.

The decoding problem for error-correcting codes can also be viewed in terms of a latent variable model – figure 34.1. In that case, the encoding matrix $\mathbf{G}$ is normally known in advance. In latent variable modelling, the parameters equivalent to $\mathbf{G}$ are usually not known, and must be inferred from the data along with the latent variables $\mathbf{s}$.

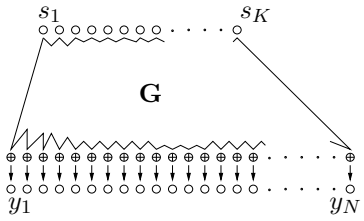


Figure 34.1. Error-correcting codes as latent variable models. The K latent variables are the independent source bits $s_1, \ldots, s_K$; these give rise to the observables via the generator matrix $\mathbf{G}$.

Usually, the latent variables have a simple distribution, often a separable distribution. Thus when we fit a latent variable model, we are finding a description of the data in terms of 'independent components'. The 'independent component analysis' algorithm corresponds to perhaps the simplest possible latent variable model with continuous latent variables.

▶ 34.2 The generative model for independent component analysis

A set of N observations $D = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$ are assumed to be generated as follows. Each J-dimensional vector $\mathbf{x}$ is a linear mixture of I underlying source signals, $\mathbf{s}$:

$$\mathbf{x} = \mathbf{G}\mathbf{s}, \tag{34.1}$$

where the matrix of mixing coefficients $\mathbf{G}$ is not known.

The simplest algorithm results if we assume that the number of sources is equal to the number of observations, i.e., $I = J$. Our aim is to recover the source variables $\mathbf{s}$ (within some multiplicative factors, and possibly permuted). To put it another way, we aim to create the inverse of $\mathbf{G}$ (within a post-multiplicative factor) given only a set of examples $\{\mathbf{x}\}$. We assume that the latent variables are independently distributed, with marginal distributions $P(s_i \,|\, \mathcal{H}) \equiv p_i(s_i)$. Here $\mathcal{H}$ denotes the assumed form of this model and the assumed probability distributions p_i of the latent variables.

The probability of the observables and the hidden variables, given $\mathbf{G}$ and

$\mathcal{H}$, is:

$$P(\{\mathbf{x}^{(n)}, \mathbf{s}^{(n)}\}_{n=1}^{N} \,|\, \mathbf{G}, \mathcal{H}) = \prod_{n=1}^{N} \left[P(\mathbf{x}^{(n)} \,|\, \mathbf{s}^{(n)}, \mathbf{G}, \mathcal{H}) P(\mathbf{s}^{(n)} \,|\, \mathcal{H}) \right] \tag{34.2}$$

$$= \prod_{n=1}^{N} \left[\left(\prod_j \delta \left(x_j^{(n)} - \textstyle\sum_i G_{ji} s_i^{(n)} \right) \right) \left(\prod_i p_i(s_i^{(n)}) \right) \right]. \tag{34.3}$$

We assume that the vector $\mathbf{x}$ is generated *without noise*. This assumption is not usually made in latent variable modelling, since noise-free data are rare; but it makes the inference problem far simpler to solve.

The likelihood function

For learning about $\mathbf{G}$ from the data D, the relevant quantity is the likelihood function

$$P(D \,|\, \mathbf{G}, \mathcal{H}) = \prod_n P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) \tag{34.4}$$

which is a product of factors each of which is obtained by marginalizing over the latent variables. When we marginalize over delta functions, remember that $\int \mathrm{d}s \, \delta(x - vs) f(s) = \frac{1}{v} f(x/v)$. We adopt summation convention at this point, such that, for example, $G_{ji} s_i^{(n)} \equiv \sum_i G_{ji} s_i^{(n)}$. A single factor in the likelihood is given by

$$P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) = \int \mathrm{d}^I \mathbf{s}^{(n)} \, P(\mathbf{x}^{(n)} \,|\, \mathbf{s}^{(n)}, \mathbf{G}, \mathcal{H}) P(\mathbf{s}^{(n)} \,|\, \mathcal{H}) \tag{34.5}$$

$$= \int \mathrm{d}^I \mathbf{s}^{(n)} \prod_j \delta \left(x_j^{(n)} - G_{ji} s_i^{(n)} \right) \prod_i p_i(s_i^{(n)}) \tag{34.6}$$

$$= \frac{1}{|\det \mathbf{G}|} \prod_i p_i(G_{ij}^{-1} x_j) \tag{34.7}$$

$$\Rightarrow \ln P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) = -\ln |\det \mathbf{G}| + \sum_i \ln p_i(G_{ij}^{-1} x_j). \tag{34.8}$$

To obtain a maximum likelihood algorithm we find the gradient of the log likelihood. If we introduce $\mathbf{W} \equiv \mathbf{G}^{-1}$, the log likelihood contributed by a single example may be written:

$$\ln P(\mathbf{x}^{(n)} \,|\, \mathbf{G}, \mathcal{H}) = \ln |\det \mathbf{W}| + \sum_i \ln p_i(W_{ij} x_j). \tag{34.9}$$

We'll assume from now on that $\det \mathbf{W}$ is positive, so that we can omit the absolute value sign. We will need the following identities:

$$\frac{\partial}{\partial G_{ji}} \ln \det \mathbf{G} = G_{ij}^{-1} = W_{ij} \tag{34.10}$$

$$\frac{\partial}{\partial G_{ji}} G_{lm}^{-1} = -G_{lj}^{-1} G_{im}^{-1} = -W_{lj} W_{im} \tag{34.11}$$

$$\frac{\partial}{\partial W_{ij}} f = -G_{jm} \left(\frac{\partial}{\partial G_{lm}} f \right) G_{li}. \tag{34.12}$$

Let us define $a_i \equiv W_{ij} x_j$,

$$\phi_i(a_i) \equiv \mathrm{d} \ln p_i(a_i) / \mathrm{d}a_i, \tag{34.13}$$

Repeat for each datapoint $\mathbf{x}$:

1. Put $\mathbf{x}$ through a linear mapping:
$$\mathbf{a} = \mathbf{W}\mathbf{x}.$$
2. Put $\mathbf{a}$ through a nonlinear map:
$$z_i = \phi_i(a_i),$$
where a popular choice for ϕ is $\phi = -\tanh(a_i)$.
3. Adjust the weights in accordance with
$$\Delta\mathbf{W} \propto [\mathbf{W}^{\mathsf{T}}]^{-1} + \mathbf{z}\mathbf{x}^{\mathsf{T}}.$$

Algorithm 34.2. Independent component analysis – online steepest ascents version. See also algorithm 34.4, which is to be preferred.

and $z_i = \phi_i(a_i)$, which indicates in which direction a_i needs to change to make the probability of the data greater. We may then obtain the gradient with respect to G_{ji} using equations (34.10) and (34.11):

$$\frac{\partial}{\partial G_{ji}} \ln P(\mathbf{x}^{(n)} \mid \mathbf{G}, \mathcal{H}) = -W_{ij} - a_i z_{i'} W_{i'j}. \tag{34.14}$$

Or alternatively, the derivative with respect to W_{ij}:

$$\frac{\partial}{\partial W_{ij}} \ln P(\mathbf{x}^{(n)} \mid \mathbf{G}, \mathcal{H}) = G_{ji} + x_j z_i. \tag{34.15}$$

If we choose to change $\mathbf{W}$ so as to ascend this gradient, we obtain the learning rule

$$\Delta\mathbf{W} \propto [\mathbf{W}^{\mathsf{T}}]^{-1} + \mathbf{z}\mathbf{x}^{\mathsf{T}}. \tag{34.16}$$

The algorithm so far is summarized in algorithm 34.2.

Choices of ϕ

The choice of the function ϕ defines the assumed prior distribution of the latent variable s.

Let's first consider the *linear* choice $\phi_i(a_i) = -\kappa a_i$, which implicitly (via equation 34.13) assumes a Gaussian distribution on the latent variables. The Gaussian distribution on the latent variables is invariant under rotation of the latent variables, so there can be no evidence favouring any particular alignment of the latent variable space. The linear algorithm is thus uninteresting in that it will never recover the matrix $\mathbf{G}$ or the original sources. Our only hope is thus that the sources are non-Gaussian. Thankfully, most real sources have non-Gaussian distributions; often they have heavier tails than Gaussians.

We thus move on to the popular tanh nonlinearity. If

$$\phi_i(a_i) = -\tanh(a_i) \tag{34.17}$$

then implicitly we are assuming

$$p_i(s_i) \propto 1/\cosh(s_i) \propto \frac{1}{e^{s_i} + e^{-s_i}}. \tag{34.18}$$

This is a heavier-tailed distribution for the latent variables than the Gaussian distribution.

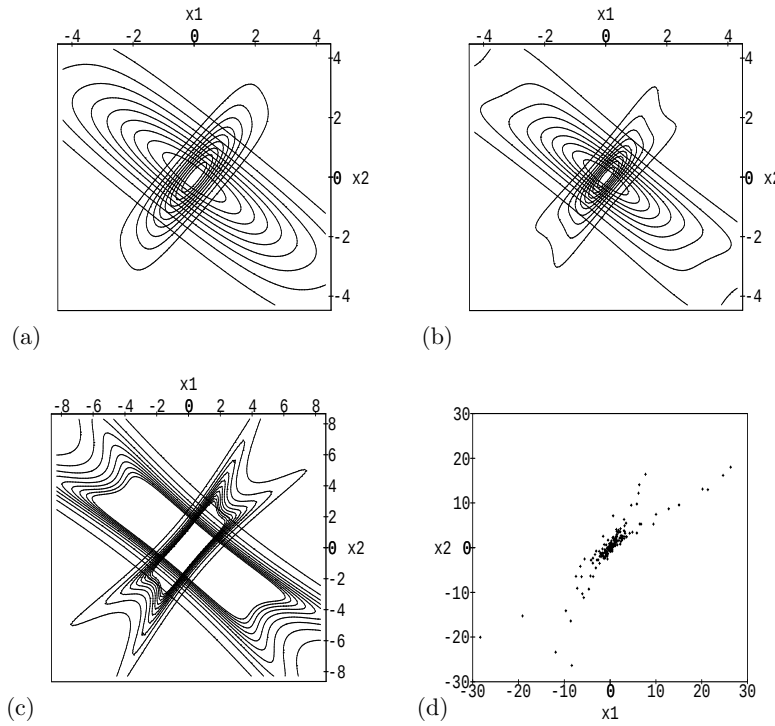


Figure 34.3. Illustration of the generative models implicit in the learning algorithm. (a) Distributions over two observables generated by $1/\cosh$ distributions on the latent variables, for $\mathbf{G} = \begin{bmatrix} 3/4 & 1/2 \\ 1/2 & 1 \end{bmatrix}$ (compact distribution) and $\mathbf{G} = \begin{bmatrix} 2 & -1 \\ -1 & 3/2 \end{bmatrix}$ (broader distribution). (b) Contours of the generative distributions when the latent variables have Cauchy distributions. The learning algorithm fits this amoeboid object to the empirical data in such a way as to maximize the likelihood. The contour plot in (b) does not adequately represent this heavy-tailed distribution. (c) Part of the tails of the Cauchy distribution, giving the contours $0.01 \ldots 0.1$ times the density at the origin. (d) Some data from one of the generative distributions illustrated in (b) and (c). Can you tell which? 200 samples were created, of which 196 fell in the plotted region.

We could also use a tanh nonlinearity with gain β, that is, $\phi_i(a_i) = -\tanh(\beta a_i)$, whose implicit probabilistic model is $p_i(s_i) \propto 1/[\cosh(\beta s_i)]^{1/\beta}$. In the limit of large β, the nonlinearity becomes a step function and the probability distribution $p_i(s_i)$ becomes a biexponential distribution, $p_i(s_i) \propto \exp(-|s|)$. In the limit $\beta \to 0$, $p_i(s_i)$ approaches a Gaussian with mean zero and variance $1/\beta$. Heavier-tailed distributions than these may also be used. The Student and Cauchy distributions spring to mind.

Example distributions

Figures 34.3(a–c) illustrate typical distributions generated by the independent components model when the components have $1/\cosh$ and Cauchy distributions. Figure 34.3d shows some samples from the Cauchy model. The Cauchy distribution, being the more heavy-tailed, gives the clearest picture of how the predictive distribution depends on the assumed generative parameters $\mathbf{G}$.

▶ 34.3 A covariant, simpler, and faster learning algorithm

We have thus derived a learning algorithm that performs steepest descents on the likelihood function. The algorithm does not work very quickly, even on toy data; the algorithm is ill-conditioned and illustrates nicely the general advice that, *while finding the gradient of an objective function is a splendid idea, ascending the gradient directly may not be.* The fact that the algorithm is ill-conditioned can be seen in the fact that it involves a matrix inverse, which can be arbitrarily large or even undefined.

Covariant optimization in general

The principle of covariance says that a consistent algorithm should give the same results independent of the units in which quantities are measured (Knuth,

1968). A prime example of a *non*-covariant algorithm is the popular steepest descents rule. A dimensionless objective function $L(\mathbf{w})$ is defined, its derivative with respect to some parameters $\mathbf{w}$ is computed, and then $\mathbf{w}$ is changed by the rule

$$\Delta w_i = \eta \frac{\partial L}{\partial w_i}. \tag{34.19}$$

This popular equation is dimensionally inconsistent: the left-hand side of this equation has dimensions of $[w_i]$ and the right-hand side has dimensions $1/[w_i]$. The behaviour of the learning algorithm (34.19) is not covariant with respect to linear rescaling of the vector $\mathbf{w}$. Dimensional inconsistency is not the end of the world, as the success of numerous gradient descent algorithms has demonstrated, and indeed if η decreases with n (during on-line learning) as $1/n$ then the Munro–Robbins theorem (Bishop, 1992, p. 41) shows that the parameters will asymptotically converge to the maximum likelihood parameters. But the non-covariant algorithm may take a very large number of iterations to achieve this convergence; indeed many former users of steepest descents algorithms prefer to use algorithms such as conjugate gradients that adaptively figure out the curvature of the objective function. The defense of equation (34.19) that points out η could be a dimensional constant is untenable if not all the parameters w_i have the same dimensions.

Here n is the number of iterations.

The algorithm would be covariant if it had the form

$$\Delta w_i = \eta \sum_{i'} M_{ii'} \frac{\partial L}{\partial w_i}, \tag{34.20}$$

where $\mathbf{M}$ is a positive-definite matrix whose i, i' element has dimensions $[w_i w_{i'}]$. From where can we obtain such a matrix? Two sources of such matrices are *metrics* and *curvatures*.

Metrics and curvatures

If there is a natural metric that defines *distances* in our parameter space $\mathbf{w}$, then a matrix $\mathbf{M}$ can be obtained from the metric. There is often a natural choice. In the special case where there is a known quadratic metric defining the length of a vector $\mathbf{w}$, then the matrix can be obtained from the quadratic form. For example if the length is $\mathbf{w}^2$ then the natural matrix is $\mathbf{M} = \mathbf{I}$, and steepest descents is appropriate.

Another way of finding a metric is to look at the curvature of the objective function, defining $\mathbf{A} \equiv -\nabla\nabla L$ (where $\nabla \equiv \partial/\partial\mathbf{w}$). Then the matrix $\mathbf{M} = \mathbf{A}^{-1}$ will give a covariant algorithm; what is more, this algorithm is the Newton algorithm, so we recognize that it will alleviate one of the principal difficulties with steepest descents, namely its slow convergence to a minimum when the objective function is at all ill-conditioned. The Newton algorithm converges to the minimum in a single step if L is quadratic.

In some problems it may be that the curvature $\mathbf{A}$ consists of both data-dependent terms and data-independent terms; in this case, one might choose to define the metric using the data-independent terms only (Gull, 1989). The resulting algorithm will still be covariant but it will not implement an exact Newton step. Obviously there are many covariant algorithms; there is no unique choice. But covariant algorithms are a small subset of the set of *all* algorithms!

Back to independent component analysis

For the present maximum likelihood problem we have evaluated the gradient with respect to $\mathbf{G}$ and the gradient with respect to $\mathbf{W} = \mathbf{G}^{-1}$. Steepest ascents in $\mathbf{W}$ is not covariant. Let us construct an alternative, covariant algorithm with the help of the curvature of the log likelihood. Taking the second derivative of the log likelihood with respect to $\mathbf{W}$ we obtain two terms, the first of which is data-independent:

$$\frac{\partial G_{ji}}{\partial W_{kl}} = -G_{jk}G_{li}, \tag{34.21}$$

and the second of which is data-dependent:

$$\frac{\partial (z_i x_j)}{\partial W_{kl}} = x_j x_l \delta_{ik} z_i', \text{ (no sum over } i) \tag{34.22}$$

where z' is the derivative of z. It is tempting to drop the data-dependent term and define the matrix $\mathbf{M}$ by $[M^{-1}]_{(ij)(kl)} = [G_{jk}G_{li}]$. However, this matrix is not positive definite (it has at least one non-positive eigenvalue), so it is a poor approximation to the curvature of the log likelihood, which must be positive definite in the neighbourhood of a maximum likelihood solution. We must therefore consult the data-dependent term for inspiration. The aim is to find a convenient approximation to the curvature and to obtain a covariant algorithm, not necessarily to implement an exact Newton step. What is the average value of $x_j x_l \delta_{ik} z_i'$? If the true value of $\mathbf{G}$ is $\mathbf{G}^*$, then

$$\left\langle x_j x_l \delta_{ik} z_i' \right\rangle = \left\langle G^*_{jm} s_m s_n G^*_{ln} \delta_{ik} z_i' \right\rangle . \tag{34.23}$$

We now make several severe approximations: we replace $\mathbf{G}^*$ by the present value of $\mathbf{G}$, and replace the correlated average $\langle s_m s_n z_i' \rangle$ by $\langle s_m s_n \rangle \langle z_i' \rangle \equiv \Sigma_{mn} D_i$. Here $\mathbf{\Sigma}$ is the variance–covariance matrix of the latent variables (which is assumed to exist), and D_i is the typical value of the curvature $\mathrm{d}^2 \ln p_i(a)/\mathrm{d}a^2$. Given that the sources are assumed to be independent, $\mathbf{\Sigma}$ and $\mathbf{D}$ are both diagonal matrices. These approximations motivate the matrix $\mathbf{M}$ given by:

$$[M^{-1}]_{(ij)(kl)} = G_{jm}\Sigma_{mn}G_{ln}\delta_{ik}D_i, \tag{34.24}$$

that is,

$$M_{(ij)(kl)} = W_{mj}\Sigma^{-1}_{mn}W_{nl}\delta_{ik}D_i^{-1}. \tag{34.25}$$

For simplicity, we further assume that the sources are similar to each other so that $\mathbf{\Sigma}$ and $\mathbf{D}$ are both homogeneous, and that $\mathbf{\Sigma D} = 1$. This will lead us to an algorithm that is covariant with respect to linear rescaling of the data $\mathbf{x}$, but not with respect to linear rescaling of the latent variables. We thus use:

$$M_{(ij)(kl)} = W_{mj}W_{ml}\delta_{ik}. \tag{34.26}$$

Multiplying this matrix by the gradient in equation (34.15) we obtain the following covariant learning algorithm:

$$\Delta W_{ij} = \eta \left(W_{ij} + W_{i'j} a_{i'} z_i \right) . \tag{34.27}$$

Notice that this expression does not require any inversion of the matrix $\mathbf{W}$. The only additional computation once $\mathbf{z}$ has been computed is a single backward pass through the weights to compute the quantity

$$x_j' = W_{i'j} a_{i'} \tag{34.28}$$

Repeat for each datapoint $\mathbf{x}$:

1. Put $\mathbf{x}$ through a linear mapping:
$$\mathbf{a} = \mathbf{W}\mathbf{x}.$$
2. Put $\mathbf{a}$ through a nonlinear map:
$$z_i = \phi_i(a_i),$$
where a popular choice for ϕ is $\phi = -\tanh(a_i)$.
3. Put $\mathbf{a}$ back through $\mathbf{W}$:
$$\mathbf{x}' = \mathbf{W}^{\mathsf{T}}\mathbf{a}.$$
4. Adjust the weights in accordance with
$$\Delta\mathbf{W} \propto \mathbf{W} + \mathbf{z}\mathbf{x}'^{\mathsf{T}}.$$

Algorithm 34.4. Independent component analysis – covariant version.

in terms of which the covariant algorithm reads:

$$\Delta W_{ij} = \eta \left(W_{ij} + x'_j z_i \right). \tag{34.29}$$

The quantity $\left(W_{ij} + x'_j z_i \right)$ on the right-hand side is sometimes called the *natural gradient*. The covariant independent component analysis algorithm is summarized in algorithm 34.4.

Further reading

ICA was originally derived using an information maximization approach (Bell and Sejnowski, 1995). Another view of ICA, in terms of energy functions, which motivates more general models, is given by Hinton *et al.* (2001). Another generalization of ICA can be found in Pearlmutter and Parra (1996, 1997). There is now an enormous literature on applications of ICA. A variational free energy minimization approach to ICA-like models is given in (Miskin, 2001; Miskin and MacKay, 2000; Miskin and MacKay, 2001). Further reading on blind separation, including non-ICA algorithms, can be found in (Jutten and Herault, 1991; Comon *et al.*, 1991; Hendin *et al.*, 1994; Amari *et al.*, 1996; Hojen-Sorensen *et al.*, 2002).

Infinite models

While latent variable models with a finite number of latent variables are widely used, it is often the case that our beliefs about the situation would be most accurately captured by a very large number of latent variables.

Consider clustering, for example. If we attack speech recognition by modelling words using a cluster model, how many clusters should we use? The number of possible words is unbounded (section 18.2), so we would really like to use a model in which it's always possible for new clusters to arise.

Furthermore, if we do a careful job of modelling the cluster corresponding to just one English word, we will probably find that the cluster for one word should itself be modelled as composed of clusters – indeed, a hierarchy of

clusters within clusters. The first levels of the hierarchy would divide male speakers from female, and would separate speakers from different regions – India, Britain, Europe, and so forth. Within each of those clusters would be subclusters for the different accents within each region. The subclusters could have subsubclusters right down to the level of villages, streets, or families.

Thus we would often like to have infinite numbers of clusters; in some cases the clusters would have a hierarchical structure, and in other cases the hierarchy would be flat. So, how should such infinite models be implemented in finite computers? And how should we set up our Bayesian models so as to avoid getting silly answers?

Infinite mixture models for categorical data are presented in Neal (1991), along with a Monte Carlo method for simulating inferences and predictions. Infinite Gaussian mixture models with a flat hierarchical structure are presented in Rasmussen (2000). Neal (2001) shows how to use Dirichlet diffusion trees to define models of hierarchical clusters. Most of these ideas build on the Dirichlet process (section 18.2). This remains an active research area (Rasmussen and Ghahramani, 2002; Beal *et al.*, 2002).

▶ 34.4 Exercises

Exercise 34.1.[3] Repeat the derivation of the algorithm, but assume a small amount of noise in $\mathbf{x}$: $\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}$; so the term $\delta\left(x_j^{(n)} - \sum_i G_{ji} s_i^{(n)}\right)$ in the joint probability (34.3) is replaced by a probability distribution over $x_j^{(n)}$ with mean $\sum_i G_{ji} s_i^{(n)}$. Show that, if this noise distribution has sufficiently small standard deviation, the identical algorithm results.

Exercise 34.2.[3] Implement the covariant ICA algorithm and apply it to toy data.

Exercise 34.3.[4-5] Create algorithms appropriate for the situations: (a) $\mathbf{x}$ includes substantial Gaussian noise; (b) more measurements than latent variables ($J > I$); (c) fewer measurements than latent variables ($J < I$).

Factor analysis assumes that the observations $\mathbf{x}$ can be described in terms of independent latent variables $\{s_k\}$ and independent additive noise. Thus the observable $\mathbf{x}$ is given by

$$\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}, \tag{34.30}$$

where $\mathbf{n}$ is a noise vector whose components have a separable probability distribution. In factor analysis it is often assumed that the probability distributions of $\{s_k\}$ and $\{n_i\}$ are zero-mean Gaussians; the noise terms may have different variances σ_i^2.

Exercise 34.4.[4] Make a maximum likelihood algorithm for inferring $\mathbf{G}$ from data, assuming the generative model $\mathbf{x} = \mathbf{G}\mathbf{s} + \mathbf{n}$ is correct and that $\mathbf{s}$ and $\mathbf{n}$ have independent Gaussian distributions. Include parameters σ_j^2 to describe the variance of each n_j, and maximize the likelihood with respect to them too. Let the variance of each s_i be 1.

Exercise 34.5.[4C] Implement the infinite Gaussian mixture model of Rasmussen (2000).

35

Random Inference Topics

▶ 35.1 What do you know if you are ignorant?

Example 35.1. A real variable x is measured in an accurate experiment. For example, x might be the half-life of the neutron, the wavelength of light emitted by a firefly, the depth of Lake Vostok, or the mass of Jupiter's moon Io.

What is the probability that the value of x starts with a '1', like the charge of the electron (in S.I. units),

$$e = 1.602\ldots \times 10^{-19}\,\mathrm{C},$$

and the Boltzmann constant,

$$k = 1.380\,66\ldots \times 10^{-23}\,\mathrm{J\,K^{-1}}?$$

And what is the probability that it starts with a '9', like the Faraday constant,

$$\mathcal{F} = 9.648\ldots \times 10^{4}\,\mathrm{C\,mol^{-1}}?$$

What about the second digit? What is the probability that the mantissa of x starts '1.1...', and what is the probability that x starts '9.9...'?

Solution. An expert on neutrons, fireflies, Antarctica, or Jove might be able to predict the value of x, and thus predict the first digit with some confidence, but what about someone with no knowledge of the topic? What is the probability distribution corresponding to 'knowing nothing'?

One way to attack this question is to notice that the units of x have not been specified. If the half-life of the neutron were measured in fortnights instead of seconds, the number x would be divided by 1 209 600; if it were measured in years, it would be divided by 3×10^7. Now, is our knowledge about x, and, in particular, our knowledge of its first digit, affected by the change in units? For the expert, the answer is yes; but let us take someone truly ignorant, for whom the answer is no; their predictions about the first digit of x are independent of the units. The arbitrariness of the units corresponds to *invariance* of the probability distribution when x is *multiplied* by any number.

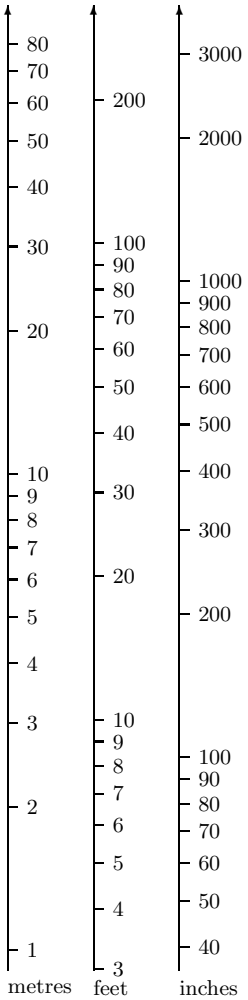


Figure 35.1. When viewed on a logarithmic scale, scales using different units are translated relative to each other.

If you don't know the units that a quantity is measured in, the probability of the first digit must be proportional to the length of the corresponding piece of logarithmic scale. The probability that the first digit of a number is 1 is thus

$$p_1 = \frac{\log 2 - \log 1}{\log 10 - \log 1} = \frac{\log 2}{\log 10}. \tag{35.1}$$

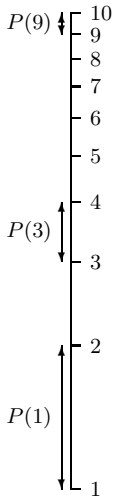
Now, $2^{10} = 1024 \simeq 10^3 = 1000$, so without needing a calculator, we have $10 \log 2 \simeq 3 \log 10$ and

$$p_1 \simeq \frac{3}{10}. \tag{35.2}$$

More generally, the probability that the first digit is d is

$$(\log(d+1) - \log(d))/(\log 10 - \log 1) = \log_{10}(1 + 1/d). \tag{35.3}$$

This observation about initial digits is known as Benford's law. Ignorance does not correspond to a uniform probability distribution over d. □



▷ Exercise 35.2.[2] A pin is thrown tumbling in the air. What is the probability distribution of the angle θ_1 between the pin and the vertical at a moment while it is in the air? The tumbling pin is photographed. What is the probability distribution of the angle θ_3 between the pin and the vertical as imaged in the photograph?

▷ Exercise 35.3.[2] Record breaking. Consider keeping track of the world record for some quantity x, say earthquake magnitude, or longjump distances jumped at world championships. If we assume that attempts to break the record take place at a steady rate, and if we assume that the underlying probability distribution of the outcome x, $P(x)$, is not changing – an assumption that I think is unlikely to be true in the case of sports endeavours, but an interesting assumption to consider nonetheless – and assuming no knowledge at all about $P(x)$, what can be predicted about successive intervals between the dates when records are broken?

▶ 35.2 The Luria–Delbrück distribution

Exercise 35.4.[3C, p.449] In their landmark paper demonstrating that bacteria could mutate from virus sensitivity to virus resistance, Luria and Delbrück (1943) wanted to estimate the mutation rate in an exponentially-growing population from the total number of mutants found at the end of the experiment. This problem is difficult because the quantity measured (the number of mutated bacteria) has a heavy-tailed probability distribution: a mutation occuring early in the experiment can give rise to a huge number of mutants. Unfortunately, Luria and Delbrück didn't know Bayes' theorem, and their way of coping with the heavy-tailed distribution involves arbitrary hacks leading to two different estimators of the mutation rate. One of these estimators (based on the mean number of mutated bacteria, averaging over several experiments) has appallingly large variance, yet sampling theorists continue to use it and base confidence intervals around it (Kepler and Oprea, 2001). In this exercise you'll do the inference right.

In each culture, a single bacterium that is *not resistant* gives rise, after g generations, to $N = 2^g$ descendants, all clones except for differences arising from mutations. The final culture is then exposed to a virus, and the number of resistant bacteria n is measured. According to the now accepted mutation hypothesis, these resistant bacteria got their resistance from random mutations that took place during the growth of the colony. The mutation rate (per cell per generation), a, is about one in a hundred million. The total number of opportunities to mutate is N, since $\sum_{i=0}^{g-1} 2^i \simeq 2^g = N$. If a bacterium mutates at the ith generation, its descendants all inherit the mutation, and the final number of resistant bacteria contributed by that one ancestor is 2^{g-i}.

Given M separate experiments, in each of which a colony of size N is created, and where the measured numbers of resistant bacteria are $\{n_m\}_{m=1}^{M}$, what can we infer about the mutation rate, a?

Make the inference given the following dataset from Luria and Delbrück, for $N = 2.4 \times 10^8$: $\{n_m\} = \{1, 0, 3, 0, 0, 5, 0, 5, 0, 6, 107, 0, 0, 0, 1, 0, 0, 64, 0, 35\}$. [A small amount of computation is required to solve this problem.]

▶ 35.3 Inferring causation

Exercise 35.5.[2, p.450] In the Bayesian graphical model community, the task of inferring which way the arrows point – that is, which nodes are parents, and which children – is one on which much has been written.

Inferring causation is tricky because of 'likelihood equivalence'. Two graphical models are likelihood-equivalent if for any setting of the parameters of either, there exists a setting of the parameters of the other such that the two joint probability distributions of all observables are identical. An example of a pair of likelihood-equivalent models are $A \rightarrow B$ and $B \rightarrow A$. The model $A \rightarrow B$ asserts that A is the parent of B, or, in very sloppy terminology, 'A causes B'. An example of a situation where '$B \rightarrow A$' is true is the case where B is the variable 'burglar in house' and A is the variable 'alarm is ringing'. Here it is literally true that B causes A. But this choice of words is confusing if applied to another example, $R \rightarrow D$, where R denotes 'it rained this morning' and D denotes 'the pavement is dry'. 'R causes D' is confusing. I'll therefore use the words 'B is a parent of A' to denote causation. Some statistical methods that use the likelihood alone are unable to use data to distinguish between likelihood-equivalent models. In a Bayesian approach, on the other hand, two likelihood-equivalent models may nevertheless be somewhat distinguished, in the light of data, since likelihood-equivalence does not force a Bayesian to use priors that assign equivalent densities over the two parameter spaces of the models.

However, many Bayesian graphical modelling folks, perhaps out of sympathy for their non-Bayesian colleagues, or from a latent urge not to appear different from them, deliberately discard this potential advantage of Bayesian methods – the ability to infer causation from data – by skewing their models so that the ability goes away; a widespread orthodoxy holds that one should identify the choices of prior for which 'prior equivalence' holds, i.e., the priors such that models that are likelihood-equivalent also have identical posterior probabilities; and then one should use one of those priors in inference and prediction. This argument motivates the use, as the prior over all probability vectors, of specially-constructed Dirichlet distributions.

In my view it is a philosophical error to use only those priors such that causation cannot be inferred. Priors should be set to describe one's assumptions; when this is done, it's likely that interesting inferences about causation *can* be made from data.

In this exercise, you'll make an example of such an inference.

Consider the toy problem where A and B are binary variables. The two models are $\mathcal{H}_{A \rightarrow B}$ and $\mathcal{H}_{B \rightarrow A}$. $\mathcal{H}_{A \rightarrow B}$ asserts that the marginal probability of A comes from a beta distribution with parameters $(1, 1)$, i.e., the uniform distribution; and that the two conditional distributions $P(b \,|\, a = 0)$ and $P(b \,|\, a = 1)$ also come independently from beta distributions with parameters $(1, 1)$. The other model assigns similar priors to the marginal probability of B and the conditional distributions of A given B. Data are gathered, and the

counts, given $F = 1000$ outcomes, are

	$a=0$	$a=1$	
$b=0$	760	5	765
$b=1$	190	45	235
	950	50	

(35.4)

What are the posterior probabilities of the two hypotheses?

Hint: it's a good idea to work this exercise out symbolically in order to spot all the simplifications that emerge.

$$\Psi(x) = \frac{\mathrm{d}}{\mathrm{d}x} \ln \Gamma(x) \simeq \ln(x) - \frac{1}{2x} + O(1/x^2). \tag{35.5}$$

The topic of inferring causation is a complex one. The fact that Bayesian inference can sensibly be used to infer the directions of arrows in graphs seems to be a neglected view, but it is certainly not the whole story. See Pearl (2000) for discussion of many other aspects of causality.

▶ 35.4 Further exercises

Exercise 35.6.[3] Photons arriving at a photon detector are believed to be emitted as a Poisson process with a time-varying rate,

$$\lambda(t) = \exp(a + b \sin(\omega t + \phi)), \tag{35.6}$$

where the parameters a, b, ω, and ϕ are known. Data are collected during the time $t = 0 \ldots T$. Given that N photons arrived at times $\{t_n\}_{n=1}^{N}$, discuss the inference of a, b, ω, and ϕ. [Further reading: Gregory and Loredo (1992).]

▷ Exercise 35.7.[2] A data file consisting of two columns of numbers has been printed in such a way that the boundaries between the columns are unclear. Here are the resulting strings.

891.10.0	912.20.0	874.10.0	870.20.0	836.10.0	861.20.0
903.10.0	937.10.0	850.20.0	916.20.0	899.10.0	907.10.0
924.20.0	861.10.0	899.20.0	849.10.0	887.20.0	840.10.0
849.20.0	891.10.0	916.20.0	891.10.0	912.20.0	875.10.0
898.20.0	924.10.0	950.20.0	958.10.0	971.20.0	933.10.0
966.20.0	908.10.0	924.20.0	983.10.0	924.20.0	908.10.0
950.20.0	911.10.0	913.20.0	921.25.0	912.20.0	917.30.0
923.50.0					

Discuss how probable it is, given these data, that the correct parsing of each item is:

(a) 891.10.0 → 891. 10.0, etc.

(b) 891.10.0 → 891.1 0.0, etc.

[A parsing of a string is a grammatical interpretation of the string. For example, 'Punch bores' could be parsed as 'Punch (noun) bores (verb)', or 'Punch (imperative verb) bores (plural noun)'.]

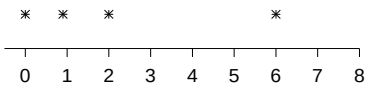
▷ Exercise 35.8.[2] In an experiment, the measured quantities $\{x_n\}$ come independently from a biexponential distribution with mean μ,

$$P(x \,|\, \mu) = \frac{1}{Z} \exp(- |x - \mu|) \,,$$

where Z is the normalizing constant, $Z = 2$. The mean μ is not known. An example of this distribution, with $\mu = 1$, is shown in figure 35.2.

Assuming the four datapoints are

$$\{x_n\} = \{0, 0.9, 2, 6\},$$



what do these data tell us about μ? Include detailed sketches in your answer. Give a range of plausible values of μ.

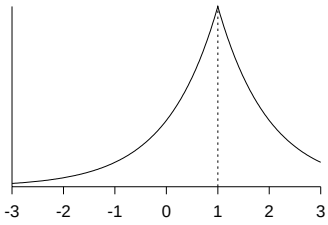


Figure 35.2. The biexponential distribution $P(x \mid \mu = 1)$.

▶ 35.5 Solutions

Solution to exercise 35.4 (p.446). A population of size N has N opportunities to mutate. The probability of the number of mutations that occurred, r, is roughly Poisson

$$P(r \mid a, N) = e^{-aN} \frac{(aN)^r}{r!}. \tag{35.7}$$

(This is slightly inaccurate because the descendants of a mutant cannot themselves undergo the same mutation.) Each mutation gives rise to a number of final mutant cells n_i that depends on the generation time of the mutation. If multiplication went like clockwork then the probability of n_i being 1 would be 1/2, the probability of 2 would be 1/4, the probability of 4 would be 1/8, and $P(n_i) = 1/(2n)$ for all n_i that are powers of two. But we don't expect the mutant progeny to divide in exact synchrony, and we don't know the precise timing of the end of the experiment compared to the division times. A smoothed version of this distribution that permits all integers to occur is

$$P(n_i) = \frac{1}{Z} \frac{1}{n_i^2}, \tag{35.8}$$

where $Z = \pi^2/6 = 1.645$. [This distribution's moments are all wrong, since n_i can never exceed N, but who cares about moments? – only sampling theory statisticians who are barking up the wrong tree, constructing 'unbiased estimators' such as $\hat{a} = (\bar{n}/N)/\log N$. The error that we introduce in the likelihood function by using the approximation to $P(n_i)$ is negligible.]

The observed number of mutants n is the sum

$$n = \sum_{i=1}^{r} n_i. \tag{35.9}$$

The probability distribution of n given r is the convolution of r identical distributions of the form (35.8). For example,

$$P(n \mid r = 2) = \sum_{n_1=1}^{n-1} \frac{1}{Z^2} \frac{1}{n_1^2} \frac{1}{(n - n_1)^2} \text{ for } n \geq 2. \tag{35.10}$$

The probability distribution of n given a, which is what we need for the Bayesian inference, is given by summing over r.

$$P(n \mid a) = \sum_{r=0}^{N} P(n \mid r) P(r \mid a, N). \tag{35.11}$$

This quantity can't be evaluated analytically, but for small a, it's easy to evaluate to any desired numerical precision by explicitly summing over r from

$r = 0$ to some $r_{\max}$, with $P(n \mid r)$ also being found for each r by $r_{\max}$ explicit convolutions for all required values of n; if $r_{\max} = n_{\max}$, the largest value of n encountered in the data, then $P(n \mid a)$ is computed exactly; but for this question's data, $r_{\max} = 9$ is plenty for an accurate result; I used $r_{\max} = 74$ to make the graphs in figure 35.3. `Octave` source code is available.[1] Incidentally, for data sets like the one in this exercise, which have a substantial number of zero counts, very little is lost by making Luria and Delbruck's second approximation, which is to retain only the count of how many n were equal to zero, and how many were non-zero. The likelihood function found using this weakened data set,

$$L(a) = (e^{-aN})^{11}(1 - e^{-aN})^{9}, \tag{35.12}$$

is scarcely distinguishable from the likelihood computed using full information.

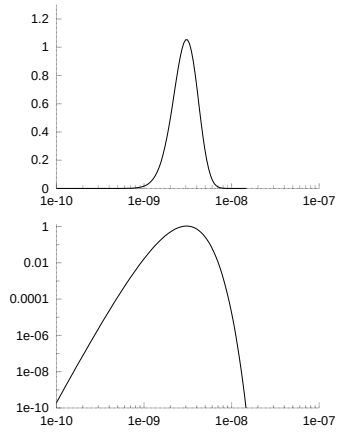


Figure 35.3. Likelihood of the mutation rate a on a linear scale and log scale, given Luria and Delbruck's data. Vertical axis: likelihood/10^{-23}; horizontal axis: a.

Solution to exercise 35.5 (p.447). From the six terms of the form

$$P(\mathbf{F} \mid \alpha\mathbf{m}) = \frac{\prod_i \Gamma(F_i + \alpha m_i)}{\Gamma(\sum_i F_i + \alpha)} \frac{\Gamma(\alpha)}{\prod_i \Gamma(\alpha m_i)}, \tag{35.13}$$

most factors cancel and all that remains is

$$\frac{P(\mathcal{H}_{A\rightarrow B} \mid \text{Data})}{P(\mathcal{H}_{B\rightarrow A} \mid \text{Data})} = \frac{(765+1)(235+1)}{(950+1)(50+1)} = \frac{3.8}{1}. \tag{35.14}$$

There is modest evidence in favour of $\mathcal{H}_{A\rightarrow B}$ because the three probabilities inferred for that hypothesis (roughly 0.95, 0.8, and 0.1) are more typical of the prior than are the three probabilities inferred for the other (0.24, 0.008, and 0.19). This statement sounds absurd if we think of the priors as 'uniform' over the three probabilities – surely, under a uniform prior, any settings of the probabilities are equally probable? But in the natural basis, the logit basis, the prior is proportional to $p(1-p)$, and the posterior probability ratio can be estimated by

$$\frac{0.95 \times 0.05 \times 0.8 \times 0.2 \times 0.1 \times 0.9}{0.24 \times 0.76 \times 0.008 \times 0.992 \times 0.19 \times 0.81} \simeq \frac{3}{1}, \tag{35.15}$$

which is not exactly right, but it does illustrate where the preference for $A \rightarrow B$ is coming from.

[1] `www.inference.phy.cam.ac.uk/itprnn/code/octave/luria0.m`

36

Decision Theory

Decision theory is trivial, apart from computational details (just like playing chess!).

You have a choice of various actions, a. The world may be in one of many states $\mathbf{x}$; which one occurs may be influenced by your action. The world's state has a probability distribution $P(\mathbf{x} \mid a)$. Finally, there is a utility function $U(\mathbf{x}, a)$ which specifies the payoff you receive when the world is in state $\mathbf{x}$ and you chose action a.

The task of decision theory is to select the action that maximizes the expected utility,

$$\mathcal{E}[U \mid a] = \int \mathrm{d}^K \mathbf{x} \; U(\mathbf{x}, a) P(\mathbf{x} \mid a). \tag{36.1}$$

That's all. The computational problem is to maximize $\mathcal{E}[U \mid a]$ over a. [Pessimists may prefer to define a loss function L instead of a utility function U and minimize the expected loss.]

Is there anything more to be said about decision theory?

Well, in a real problem, the choice of an appropriate utility function may be quite difficult. Furthermore, when a sequence of actions is to be taken, with each action providing information about $\mathbf{x}$, we have to take into account the effect that this anticipated information may have on our subsequent actions. The resulting mixture of forward probability and inverse probability computations in a decision problem is distinctive. In a realistic problem such as playing a board game, the tree of possible cogitations and actions that must be considered becomes enormous, and 'doing the right thing' is not simple, because the expected utility of an action cannot be computed exactly (Russell and Wefald, 1991; Baum and Smith, 1993; Baum and Smith, 1997).

Let's explore an example.

▶ 36.1 Rational prospecting

Suppose you have the task of choosing the site for a Tanzanite mine. Your final action will be to select the site from a list of N sites. The nth site has a net value called the return x_n which is initially unknown, and will be found out exactly only after site n has been chosen. [x_n equals the revenue earned from selling the Tanzanite from that site, minus the costs of buying the site, paying the staff, and so forth.] At the outset, the return x_n has a probability distribution $P(x_n)$, based on the information already available.

Before you take your final action you have the opportunity to do some prospecting. Prospecting at the nth site has a cost c_n and yields data d_n which reduce the uncertainty about x_n. [We'll assume that the returns of

the N sites are unrelated to each other, and that prospecting at one site only yields information about that site and doesn't affect the return from that site.]

Your decision problem is:

> given the initial probability distributions $P(x_1)$, $P(x_2)$, ..., $P(x_N)$, first, decide whether to prospect, and at which sites; then, in the light of your prospecting results, choose which site to mine.

For simplicity, let's make everything in the problem Gaussian and focus on the question of whether to prospect once or not. We'll assume our utility function is linear in x_n; we wish to maximize our expected return. The utility function is

The notation $P(y) = \text{Normal}(y; \mu, \sigma^2)$ indicates that y has Gaussian distribution with mean μ and variance σ^2.

$$U = x_{n_a}, \tag{36.2}$$

if no prospecting is done, where n_a is the chosen 'action' site; and, if prospecting is done, the utility is

$$U = -c_{n_p} + x_{n_a}, \tag{36.3}$$

where n_p is the site at which prospecting took place.

The prior distribution of the return of site n is

$$P(x_n) = \text{Normal}(x_n; \mu_n, \sigma_n^2). \tag{36.4}$$

If you prospect at site n, the datum d_n is a noisy version of x_n:

$$P(d_n \,|\, x_n) = \text{Normal}(d_n; x_n, \sigma^2). \tag{36.5}$$

▷ Exercise 36.1.[2] Given these assumptions, show that the prior probability distribution of d_n is

$$P(d_n) = \text{Normal}(d_n; \mu_n, \sigma^2 + \sigma_n^2) \tag{36.6}$$

(mnemonic: when independent variables add, variances add), and that the posterior distribution of x_n given d_n is

$$P(x_n \,|\, d_n) = \text{Normal}\left(x_n; \mu_n', \sigma_n^{2\prime}\right) \tag{36.7}$$

where

$$\mu_n' = \frac{d_n/\sigma^2 + \mu_n/\sigma_n^2}{1/\sigma^2 + 1/\sigma_n^2} \quad \text{and} \quad \frac{1}{\sigma_n^{2\prime}} = \frac{1}{\sigma^2} + \frac{1}{\sigma_n^2} \tag{36.8}$$

(mnemonic: when Gaussians multiply, precisions add).

To start with, let's evaluate the expected utility if we do no prospecting (i.e., choose the site immediately); then we'll evaluate the expected utility if we first prospect at one site and then make our choice. From these two results we will be able to decide whether to prospect once or zero times, and, if we prospect once, at which site.

So, first we consider the expected utility without any prospecting.

Exercise 36.2.[2] Show that the optimal action, assuming no prospecting, is to select the site with biggest mean

$$n_a = \underset{n}{\text{argmax}}\ \mu_n, \tag{36.9}$$

and the expected utility of this action is

$$\mathcal{E}[U \,|\, \text{optimal } n] = \max_n \mu_n. \tag{36.10}$$

[If your intuition says 'surely the optimal decision should take into account the different uncertainties σ_n too?', the answer to this question is 'reasonable – if so, then the utility function should be *nonlinear* in x'.]

Now the exciting bit. Should we prospect? Once we have prospected at site n_p, we will choose the site using the decision rule (36.9) with the value of mean μ_{n_p} replaced by the updated value μ'_n given by (36.8). What makes the problem exciting is that we don't yet know the value of d_n, so we don't know what our action n_a will be; indeed the whole value of doing the prospecting comes from the fact that the outcome d_n may alter the action from the one that we would have taken in the absence of the experimental information.

From the expression for the new mean in terms of d_n (36.8), and the known variance of d_n (36.6), we can compute the probability distribution of the key quantity, μ'_n, and can work out the expected utility by integrating over all possible outcomes and their associated actions.

Exercise 36.3.[2] Show that the probability distribution of the new mean μ'_n (36.8) is Gaussian with mean μ_n and variance

$$s^2 \equiv \sigma_n^2 \frac{\sigma_n^2}{\sigma^2 + \sigma_n^2}. \tag{36.11}$$

Consider prospecting at site n. Let the biggest mean of the other sites be μ_1. When we obtain the new value of the mean, μ'_n, we will choose site n and get an expected return of μ'_n if $\mu'_n > \mu_1$, and we will choose site 1 and get an expected return of μ_1 if $\mu'_n < \mu_1$.

So the expected utility of prospecting at site n, then picking the best site, is

$$\mathcal{E}[U \,|\, \text{prospect at } n] = -c_n + P(\mu'_n < \mu_1)\,\mu_1 + \int_{\mu_1}^{\infty} \mathrm{d}\mu'_n \, \mu'_n \, \text{Normal}(\mu'_n; \mu_n, s^2). \tag{36.12}$$

The difference in utility between prospecting and not prospecting is the quantity of interest, and it depends on what we would have done without prospecting; and that depends on whether μ_1 is bigger than μ_n.

$$\mathcal{E}[U \,|\, \text{no prospecting}] \;=\; \begin{cases} -\mu_1 & \text{if } \mu_1 \geq \mu_n \\ -\mu_n & \text{if } \mu_1 \leq \mu_n. \end{cases} \tag{36.13}$$

So

$$\mathcal{E}[U \,|\, \text{prospect at } n] - \mathcal{E}[U \,|\, \text{no prospecting}] = \begin{cases} -c_n + \displaystyle\int_{\mu_1}^{\infty} \mathrm{d}\mu'_n \, (\mu'_n - \mu_1) \, \text{Normal}(\mu'_n; \mu_n, s^2) & \text{if } \mu_1 \geq \mu_n \\ -c_n + \displaystyle\int_{-\infty}^{\mu_1} \mathrm{d}\mu'_n \, (\mu_1 - \mu'_n) \, \text{Normal}(\mu'_n; \mu_n, s^2) & \text{if } \mu_1 \leq \mu_n. \end{cases} \tag{36.14}$$

We can plot the change in expected utility due to prospecting (omitting c_n) as a function of the difference $(\mu_n - \mu_1)$ (horizontal axis) and the initial standard deviation σ_n (vertical axis). In the figure the noise variance is $\sigma^2 = 1$.

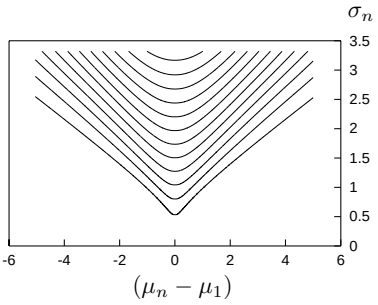


Figure 36.1. Contour plot of the gain in expected utility due to prospecting. The contours are equally spaced from 0.1 to 1.2 in steps of 0.1. To decide whether it is worth prospecting at site n, find the contour equal to c_n (the cost of prospecting); all points $[(\mu_n - \mu_1), \sigma_n]$ above that contour are worthwhile.

▶ 36.2 Further reading

If the world in which we act is a little more complicated than the prospecting problem – for example, if multiple iterations of prospecting are possible, and the cost of prospecting is uncertain – then finding the optimal balance between exploration and exploitation becomes a much harder computational problem. *Reinforcement learning* addresses approximate methods for this problem (Sutton and Barto, 1998).

▶ 36.3 Further exercises

▷ Exercise 36.4.[2] The four doors problem.

A new game show uses rules similar to those of the three doors (exercise 3.8 (p.57)), but there are four doors, and the host explains: 'First you will point to one of the doors, and then I will open one of the other doors, guaranteeing to choose a non-winner. Then you decide whether to stick with your original pick or switch to one of the remaining doors. Then I will open another non-winner (but never the current pick). You will then make your final decision by sticking with the door picked on the previous decision or by switching to the only other remaining door.'

What is the optimal strategy? Should you switch on the first opportunity? Should you switch on the second opportunity?

▷ Exercise 36.5.[3] One of the challenges of decision theory is figuring out exactly what the utility function is. The utility of money, for example, is notoriously nonlinear for most people.

In fact, the behaviour of many people cannot be captured by a coherent utility function, as illustrated by the *Allais paradox*, which runs as follows.

Which of these choices do you find most attractive?

A. £1 million guaranteed.
B. 89% chance of £1 million;
10% chance of £2.5 million;
1% chance of nothing.

Now consider these choices:

C. 89% chance of nothing;
11% chance of £1 million.
D. 90% chance of nothing;
10% chance of £2.5 million.

Many people prefer A to B, and, at the same time, D to C. Prove that these preferences are inconsistent with any utility function $U(x)$ for money.

Exercise 36.6.[4] Optimal stopping.

A large queue of N potential partners is waiting at your door, all asking to marry you. They have arrived in random order. As you meet each partner, you have to decide on the spot, based on the information so far, whether to marry them or say no. Each potential partner has a desirability d_n, which you find out if and when you meet them. You must marry one of them, but you are not allowed to go back to anyone you have said no to.

There are several ways to define the precise problem.

(a) Assuming your aim is to maximize the desirability d_n, i.e., your utility function is $d_{\hat{n}}$, where $\hat{n}$ is the partner selected, what strategy should you use?

(b) Assuming you wish very much to marry *the most desirable* person (i.e., your utility function is 1 if you achieve that, and zero otherwise); what strategy should you use?

(c) Assuming you wish very much to marry the most desirable person, and that your strategy will be 'strategy M':

> Strategy M – Meet the first M partners and say no to all of them. Memorize the maximum desirability $d_{\max}$ among them. Then meet the others in sequence, waiting until a partner with $d_n > d_{\max}$ comes along, and marry them. If none more desirable comes along, marry the final Nth partner (and feel miserable).

– what is the optimal value of M?

Exercise 36.7.[3] Regret as an objective function?

The preceding exercise (parts b and c) involved a utility function based on regret. If one married the tenth most desirable candidate, the utility function asserts that one would feel regret for having not chosen the most desirable.

Many people working in learning theory and decision theory use 'minimizing the maximal possible regret' as an objective function, but does this make sense?

Imagine that Fred has bought a lottery ticket, and offers to sell it to you before it's known whether the ticket is a winner. For simplicity say the probability that the ticket is a winner is 1/100, and if it is a winner, it is worth £10. Fred offers to sell you the ticket for £1. Do you buy it?

The possible actions are 'buy' and 'don't buy'. The utilities of the four possible action–outcome pairs are shown in table 36.2. I have assumed that the utility of small amounts of money for you is linear. If you don't buy the ticket then the utility is zero regardless of whether the ticket proves to be a winner. If you do buy the ticket you end up either losing one pound (with probability 99/100) or gaining nine (with probability 1/100). In the minimax regret community, actions are chosen to minimize the maximum possible regret. The four possible regret outcomes are shown in table 36.3. If you buy the ticket and it doesn't win, you have a regret of £1, because if you had not bought it you would have been £1 better off. If you do not buy the ticket and it wins, you have a regret of £9, because if you had bought it you would have been £9 better off. The action that minimizes the maximum possible regret is thus to buy the ticket.

	Action	
Outcome	Buy	Don't buy
No win	−1	0
Wins	+9	0

Table 36.2. Utility in the lottery ticket problem.

	Action	
Outcome	Buy	Don't buy
No win	1	0
Wins	0	9

Table 36.3. Regret in the lottery ticket problem.

Discuss whether this use of regret to choose actions can be philosophically justified.

The above problem can be turned into an investment portfolio decision problem by imagining that you have been given one pound to invest in two possible funds for one day: Fred's lottery fund, and the cash fund. If you put £f_1 into Fred's lottery fund, Fred promises to return £$9f_1$ to you if the lottery ticket is a winner, and otherwise nothing. The remaining £f_0 (with $f_0 = 1 - f_1$) is kept as cash. What is the best investment? Show that the minimax regret community will invest $f_1 = 9/10$ of their money in the high risk, high return lottery fund, and only $f_0 = 1/10$ in cash. Can this investment method be justified?

Exercise 36.8.[3] Gambling oddities (from Cover and Thomas (1991)). A horse race involving I horses occurs repeatedly, and you are obliged to bet all your money each time. Your bet at time t can be represented by

a normalized probability vector $\mathbf{b}$ multiplied by your money $m(t)$. The odds offered by the bookies are such that if horse i wins then your return is $m(t+1) = b_i o_i m(t)$. Assuming the bookies' odds are 'fair', that is,

$$\sum_i \frac{1}{o_i} = 1, \tag{36.15}$$

and assuming that the probability that horse i wins is p_i, work out the optimal betting strategy if your aim is *Cover's aim*, namely, to maximize the *expected value of* $\log m(T)$. Show that the optimal strategy sets $\mathbf{b}$ equal to $\mathbf{p}$, independent of the bookies' odds $\mathbf{o}$. Show that when this strategy is used, the money is expected to grow exponentially as:

$$2^{nW(\mathbf{b},\mathbf{p})} \tag{36.16}$$

where $W = \sum_i p_i \log b_i o_i$.

If you only bet once, is the optimal strategy any different?

Do you think this optimal strategy makes sense? Do you think that it's 'optimal', in common language, to ignore the bookies' odds? What can you conclude about 'Cover's aim'?

Exercise 36.9.[3] Two ordinary dice are thrown repeatedly; the outcome of each throw is the sum of the two numbers. Joe Shark, who says that 6 and 8 are his lucky numbers, bets even money that a 6 will be thrown before the first 7 is thrown. If you were a gambler, would you take the bet? What is your probability of winning? Joe then bets even money that an 8 will be thrown before the first 7 is thrown. Would you take the bet?

Having gained your confidence, Joe suggests combining the two bets into a single bet: he bets a larger sum, still at even odds, that an 8 and a 6 will be thrown before two 7s have been thrown. Would you take the bet? What is your probability of winning?

37

Bayesian Inference and Sampling Theory

There are two schools of statistics. Sampling theorists concentrate on having methods guaranteed to work most of the time, given minimal assumptions. Bayesians try to make inferences that take into account all available information and answer the question of interest given the particular data set. As you have probably gathered, I strongly recommend the use of Bayesian methods.

Sampling theory is the widely used approach to statistics, and most papers in most journals report their experiments using quantities like confidence intervals, significance levels, and p-values. A p-value (e.g. $p = 0.05$) is the probability, given a null hypothesis for the probability distribution of the data, that the outcome would be as extreme as, or more extreme than, the observed outcome. Untrained readers – and perhaps, more worryingly, the authors of many papers – usually interpret such a p-value as if it is a Bayesian probability (for example, the posterior probability of the null hypothesis), an interpretation that both sampling theorists and Bayesians would agree is incorrect.

In this chapter we study a couple of simple inference problems in order to compare these two approaches to statistics.

While in some cases, the answers from a Bayesian approach and from sampling theory are very similar, we can also find cases where there are significant differences. We have already seen such an example in exercise 3.15 (p.59), where a sampling theorist got a p-value smaller than 7%, and viewed this as strong evidence *against* the null hypothesis, whereas the data actually *favoured* the null hypothesis over the simplest alternative. On p.64, another example was given where the p-value was smaller than the mystical value of 5%, yet the data again favoured the null hypothesis. Thus in some cases, sampling theory can be trigger-happy, declaring results to be 'sufficiently improbable that the null hypothesis should be rejected', when those results actually weakly support the null hypothesis. As we will now see, there are also inference problems where sampling theory fails to detect 'significant' evidence where a Bayesian approach and everyday intuition agree that the evidence is strong. Most telling of all are the inference problems where the 'significance' assigned by sampling theory changes depending on irrelevant factors concerned with the design of the experiment.

This chapter is only provided for those readers who are curious about the sampling theory / Bayesian methods debate. If you find any of this chapter tough to understand, please skip it. There is no point trying to understand the debate. Just use Bayesian methods – they are much easier to understand than the debate itself!

▶ 37.1 A medical example

> We are trying to reduce the incidence of an unpleasant disease called *microsoftus*. Two vaccinations, A and B, are tested on a group of volunteers. Vaccination B is a control treatment, a placebo treatment with no active ingredients. Of the 40 subjects, 30 are randomly assigned to have treatment A and the other 10 are given the control treatment B. We observe the subjects for one year after their vaccinations. Of the 30 in group A, one contracts *microsoftus*. Of the 10 in group B, three contract *microsoftus*.
>
> Is treatment A better than treatment B?

Sampling theory has a go

The standard sampling theory approach to the question 'is A better than B?' is to construct a *statistical test*. The test usually compares a hypothesis such as

$\mathcal{H}_1$: 'A and B have different effectivenesses'

with a null hypothesis such as

$\mathcal{H}_0$: 'A and B have exactly the same effectivenesses as each other'.

A novice might object 'no, no, I want to compare the hypothesis "A is better than B" with the alternative "B is better than A"!' but such objections are not welcome in sampling theory.

Once the two hypotheses have been defined, the first hypothesis is scarcely mentioned again – attention focuses solely on the null hypothesis. It makes me laugh to write this, but it's true! The null hypothesis is accepted or rejected purely on the basis of how unexpected the data were to $\mathcal{H}_0$, not on how much better $\mathcal{H}_1$ predicted the data. One chooses a *statistic* which measures how much a data set deviates from the null hypothesis. In the example here, the standard statistic to use would be one called χ^2 (chi-squared). To compute χ^2, we take the difference between each data measurement and its *expected* value *assuming the null hypothesis to be true*, and divide the square of that difference by the *variance* of the measurement, *assuming the null hypothesis to be true*. In the present problem, the four data measurements are the integers F_{A+}, F_{A-}, F_{B+}, and F_{B-}, that is, the number of subjects given treatment A who contracted *microsoftus* (F_{A+}), the number of subjects given treatment A who didn't (F_{A-}), and so forth. The definition of χ^2 is:

$$\chi^2 = \sum_i \frac{(F_i - \langle F_i \rangle)^2}{\langle F_i \rangle}. \tag{37.1}$$

Actually, in my elementary statistics book (Spiegel, 1988) I find Yates's correction is recommended:

$$\chi^2 = \sum_i \frac{(|F_i - \langle F_i \rangle| - 0.5)^2}{\langle F_i \rangle}. \tag{37.2}$$

If you want to know about Yates's correction, read a sampling theory textbook. The point of this chapter is not to teach sampling theory; I merely mention Yates's correction because it is what a professional sampling theorist might use.

In this case, given the null hypothesis that treatments A and B are equally effective, and have rates f_+ and f_- for the two outcomes, the expected counts are:

$$\begin{array}{ll} \langle F_{A+} \rangle = f_+ N_A & \langle F_{A-} \rangle = f_- N_A \\ \langle F_{B+} \rangle = f_+ N_B & \langle F_{B-} \rangle = f_- N_B. \end{array} \tag{37.3}$$

The test accepts or rejects the null hypothesis on the basis of how big χ^2 is. To make this test precise, and give it a 'significance level', we have to work out what the *sampling distribution* of χ^2 is, taking into account the fact that the four data points are not independent (they satisfy the two constraints $F_{A+} + F_{A-} = N_A$ and $F_{B+} + F_{B-} = N_B$) and the fact that the parameters $f_\pm$ are not known. These three constraints reduce the *number of degrees of freedom* in the data from four to one. [If you want to learn more about computing the 'number of degrees of freedom', read a sampling theory book; in Bayesian methods we don't need to know all that, and quantities equivalent to the number of degrees of freedom pop straight out of a Bayesian analysis when they are appropriate.] These sampling distributions are tabulated by sampling theory gnomes and come accompanied by warnings about the conditions under which they are accurate. For example, standard tabulated distributions for χ^2 are only accurate if the expected numbers F_i are about 5 or more.

The sampling distribution of a statistic is the probability distribution of its value under repetitions of the experiment, assuming that the null hypothesis is true.

Once the data arrive, sampling theorists estimate the unknown parameters $f_\pm$ of the null hypothesis from the data:

$$\hat{f}_+ = \frac{F_{A+} + F_{B+}}{N_A + N_B}, \quad \hat{f}_- = \frac{F_{A-} + F_{B-}}{N_A + N_B}, \tag{37.4}$$

and evaluate χ^2. At this point, the sampling theory school divides itself into two camps. One camp uses the following protocol: first, before looking at the data, pick the significance level of the test (e.g. 5%), and determine the critical value of χ^2 above which the null hypothesis will be rejected. (The significance level is the fraction of times that the statistic χ^2 would exceed the critical value, if the null hypothesis were true.) Then evaluate χ^2, compare with the critical value, and declare the outcome of the test, and its significance level (which was fixed beforehand).

The second camp looks at the data, finds χ^2, then looks in the table of χ^2-distributions for the significance level, p, for which the observed value of χ^2 would be the critical value. The result of the test is then reported by giving this value of p, which is the fraction of times that a result as extreme as the one observed, or more extreme, would be expected to arise if the null hypothesis were true.

Let's apply these two methods. First camp: let's pick 5% as our significance level. The critical value for χ^2 with one degree of freedom is $\chi^2_{0.05} = 3.84$. The estimated values of $f_\pm$ are

$$f_+ = 1/10, \quad f_- = 9/10. \tag{37.5}$$

The expected values of the four measurements are

$$\langle F_{A+} \rangle = 3 \tag{37.6}$$
$$\langle F_{A-} \rangle = 27 \tag{37.7}$$
$$\langle F_{B+} \rangle = 1 \tag{37.8}$$
$$\langle F_{B-} \rangle = 9 \tag{37.9}$$

and χ^2 (as defined in equation (37.1)) is

$$\chi^2 = 5.93. \tag{37.10}$$

Since this value exceeds 3.84, we reject the null hypothesis that the two treatments are equivalent at the 0.05 significance level. However, if we use Yates's correction, we find $\chi^2 = 3.33$, and therefore accept the null hypothesis.

Camp two runs a finger across the χ^2 table found at the back of any good sampling theory book and finds $\chi^2_{.10} = 2.71$. Interpolating between $\chi^2_{.10}$ and $\chi^2_{.05}$, camp two reports 'the p-value is $p = 0.07$'.

Notice that this answer does not say how much more effective A is than B, it simply says that A is 'significantly' different from B. And here, 'significant' means only 'statistically significant', not practically significant.

The man in the street, reading the statement that 'the treatment was significantly different from the control ($p = 0.07$)', might come to the conclusion that 'there is a 93% chance that the treatments differ in effectiveness'. But what '$p = 0.07$' actually means is 'if you did this experiment many times, and the two treatments *had* equal effectiveness, then 7% of the time you would find a value of χ^2 more extreme than the one that happened here'. This has almost nothing to do with what we want to know, which is how likely it is that treatment A is better than B.

Let me through, I'm a Bayesian

OK, now let's *infer* what we really want to know. We scrap the hypothesis that the two treatments have exactly equal effectivenesses, since we do not believe it. There are two unknown parameters, p_{A+} and p_{B+}, which are the probabilities that people given treatments A and B, respectively, contract the disease.

Given the data, we can infer these two probabilities, and we can answer questions of interest by examining the posterior distribution.

The posterior distribution is

$$P(p_{A+}, p_{B+} \,|\, \{F_i\}) = \frac{P(\{F_i\} \,|\, p_{A+}, p_{B+}) P(p_{A+}, p_{B+})}{P(\{F_i\})}. \tag{37.11}$$

The likelihood function is

$$P(\{F_i\} \,|\, p_{A+}, p_{B+}) = \binom{N_A}{F_{A+}} p_{A+}^{F_{A+}} p_{A-}^{F_{A-}} \binom{N_B}{F_{B+}} p_{B+}^{F_{B+}} p_{B-}^{F_{B-}} \tag{37.12}$$

$$= \binom{30}{1} p_{A+}^{1} p_{A-}^{29} \binom{10}{3} p_{B+}^{3} p_{B-}^{7}. \tag{37.13}$$

What prior distribution should we use? The prior distribution gives us the opportunity to include knowledge from other experiments, or a prior belief that the two parameters p_{A+} and p_{B+}, while different from each other, are expected to have similar values.

Here we will use the simplest vanilla prior distribution, a uniform distribution over each parameter.

$$P(p_{A+}, p_{B+}) = 1. \tag{37.14}$$

We can now plot the posterior distribution. Given the assumption of a separable prior on p_{A+} and p_{B+}, the posterior distribution is also separable:

$$P(p_{A+}, p_{B+} \,|\, \{F_i\}) = P(p_{A+} \,|\, F_{A+}, F_{A-}) P(p_{B+} \,|\, F_{B+}, F_{B-}). \tag{37.15}$$

The two posterior distributions are shown in figure 37.1 (except the graphs are not normalized) and the joint posterior probability is shown in figure 37.2.

If we want to know the answer to the question 'how probable is it that p_{A+} is smaller than p_{B+}?', we can answer exactly that question by computing the posterior probability

$$P(p_{A+} < p_{B+} \,|\, \text{Data}), \tag{37.16}$$

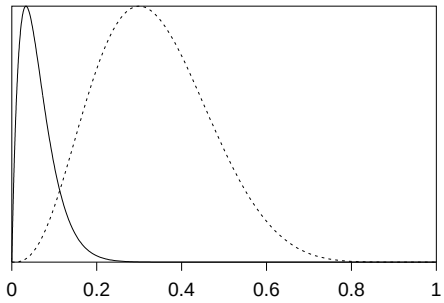


Figure 37.1. Posterior probabilities of the two effectivenesses. Treatment A – solid line; B – dotted line.

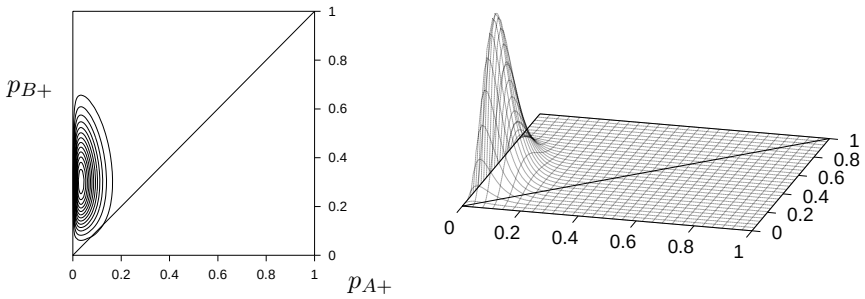


Figure 37.2. Joint posterior probability of the two effectivenesses – contour plot and surface plot.

which is the integral of the joint posterior probability $P(p_{A+}, p_{B+} \,|\, \text{Data})$ shown in figure 37.2 over the region in which $p_{A+} < p_{B+}$, i.e., the shaded triangle in figure 37.3. The value of this integral (obtained by a straightforward numerical integration of the likelihood function (37.13) over the relevant region) is $P(p_{A+} < p_{B+} \,|\, \text{Data}) = 0.990$.

Thus there is a 99% chance, given the data and our prior assumptions, that treatment A is superior to treatment B. In conclusion, according to our Bayesian model, the data (1 out of 30 contracted the disease after vaccination A, and 3 out of 10 contracted the disease after vaccination B) give very strong evidence – about 99 to one – that treatment A is superior to treatment B.

In the Bayesian approach, it is also easy to answer other relevant questions. For example, if we want to know 'how likely is it that treatment A is ten times more effective than treatment B?', we can integrate the joint posterior probability $P(p_{A+}, p_{B+} \,|\, \text{Data})$ over the region in which $p_{A+} < 10\, p_{B+}$ (figure 37.4).

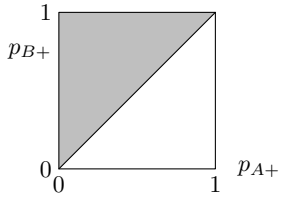


Figure 37.3. The proposition $p_{A+} < p_{B+}$ is true for all points in the shaded triangle. To find the probability of this proposition we integrate the joint posterior probability $P(p_{A+}, p_{B+} \,|\, \text{Data})$ (figure 37.2) over this region.

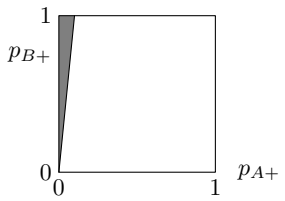


Figure 37.4. The proposition $p_{A+} < 10\, p_{B+}$ is true for all points in the shaded triangle.

Model comparison

If there were a situation in which we really did want to compare the two hypotheses $\mathcal{H}_0$: $p_{A+} = p_{B+}$ and $\mathcal{H}_1$: $p_{A+} \neq p_{B+}$, we can of course do this directly with Bayesian methods also.

As an example, consider the data set:

D: One subject, given treatment A, subsequently contracted *microsoftus*.
One subject, given treatment B, did not.

Treatment	A	B
Got disease	1	0
Did not	0	1
Total treated	1	1

How strongly does this data set favour $\mathcal{H}_1$ over $\mathcal{H}_0$?

We answer this question by computing the evidence for each hypothesis. Let's assume uniform priors over the unknown parameters of the models. The first hypothesis $\mathcal{H}_0$: $p_{A+} = p_{B+}$ has just one unknown parameter, let's call it p.

$$P(p \,|\, \mathcal{H}_0) = 1 \quad p \in (0,1). \tag{37.17}$$

We'll use the uniform prior over the two parameters of model $\mathcal{H}_1$ that we used before:

$$P(p_{A+}, p_{B+} \,|\, \mathcal{H}_1) = 1 \quad p_{A+} \in (0,1),\ p_{B+} \in (0,1). \tag{37.18}$$

Now, the probability of the data D under model $\mathcal{H}_0$ is the normalizing constant from the inference of p given D:

$$\begin{aligned} P(D \,|\, \mathcal{H}_0) &= \int \mathrm{d}p\, P(D \,|\, p) P(p \,|\, \mathcal{H}_0) && (37.19)\\ &= \int \mathrm{d}p\, p(1-p) \times 1 && (37.20)\\ &= 1/6. && (37.21) \end{aligned}$$

The probability of the data D under model $\mathcal{H}_1$ is given by a simple two-dimensional integral:

$$\begin{aligned} P(D \,|\, \mathcal{H}_1) &= \int\int \mathrm{d}p_{A+}\, \mathrm{d}p_{B+}\, P(D \,|\, p_{A+}, p_{B+}) P(p_{A+}, p_{B+} \,|\, \mathcal{H}_1) && (37.22)\\ &= \int \mathrm{d}p_{A+}\, p_{A+} \int \mathrm{d}p_{B+}\, (1 - p_{B+}) && (37.23)\\ &= 1/2 \times 1/2 && (37.24)\\ &= 1/4. && (37.25) \end{aligned}$$

Thus the evidence ratio in favour of model $\mathcal{H}_1$, which asserts that the two effectivenesses are unequal, is

$$\frac{P(D \,|\, \mathcal{H}_1)}{P(D \,|\, \mathcal{H}_0)} = \frac{1/4}{1/6} = \frac{0.6}{0.4}. \tag{37.26}$$

So if the prior probability over the two hypotheses was 50:50, the posterior probability is 60:40 in favour of $\mathcal{H}_1$. □

Is it not easy to get sensible answers to well-posed questions using Bayesian methods?

[The sampling theory answer to this question would involve the identical significance test that was used in the preceding problem; that test would yield a 'not significant' result. I think it is greatly preferable to acknowledge what is obvious to the intuition, namely that the data D do give *weak* evidence in favour of $\mathcal{H}_1$. Bayesian methods quantify how weak the evidence is.]

▶ 37.2 Dependence of p-values on irrelevant information

In an expensive laboratory, Dr. Bloggs tosses a coin labelled a and b twelve times and the outcome is the string

$$aaabaaaabaab,$$

which contains three bs and nine as.

What evidence do these data give that the coin is biased in favour of a?

Dr. Bloggs consults his sampling theory friend who says 'let r be the number of bs and $n = 12$ be the total number of tosses; I view r as the random variable and find the probability of r taking on the value $r = 3$ or a more extreme value, assuming the null hypothesis $p_a = 0.5$ to be true'. He thus computes

$$\begin{aligned} P(r \leq 3 \,|\, n=12, \mathcal{H}_0) &= \sum_{r=0}^{3} \binom{n}{r} 1/2^n = \left(\binom{12}{0} + \binom{12}{1} + \binom{12}{2} + \binom{12}{3} \right) 1/2^{12} \\ &= 0.07, \end{aligned} \tag{37.27}$$

and reports 'at the significance level of 5%, there is not significant evidence of bias in favour of a'. Or, if the friend prefers to report p-values rather than simply compare p with 5%, he would report 'the p-value is 7%, which is not conventionally viewed as significantly small'. If a two-tailed test seemed more appropriate, he might compute the two-tailed area, which is twice the above probability, and report 'the p-value is 15%, which is not significantly small'. We won't focus on the issue of the choice between the one-tailed and two-tailed tests, as we have bigger fish to catch.

Dr. Bloggs pays careful attention to the calculation (37.27), and responds 'no, no, the random variable in the experiment was not r: I decided before running the experiment that I would keep tossing the coin until I saw three bs; the random variable is thus n'.

> Such experimental designs are not unusual. In my experiments on error-correcting codes I often simulate the decoding of a code until a chosen number r of block errors (bs) has occurred, since the error on the inferred value of $\log p_{\mathrm{b}}$ goes roughly as $\sqrt{r}$, independent of n.

Exercise 37.1.[2] Find the Bayesian inference about the bias p_a of the coin given the data, and determine whether a Bayesian's inferences depend on what stopping rule was in force.

According to sampling theory, a different calculation is required in order to assess the 'significance' of the result $n = 12$. The probability distribution of n given $\mathcal{H}_0$ is the probability that the first $n-1$ tosses contain exactly $r-1$ bs and then the nth toss is a b.

$$P(n \,|\, \mathcal{H}_0, r) = \binom{n-1}{r-1} 1/2^n. \tag{37.28}$$

The sampling theorist thus computes

$$P(n \geq 12 \,|\, r=3, \mathcal{H}_0) = 0.03. \tag{37.29}$$

He reports back to Dr. Bloggs, 'the p-value is 3% – there *is* significant evidence of bias after all!'

What do you think Dr. Bloggs should do? Should he publish the result, with this marvellous p-value, in one of the journals that insists that all experimental results have their 'significance' assessed using sampling theory? Or should he boot the sampling theorist out of the door and seek a coherent method of assessing significance, one that does not depend on the stopping rule?

At this point the audience divides in two. Half the audience intuitively feel that the stopping rule is irrelevant, and don't need any convincing that the answer to exercise 37.1 (p.463) is 'the inferences about p_a do not depend on the stopping rule'. The other half, perhaps on account of a thorough

training in sampling theory, intuitively feel that Dr. Bloggs's stopping rule, which stopped tossing the moment the third b appeared, may have biased the experiment somehow. If you are in the second group, I encourage you to reflect on the situation, and hope you'll eventually come round to the view that is consistent with the likelihood principle, which is that the stopping rule is *not* relevant to what we have learned about p_a.

As a thought experiment, consider some onlookers who (in order to save money) are spying on Dr. Bloggs's experiments: each time he tosses the coin, the spies update the values of r and n. The spies are eager to make inferences from the data as soon as each new result occurs. Should the spies' beliefs about the bias of the coin depend on Dr. Bloggs's intentions regarding the continuation of the experiment?

The fact that the p-values of sampling theory *do* depend on the stopping rule (indeed, whole volumes of the sampling theory literature are concerned with the task of assessing 'significance' when a complicated stopping rule is required – 'sequential probability ratio tests', for example) seems to me a compelling argument for having nothing to do with p-values at all. A Bayesian solution to this inference problem was given in sections 3.2 and 3.3 and exercise 3.15 (p.59).

Would it help clarify this issue if I added one more scene to the story? The janitor, who's been eavesdropping on Dr. Bloggs's conversation, comes in and says 'I happened to notice that just after you stopped doing the experiments on the coin, the Officer for Whimsical Departmental Rules ordered the immediate destruction of all such coins. Your coin was therefore destroyed by the departmental safety officer. There is no way you could have continued the experiment much beyond $n = 12$ tosses. Seems to me, you need to recompute your p-value?'

▶ 37.3 Confidence intervals

In an experiment in which data D are obtained from a system with an unknown parameter θ, a standard concept in sampling theory is the idea of a *confidence interval* for θ. Such an interval $(\theta_{\min}(D), \theta_{\max}(D))$ has associated with it a *confidence level* such as 95% which is informally interpreted as 'the probability that θ lies in the confidence interval'.

Let's make precise what the confidence level really means, then give an example. A confidence interval is a function $(\theta_{\min}(D), \theta_{\max}(D))$ of the data set D. The confidence level of the confidence interval is a property that we can compute before the data arrive. We imagine generating many data sets from a particular true value of θ, and calculating the interval $(\theta_{\min}(D), \theta_{\max}(D))$, and then checking whether the true value of θ lies in that interval. If, averaging over all these imagined repetitions of the experiment, the true value of θ lies in the confidence interval a fraction f of the time, and this property holds for all true values of θ, then the confidence level of the confidence interval is f.

For example, if θ is the mean of a Gaussian distribution which is known to have standard deviation 1, and D is a sample from that Gaussian, then $(\theta_{\min}(D), \theta_{\max}(D)) = (D-2, D+2)$ is a 95% confidence interval for θ.

Let us now look at a simple example where the meaning of the confidence level becomes clearer. Let the parameter θ be an integer, and let the data be a pair of points x_1, x_2, drawn independently from the following distribution:

$$P(x \mid \theta) = \begin{cases} 1/2 & x = \theta \\ 1/2 & x = \theta + 1 \\ 0 & \text{for other values of } x. \end{cases} \tag{37.30}$$

For example, if θ were 39, then we could expect the following data sets:

$$\begin{array}{rl} D = (x_1, x_2) = (39, 39) & \text{with probability } 1/4; \\ (x_1, x_2) = (39, 40) & \text{with probability } 1/4; \\ (x_1, x_2) = (40, 39) & \text{with probability } 1/4; \\ (x_1, x_2) = (40, 40) & \text{with probability } 1/4. \end{array} \tag{37.31}$$

We now consider the following confidence interval:

$$[\theta_{\min}(D), \theta_{\max}(D)] = [\min(x_1, x_2), \min(x_1, x_2)]. \tag{37.32}$$

For example, if $(x_1, x_2) = (40, 39)$, then the confidence interval for θ would be $[\theta_{\min}(D), \theta_{\max}(D)] = [39, 39]$.

Let's think about this confidence interval. What is its confidence level? By considering the four possibilities shown in (37.31), we can see that there is a 75% chance that the confidence interval will contain the true value. The confidence interval therefore has a confidence level of 75%, by definition.

Now, what if the data we acquire are $(x_1, x_2) = (29, 29)$? Well, we can compute the confidence interval, and it is $[29, 29]$. So shall we report this interval, and its associated confidence level, 75%? This would be correct by the rules of sampling theory. But does this make sense? What do we actually know in this case? Intuitively, or by Bayes' theorem, it is clear that θ could either be 29 or 28, and both possibilities are equally likely (if the prior probabilities of 28 and 29 were equal). The posterior probability of θ is 50% on 29 and 50% on 28.

What if the data are $(x_1, x_2) = (29, 30)$? In this case, the confidence interval is still $[29, 29]$, and its associated confidence level is 75%. But in this case, by Bayes' theorem, or common sense, we are 100% sure that θ is 29.

In neither case is the probability that θ lies in the '75% confidence interval' equal to 75%!

Thus

1. the way in which many people interpret the confidence levels of sampling theory is *incorrect*;

2. given some data, what people usually want to know (whether they know it or not) is a Bayesian posterior probability distribution.

Are all these examples contrived? Am I making a fuss about nothing? If you are sceptical about the dogmatic views I have expressed, I encourage you to look at a case study: look in depth at exercise 35.4 (p.446) and the reference (Kepler and Oprea, 2001), in which sampling theory estimates and confidence intervals for a mutation rate are constructed. Try both methods on simulated data – the Bayesian approach based on simply computing the likelihood function, and the confidence interval from sampling theory; and let me know if you don't find that the Bayesian answer is always better than the sampling theory answer; and often much, much better. This suboptimality of sampling theory, achieved with great effort, is why I am passionate about Bayesian methods. Bayesian methods are straightforward, and they optimally use all the information in the data.

▶ 37.4 Some compromise positions

Let's end on a conciliatory note. Many sampling theorists are pragmatic – they are happy to choose from a selection of statistical methods, choosing whichever has the 'best' long-run properties. In contrast, I have no problem

with the idea that there is only *one* answer to a well-posed problem; but it's not essential to convert sampling theorists to this viewpoint: instead, we can offer them Bayesian estimators and Bayesian confidence intervals, and request that the sampling theoretical properties of these methods be evaluated. We don't need to mention that the methods are derived from a Bayesian perspective. If the sampling properties are good then the pragmatic sampling theorist will choose to use the Bayesian methods. It is indeed the case that many Bayesian methods have good sampling-theoretical properties. Perhaps it's not surprising that a method that gives the optimal answer for each individual case should also be good in the long run!

Another piece of common ground can be conceded: while I believe that most well-posed inference problems have a unique correct answer, which can be found by Bayesian methods, not all problems are well-posed. A common question arising in data modelling is 'am I using an appropriate model?' Model criticism, that is, hunting for defects in a current model, is a task that may be aided by sampling theory tests, in which the null hypothesis ('the current model is correct') is well defined, but the alternative model is not specified. One could use sampling theory measures such as p-values to guide one's search for the aspects of the model most in need of scrutiny.

Further reading

My favourite reading on this topic includes (Jaynes, 1983; Gull, 1988; Loredo, 1990; Berger, 1985; Jaynes, 2003). Treatises on Bayesian statistics from the statistics community include (Box and Tiao, 1973; O'Hagan, 1994).

▶ 37.5 Further exercises

▷ Exercise 37.2.[3C] A traffic survey records traffic on two successive days. On Friday morning, there are 12 vehicles in one hour. On Saturday morning, there are 9 vehicles in half an hour. Assuming that the vehicles are Poisson distributed with rates λ_F and λ_S (in vehicles per hour) respectively,

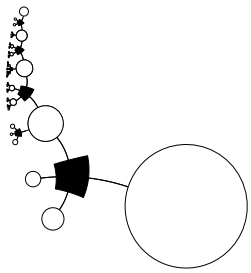
(a) is λ_S greater than λ_F?

(b) by what factor is λ_S bigger or smaller than λ_F?

▷ Exercise 37.3.[3C] Write a program to compare treatments A and B given data F_{A+}, F_{A-}, F_{B+}, F_{B-} as described in section 37.1. The outputs of the program should be (a) the probability that treatment A is more effective than treatment B; (b) the probability that $p_{A+} < 10\,p_{B+}$; (c) the probability that $p_{B+} < 10\,p_{A+}$.

Part V

Neural networks



38

Introduction to Neural Networks

In the field of neural networks, we study the properties of networks of idealized 'neurons'.

Three motivations underlie work in this broad and interdisciplinary field.

Biology. The task of understanding how the brain works is one of the outstanding unsolved problems in science. Some neural network models are intended to shed light on the way in which computation and memory are performed by brains.

Engineering. Many researchers would like to create machines that can 'learn', perform 'pattern recognition' or 'discover patterns in data'.

Complex systems. A third motivation for being interested in neural networks is that they are complex adaptive systems whose properties are interesting in their own right.

I should emphasize several points at the outset.

- This book gives only a taste of this field. There are many interesting neural network models which we will not have time to touch on.
- The models that we discuss are not intended to be faithful models of biological systems. If they are at all relevant to biology, their relevance is on an abstract level.
- I will describe some neural network methods that are widely used in nonlinear data modelling, but I will not be able to give a full description of the state of the art. If you wish to solve real problems with neural networks, please read the relevant papers.

▶ 38.1 Memories

In the next few chapters we will meet several neural network models which come with simple learning algorithms which make them function as *memories*. Perhaps we should dwell for a moment on the conventional idea of memory in digital computation. A memory (a string of 5000 bits describing the name of a person and an image of their face, say) is stored in a digital computer at an *address*. To retrieve the memory you need to know the address. The address has nothing to do with the memory itself. Notice the properties that this scheme does *not* have:

1. Address-based memory is *not* associative. Imagine you know half of a memory, say someone's face, and you would like to recall the rest of the

memory – their name. If your memory is address-based then you can't get at a memory without knowing the address. [Computer scientists have devoted effort to wrapping traditional address-based memories inside cunning software to produce content-addressable memories, but content-addressability does not come naturally. It has to be added on.]

2. Address-based memory is *not* robust or fault-tolerant. If a one-bit mistake is made in specifying the *address* then a completely different memory will be retrieved. If one bit of a *memory* is flipped then whenever that memory is retrieved the error will be present. Of course, in all modern computers, error-correcting codes are used in the memory, so that small numbers of errors can be detected and corrected. But this error-tolerance is not an intrinsic property of the memory system. If minor damage occurs to certain hardware that implements memory retrieval, it is likely that all functionality will be catastrophically lost.

3. Address-based memory is not distributed. In a serial computer that is accessing a particular memory, only a tiny fraction of the devices participate in the memory recall: the CPU and the circuits that are storing the required byte. All the other millions of devices in the machine are sitting idle.

   Are there models of truly parallel computation, in which multiple devices participate in all computations? [Present-day parallel computers scarcely differ from serial computers from this point of view. Memory retrieval works in just the same way, and control of the computation process resides in CPUs. There are simply a few more CPUs. Most of the devices sit idle most of the time.]

Biological memory systems are completely different.

1. Biological memory is associative. Memory recall is *content-addressable*. Given a person's name, we can often recall their face; and *vice versa*. Memories are apparently recalled spontaneously, not just at the request of some CPU.

2. Biological memory recall is error-tolerant and robust.

   - Errors in the cues for memory recall can be corrected. An example asks you to recall 'An American politician who was very intelligent and whose politician father did not like broccoli'. Many people think of president Bush – even though one of the cues contains an error.
   - Hardware faults can also be tolerated. Our brains are noisy lumps of meat that are in a continual state of change, with cells being damaged by natural processes, alcohol, and boxing. While the cells in our brains and the proteins in our cells are continually changing, many of our memories persist unaffected.

3. Biological memory is parallel and distributed – not *completely* distributed throughout the whole brain: there does appear to be some functional specialization – but in the parts of the brain where memories are stored, it seems that many neurons participate in the storage of multiple memories.

These properties of biological memory systems motivate the study of 'artificial neural networks' – parallel distributed computational systems consisting

of many interacting simple elements. The hope is that these model systems might give some hints as to how neural computation is achieved in real biological neural networks.

▶ 38.2 Terminology

Each time we describe a neural network algorithm we will typically specify three things. [If any of this terminology is hard to understand, it's probably best to dive straight into the next chapter.]

Architecture. The architecture specifies what variables are involved in the network and their topological relationships – for example, the variables involved in a neural net might be the *weights* of the connections between the neurons, along with the *activities* of the neurons.

Activity rule. Most neural network models have short time-scale dynamics: local rules define how the *activities* of the neurons change in response to each other. Typically the activity rule depends on the *weights* (the parameters) in the network.

Learning rule. The learning rule specifies the way in which the neural network's *weights* change with time. This learning is usually viewed as taking place on a longer time scale than the time scale of the dynamics under the activity rule. Usually the learning rule will depend on the *activities* of the neurons. It may also depend on the values of *target* values supplied by a *teacher* and on the current value of the weights.

Where do these rules come from? Often, activity rules and learning rules are invented by imaginative researchers. Alternatively, activity rules and learning rules may be *derived* from carefully chosen *objective functions*.

Neural network algorithms can be roughly divided into two classes.

Supervised neural networks are given data in the form of *inputs* and *targets*, the targets being a *teacher*'s specification of what the neural network's response to the input should be.

Unsupervised neural networks are given data in an undivided form – simply a set of examples $\{\mathbf{x}\}$. Some learning algorithms are intended simply to memorize these data in such a way that the examples can be recalled in the future. Other algorithms are intended to 'generalize', to discover 'patterns' in the data, or extract the underlying 'features' from them.

Some unsupervised algorithms are able to make predictions – for example, some algorithms can 'fill in' missing variables in an example $\mathbf{x}$ – and so can also be viewed as supervised networks.

39

The Single Neuron as a Classifier

▶ 39.1 The single neuron

We will study a single neuron for two reasons. First, many neural network models are built out of single neurons, so it is good to understand them in detail. And second, a single neuron is itself capable of 'learning' – indeed, various standard statistical methods can be viewed in terms of single neurons – so this model will serve as a first example of a *supervised neural network*.

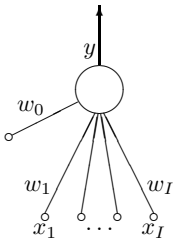


Figure 39.1. A single neuron

Definition of a single neuron

We will start by defining the architecture and the activity rule of a single neuron, and we will then derive a learning rule.

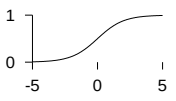
Architecture. A single neuron has a number I of *inputs* x_i and one *output* which we will here call y. (See figure 39.1.) Associated with each input is a *weight* w_i ($i = 1, \ldots, I$). There may be an additional parameter w_0 of the neuron called a *bias* which we may view as being the weight associated with an input x_0 that is permanently set to 1. The single neuron is a *feedforward* device – the connections are directed from the inputs to the output of the neuron.

Activity rule. The activity rule has two steps.

1. First, in response to the imposed inputs $\mathbf{x}$, we compute the *activation* of the neuron,
$$a = \sum_i w_i x_i, \tag{39.1}$$
where the sum is over $i = 0, \ldots, I$ if there is a bias and $i = 1, \ldots, I$ otherwise.
2. Second, the *output* y is set as a function $f(a)$ of the activation. The output is also called the *activity* of the neuron, not to be confused with the activation a. There are several possible *activation functions*; here are the most popular.

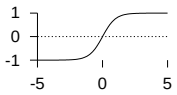
activation → activity
$a \quad \rightarrow \quad y(a)$

   (a) Deterministic activation functions:
      i. Linear.
$$y(a) = a. \tag{39.2}$$
      ii. Sigmoid (logistic function).
$$y(a) = \frac{1}{1 + e^{-a}} \qquad (y \in (0, 1)). \tag{39.3}$$



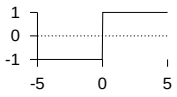
iii. Sigmoid (tanh).

$$y(a) = \tanh(a) \qquad (y \in (-1, 1)). \tag{39.4}$$



iv. Threshold function.

$$y(a) = \Theta(a) \equiv \begin{cases} 1 & a > 0 \\ -1 & a \leq 0. \end{cases} \tag{39.5}$$



(b) Stochastic activation functions: y is stochastically selected from ± 1.

i. Heat bath.

$$y(a) = \begin{cases} 1 & \text{with probability } \dfrac{1}{1+e^{-a}} \\ -1 & \text{otherwise.} \end{cases} \tag{39.6}$$

ii. The Metropolis rule produces the output in a way that depends on the previous output state y:

Compute $\Delta = ay$
If $\Delta \leq 0$, flip y to the other state
Else flip y to the other state with probability $e^{-\Delta}$.

▶ 39.2 Basic neural network concepts

A neural network implements a function $y(\mathbf{x}; \mathbf{w})$; the 'output' of the network, y, is a nonlinear function of the 'inputs' $\mathbf{x}$; this function is parameterized by 'weights' $\mathbf{w}$.

We will study a single neuron which produces an output between 0 and 1 as the following function of $\mathbf{x}$:

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}. \tag{39.7}$$

Exercise 39.1.[1] In what contexts have we encountered the function $y(\mathbf{x}; \mathbf{w}) = 1/(1 + e^{-\mathbf{w}\cdot\mathbf{x}})$ already?

Motivations for the linear logistic function

In section 11.2 we studied 'the best detection of pulses', assuming that one of two signals $\mathbf{x}_0$ and $\mathbf{x}_1$ had been transmitted over a Gaussian channel with variance–covariance matrix $\mathbf{A}^{-1}$. We found that the probability that the source signal was $s = 1$ rather than $s = 0$, given the received signal $\mathbf{y}$, was

$$P(s = 1 \,|\, \mathbf{y}) = \frac{1}{1 + \exp(-a(\mathbf{y}))}, \tag{39.8}$$

where $a(\mathbf{y})$ was a linear function of the received vector,

$$a(\mathbf{y}) = \mathbf{w}^{\mathsf{T}}\mathbf{y} + \theta, \tag{39.9}$$

with $\mathbf{w} \equiv \mathbf{A}(\mathbf{x}_1 - \mathbf{x}_0)$.

The linear logistic function can be motivated in several other ways – see the exercises.

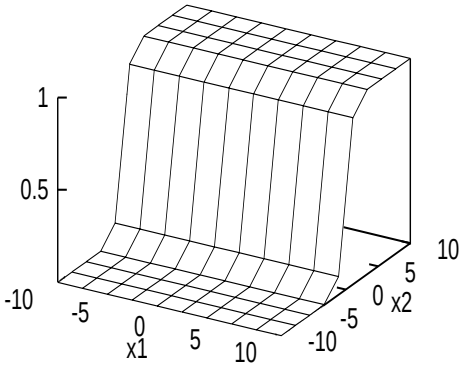


Figure 39.2. Output of a simple neural network as a function of its input.

Input space and weight space

For convenience let us study the case where the input vector $\mathbf{x}$ and the parameter vector $\mathbf{w}$ are both two-dimensional: $\mathbf{x} = (x_1, x_2)$, $\mathbf{w} = (w_1, w_2)$. Then we can spell out the function performed by the neuron thus:

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}. \tag{39.10}$$

Figure 39.2 shows the output of the neuron as a function of the input vector, for $\mathbf{w} = (0, 2)$. The two horizontal axes of this figure are the inputs x_1 and x_2, with the output y on the vertical axis. Notice that on any line perpendicular to $\mathbf{w}$, the output is constant; and along a line in the direction of $\mathbf{w}$, the output is a sigmoid function.

We now introduce the idea of *weight space*, that is, the parameter space of the network. In this case, there are two parameters w_1 and w_2, so the weight space is two dimensional. This weight space is shown in figure 39.3. For a selection of values of the parameter vector $\mathbf{w}$, smaller inset figures show the function of $\mathbf{x}$ performed by the network when $\mathbf{w}$ is set to those values. Each of these smaller figures is equivalent to figure 39.2. Thus each *point* in $\mathbf{w}$ space corresponds to a *function* of $\mathbf{x}$. Notice that the gain of the sigmoid function (the gradient of the ramp) increases as the magnitude of $\mathbf{w}$ increases.

Now, the central idea of supervised neural networks is this. Given *examples* of a relationship between an input vector $\mathbf{x}$, and a target t, we hope to make the neural network 'learn' a model of the relationship between $\mathbf{x}$ and t. A successfully trained network will, for any given $\mathbf{x}$, give an output y that is close (in some sense) to the target value t. *Training* the network involves searching in the weight space of the network for a value of $\mathbf{w}$ that produces a function that fits the provided training data well.

Typically an *objective function* or *error function* is defined, as a function of $\mathbf{w}$, to measure how well the network with weights set to $\mathbf{w}$ solves the task. The objective function is a sum of terms, one for each input/target pair $\{\mathbf{x}, t\}$, measuring how close the output $y(\mathbf{x}; \mathbf{w})$ is to the target t. The training process is an exercise in *function minimization* – i.e., adjusting $\mathbf{w}$ in such a way as to find a $\mathbf{w}$ that minimizes the objective function. Many function-minimization algorithms make use not only of the objective function, but also its *gradient* with respect to the parameters $\mathbf{w}$. For general feedforward neural networks the *backpropagation* algorithm efficiently evaluates the gradient of the output y with respect to the parameters $\mathbf{w}$, and thence the gradient of the objective function with respect to $\mathbf{w}$.

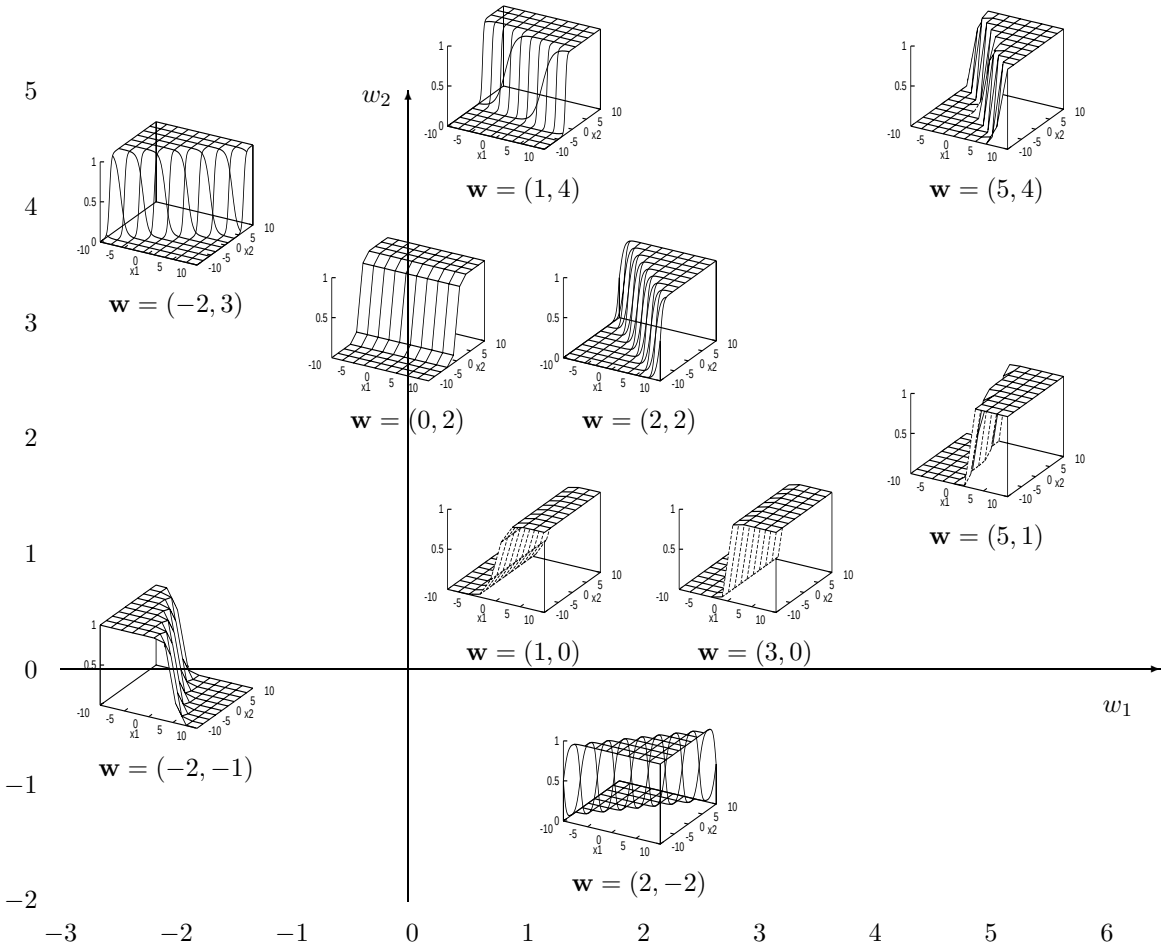


Figure 39.3. Weight space.

▶ 39.3 Training the single neuron as a binary classifier

We assume we have a data set of inputs $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$ with binary labels $\{t^{(n)}\}_{n=1}^{N}$, and a neuron whose output $y(\mathbf{x};\mathbf{w})$ is bounded between 0 and 1. We can then write down the following *error function*:

$$G(\mathbf{w}) = -\sum_n \left[t^{(n)} \ln y(\mathbf{x}^{(n)};\mathbf{w}) + (1-t^{(n)}) \ln(1-y(\mathbf{x}^{(n)};\mathbf{w})) \right]. \quad (39.11)$$

Each term in this objective function may be recognized as the *information content* of one outcome. It may also be described as the relative entropy between the empirical probability distribution $(t^{(n)}, 1-t^{(n)})$ and the probability distribution implied by the output of the neuron $(y, 1-y)$. The objective function is bounded below by zero and only attains this value if $y(\mathbf{x}^{(n)};\mathbf{w}) = t^{(n)}$ for all n.

We now differentiate this objective function with respect to $\mathbf{w}$.

Exercise 39.2.[2] The backpropagation algorithm. Show that the derivative $\mathbf{g} = \partial G/\partial \mathbf{w}$ is given by:

$$g_j = \frac{\partial G}{\partial w_j} = \sum_{n=1}^{N} -(t^{(n)} - y^{(n)}) x_j^{(n)}. \quad (39.12)$$

Notice that the quantity $e^{(n)} \equiv t^{(n)} - y^{(n)}$ is the *error* on example n – the difference between the target and the output. The simplest thing to do with a gradient of an error function is to *descend* it (even though this is often dimensionally incorrect, since a gradient has dimensions [1/parameter], whereas a change in a parameter has dimensions [parameter]). Since the derivative $\partial G/\partial \mathbf{w}$ is a sum of terms $\mathbf{g}^{(n)}$ defined by

$$g_j^{(n)} \equiv -(t^{(n)} - y^{(n)}) x_j^{(n)} \quad (39.13)$$

for $n = 1, \ldots, N$, we can obtain a simple on-line algorithm by putting each input through the network one at a time, and adjusting $\mathbf{w}$ a little in a direction opposite to $\mathbf{g}^{(n)}$.

We summarize the whole learning algorithm.

The on-line gradient-descent learning algorithm

Architecture. A single neuron has a number I of *inputs* x_i and one *output* y. Associated with each input is a weight w_i $(i = 1, \ldots, I)$.

Activity rule. 1. First, in response to the received inputs $\mathbf{x}$ (which may be arbitrary real numbers), we compute the *activation* of the neuron,

$$a = \sum_i w_i x_i, \quad (39.14)$$

where the sum is over $i = 0, \ldots, I$ if there is a bias and $i = 1, \ldots, I$ otherwise.

2. Second, the *output* y is set as a sigmoid function of the activation.

$$y(a) = \frac{1}{1+e^{-a}}. \quad (39.15)$$

This output might be viewed as stating the probability, according to the neuron, that the given input is in class 1 rather than class 0.

Learning rule. The *teacher* supplies a *target* value $t \in \{0, 1\}$ which says what the correct answer is for the given input. We compute the *error signal*

$$e = t - y \tag{39.16}$$

then adjust the weights $\mathbf{w}$ in a direction that would reduce the magnitude of this error:

$$\Delta w_i = \eta e x_i, \tag{39.17}$$

where η is the 'learning rate'. Commonly η is set by trial and error to a constant value or to a decreasing function of simulation time τ such as η_0/τ.

The activity rule and learning rule are repeated for each input/target pair $(\mathbf{x}, t)$ that is presented. If there is a fixed data set of size N, we can cycle through the data multiple times.

Batch learning versus on-line learning

Here we have described the *on-line* learning algorithm, in which a change in the weights is made after every example is presented. An alternative paradigm is to go through a *batch* of examples, computing the outputs and errors and accumulating the changes specified in equation (39.17) which are then made at the end of the batch.

Batch learning for the single neuron classifier

For each input/target pair $(\mathbf{x}^{(n)}, t^{(n)})$ $(n = 1, \ldots, N)$, compute $y^{(n)} = y(\mathbf{x}^{(n)}; \mathbf{w})$, where

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\sum_i w_i x_i)}, \tag{39.18}$$

define $e^{(n)} = t^{(n)} - y^{(n)}$, and compute for each weight w_i

$$g_i^{(n)} = -e^{(n)} x_i^{(n)}. \tag{39.19}$$

Then let

$$\Delta w_i = -\eta \sum_n g_i^{(n)}. \tag{39.20}$$

This batch learning algorithm is a *gradient descent algorithm*, whereas the on-line algorithm is a *stochastic gradient descent* algorithm. Source code implementing batch learning is given in algorithm 39.5. This algorithm is demonstrated in figure 39.4 for a neuron with two inputs with weights w_1 and w_2 and a bias w_0, performing the function

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}. \tag{39.21}$$

The bias w_0 is included, in contrast to figure 39.3, where it was omitted. The neuron is trained on a data set of ten labelled examples.

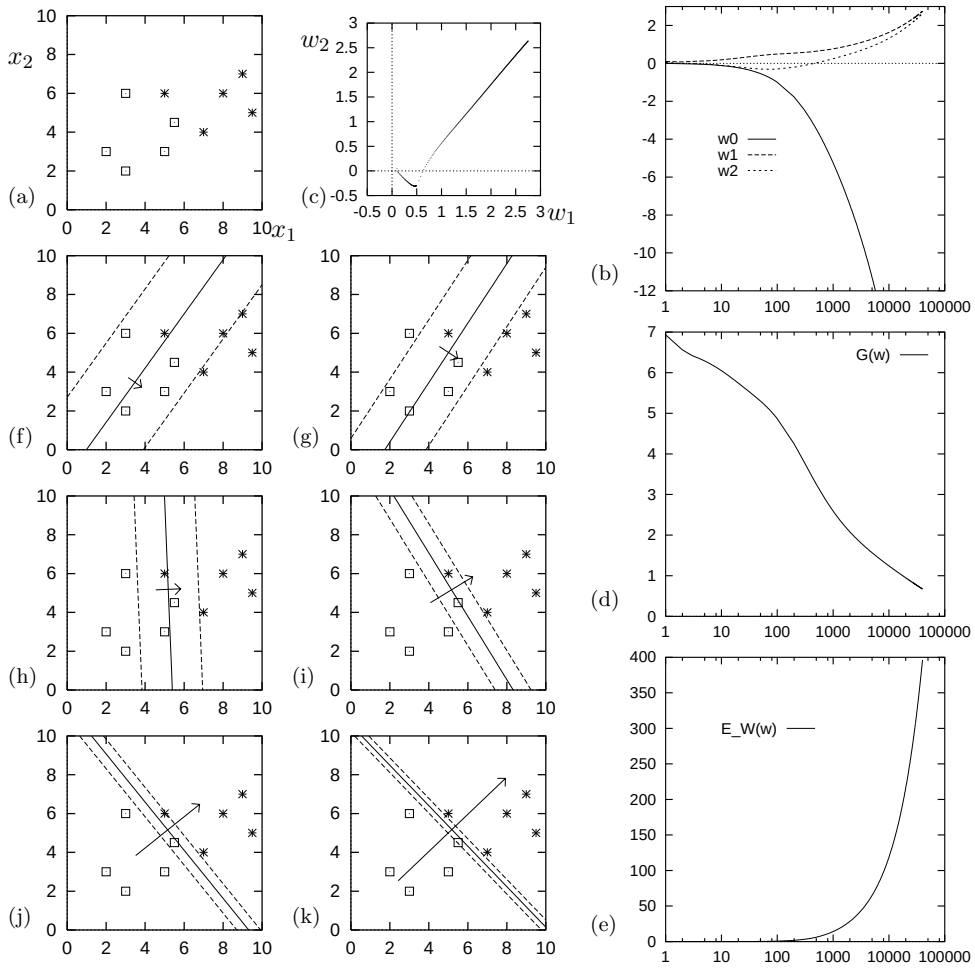


Figure 39.4. A single neuron learning to classify by gradient descent. The neuron has two weights w_1 and w_2 and a bias w_0. The learning rate was set to $\eta = 0.01$ and batch-mode gradient descent was performed using the code displayed in algorithm 39.5. (a) The training data. (b) Evolution of weights w_0, w_1 and w_2 as a function of number of iterations (on log scale). (c) Evolution of weights w_1 and w_2 in weight space. (d) The objective function $G(\mathbf{w})$ as a function of number of iterations. (e) The magnitude of the weights $E_W(\mathbf{w})$ as a function of time. (f–k) The function performed by the neuron (shown by three of its contours) after 30, 80, 500, 3000, 10 000 and 40 000 iterations. The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. Also shown is a vector proportional to (w_1, w_2). The larger the weights are, the bigger this vector becomes, and the closer together are the contours.

```
global x ;        # x is an N * I matrix containing all the input vectors
global t ;        # t is a vector of length N containing all the targets

for l = 1:L       # loop L times

   a = x * w  ;                          # compute all activations
   y = sigmoid(a) ;                      # compute outputs
   e = t - y   ;                         # compute errors
   g = - x' * e ;                        # compute the gradient vector
   w = w - eta * ( g + alpha * w )  ;    # make step, using learning rate eta
                                         #    and weight decay alpha
endfor

function f = sigmoid ( v )
   f = 1.0 ./ ( 1.0 .+ exp ( - v ) ) ;
endfunction
```

Algorithm 39.5. `Octave` source code for a gradient descent optimizer of a single neuron, batch learning, with optional weight decay (rate `alpha`). `Octave` notation: the instruction `a = x * w` causes the $(N \times I)$ *matrix* `x` consisting of all the input vectors to be multiplied by the weight vector `w`, giving the *vector* `a` listing the activations for all N input vectors; `x'` means `x`-transpose; the single command `y = sigmoid(a)` computes the sigmoid function of all elements of the vector `a`.

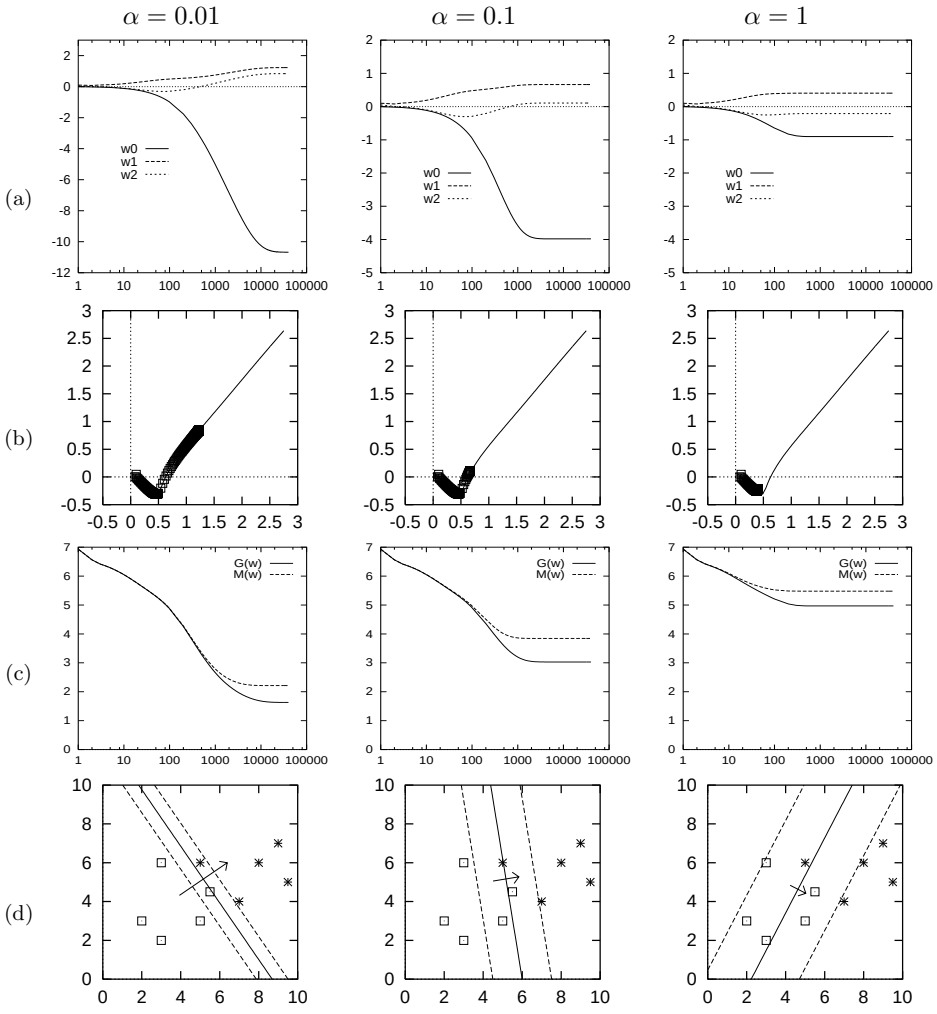


Figure 39.6. The influence of weight decay on a single neuron's learning. The objective function is $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$. The learning method was as in figure 39.4. (a) Evolution of weights w_0, w_1 and w_2. (b) Evolution of weights w_1 and w_2 in weight space shown by points, contrasted with the trajectory followed in the case of zero weight decay, shown by a thin line (from figure 39.4). Notice that for this problem weight decay has an effect very similar to 'early stopping'. (c) The objective function $M(\mathbf{w})$ and the error function $G(\mathbf{w})$ as a function of number of iterations. (d) The function performed by the neuron after 40 000 iterations.

▶ 39.4 Beyond descent on the error function: regularization

If the parameter η is set to an appropriate value, this algorithm works: the algorithm finds a setting of $\mathbf{w}$ that correctly classifies as many of the examples as possible.

If the examples are in fact *linearly separable* then the neuron finds this linear separation and its weights diverge to ever-larger values as the simulation continues. This can be seen happening in figure 39.4(f–k). This is an example of *overfitting*, where a model fits the data so well that its generalization performance is likely to be adversely affected.

This behaviour may be viewed as undesirable. How can it be rectified?

An ad hoc solution to overfitting is to use *early stopping*, that is, use an algorithm originally intended to minimize the error function $G(\mathbf{w})$, then prevent it from doing so by halting the algorithm at some point.

A more principled solution to overfitting makes use of *regularization*. Regularization involves modifying the objective function in such a way as to incorporate a bias against the sorts of solution $\mathbf{w}$ which we dislike. In the above example, what we dislike is the development of a very sharp decision boundary in figure 39.4k; this sharp boundary is associated with large weight values, so we use a regularizer that penalizes large weight values. We modify the objective function to:

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{39.22}$$

where the simplest choice of regularizer is the *weight decay* regularizer

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_i w_i^2. \tag{39.23}$$

The *regularization constant* α is called the weight decay rate. This additional term favours small values of $\mathbf{w}$ and decreases the tendency of a model to overfit fine details of the training data. The quantity α is known as a *hyperparameter*. Hyperparameters play a role in the learning algorithm but play no role in the activity rule of the network.

Exercise 39.3.[1] Compute the derivative of $M(\mathbf{w})$ with respect to w_i. Why is the above regularizer known as the 'weight decay' regularizer?

The gradient descent source code of algorithm 39.5 implements weight decay. This gradient descent algorithm is demonstrated in figure 39.6 using weight decay rates $\alpha = 0.01$, 0.1, and 1. As the weight decay rate is increased the solution becomes biased towards broader sigmoid functions with decision boundaries that are closer to the origin.

Note

Gradient descent with a step size η is in general *not* the most efficient way to minimize a function. A modification of gradient descent known as *momentum*, while improving convergence, is also not recommended. Most neural network experts use more advanced optimizers such as conjugate gradient algorithms. [Please do not confuse momentum, which is sometimes given the symbol α, with weight decay.]

▶ 39.5 Further exercises

More motivations for the linear neuron

▷ Exercise 39.4.[2] Consider the task of recognizing which of two Gaussian distributions a vector $\mathbf{z}$ comes from. Unlike the case studied in section 11.2, where the distributions had different means but a common variance–covariance matrix, we will assume that the two distributions have exactly the same mean but different variances. Let the probability of $\mathbf{z}$ given s ($s \in \{0,1\}$) be

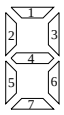
$$P(\mathbf{z} \mid s) = \prod_{i=1}^{I} \text{Normal}(z_i; 0, \sigma_{si}^2), \tag{39.24}$$

where σ_{si}^2 is the variance of z_i when the source symbol is s. Show that $P(s=1 \mid \mathbf{z})$ can be written in the form

$$P(s=1 \mid \mathbf{z}) = \frac{1}{1+\exp(-\mathbf{w}^{\mathsf{T}}\mathbf{x}+\theta)}, \tag{39.25}$$

where x_i is an appropriate function of z_i, $x_i = g(z_i)$.

Exercise 39.5.[2] **The noisy LED**.



$\mathbf{c}(2) =$  $\mathbf{c}(3) =$  $\mathbf{c}(8) =$ 

Consider an LED display with 7 elements numbered as shown above. The state of the display is a vector $\mathbf{x}$. When the controller wants the display to show character number s, e.g. $s=2$, each element x_j ($j = 1, \ldots, 7$) either adopts its intended state $c_j(s)$, with probability $1-f$, or is flipped, with probability f. Let's call the two states of x '+1' and '−1'.

(a) Assuming that the intended character s is actually a 2 or a 3, what is the probability of s, given the state $\mathbf{x}$? Show that $P(s=2 \mid \mathbf{x})$ can be written in the form

$$P(s=2 \mid \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^{\mathsf{T}}\mathbf{x}+\theta)}, \tag{39.26}$$

and compute the values of the weights $\mathbf{w}$ in the case $f = 0.1$.

(b) Assuming that s is one of $\{0, 1, 2, \ldots, 9\}$, with prior probabilities p_s, what is the probability of s, given the state $\mathbf{x}$? Put your answer in the form

$$P(s \mid \mathbf{x}) = \frac{e^{a_s}}{\sum_{s'} e^{a_{s'}}}, \tag{39.27}$$

where $\{a_s\}$ are functions of $\{c_j(s)\}$ and $\mathbf{x}$.

Could you make a better alphabet of 10 characters for a noisy LED, i.e., an alphabet less susceptible to confusion?

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

Table 39.7. An alternative 15-character alphabet for the 7-element LED display.

▷ Exercise 39.6.[2] A $(3,1)$ error-correcting code consists of the two codewords $\mathbf{x}^{(1)} = (1,0,0)$ and $\mathbf{x}^{(2)} = (0,0,1)$. A source bit $s \in \{1,2\}$ having probability distribution $\{p_1, p_2\}$ is used to select one of the two codewords for transmission over a binary symmetric channel with noise level f. The

received vector is $\mathbf{r}$. Show that the posterior probability of s given $\mathbf{r}$ can be written in the form

$$P(s=1 \,|\, \mathbf{r}) = \frac{1}{1 + \exp\left(-w_0 - \sum_{n=1}^{3} w_n r_n\right)},$$

and give expressions for the coefficients $\{w_n\}_{n=1}^{3}$ and the bias, w_0.

Describe, with a diagram, how this optimal decoder can be expressed in terms of a 'neuron'.

Problems to look at before Chapter 40

▷ Exercise 40.1.[2] What is $\sum_{K=0}^{N} \binom{N}{K}$?

[The symbol $\binom{N}{K}$ means the combination $\frac{N!}{K!(N-K)!}$.]

▷ Exercise 40.2.[2] If the top row of Pascal's triangle (which contains the single number '1') is denoted row zero, what is the sum of all the numbers in the triangle above row N?

▷ Exercise 40.3.[2] 3 points are selected at random on the surface of a sphere. What is the probability that all of them lie on a single hemisphere?

This chapter's material is originally due to Polya (1954) and Cover (1965) and the exposition that follows is Yaser Abu-Mostafa's.

40

Capacity of a Single Neuron

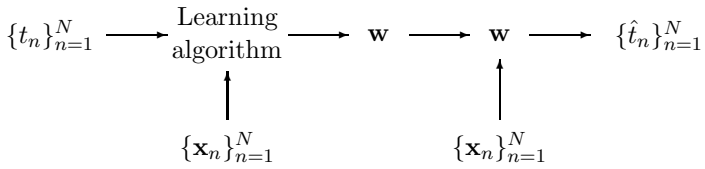


Figure 40.1. Neural network learning viewed as communication.

▶ 40.1 Neural network learning as communication

Many neural network models involve the adaptation of a set of weights $\mathbf{w}$ in response to a set of data points, for example a set of N target values $D_N = \{t_n\}_{n=1}^N$ at given locations $\{\mathbf{x}_n\}_{n=1}^N$. The adapted weights are then used to process subsequent input data. This process can be viewed as a communication process, in which the sender examines the data D_N and creates a message $\mathbf{w}$ that depends on those data. The receiver then uses $\mathbf{w}$; for example, the receiver might use the weights to try to reconstruct what the data D_N was. [In neural network parlance, this is using the neuron for 'memory' rather than for 'generalization'; 'generalizing' means extrapolating from the observed data to the value of t_{N+1} at some new location $\mathbf{x}_{N+1}$.] Just as a disk drive is a communication channel, the adapted network weights $\mathbf{w}$ therefore play the role of a communication channel, conveying information about the training data to a future user of that neural net. The question we now address is, 'what is the capacity of this channel?' – that is, 'how much information can be stored by training a neural network?'

If we had a learning algorithm that either produces a network whose response to all inputs is +1 or a network whose response to all inputs is 0, depending on the training data, then the weights allow us to distinguish between just two sorts of data set. The maximum information such a learning algorithm could convey about the data is therefore 1 bit, this information content being achieved if the two sorts of data set are equiprobable. How much more information can be conveyed if we make full use of a neural network's ability to represent other functions?

▶ 40.2 The capacity of a single neuron

We will look at the simplest case, that of a single binary threshold neuron. We will find that the capacity of such a neuron is *two bits per weight*. A neuron with K inputs can store $2K$ bits of information.

To obtain this interesting result we lay down some rules to exclude less interesting answers, such as: 'the capacity of a neuron is infinite, because each

of its weights is a real number and so can convey an infinite number of bits'. We exclude this answer by saying that the receiver is not able to examine the weights directly, nor is the receiver allowed to probe the weights by observing the output of the neuron for arbitrarily chosen inputs. We constrain the receiver to observe the output of the neuron at the same fixed set of N points $\{\mathbf{x}_n\}$ that were in the training set. What matters now is how many different distinguishable functions our neuron can produce, given that we can observe the function only at these N points. How many different binary labellings of N points can a linear threshold function produce? And how does this number compare with the maximum possible number of binary labellings, 2^N? If nearly all of the 2^N labellings can be realized by our neuron, then it is a communication channel that can convey all N bits (the target values $\{t_n\}$) with small probability of error. We will identify the capacity of the neuron as the maximum value that N can have such that the probability of error is very small. [We are departing a little from the definition of capacity in Chapter 9.]

We thus examine the following scenario. The sender is given a neuron with K inputs and a data set D_N which is a labelling of N points. The sender uses an adaptive algorithm to try to find a $\mathbf{w}$ that can reproduce this labelling exactly. We will assume the algorithm finds such a $\mathbf{w}$ if it exists. The receiver then evaluates the threshold function on the N input values. What is the probability that *all* N bits are correctly reproduced? How large can N become, for a given K, without this probability becoming substantially less than one?

General position

One technical detail needs to be pinned down: what set of inputs $\{\mathbf{x}_n\}$ are we considering? Our answer might depend on this choice. We will assume that the points are in *general position*.

Definition 40.1 *A set of points $\{\mathbf{x}_n\}$ in K-dimensional space are in* general position *if any subset of size $\leq K$ is linearly independent, and no $K+1$ of them lie in a $(K-1)$-dimensional plane.*

In $K = 3$ dimensions, for example, a set of points are in general position if no three points are colinear and no four points are coplanar. The intuitive idea is that points in general position are like random points in the space, in terms of the linear dependences between points. You don't expect three random points in three dimensions to lie on a straight line.

The linear threshold function

The neuron we will consider performs the function

$$y = f\left(\sum_{k=1}^{K} w_k x_k\right) \tag{40.1}$$

where

$$f(a) = \begin{cases} 1 & a > 0 \\ 0 & a \leq 0. \end{cases} \tag{40.2}$$

We will not have a bias w_0; the capacity for a neuron with a bias can be obtained by replacing K by $K+1$ in the final result below, i.e., considering one of the inputs to be fixed to 1. (These input points would not then be in general position; the derivation still works.)

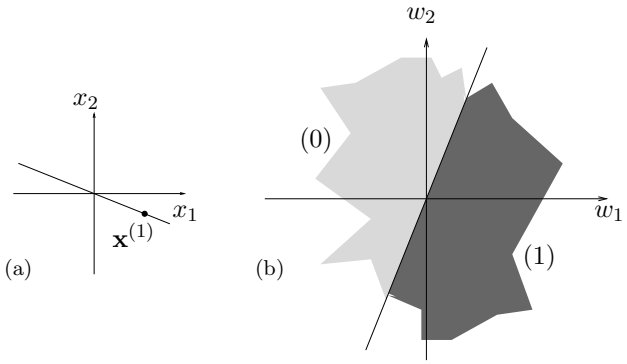


Figure 40.2. One data point in a two-dimensional input space, and the two regions of weight space that give the two alternative labellings of that point.

▶ 40.3 Counting threshold functions

Let us denote by $T(N, K)$ the number of distinct threshold functions on N points in general position in K dimensions. We will derive a formula for $T(N, K)$.

To start with, let us work out a few cases by hand.

In $K = 1$ dimension, for any N

The N points lie on a line. By changing the sign of the one weight w_1 we can label all points on the right side of the origin 1 and the others 0, or *vice versa*. Thus there are two distinct threshold functions. $T(N, 1) = 2$.

With $N = 1$ point, for any K

If there is just one point $\mathbf{x}^{(1)}$ then we can realize both possible labellings by setting $\mathbf{w} = \pm\mathbf{x}^{(1)}$. Thus $T(1, K) = 2$.

In $K = 2$ dimensions

In two dimensions with N points, we are free to spin the separating line around the origin. Each time the line passes over a point we obtain a new function. Once we have spun the line through 360 degrees we reproduce the function we started from. Because the points are in general position, the separating plane (line) crosses only one point at a time. In one revolution, every point is passed over twice. There are therefore $2N$ distinct threshold functions. $T(N, 2) = 2N$.

Comparing with the total number of binary functions, 2^N, we may note that for $N \geq 3$, not all binary functions can be realized by a linear threshold function. One famous example of an unrealizable function with $N = 4$ and $K = 2$ is the exclusive-or function on the points $\mathbf{x} = (\pm 1, \pm 1)$. [These points are not in general position, but you may confirm that the function remains unrealizable even if the points are perturbed into general position.]

In $K = 2$ dimensions, from the point of view of weight space

There is another way of visualizing this problem. Instead of visualizing a plane separating points in the two-dimensional input space, we can consider the two-dimensional *weight space*, colouring regions in weight space different colours if they label the given datapoints differently. We can then count the number of threshold functions by counting how many distinguishable regions there are in weight space. Consider first the set of weight vectors in weight

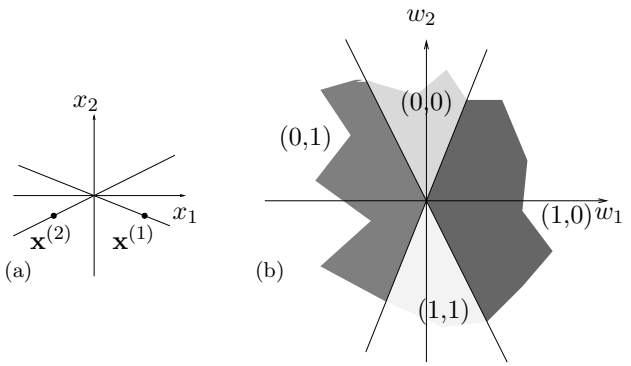


Figure 40.3. Two data points in a two-dimensional input space, and the four regions of weight space that give the four alternative labellings.

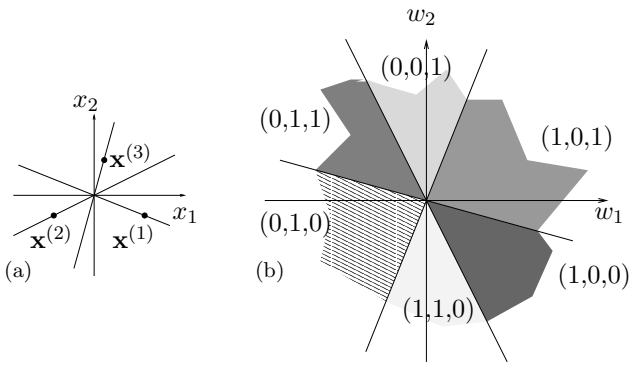


Figure 40.4. Three data points in a two-dimensional input space, and the six regions of weight space that give alternative labellings of those points. In this case, the labellings $(0,0,0)$ and $(1,1,1)$ cannot be realized. For any three points in general position there are always two labellings that cannot be realized.

space that classify a particular example $\mathbf{x}^{(n)}$ as a 1. For example, figure 40.2a shows a single point in our two-dimensional $\mathbf{x}$-space, and figure 40.2b shows the two corresponding sets of points in $\mathbf{w}$-space. One set of weight vectors occupy the half space

$$\mathbf{x}^{(n)} \cdot \mathbf{w} > 0, \tag{40.3}$$

and the others occupy $\mathbf{x}^{(n)} \cdot \mathbf{w} < 0$. In figure 40.3a we have added a second point in the input space. There are now 4 possible labellings: $(1,1)$, $(1,0)$, $(0,1)$, and $(0,0)$. Figure 40.3b shows the two hyperplanes $\mathbf{x}^{(1)} \cdot \mathbf{w} = 0$ and $\mathbf{x}^{(2)} \cdot \mathbf{w} = 0$ which separate the sets of weight vectors that produce each of these labellings. When $N = 3$ (figure 40.4), weight space is divided by three hyperplanes into six regions. Not all of the eight conceivable labellings can be realized. Thus $T(3,2) = 6$.

In $K = 3$ dimensions

We now use this weight space visualization to study the three dimensional case.

Let us imagine adding one point at a time and count the number of threshold functions as we do so. When $N = 2$, weight space is divided by two hyperplanes $\mathbf{x}^{(1)} \cdot \mathbf{w} = 0$ and $\mathbf{x}^{(2)} \cdot \mathbf{w} = 0$ into four regions; in any one region all vectors $\mathbf{w}$ produce the same function on the 2 input vectors. Thus $T(2,3) = 4$.

Adding a third point in general position produces a third plane in $\mathbf{w}$ space, so that there are 8 distinguishable regions. $T(3,3) = 8$. The three bisecting planes are shown in figure 40.5a.

At this point matters become slightly more tricky. As figure 40.5b illustrates, the fourth plane in the three-dimensional $\mathbf{w}$ space cannot transect all eight of the sets created by the first three planes. Six of the existing regions are cut in two and the remaining two are unaffected. So $T(4,3) = 14$. Two

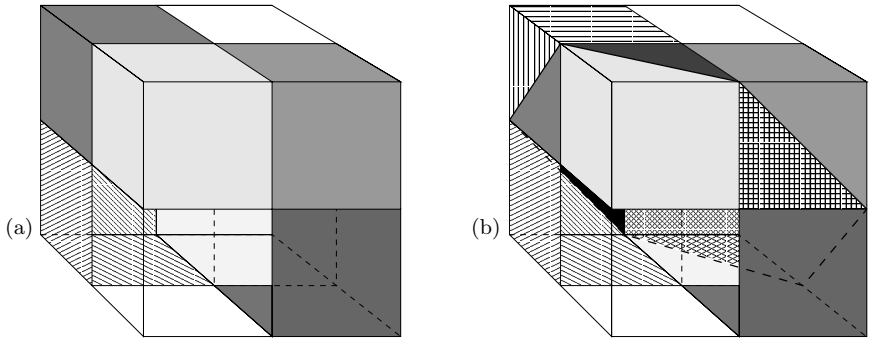


Figure 40.5. Weight space illustrations for $T(3,3)$ and $T(4,3)$. (a) $T(3,3) = 8$. Three hyperplanes (corresponding to three points in general position) divide 3-space into 8 regions, shown here by colouring the relevant part of the surface of a hollow, semi-transparent cube centred on the origin. (b) $T(4,3) = 14$. Four hyperplanes divide 3-space into 14 regions, of which this figure shows 13 (the 14th region is out of view on the right-hand face. Compare with figure 40.5a: all of the regions that are not coloured white have been cut into two.

	K							
N	1	2	3	4	5	6	7	8
1	2	2	2	2	2	2	2	2
2	2	4	4					
3	2	6	8					
4	2	8	14					
5	2	10						
6	2	12						

Table 40.6. Values of $T(N,K)$ deduced by hand.

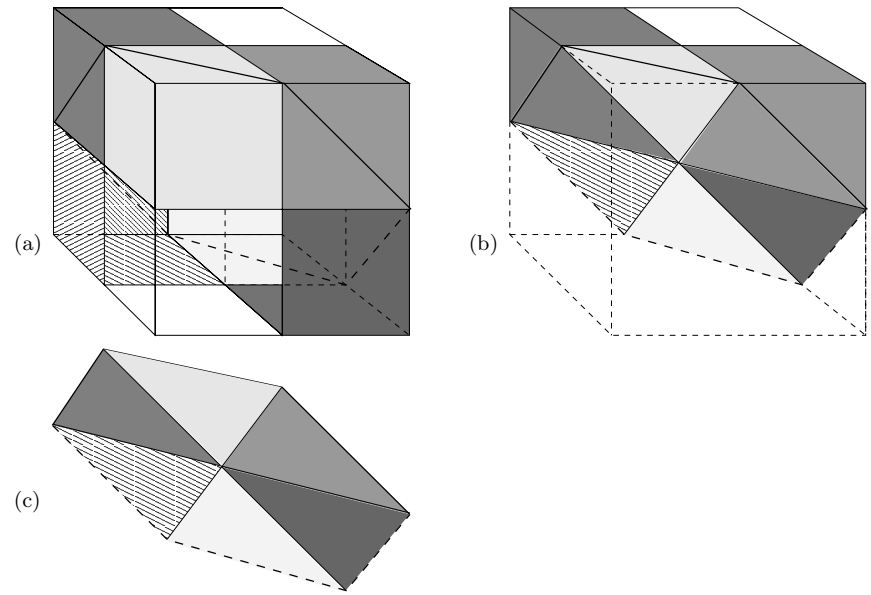


Figure 40.7. Illustration of the cutting process going from $T(3,3)$ to $T(4,3)$. (a) The eight regions of figure 40.5a with one added hyperplane. All of the regions that are not coloured white have been cut into two. (b) Here, the hollow cube has been made solid, so we can see which regions are cut by the fourth plane. The front half of the cube has been cut away. (c) This figure shows the new two dimensional hyperplane, which is divided into six regions by the three one-dimensional hyperplanes (lines) which cross it. Each of these regions corresponds to one of the three-dimensional regions in figure 40.7a which is cut into two by this new hyperplane. This shows that $T(4,3) - T(3,3) = 6$. Figure 40.7c should be compared with figure 40.4b.

of the binary functions on 4 points in 3 dimensions cannot be realized by a linear threshold function.

We have now filled in the values of $T(N, K)$ shown in table 40.6. Can we obtain any insights into our derivation of $T(4, 3)$ in order to fill in the rest of the table for $T(N, K)$? Why was $T(4, 3)$ greater than $T(3, 3)$ by six?

Six is the number of regions that the new hyperplane bisected in **w**-space (figure 40.7a b). Equivalently, if we look in the $K-1$ dimensional subspace that is the Nth hyperplane, that subspace is divided into six regions by the $N-1$ previous hyperplanes (figure 40.7c). Now this is a concept we have met before. Compare figure 40.7c with figure 40.4b. How many regions are created by $N-1$ hyperplanes in a $K-1$ dimensional space? Why, $T(N-1, K-1)$, of course! In the present case $N = 4$, $K = 3$, we can look up $T(3, 2) = 6$ in the previous section. So

$$T(4, 3) = T(3, 3) + T(3, 2). \tag{40.4}$$

Recurrence relation for any N, K

Generalizing this picture, we see that when we add an Nth hyperplane in K dimensions, it will bisect $T(N-1, K-1)$ of the $T(N-1, K)$ regions that were created by the previous $N-1$ hyperplanes. Therefore, the total number of regions obtained after adding the Nth hyperplane is $2T(N-1, K-1)$ (since $T(N-1, K-1)$ out of $T(N-1, K)$ regions are split in two) plus the remaining $T(N-1, K) - T(N-1, K-1)$ regions not split by the Nth hyperplane, which gives the following equation for $T(N, K)$:

$$T(N, K) = T(N-1, K) + T(N-1, K-1). \tag{40.5}$$

Now all that remains is to solve this recurrence relation given the boundary conditions $T(N, 1) = 2$ and $T(1, K) = 2$.

Does the recurrence relation (40.5) look familiar? Maybe you remember building Pascal's triangle by adding together two adjacent numbers in one row to get the number below. The N, K element of Pascal's triangle is equal to

$$C(N, K) \equiv \binom{N}{K} \equiv \frac{N!}{(N-K)!K!}. \tag{40.6}$$

Table 40.8. Pascal's triangle.

	K							
N	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		

Combinations $\binom{N}{K}$ satisfy the equation

$$C(N, K) = C(N-1, K-1) + C(N-1, K), \quad \text{for all } N > 0. \tag{40.7}$$

[Here we are adopting the convention that $\binom{N}{K} \equiv 0$ if $K > N$ or $K < 0$.] So $\binom{N}{K}$ satisfies the required recurrence relation (40.5). This doesn't mean $T(N, K) = \binom{N}{K}$, since many functions can satisfy one recurrence relation.

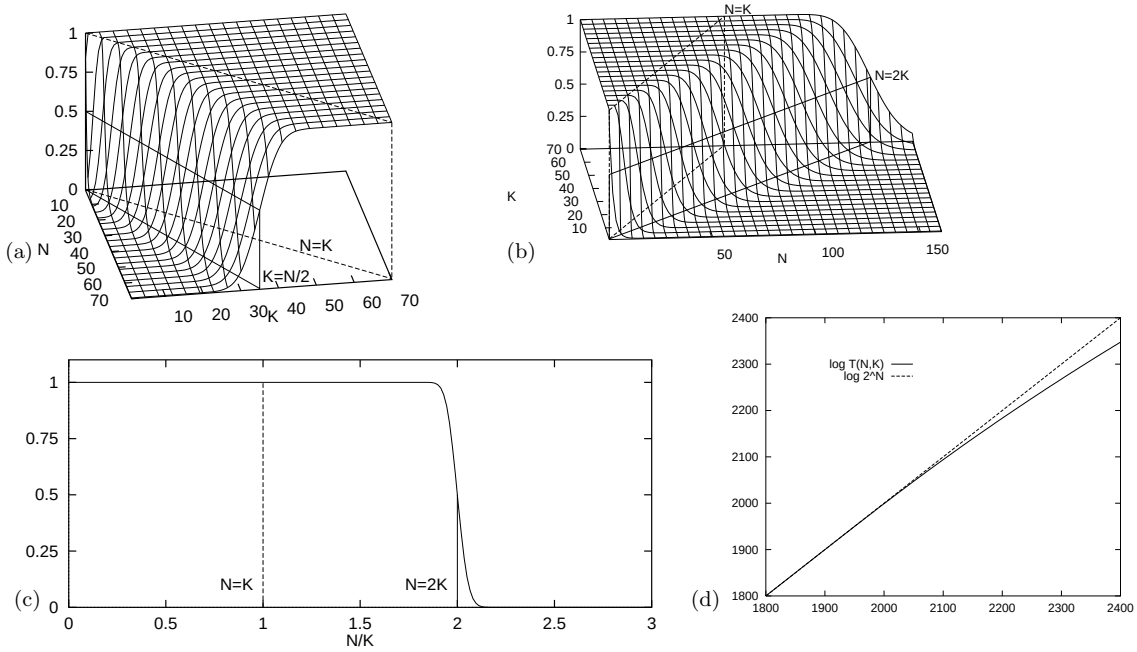


Figure 40.9. The fraction of functions on N points in K dimensions that are linear threshold functions, $T(N,K)/2^N$, shown from various viewpoints. In (a) we see the dependence on K, which is approximately an error function passing through 0.5 at $K = N/2$; the fraction reaches 1 at $K = N$. In (b) we see the dependence on N, which is 1 up to $N = K$ and drops sharply at $N = 2K$. Panel (c) shows the dependence on N/K for $K = 1000$. There is a sudden drop in the fraction of realizable labellings when $N = 2K$. Panel (d) shows the values of $\log_2 T(N,K)$ and $\log_2 2^N$ as a function of N for $K = 1000$. These figures were plotted using the approximation of $T/2^N$ by the error function.

But perhaps we can express $T(N,K)$ as a linear superposition of combination functions of the form $C_{\alpha,\beta}(N,K) \equiv \binom{N+\alpha}{K+\beta}$. By comparing tables 40.8 and 40.6 we can see how to satisfy the boundary conditions: we simply need to translate Pascal's triangle to the right by 1, 2, 3, ...; superpose; add; multiply by two, and drop the whole table by one line. Thus:

$$T(N,K) = 2\sum_{k=0}^{K-1} \binom{N-1}{k}. \tag{40.8}$$

Using the fact that the Nth row of Pascal's triangle sums to 2^N, that is, $\sum_{k=0}^{N-1} \binom{N-1}{k} = 2^{N-1}$, we can simplify the cases where $K-1 \geq N-1$.

$$T(N,K) = \begin{cases} 2^N & K \geq N \\ 2\sum_{k=0}^{K-1} \binom{N-1}{k} & K < N. \end{cases} \tag{40.9}$$

Interpretation

It is natural to compare $T(N,K)$ with the total number of binary functions on N points, 2^N. The ratio $T(N,K)/2^N$ tells us the probability that an arbitrary labelling $\{t_n\}_{n=1}^N$ can be memorized by our neuron. The two functions are equal for all $N \leq K$. The line $N = K$ is thus a special line, defining the maximum number of points on which *any* arbitrary labelling can be realized. This number of points is referred to as the *Vapnik–Chervonenkis dimension* (VC dimension) of the class of functions. The VC dimension of a binary threshold function on K dimensions is thus K.

What is interesting is (for large K) the number of points N such that *almost* any labelling can be realized. The ratio $T(N,K)/2^N$ is, for $N < 2K$, still greater than $1/2$, and for large K the ratio is very close to 1.

For our purposes the sum in equation (40.9) is well approximated by the error function,

$$\sum_{0}^{K} \binom{N}{k} \simeq 2^N \, \Phi\left(\frac{K-(N/2)}{\sqrt{N}/2}\right), \tag{40.10}$$

where $\Phi(z) \equiv \int_{-\infty}^{z} \exp(-z^2/2)/\sqrt{2\pi}$. Figure 40.9 shows the realizable fraction $T(N,K)/2^N$ as a function of N and K. The take-home message is shown in figure 40.9c: although the fraction $T(N,K)/2^N$ is less than 1 for $N > K$, it is only negligibly less than 1 up to $N = 2K$; there, there is a catastrophic drop to zero, so that for $N > 2K$, only a tiny fraction of the binary labellings can be realized by the threshold function.

Conclusion

The capacity of a linear threshold neuron, for large K, is 2 bits per weight.

A single neuron can almost certainly memorize up to $N = 2K$ random binary labels perfectly, but will almost certainly fail to memorize more.

40.4 Further exercises

▷ Exercise 40.4.[2] Can a finite set of $2N$ distinct points in a two-dimensional space be split in half by a straight line

- if the points are in general position?
- if the points are not in general position?

Can $2N$ points in a K dimensional space be split in half by a $K-1$ dimensional hyperplane?

Exercise 40.5.[2, p.491] Four points are selected at random on the surface of a sphere. What is the probability that all of them lie on a single hemisphere? How does this question relate to $T(N,K)$?

Exercise 40.6.[2] Consider the binary threshold neuron in $K = 3$ dimensions, and the set of points $\{\mathbf{x}\} = \{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}$. Find a parameter vector $\mathbf{w}$ such that the neuron memorizes the labels: (a) $\{t\} = \{1,1,1,1\}$; (b) $\{t\} = \{1,1,0,0\}$.

Find an unrealizable labelling $\{t\}$.

▷ Exercise 40.7.[3] In this chapter we constrained all our hyperplanes to go through the origin. In this exercise, we remove this constraint.

How many regions in a plane are created by N lines in general position?

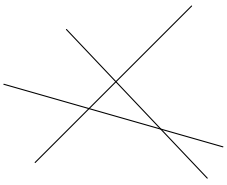


Figure 40.10. Three lines in a plane create seven regions.

Exercise 40.8.[2] Estimate in bits the total sensory experience that you have had in your life – visual information, auditory information, etc. Estimate how much information you have memorized. Estimate the information content of the works of Shakespeare. Compare these with the capacity of your brain assuming you have 10^{11} neurons each making 1000 synaptic connections, and that the capacity result for one neuron (two bits per connection) applies. Is your brain full yet?

▷ Exercise 40.9.[3] What is the capacity of the axon of a spiking neuron, viewed as a communication channel, in bits per second? [See MacKay and McCulloch (1952) for an early publication on this topic.] Multiply by the number of axons in the optic nerve (about 10^6) or cochlear nerve (about 50 000 per ear) to estimate again the rate of acquisition sensory experience.

▶ 40.5 Solutions

Solution to exercise 40.5 (p.490). The probability that all four points lie on a single hemisphere is

$$T(4,3)/2^4 = 14/16 = 7/8. \tag{40.11}$$

41

Learning as Inference

▶ 41.1 Neural network learning as inference

In Chapter 39 we trained a simple neural network as a classifier by minimizing an objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{41.1}$$

made up of an error function

$$G(\mathbf{w}) = -\sum_n \left[t^{(n)} \ln y(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \ln(1 - y(\mathbf{x}^{(n)}; \mathbf{w})) \right] \tag{41.2}$$

and a regularizer

$$E_W(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2. \tag{41.3}$$

This neural network learning process can be given the following probabilistic interpretation.

We interpret the output $y(\mathbf{x}; \mathbf{w})$ of the neuron literally as defining (when its parameters $\mathbf{w}$ are specified) the probability that an input $\mathbf{x}$ belongs to class $t = 1$, rather than the alternative $t = 0$. Thus $y(\mathbf{x}; \mathbf{w}) \equiv P(t = 1 \,|\, \mathbf{x}, \mathbf{w})$. Then each value of $\mathbf{w}$ defines a different hypothesis about the probability of class 1 relative to class 0 as a function of $\mathbf{x}$.

We define the observed data D to be the *targets* $\{t\}$ – the inputs $\{\mathbf{x}\}$ are assumed to be given, and not to be modelled. To infer $\mathbf{w}$ given the data, we require a likelihood function and a prior probability over $\mathbf{w}$. The likelihood function measures how well the parameters $\mathbf{w}$ predict the observed data; it is the probability assigned to the observed t values by the model with parameters set to $\mathbf{w}$. Now the two equations

$$\begin{array}{rcl} P(t = 1 \,|\, \mathbf{w}, \mathbf{x}) & = & y \\ P(t = 0 \,|\, \mathbf{w}, \mathbf{x}) & = & 1 - y \end{array} \tag{41.4}$$

can be rewritten as the single equation

$$P(t \,|\, \mathbf{w}, \mathbf{x}) = y^t (1 - y)^{1-t} = \exp[t \ln y + (1 - t) \ln(1 - y)]\,. \tag{41.5}$$

So the error function G can be interpreted as minus the log likelihood:

$$P(D \,|\, \mathbf{w}) = \exp[-G(\mathbf{w})]. \tag{41.6}$$

Similarly the regularizer can be interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w} \,|\, \alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{41.7}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$, and $1/Z_W(\alpha)$ is equal to $(\alpha/2\pi)^{K/2}$, where K is the number of parameters in the vector $\mathbf{w}$.

The objective function $M(\mathbf{w})$ then corresponds to the *inference* of the parameters $\mathbf{w}$, given the data:

$$P(\mathbf{w} \mid D, \alpha) = \frac{P(D \mid \mathbf{w})P(\mathbf{w} \mid \alpha)}{P(D \mid \alpha)} \tag{41.8}$$

$$= \frac{e^{-G(\mathbf{w})}\, e^{-\alpha E_W(\mathbf{w})}/Z_W(\alpha)}{P(D \mid \alpha)} \tag{41.9}$$

$$= \frac{1}{Z_M} \exp(-M(\mathbf{w})). \tag{41.10}$$

So the $\mathbf{w}$ *found by (locally) minimizing* $M(\mathbf{w})$ *can be interpreted as the (locally) most probable parameter vector,* $\mathbf{w}^*$. From now on we will refer to $\mathbf{w}^*$ as $\mathbf{w}_{\rm MP}$.

Why is it natural to interpret the error functions as *log* probabilities? Error functions are usually additive. For example, G is a *sum* of information contents, and E_W is a *sum* of squared weights. Probabilities, on the other hand, are multiplicative: for independent events X and Y, the joint probability is $P(x, y) = P(x)P(y)$. The logarithmic mapping maintains this correspondence.

The interpretation of $M(\mathbf{w})$ as a log probability has numerous benefits, some of which we will discuss in a moment.

▶ 41.2 Illustration for a neuron with two weights

In the case of a neuron with just two inputs and no bias,

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}, \tag{41.11}$$

we can plot the posterior probability of $\mathbf{w}$, $P(\mathbf{w} \mid D, \alpha) \propto \exp(-M(\mathbf{w}))$. Imagine that we receive some data as shown in the left column of figure 41.1. Each data point consists of a two-dimensional input vector $\mathbf{x}$ and a t value indicated by $\times$ $(t = 1)$ or $\Box$ $(t = 0)$. The likelihood function $\exp(-G(\mathbf{w}))$ is shown as a function of $\mathbf{w}$ in the second column. It is a product of functions of the form (41.11).

The product of traditional learning is a point in $\mathbf{w}$-space, the estimator $\mathbf{w}^*$, which maximizes the posterior probability density. In contrast, in the Bayesian view, the product of learning is an *ensemble* of plausible parameter values (bottom right of figure 41.1). We do not choose one particular hypothesis $\mathbf{w}$; rather we evaluate their posterior probabilities. The posterior distribution is obtained by multiplying the likelihood by a prior distribution over $\mathbf{w}$ space (shown as a broad Gaussian at the upper right of figure 41.1). The posterior ensemble (within a multiplicative constant) is shown in the third column of figure 41.1, and as a contour plot in the fourth column. As the amount of data increases (from top to bottom), the posterior ensemble becomes increasingly concentrated around the most probable value $\mathbf{w}^*$.

▶ 41.3 Beyond optimization: making predictions

Let us consider the task of making predictions with the neuron which we trained as a classifier in section 39.3. This was a neuron with two inputs and a bias.

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}. \tag{41.12}$$

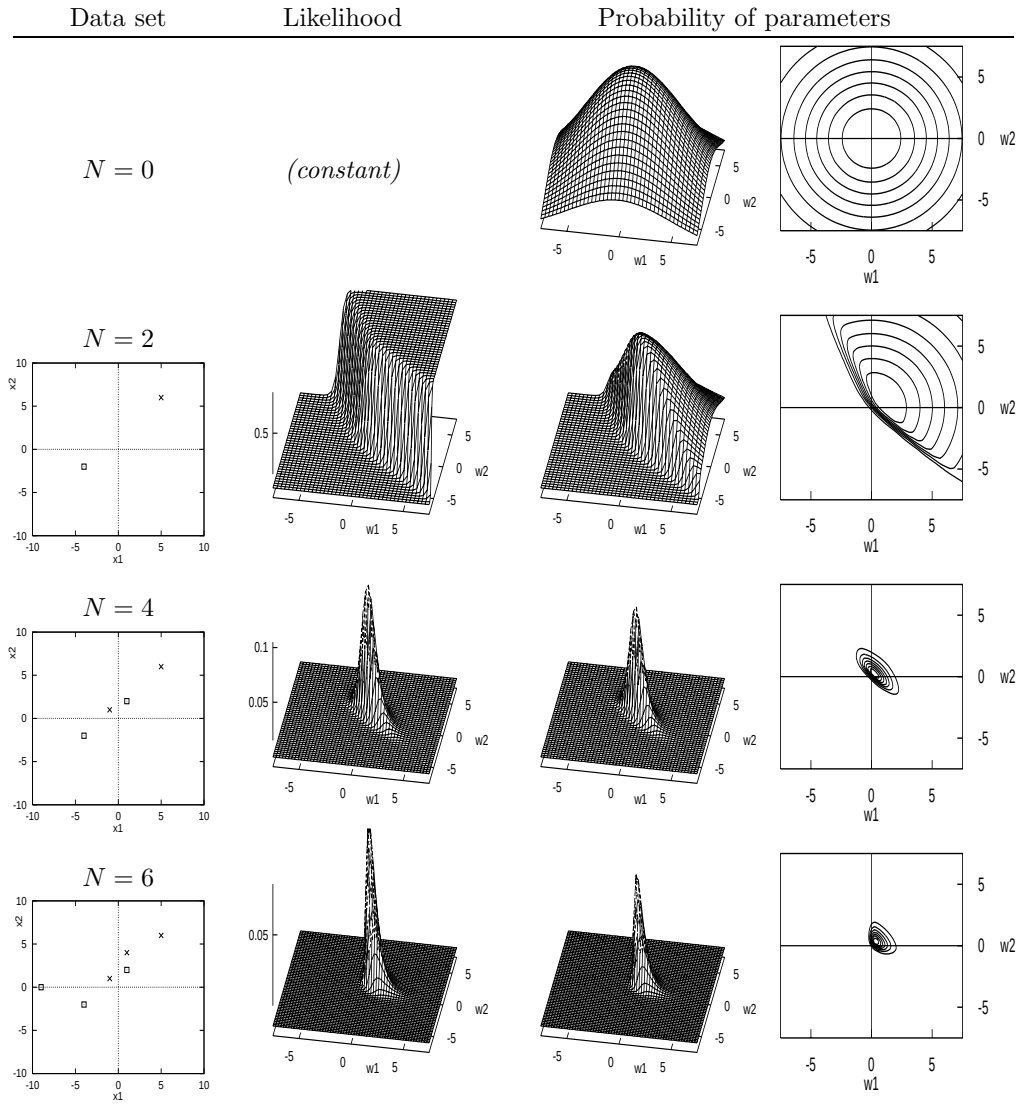


Figure 41.1. The Bayesian interpretation and generalization of traditional neural network learning. Evolution of the probability distribution over parameters as data arrive.

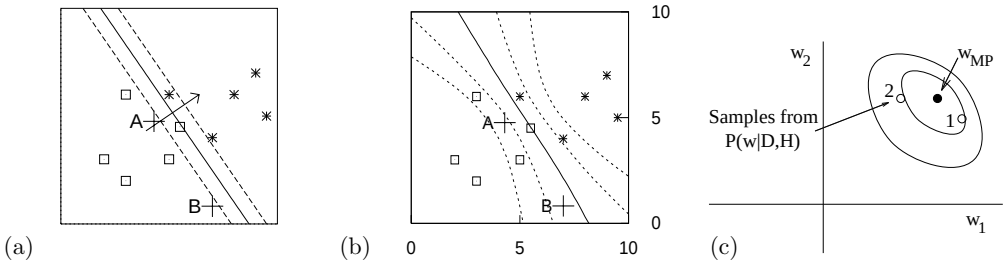


Figure 41.2. Making predictions. (a) The function performed by an optimized neuron $\mathbf{w}_{\rm MP}$ (shown by three of its contours) trained with weight decay, $\alpha = 0.01$ (from figure 39.6). The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. (b) Are these predictions more reasonable? (Contours shown are for $y = 0.5, 0.27$, 0.73, 0.12 and 0.88.) (c) The posterior probability of $\mathbf{w}$ (schematic); the Bayesian predictions shown in (b) were obtained by averaging together the predictions made by each possible value of the weights $\mathbf{w}$, with each value of $\mathbf{w}$ receiving a vote proportional to its probability under the posterior ensemble. The method used to create (b) is described in section 41.4.

When we last played with it, we trained it by minimizing the objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}). \tag{41.13}$$

The resulting optimized function for the case $\alpha = 0.01$ is reproduced in figure 41.2a.

We now consider the task of predicting the class $t^{(N+1)}$ corresponding to a new input $\mathbf{x}^{(N+1)}$. It is common practice, when making predictions, simply to use a neural network with its weights fixed to their optimized value $\mathbf{w}_{\rm MP}$, but this is not optimal, as can be seen intuitively by considering the predictions shown in figure 41.2a. Are these reasonable predictions? Consider new data arriving at points A and B. The best-fit model assigns both of these examples probability 0.2 of being in class 1, because they have the same value of $\mathbf{w}_{\rm MP} \cdot \mathbf{x}$. If we really knew that $\mathbf{w}$ was equal to $\mathbf{w}_{\rm MP}$, then these predictions would be correct. But we do not know $\mathbf{w}$. The parameters are *uncertain*. Intuitively we might be inclined to assign a less confident probability (closer to 0.5) at B than at A, as shown in figure 41.2b, since point B is far from the training data. *The best-fit parameters* $\mathbf{w}_{\rm MP}$ *often give over-confident predictions.* A non-Bayesian approach to this problem is to downweight all predictions uniformly, by an empirically determined factor (Copas, 1983). This is not ideal, since intuition suggests the strength of the predictions at B should be downweighted more than those at A. A Bayesian viewpoint helps us to understand the cause of the problem, and provides a straightforward solution. In a nutshell, we obtain Bayesian predictions by taking into account the whole posterior ensemble, shown schematically in figure 41.2c.

The Bayesian prediction of a new datum $\mathbf{t}^{(N+1)}$ involves *marginalizing* over the parameters (and over anything else about which we are uncertain). For simplicity, let us assume that the weights $\mathbf{w}$ are the only uncertain quantities – the weight decay rate α and the model $\mathcal{H}$ itself are assumed to be fixed. Then by the sum rule, the predictive probability of a new target $\mathbf{t}^{(N+1)}$ at a location $\mathbf{x}^{(N+1)}$ is:

$$P(\mathbf{t}^{(N+1)} \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K \mathbf{w}\, P(\mathbf{t}^{(N+1)} \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) P(\mathbf{w} \,|\, D, \alpha), \tag{41.14}$$

where K is the dimensionality of $\mathbf{w}$, three in the toy problem. Thus the predictions are obtained by weighting the prediction for each possible $\mathbf{w}$,

$$\begin{array}{rcl} P(\mathbf{t}^{(N+1)} = 1 \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & y(\mathbf{x}^{(N+1)}; \mathbf{w}) \\ P(\mathbf{t}^{(N+1)} = 0 \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & 1 - y(\mathbf{x}^{(N+1)}; \mathbf{w}), \end{array} \tag{41.15}$$

with a weight given by the posterior probability of $\mathbf{w}$, $P(\mathbf{w} \,|\, D, \alpha)$, which we most recently wrote down in equation (41.10). This posterior probability is

$$P(\mathbf{w} \,|\, D, \alpha) = \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{41.16}$$

where

$$Z_M = \int \mathrm{d}^K \mathbf{w} \exp(-M(\mathbf{w})). \tag{41.17}$$

In summary, we can get the Bayesian predictions if we can find a way of computing the integral

$$P(\mathbf{t}^{(N+1)} = 1 \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K \mathbf{w}\ \ y(\mathbf{x}^{(N+1)}; \mathbf{w}) \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{41.18}$$

which is the average of the output of the neuron at $\mathbf{x}^{(N+1)}$ under the posterior distribution of $\mathbf{w}$.

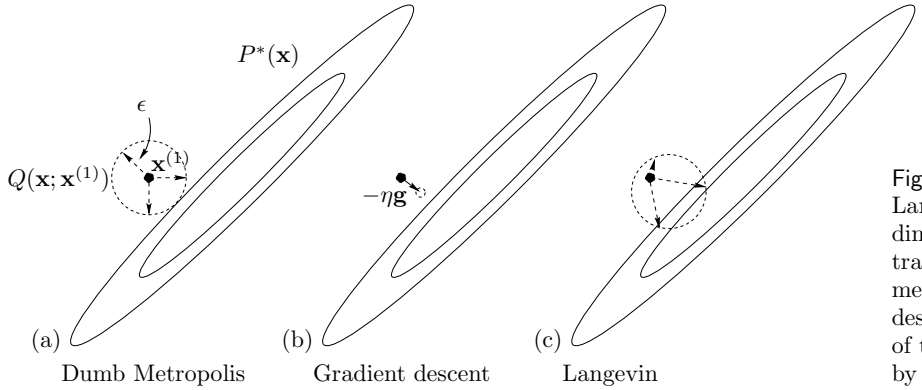


Figure 41.3. One step of the Langevin method in two dimensions (c), contrasted with a traditional 'dumb' Metropolis method (a) and with gradient descent (b). The proposal density of the Langevin method is given by 'gradient descent with noise'.

Implementation

How shall we compute the integral (41.18)? For our toy problem, the weight space is three dimensional; for a realistic neural network the dimensionality K might be in the thousands.

Bayesian inference for general data modelling problems may be implemented by exact methods (Chapter 25), by Monte Carlo sampling (Chapter 29), or by deterministic approximate methods, for example, methods that make Gaussian approximations to $P(\mathbf{w} \mid D, \alpha)$ using Laplace's method (Chapter 27) or variational methods (Chapter 33). For neural networks there are few exact methods. The two main approaches to implementing Bayesian inference for neural networks are the Monte Carlo methods developed by Neal (1996) and the Gaussian approximation methods developed by MacKay (1991).

▶ 41.4 Monte Carlo implementation of a single neuron

First we will use a Monte Carlo approach in which the task of evaluating the integral (41.18) is solved by treating $y(\mathbf{x}^{(N+1)}; \mathbf{w})$ as a function f of $\mathbf{w}$ whose mean we compute using

$$\langle f(\mathbf{w}) \rangle \simeq \frac{1}{R} \sum_r f(\mathbf{w}^{(r)}) \tag{41.19}$$

where $\{\mathbf{w}^{(r)}\}$ are samples from the posterior distribution $\frac{1}{Z_M}\exp(-M(\mathbf{w}))$ (cf. equation (29.6)). We obtain the samples using a Metropolis method (section 29.4). As an aside, a possible disadvantage of this Monte Carlo approach is that it is a poor way of estimating the probability of an improbable event, i.e., a $P(t \mid D, \mathcal{H})$ that is very close to zero, if the improbable event is most likely to occur in conjunction with improbable parameter values.

How to generate the samples $\{\mathbf{w}^{(r)}\}$? Radford Neal introduced the *Hamiltonian Monte Carlo* method to neural networks. We met this sophisticated Metropolis method, which makes use of gradient information, in Chapter 30. The method we now demonstrate is a simple version of Hamiltonian Monte Carlo called the *Langevin Monte Carlo method*.

The Langevin Monte Carlo method

The Langevin method (algorithm 41.4) may be summarized as 'gradient descent with added noise', as shown pictorially in figure 41.3. A noise vector $\mathbf{p}$ is generated from a Gaussian with unit variance. The gradient $\mathbf{g}$ is computed,

```
g = gradM ( w ) ;                          # set gradient using initial w
M = findM ( w ) ;                          # set objective function too

for l = 1:L                                # loop L times
   p = randn ( size(w) ) ;                 # initial momentum is Normal(0,1)
   H = p' * p / 2 + M ;                    # evaluate H(w,p)

*  p = p - epsilon * g / 2 ;               # make half-step in p
*  wnew = w + epsilon * p ;                # make step in w
*  gnew = gradM ( wnew ) ;                 # find new gradient
*  p = p - epsilon * gnew / 2 ;            # make half-step in p

   Mnew = findM ( wnew ) ;                 # find new objective function
   Hnew = p' * p / 2 + Mnew ;              # evaluate new value of H
   dH = Hnew - H ;                         # decide whether to accept
   if ( dH < 0 )                 accept = 1 ;
   elseif ( rand() < exp(-dH) ) accept = 1 ;  # compare with a uniform
   else                          accept = 0 ;  #                variate
   endif
   if ( accept )      g = gnew ;   w = wnew ;    M = Mnew ;    endif
endfor

function gM = gradM ( w )                  # gradient of objective function
   a = x * w  ;                            # compute activations
   y = sigmoid(a) ;                        # compute outputs
   e = t - y   ;                           # compute errors
   g = - x' * e ;                          # compute the gradient of G(w)
   gM = alpha * w + g ;
endfunction

function M = findM ( w )                   # objective function
   G = - (t' * log(y) + (1-t') * log( 1-y )) ;
   EW = w' * w / 2 ;
   M = G + alpha * EW ;
endfunction
```

Algorithm 41.4. `Octave` source code for the Langevin Monte Carlo method. To obtain the Hamiltonian Monte Carlo method, we repeat the four lines marked * multiple times (algorithm 41.8).

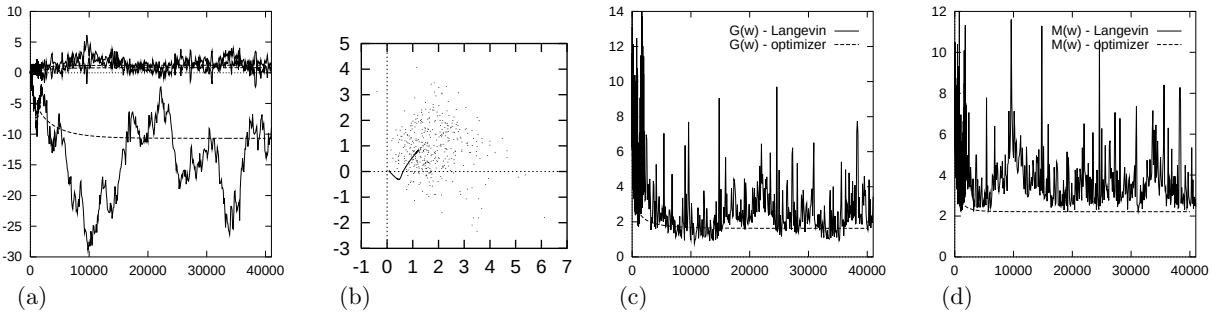


Figure 41.5. A single neuron learning under the Langevin Monte Carlo method. (a) Evolution of weights w_0, w_1 and w_2 as a function of number of iterations. (b) Evolution of weights w_1 and w_2 in weight space. Also shown by a line is the evolution of the weights using the optimizer of figure 39.6. (c) The error function $G(\mathbf{w})$ as a function of number of iterations. Also shown is the error function during the optimization of figure 39.6. (d) The objective function $M(\mathbf{x})$ as a function of number of iterations. See also figures 41.6 and 41.7.

and a step in $\mathbf{w}$ is made, given by

$$\Delta\mathbf{w} = -\tfrac{1}{2}\epsilon^2\mathbf{g} + \epsilon\mathbf{p}. \tag{41.20}$$

Notice that if the $\epsilon\mathbf{p}$ term were omitted this would simply be gradient descent with learning rate $\eta = \frac{1}{2}\epsilon^2$. This step in $\mathbf{w}$ is accepted or rejected depending on the change in the value of the objective function $M(\mathbf{w})$ and on the change in gradient, with a probability of acceptance such that detailed balance holds.

The Langevin method has one free parameter, ϵ, which controls the typical step size. If ϵ is set to too large a value, moves may be rejected. If it is set to a very small value, progress around the state space will be slow.

Demonstration of Langevin method

The Langevin method is demonstrated in figures 41.5, 41.6 and 41.7. Here, the objective function is $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$, with $\alpha = 0.01$. These figures include, for comparison, the results of the previous optimization method using gradient descent on the same objective function (figure 39.6). It can be seen that the mean evolution of $\mathbf{w}$ is similar to the evolution of the parameters under gradient descent. The Monte Carlo method appears to have converged to the posterior distribution after about 10 000 iterations.

The average acceptance rate during this simulation was 93%; only 7% of the proposed moves were rejected. Probably, faster progress around the state space would have been made if a larger step size ϵ had been used, but the value was chosen so that the 'descent rate' $\eta = \frac{1}{2}\epsilon^2$ matched the step size of the earlier simulations.

Making Bayesian predictions

From iteration 10,000 to 40,000, the weights were sampled every 1000 iterations and the corresponding functions of $\mathbf{x}$ are plotted in figure 41.6. There is a considerable variety of plausible functions. We obtain a Monte Carlo approximation to the Bayesian predictions by averaging these thirty functions of $\mathbf{x}$ together. The result is shown in figure 41.7 and contrasted with the predictions given by the optimized parameters. The Bayesian predictions become satisfyingly moderate as we move away from the region of highest data density.

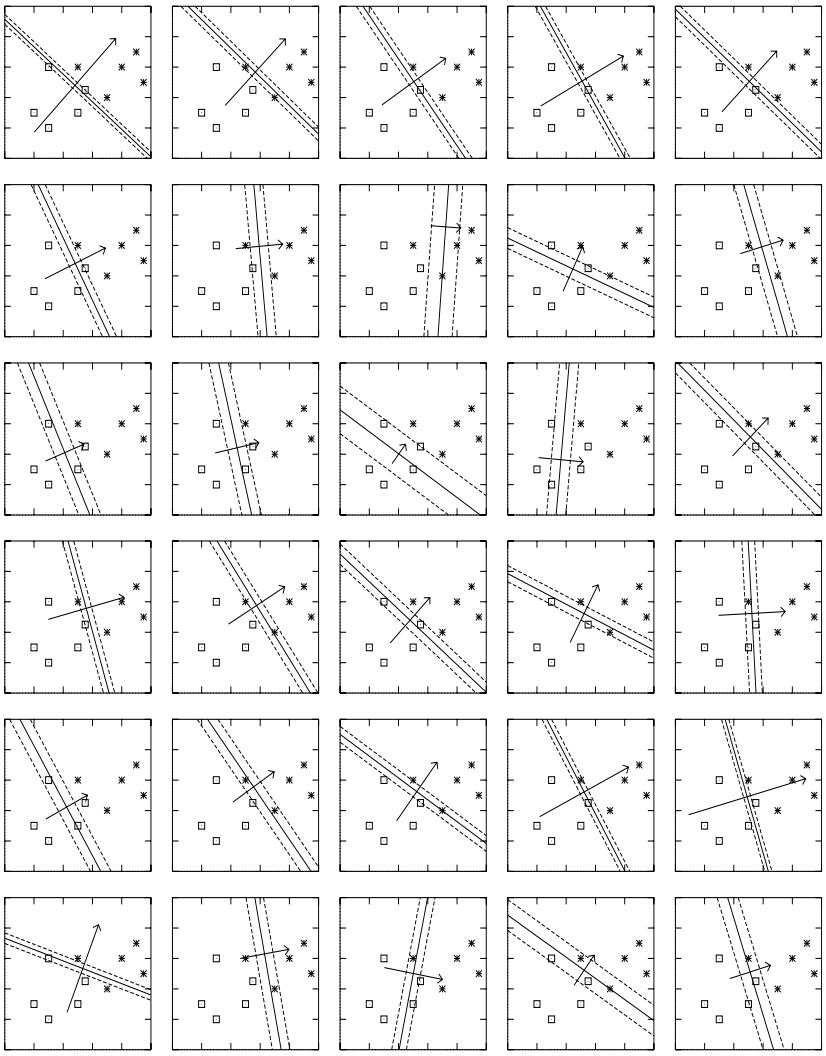


Figure 41.6. Samples obtained by the Langevin Monte Carlo method. The learning rate was set to $\eta = 0.01$ and the weight decay rate to $\alpha = 0.01$. The step size is given by $\epsilon = \sqrt{2\eta}$. The function performed by the neuron is shown by three of its contours every 1000 iterations from iteration 10 000 to 40 000. The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. Also shown is a vector proportional to (w_1, w_2).

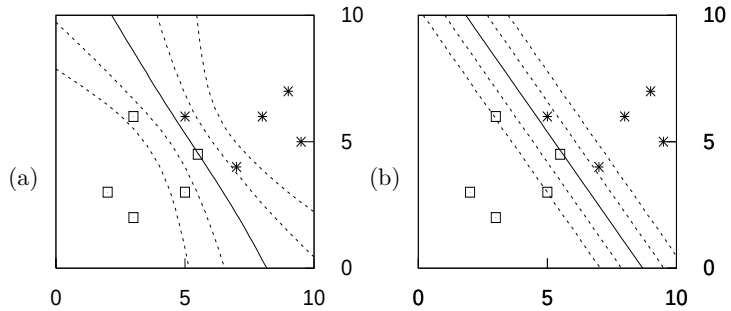


Figure 41.7. Bayesian predictions found by the Langevin Monte Carlo method compared with the predictions using the optimized parameters. (a) The predictive function obtained by averaging the predictions for 30 samples uniformly spaced between iterations 10 000 and 40 000, shown in figure 41.6. The contours shown are those corresponding to $a = 0, \pm 1, \pm 2$, namely $y = 0.5, 0.27$, 0.73, 0.12 and 0.88. (b) For contrast, the predictions given by the 'most probable' setting of the neuron's parameters, as given by optimization of $M(\mathbf{w})$.

```
wnew = w ;
gnew = g ;
for tau = 1:Tau

   p = p - epsilon * gnew / 2 ;     # make half-step in p
   wnew = wnew + epsilon * p ;      # make step in w

   gnew = gradM ( wnew ) ;          # find new gradient
   p = p - epsilon * gnew / 2 ;     # make half-step in p

endfor
```

Algorithm 41.8. `Octave` source code for the Hamiltonian Monte Carlo method. The algorithm is identical to the Langevin method in algorithm 41.4, except for the replacement of the four lines marked `*` in that algorithm by the fragment shown here.

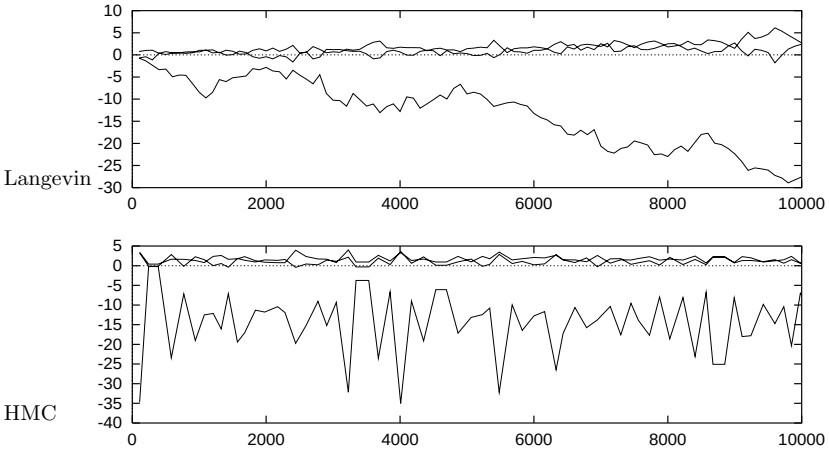


Figure 41.9. Comparison of sampling properties of the Langevin Monte Carlo method and the Hamiltonian Monte Carlo (HMC) method. The horizontal axis is the number of gradient evaluations made. Each figure shows the weights during the first 10,000 iterations. The rejection rate during this Hamiltonian Monte Carlo simulation was 8%.

The Bayesian classifier is better able to identify the points where the classification is uncertain. This pleasing behaviour results simply from a mechanical application of the rules of probability.

Optimization and typicality

A final observation concerns the behaviour of the functions $G(\mathbf{w})$ and $M(\mathbf{w})$ during the Monte Carlo sampling process, compared with the values of G and M at the optimum $\mathbf{w}_{\rm MP}$ (figure 41.5). The function $G(\mathbf{w})$ fluctuates around the value of $G(\mathbf{w}_{\rm MP})$, though not in a symmetrical way. The function $M(\mathbf{w})$ also fluctuates, but it does not fluctuate *around* $M(\mathbf{w}_{\rm MP})$ – obviously it cannot, because M is minimized at $\mathbf{w}_{\rm MP}$, so M could not go any smaller – furthermore, M only rarely drops close to $M(\mathbf{w}_{\rm MP})$. In the language of information theory, *the typical set of* $\mathbf{w}$ *has different properties from the most probable state* $\mathbf{w}_{\rm MP}$.

A general message therefore emerges – applicable to all data models, not just neural networks: one should be cautious about making use of *optimized* parameters, as the properties of optimized parameters may be unrepresentative of the properties of typical, plausible parameters; and the predictions obtained using optimized parameters alone will often be unreasonably overconfident.

Reducing random walk behaviour using Hamiltonian Monte Carlo

As a final study of Monte Carlo methods, we now compare the Langevin Monte Carlo method with its big brother, the Hamiltonian Monte Carlo method. The change to Hamiltonian Monte Carlo is simple to implement, as shown in algorithm 41.8. Each single proposal makes use of multiple gradient evaluations

along a dynamical trajectory in $\mathbf{w}, \mathbf{p}$ space, where $\mathbf{p}$ are the extra 'momentum' variables of the Langevin and Hamiltonian Monte Carlo methods. The number of steps '`Tau`' was set at random to a number between 100 and 200 for each trajectory. The step size ϵ was kept fixed so as to retain comparability with the simulations that have gone before; it is recommended that one randomize the step size in practical applications, however.

Figure 41.9 compares the sampling properties of the Langevin and Hamiltonian Monte Carlo methods. The autocorrelation of the state of the Hamiltonian Monte Carlo simulation falls much more rapidly with simulation time than that of the Langevin method. For this toy problem, Hamiltonian Monte Carlo is at least ten times more efficient in its use of computer time.

▶ 41.5 Implementing inference with Gaussian approximations

Physicists love to take nonlinearities and locally linearize them, and they love to approximate probability distributions by Gaussians. Such approximations offer an alternative strategy for dealing with the integral

$$P(\mathbf{t}^{(N+1)}=1 \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K\mathbf{w}\; y(\mathbf{x}^{(N+1)};\mathbf{w}) \frac{1}{Z_M} \exp(-M(\mathbf{w})), \quad (41.21)$$

which we just evaluated using Monte Carlo methods.

We start by making a Gaussian approximation to the posterior probability. We go to the minimum of $M(\mathbf{w})$ (using a gradient-based optimizer) and Taylor-expand M there:

$$M(\mathbf{w}) \simeq M(\mathbf{w}_{\mathrm{MP}}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}})^{\mathsf{T}}\mathbf{A}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}}) + \cdots, \quad (41.22)$$

where $\mathbf{A}$ is the matrix of second derivatives, also known as the *Hessian*, defined by

$$A_{ij} \equiv \frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}_{\mathrm{MP}}} . \quad (41.23)$$

We thus define our Gaussian approximation:

$$Q(\mathbf{w}; \mathbf{w}_{\mathrm{MP}}, \mathbf{A}) = [\det(\mathbf{A}/2\pi)]^{1/2} \exp\left[-\frac{1}{2}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}})^{\mathsf{T}}\mathbf{A}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}})\right]. \quad (41.24)$$

We can think of the matrix $\mathbf{A}$ as defining *error bars* on $\mathbf{w}$. To be precise, Q is a normal distribution whose variance–covariance matrix is $\mathbf{A}^{-1}$.

Exercise 41.1.[2] Show that the second derivative of $M(\mathbf{w})$ with respect to $\mathbf{w}$ is given by

$$\frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) = \sum_{n=1}^{N} f'(a^{(n)}) x_i^{(n)} x_j^{(n)} + \alpha\delta_{ij}, \quad (41.25)$$

where $f'(a)$ is the first derivative of $f(a) \equiv 1/(1+e^{-a})$, which is

$$f'(a) = \frac{\mathrm{d}}{\mathrm{d}a} f(a) = f(a)(1-f(a)), \quad (41.26)$$

and

$$a^{(n)} = \sum_j w_j x_j^{(n)}. \quad (41.27)$$

Having computed the Hessian, our task is then to perform the integral (41.21) using our Gaussian approximation.

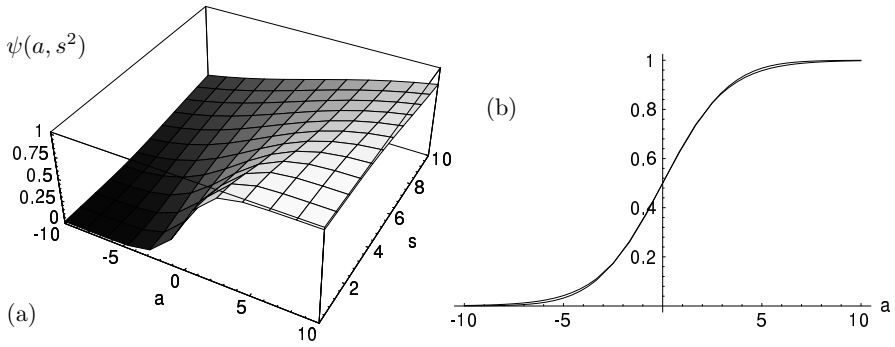


Figure 41.10. The marginalized probability, and an approximation to it. (a) The function $\psi(a, s^2)$, evaluated numerically. In (b) the functions $\psi(a, s^2)$ and $\phi(a, s^2)$ defined in the text are shown as a function of a for $s^2 = 4$. From MacKay (1992b).

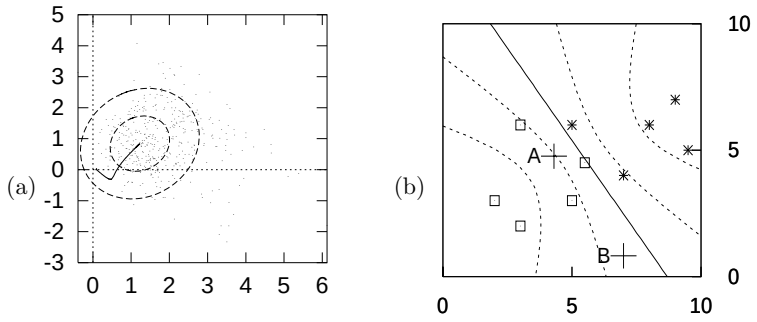


Figure 41.11. The Gaussian approximation in weight space and its approximate predictions in input space. (a) A projection of the Gaussian approximation onto the (w_1, w_2) plane of weight space. The one- and two-standard-deviation contours are shown. Also shown are the trajectory of the optimizer, and the Monte Carlo method's samples. (b) The predictive function obtained from the Gaussian approximation and equation (41.30). (cf. figure 42.2.)

Calculating the marginalized probability

The output $y(\mathbf{x}; \mathbf{w})$ depends on $\mathbf{w}$ only through the scalar $a(\mathbf{x}; \mathbf{w})$, so we can reduce the dimensionality of the integral by finding the probability density of a. We are assuming a locally Gaussian posterior probability distribution over $\mathbf{w} = \mathbf{w}_{\rm MP} + \Delta\mathbf{w}$, $P(\mathbf{w} \,|\, D, \alpha) \simeq (1/Z_Q) \exp(-\frac{1}{2}\Delta\mathbf{w}^{\mathsf{T}}\mathbf{A}\Delta\mathbf{w})$. For our single neuron, the activation $a(\mathbf{x}; \mathbf{w})$ is a linear function of $\mathbf{w}$ with $\partial a/\partial \mathbf{w} = \mathbf{x}$, so for any $\mathbf{x}$, the activation a is Gaussian-distributed.

▷ Exercise 41.2.[2] Assuming $\mathbf{w}$ is Gaussian-distributed with mean $\mathbf{w}_{\rm MP}$ and variance–covariance matrix $\mathbf{A}^{-1}$, show that the probability distribution of $a(\mathbf{x})$ is

$$P(a \,|\, \mathbf{x}, D, \alpha) = \text{Normal}(a_{\rm MP}, s^2) = \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{(a - a_{\rm MP})^2}{2s^2}\right), \tag{41.28}$$

where $a_{\rm MP} = a(\mathbf{x}; \mathbf{w}_{\rm MP})$ and $s^2 = \mathbf{x}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{x}$.

This means that the marginalized output is:

$$P(t=1 \,|\, \mathbf{x}, D, \alpha) = \psi(a_{\rm MP}, s^2) \equiv \int \mathrm{d}a \, f(a) \, \text{Normal}(a_{\rm MP}, s^2). \tag{41.29}$$

This is to be contrasted with $y(\mathbf{x}; \mathbf{w}_{\rm MP}) = f(a_{\rm MP})$, the output of the most probable network. The integral of a sigmoid times a Gaussian can be approximated by:

$$\psi(a_{\rm MP}, s^2) \simeq \phi(a_{\rm MP}, s^2) \equiv f(\kappa(s) a_{\rm MP}) \tag{41.30}$$

with $\kappa = 1/\sqrt{1 + \pi s^2/8}$ (figure 41.10).

Demonstration

Figure 41.11 shows the result of fitting a Gaussian approximation at the optimum $\mathbf{w}_{\rm MP}$, and the results of using that Gaussian approximation and equa-

tion (41.30) to make predictions. Comparing these predictions with those of the Langevin Monte Carlo method (figure 41.7) we observe that, whilst qualitatively the same, the two are clearly numerically different. So at least one of the two methods is not completely accurate.

▷ Exercise 41.3.[2] Is the Gaussian approximation to $P(\mathbf{w} \mid D, \alpha)$ too heavy-tailed or too light-tailed, or both? It may help to consider $P(\mathbf{w} \mid D, \alpha)$ as a function of one parameter w_i and to think of the two distributions on a logarithmic scale. Discuss the conditions under which the Gaussian approximation is most accurate.

Why marginalize?

If the output is immediately used to make a (0/1) decision and the costs associated with error are symmetrical, then the use of marginalized outputs under this Gaussian approximation will make no difference to the performance of the classifier, compared with using the outputs given by the most probable parameters, since both functions pass through 0.5 at $a_{\mathrm{MP}} = 0$. But these Bayesian outputs will make a difference if, for example, there is an option of saying 'I don't know', in addition to saying 'I guess 0' and 'I guess 1'. And even if there are just the two choices '0' and '1', if the costs associated with error are unequal, then the decision boundary will be some contour other than the 0.5 contour, and the boundary will be affected by marginalization.

Postscript on Supervised Neural Networks

One of my students, Robert, asked:

> Maybe I'm missing something fundamental, but supervised neural networks seem equivalent to fitting a pre-defined function to some given data, then extrapolating – what's the difference?

I agree with Robert. The supervised neural networks we have studied so far are simply parameterized nonlinear functions which can be fitted to data. Hopefully you will agree with another comment that Robert made:

> Unsupervised networks seem much more interesting than their supervised counterparts. I'm amazed that it works!

42

Hopfield Networks

We have now spent three chapters studying the single neuron. The time has come to connect multiple neurons together, making the output of one neuron be the input to another, so as to make neural networks.

Neural networks can be divided into two classes on the basis of their connectivity.

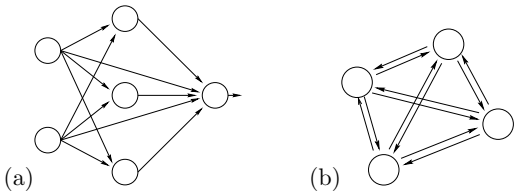


Figure 42.1. (a) A feedforward network. (b) A feedback network.

Feedforward networks. In a feedforward network, all the connections are directed such that the network forms a directed acyclic graph.

Feedback networks. Any network that is not a feedforward network will be called a feedback network.

In this chapter we will discuss a fully connected feedback network called the Hopfield network. The weights in the Hopfield network are constrained to be *symmetric*, i.e., the weight from neuron i to neuron j is equal to the weight from neuron j to neuron i.

Hopfield networks have two applications. First, they can act as *associative memories*. Second, they can be used to solve *optimization problems*. We will first discuss the idea of associative memory, also known as content-addressable memory.

▶ 42.1 Hebbian learning

In Chapter 38, we discussed the contrast between traditional digital memories and biological memories. Perhaps the most striking difference is the *associative* nature of biological memory.

A simple model due to Donald Hebb (1949) captures the idea of associative memory. Imagine that the weights between neurons whose activities are *positively correlated* are *increased*:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} \sim \mathrm{Correlation}(x_i, x_j). \tag{42.1}$$

Now imagine that when stimulus m is present (for example, the smell of a banana), the activity of neuron m increases; and that neuron n is associated

with another stimulus, n (for example, the sight of a yellow object). If these two stimuli – a yellow sight and a banana smell – co-occur in the environment, then the Hebbian learning rule (42.1) will increase the weights w_{nm} and w_{mn}. This means that when, on a later occasion, stimulus n occurs in isolation, making the activity x_n large, the positive weight from n to m will cause neuron m also to be activated. Thus the response to the sight of a yellow object is an automatic association with the smell of a banana. We could call this 'pattern completion'. No teacher is required for this associative memory to work. No signal is needed to indicate that a correlation has been detected or that an association should be made. The unsupervised, local learning algorithm and the unsupervised, local activity rule spontaneously produce associative memory.

This idea seems so simple and so effective that it must be relevant to how memories work in the brain.

▶ 42.2 Definition of the binary Hopfield network

Convention for weights. Our convention in general will be that w_{ij} denotes the connection *from* neuron j *to* neuron i.

Architecture. A Hopfield network consists of I neurons. They are fully connected through *symmetric, bidirectional* connections with weights $w_{ij} = w_{ji}$. There are no self-connections, so $w_{ii} = 0$ for all i. Biases w_{i0} may be included (these may be viewed as weights from a neuron '0' whose activity is permanently $x_0 = 1$). We will denote the activity of neuron i (its output) by x_i.

Activity rule. Roughly, a Hopfield network's activity rule is for each neuron to update its state as if it were a single neuron with the threshold activation function

$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0. \end{cases} \tag{42.2}$$

Since there is *feedback* in a Hopfield network (every neuron's output is an input to all the other neurons) we will have to specify an order for the updates to occur. The updates may be synchronous or asynchronous.

Synchronous updates – all neurons compute their activations

$$a_i = \sum_j w_{ij} x_j \tag{42.3}$$

then update their states simultaneously to

$$x_i = \Theta(a_i). \tag{42.4}$$

Asynchronous updates – one neuron at a time computes its activation and updates its state. The sequence of selected neurons may be a fixed sequence or a random sequence.

The properties of a Hopfield network may be sensitive to the above choices.

Learning rule. The learning rule is intended to make a set of desired *memories* $\{\mathbf{x}^{(n)}\}$ be *stable states* of the Hopfield network's activity rule. Each memory is a binary pattern, with $x_i \in \{-1, 1\}$.

(a)

moscow------russia
lima----------peru
london-----england
tokyo--------japan
edinburgh-scotland
ottawa------canada
oslo--------norway
stockholm---sweden
paris-------france

(b)
```
moscow---::::::::::   ⟹   moscow------russia
::::::::::--canada    ⟹   ottawa------canada
```

(c)
```
otowa-------canada    ⟹   ottawa------canada
egindurrh-sxotland    ⟹   edinburgh-scotland
```

Figure 42.2. Associative memory (schematic). (a) A list of desired memories. (b) The first purpose of an associative memory is pattern completion, given a partial pattern. (c) The second purpose of a memory is error correction.

The weights are set using the *sum of outer products* or *Hebb rule*,

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)}, \tag{42.5}$$

where η is an unimportant constant. To prevent the largest possible weight from growing with N we might choose to set $\eta = 1/N$.

Exercise 42.1.[1] Explain why the value of η is not important for the Hopfield network defined above.

▶ 42.3 Definition of the continuous Hopfield network

Using the identical architecture and learning rule we can define a Hopfield network whose activities are real numbers between -1 and 1.

Activity rule. A Hopfield network's activity rule is for each neuron to update its state as if it were a single neuron with a sigmoid activation function. The updates may be synchronous or asynchronous, and involve the equations

$$a_i = \sum_j w_{ij} x_j \tag{42.6}$$

and

$$x_i = \tanh(a_i). \tag{42.7}$$

The learning rule is the same as in the binary Hopfield network, but the value of η becomes relevant. Alternatively, we may fix η and introduce a *gain* $\beta \in (0, \infty)$ into the activation function:

$$x_i = \tanh(\beta a_i). \tag{42.8}$$

Exercise 42.2.[1] Where have we encountered equations 42.6, 42.7, and 42.8 before?

▶ 42.4 Convergence of the Hopfield network

The hope is that the Hopfield networks we have defined will perform associative memory recall, as shown schematically in figure 42.2. We hope that the activity rule of a Hopfield network will take a partial memory or a corrupted memory, and perform pattern completion or error correction to restore the original memory.

But why should we expect *any* pattern to be stable under the activity rule, let alone the desired memories?

We address the continuous Hopfield network, since the binary network is a special case of it. We have already encountered the activity rule (42.6, 42.8)

when we discussed variational methods (section 33.2): when we approximated the spin system whose energy function was

$$E(\mathbf{x};\mathbf{J}) = -\frac{1}{2}\sum_{m,n} J_{mn}x_m x_n - \sum_n h_n x_n \tag{42.9}$$

with a separable distribution

$$Q(\mathbf{x};\mathbf{a}) = \frac{1}{Z_Q}\exp\left(\sum_n a_n x_n\right) \tag{42.10}$$

and optimized the latter so as to minimize the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})E(\mathbf{x};\mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})\ln\frac{1}{Q(\mathbf{x};\mathbf{a})}, \tag{42.11}$$

we found that the pair of iterative equations

$$a_m = \beta\left(\sum_n J_{mn}\bar{x}_n + h_m\right) \tag{42.12}$$

and

$$\bar{x}_n = \tanh(a_n) \tag{42.13}$$

were guaranteed to decrease the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\left(-\frac{1}{2}\sum_{m,n} J_{mn}\bar{x}_m\bar{x}_n - \sum_n h_n\bar{x}_n\right) - \sum_n H_2^{(e)}(q_n). \tag{42.14}$$

If we simply replace J by w, $\bar{x}$ by x, and h_n by w_{i0}, we see that the equations of the Hopfield network are identical to a set of mean-field equations that minimize

$$\beta\tilde{F}(\mathbf{x}) = -\beta\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x} - \sum_i H_2^{(e)}[(1+x_i)/2]. \tag{42.15}$$

There is a general name for a function that decreases under the dynamical evolution of a system and that is bounded below: such a function is a *Lyapunov function* for the system. It is useful to be able to prove the existence of Lyapunov functions: if a system has a Lyapunov function then its dynamics are bound to settle down to a *fixed point*, which is a local minimum of the Lyapunov function, or a *limit cycle*, along which the Lyapunov function is a constant. Chaotic behaviour is not possible for a system with a Lyapunov function. If a system has a Lyapunov function then its state space can be divided into *basins of attraction*, one basin associated with each attractor.

So, the continuous Hopfield network's activity rules (if implemented asynchronously) have a Lyapunov function. This Lyapunov function is a convex function of each parameter a_i so a Hopfield network's dynamics will always converge to a stable fixed point.

This convergence proof depends crucially on the fact that the Hopfield network's connections are *symmetric*. It also depends on the updates being made asynchronously.

Exercise 42.3.[2, p.520] Show by constructing an example that if a feedback network does not have symmetric connections then its dynamics may fail to converge to a fixed point.

Exercise 42.4.[2, p.521] Show by constructing an example that if a Hopfield network is updated synchronously that, from some initial conditions, it may fail to converge to a fixed point.

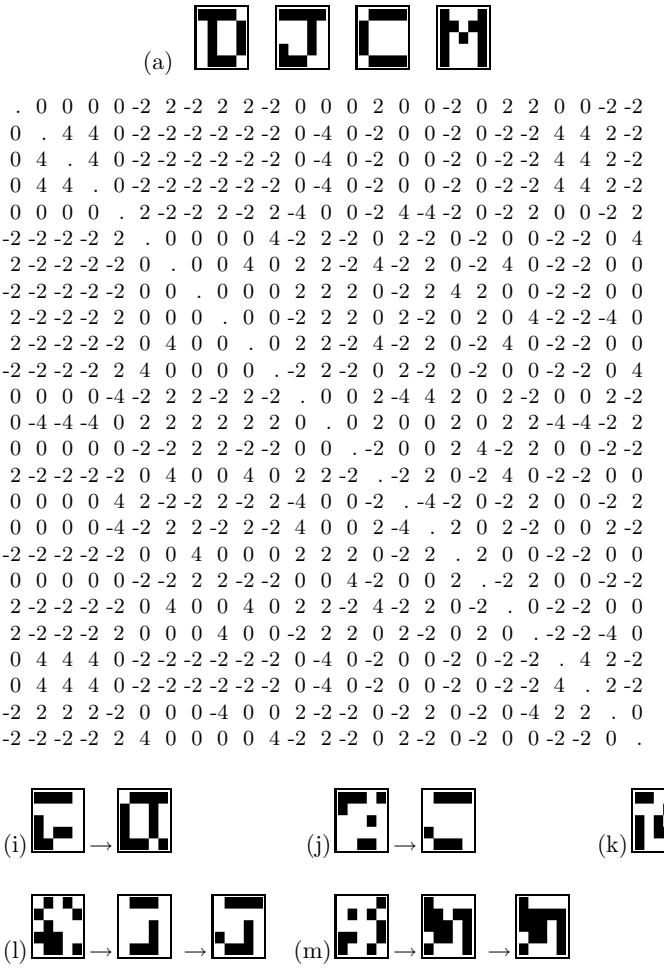


Figure 42.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to stable states other than the four memories; in (i), the stable state looks like a mixture of two memories, 'D' and 'J'; stable state (j) is like a mixture of 'J' and 'C'; in (k), we find a corrupted version of the 'M' memory (two bits distant); in (l) a corrupted version of 'J' (four bits distant) and in (m), a state which looks spurious until we recognize that it is the inverse of the stable state (l).

42.5 The associative memory in action

Figure 42.3 shows the dynamics of a 25-unit binary Hopfield network that has learnt four patterns by Hebbian learning. The four patterns are displayed as five by five binary images in figure 42.3a. For twelve initial conditions, panels (b–m) show the state of the network, iteration by iteration, all 25 units being updated asynchronously in each iteration. For an initial condition randomly perturbed from a memory, it often only takes one iteration for all the errors to be corrected. The network has more stable states in addition to the four desired memories: the inverse of any stable state is also a stable state; and there are several stable states that can be interpreted as mixtures of the memories.

Brain damage

The network can be severely damaged and still work fine as an associative memory. If we take the 300 weights of the network shown in figure 42.3 and randomly set 50 or 100 of them to zero, we still find that the desired memories are attracting stable states. Imagine a digital computer that still works fine even when 20% of its components are destroyed!

▷ Exercise 42.5.[3C] Implement a Hopfield network and confirm this amazing robust error-correcting capability.

More memories

We can squash more memories into the network too. Figure 42.4a shows a set of five memories. When we train the network with Hebbian learning, all five memories are stable states, even when 26 of the weights are randomly deleted (as shown by the 'x's in the weight matrix). However, the basins of attraction are smaller than before: figures 42.4(b–f) show the dynamics resulting from randomly chosen starting states close to each of the memories (3 bits flipped). Only three of the memories are recovered correctly.

If we try to store too many patterns, the associative memory fails catastrophically. When we add a sixth pattern, as shown in figure 42.5, only one of the patterns is stable; the others all flow into one of two spurious stable states.

42.6 The continuous-time continuous Hopfield network

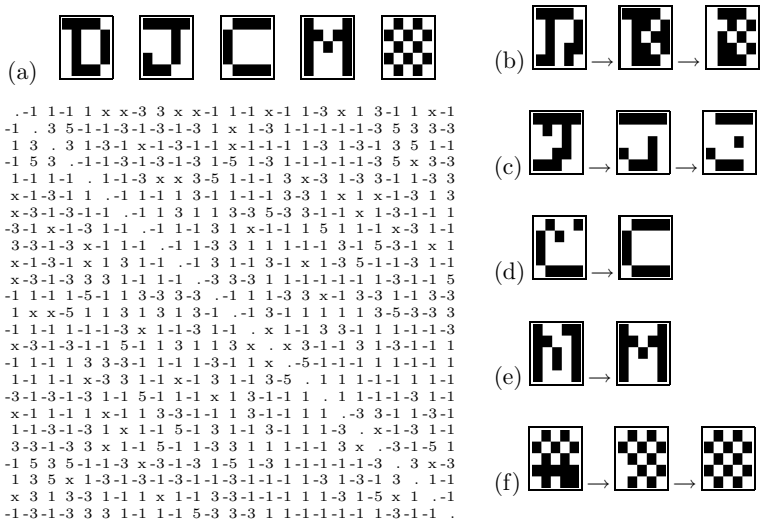
The fact that the Hopfield network's properties are not robust to the minor change from asynchronous to synchronous updates might be a cause for concern; can this model be a useful model of biological networks? It turns out that once we move to a continuous-time version of the Hopfield networks, this issue melts away.

We assume that each neuron's activity x_i is a continuous function of time $x_i(t)$ and that the activations $a_i(t)$ are computed instantaneously in accordance with

$$a_i(t) = \sum_j w_{ij} x_j(t). \tag{42.16}$$

The neuron's response to its activation is assumed to be mediated by the differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)), \tag{42.17}$$



```
 .-1 1-1 1 x x-3 3 x x-1 1-1 x-1 1-3 x 1 3-1 1 x-1
-1  . 3 5-1-1-3-1-3-1-3 1 x 1-3 1-1-1-1-1-3 5 3 3-3
 1 3  . 3 1-3-1 x-1-3-1-1 x-1-1-1 1-3 1-3-1 3 5 1-1
-1 5 3  .-1-1-3-1-3-1-3 1-5 1-3 1-1-1-1-1-3 5 x 3-3
 1-1 1-1  . 1-1-3 x x 3-5 1-1-1 3 x-3 1-3 3-1 1-3 3
 x-1-3-1 1  .-1 1-1 1 3-1 1-1-1 3-3 1 x 1 x-1-3 1 3
 x-3-1-3-1-1  .-1 1 3 1 1 3-3 5-3 3-1-1 x 1-3-1-1 1
-3-1 x-1-3 1-1  .-1 1-1 3 1 x-1-1 1 5 1 1-1 x-3 1-1
 3-3-1-3 x-1 1-1  .-1 1-3 3 1 1 1-1-1 3-1 5-3-1 x 1
 x-1-3-1 x 1 3 1-1  .-1 3 1-1 3-1 x 1-3 5-1-1-3 1-1
 x-3-1-3 3 3 1-1 1-1  .-3 3-3 1 1-1-1-1-1 1-3-1-1 5
-1 1-1 1-5-1 1 3-3 3-3  .-1 1 1-3 3 x-1 3-3 1-1 3-3
 1 x x-5 1 1 3 1 3 1 3-1  .-1 3-1 1 1 1 1 3-5-3-3 3
-1 1-1 1-1-1-3 x 1-1-3 1-1  . x 1-1 3 3-1 1 1-1-1-3
 x-3-1-3-1-1 5-1 1 3 1 1 3 x  . x 3-1-1 3 1-3-1-1 1
-1 1-1 1 3 3-3-1 1-1 1-3-1 1 x  .-5-1-1-1 1 1-1-1 1
 1-1 1-1 x-3 3 1-1 x-1 3 1-1 3-5  . 1 1 1-1-1 1 1-1
-3-1-3-1-3 1-1 5-1 1-1 x 1 3-1-1 1  . 1 1-1-1-3 1-1
 x-1 1-1 1 x-1 1 3-3-1-1 1 3-1-1 1 1  .-3 3-1 1-3-1
 1-1-3-1-3 1 x 1-1 5-1 3 1-1 3-1 1 1-3  . x-1-3 1-1
 3-3-1-3 3 x 1-1 5-1 1-3 3 1 1 1-1-1 3 x  .-3-1-5 1
-1 5 3 5-1-1-3 x-3-1-3 1-5 1-3 1-1-1-1-1-3  . 3 x-3
 1 3 5 x 1-3-1-3-1-3-1-1-3-1-1-1 1-3 1-3-1 3  . 1-1
 x 3 1 3-3 1-1 1 x 1-1 3-3-1-1-1 1 1-3 1-5 x 1  .-1
-1-3-1-3 3 3 1-1 1-1 5-3 3-3 1 1-1-1-1-1 1-3-1-1  .
```

Figure 42.4. Hopfield network storing five memories, and suffering deletion of 26 of its 300 weights. (a) The five memories, and the weights of the network, with deleted weights shown by 'x'. (b–f) Initial states that differ by three random bits from a memory: some are restored, but some converge to other states.

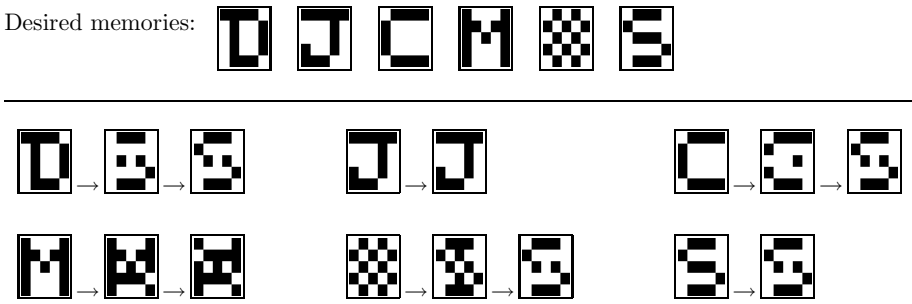


Figure 42.5. An overloaded Hopfield network trained on six memories, most of which are not stable.

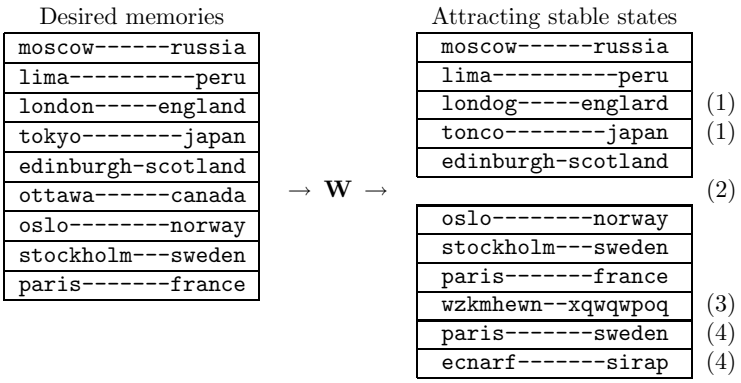


Figure 42.6. Failure modes of a Hopfield network (highly schematic). A list of desired memories, and the resulting list of attracting stable states. Notice (1) some memories that are retained with a small number of errors; (2) desired memories that are completely lost (there is no attracting stable state at the desired memory or near it); (3) spurious stable states unrelated to the original list; (4) spurious stable states that are confabulations of desired memories.

where $f(a)$ is the activation function, for example $f(a) = \tanh(a)$. For a steady activation a_i, the activity $x_i(t)$ relaxes exponentially to $f(a_i)$ with time-constant τ.

Now, here is the nice result: as long as the weight matrix is symmetric, this system has the variational free energy (42.15) as its Lyapunov function.

▷ Exercise 42.6.[1] By computing $\frac{\mathrm{d}}{\mathrm{d}t}\tilde{F}$, prove that the variational free energy $\tilde{F}(\mathbf{x})$ is a Lyapunov function for the continuous-time Hopfield network.

It is particularly easy to prove that a function L is a Lyapunov function if the system's dynamics perform steepest descent on L, with $\frac{\mathrm{d}}{\mathrm{d}t}x_i(t) \propto \frac{\partial}{\partial x_i}L$. In the case of the continuous-time continuous Hopfield network, it is not quite so simple, but every component of $\frac{\mathrm{d}}{\mathrm{d}t}x_i(t)$ does have the same sign as $\frac{\partial}{\partial x_i}\tilde{F}$, which means that with an appropriately defined metric, the Hopfield network dynamics do perform steepest descents on $\tilde{F}(\mathbf{x})$.

▶ 42.7 The capacity of the Hopfield network

One way in which we viewed learning in the single neuron was as communication – communication of the labels of the training data set from one point in time to a later point in time. We found that the capacity of a linear threshold neuron was 2 bits per weight.

Similarly, we might view the Hopfield associative memory as a communication channel (figure 42.6). A list of desired memories is encoded into a set of weights $\mathbf{W}$ using the Hebb rule of equation (42.5), or perhaps some other learning rule. The receiver, receiving the weights $\mathbf{W}$ only, finds the stable states of the Hopfield network, which he interprets as the original memories. This communication system can fail in various ways, as illustrated in the figure.

1. Individual bits in some memories might be corrupted, that is, a stable state of the Hopfield network is displaced a little from the desired memory.

2. Entire memories might be absent from the list of attractors of the network; or a stable state might be present but have such a small basin of attraction that it is of no use for pattern completion and error correction.

3. Spurious additional memories unrelated to the desired memories might be present.

4. Spurious additional memories derived from the desired memories by operations such as mixing and inversion may also be present.

Of these failure modes, modes 1 and 2 are clearly undesirable, mode 2 especially so. Mode 3 might not matter so much as long as each of the desired memories has a large basin of attraction. The fourth failure mode might in some contexts actually be viewed as beneficial. For example, if a network is required to memorize examples of valid sentences such as 'John loves Mary' and 'John gets cake', we might be happy to find that 'John loves cake' was also a stable state of the network. We might call this behaviour 'generalization'.

The capacity of a Hopfield network with I neurons might be defined to be the number of random patterns N that can be stored without failure-mode 2 having substantial probability. If we also require failure-mode 1 to have tiny probability then the resulting capacity is much smaller. We now study these alternative definitions of the capacity.

The capacity of the Hopfield network – stringent definition

We will first explore the information storage capabilities of a binary Hopfield network that learns using the Hebb rule by considering the stability of just one bit of one of the desired patterns, assuming that the state of the network is set to that desired pattern $\mathbf{x}^{(n)}$. We will assume that the patterns to be stored are randomly selected binary patterns.

The activation of a particular neuron i is

$$a_i = \sum_j w_{ij} x_j^{(n)}, \tag{42.18}$$

where the weights are, for $i \neq j$,

$$w_{ij} = x_i^{(n)} x_j^{(n)} + \sum_{m \neq n} x_i^{(m)} x_j^{(m)}. \tag{42.19}$$

Here we have split $\mathbf{W}$ into two terms, the first of which will contribute 'signal', reinforcing the desired memory, and the second 'noise'. Substituting for w_{ij}, the activation is

$$a_i = \sum_{j \neq i} x_i^{(n)} x_j^{(n)} x_j^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)} \tag{42.20}$$

$$= (I-1) x_i^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)}. \tag{42.21}$$

The first term is $(I-1)$ times the desired state $x_i^{(n)}$. If this were the only term, it would keep the neuron firmly clamped in the desired state. The second term is a sum of $(I-1)(N-1)$ random quantities $x_i^{(m)} x_j^{(m)} x_j^{(n)}$. A moment's reflection confirms that these quantities are independent random binary variables with mean 0 and variance 1.

Thus, considering the statistics of a_i under the ensemble of random patterns, we conclude that a_i has mean $(I-1)x_i^{(n)}$ and variance $(I-1)(N-1)$.

For brevity, we will now assume I and N are large enough that we can neglect the distinction between I and $I-1$, and between N and $N-1$. Then we can restate our conclusion: a_i is Gaussian-distributed with mean $I x_i^{(n)}$ and variance IN.

What then is the probability that the selected bit is stable, if we put the network into the state $\mathbf{x}^{(n)}$? The probability that bit i will flip on the first iteration of the Hopfield network's dynamics is

$$P(i \text{ unstable}) = \Phi\left(-\frac{I}{\sqrt{IN}}\right) = \Phi\left(-\frac{1}{\sqrt{N/I}}\right), \tag{42.22}$$

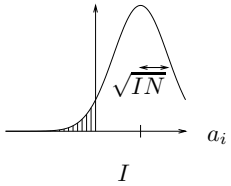


Figure 42.7. The probability density of the activation a_i in the case $x_i^{(n)} = 1$; the probability that bit i becomes flipped is the area of the tail.

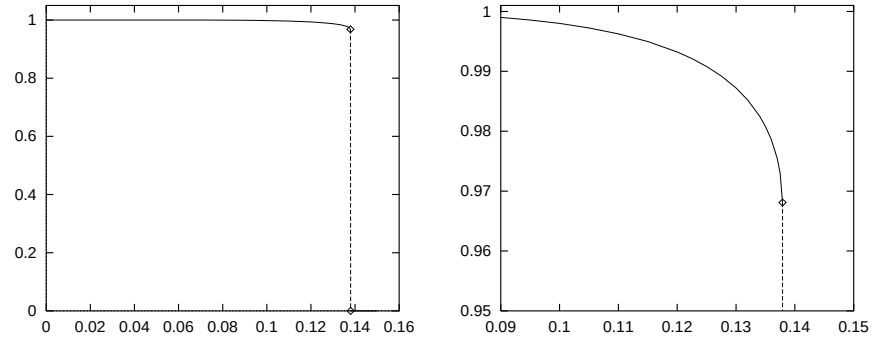


Figure 42.8. Overlap between a desired memory and the stable state nearest to it as a function of the loading fraction N/I. The overlap is defined to be the scaled inner product $\sum_i x_i x_i^{(n)}/I$, which is 1 when recall is perfect and zero when the stable state has 50% of the bits flipped. There is an abrupt transition at $N/I = 0.138$, where the overlap drops from 0.97 to zero.

where

$$\Phi(z) \equiv \int_{-\infty}^{z} \mathrm{d}z \, \frac{1}{\sqrt{2\pi}} e^{-z^2/2}. \tag{42.23}$$

The important quantity N/I is the ratio of the number of patterns stored to the number of neurons. If, for example, we try to store $N \simeq 0.18I$ patterns in the Hopfield network then there is a chance of 1% that a specified bit in a specified pattern will be unstable on the first iteration.

We are now in a position to derive our first capacity result, for the case where no corruption of the desired memories is permitted.

▷ Exercise 42.7.[2] Assume that we wish all the desired patterns to be completely stable – we don't want any of the bits to flip when the network is put into any desired pattern state – and the total probability of any error at all is required to be less than a small number ϵ. Using the approximation to the error function for large z,

$$\Phi(-z) \simeq \frac{1}{\sqrt{2\pi}} \frac{e^{-z^2/2}}{z}, \tag{42.24}$$

show that the maximum number of patterns that can be stored, $N_{\max}$, is

$$N_{\max} \simeq \frac{I}{4 \ln I + 2 \ln(1/\epsilon)}. \tag{42.25}$$

If, however, we allow a small amount of corruption of memories to occur, the number of patterns that can be stored increases.

The statistical physicists' capacity

The analysis that led to equation (42.22) tells us that if we try to store $N \simeq 0.18I$ patterns in the Hopfield network then, starting from a desired memory, about 1% of the bits will be unstable on the first iteration. Our analysis does not shed light on what is expected to happen on subsequent iterations. The flipping of these bits might make some of the other bits unstable too, causing an increasing number of bits to be flipped. This process might lead to an avalanche in which the network's state ends up a long way from the desired memory.

In fact, when N/I is large, such avalanches do happen. When N/I is small, they tend not to – there is a stable state near to each desired memory. For the limit of large I, Amit *et al.* (1985) have used methods from statistical physics to find numerically the transition between these two behaviours. There is a sharp discontinuity at

$$N_{\mathrm{crit}} = 0.138I. \tag{42.26}$$

Below this critical value, there is likely to be a stable state near every desired memory, in which a small fraction of the bits are flipped. When N/I exceeds 0.138, the system has only spurious stable states, known as *spin glass states*, none of which is correlated with any of the desired memories. Just below the critical value, the fraction of bits that are flipped when a desired memory has evolved to its associated stable state is 1.6%. Figure 42.8 shows the overlap between the desired memory and the nearest stable state as a function of N/I.

Some other transitions in properties of the model occur at some additional values of N/I, as summarized below.

For all N/I, stable spin glass states exist, uncorrelated with the desired memories.

For $N/I > 0.138$, these spin glass states are the only stable states.

For $N/I \in (0, 0.138)$, there are stable states close to the desired memories.

For $N/I \in (0, 0.05)$, the stable states associated with the desired memories have lower energy than the spurious spin glass states.

For $N/I \in (0.05, 0.138)$, the spin glass states dominate – there are spin glass states that have lower energy than the stable states associated with the desired memories.

For $N/I \in (0, 0.03)$, there are additional *mixture* states, which are combinations of several desired memories. These stable states do not have as low energy as the stable states associated with the desired memories.

In conclusion, the capacity of the Hopfield network with I neurons, if we define the capacity in terms of the abrupt discontinuity discussed above, is $0.138I$ random binary patterns, each of length I, each of which is received with 1.6% of its bits flipped. In bits, this capacity is

$$0.138I^2 \times (1 - H_2(0.016)) = 0.122\, I^2 \text{ bits.} \tag{42.27}$$

This expression for the capacity omits a smaller negative term of order $N \log_2 N$ bits, associated with the arbitrary order of the memories.

Since there are $I^2/2$ weights in the network, we can also express the capacity as *0.24 bits per weight.*

▶ 42.8 Improving on the capacity of the Hebb rule

The capacities discussed in the previous section are the capacities of the Hopfield network whose weights are set using the Hebbian learning rule. We can do better than the Hebb rule by defining an objective function that measures how well the network stores all the memories, and minimizing it.

For an associative memory to be useful, it must be able to correct at least one flipped bit. Let's make an objective function that measures whether flipped bits tend to be restored correctly. Our intention is that, for every neuron i in the network, the weights to that neuron should satisfy this rule:

> for every pattern $\mathbf{x}^{(n)}$, if the neurons other than i are set correctly to $x_j = x_j^{(n)}$, then the activation of neuron i should be such that its preferred output is $x_i = x_i^{(n)}$.

Is this rule a familiar idea? Yes, it is precisely what we wanted the single neuron of Chapter 39 to do. Each pattern $\mathbf{x}^{(n)}$ defines an input, target pair for the single neuron i. And it defines an input, target pair for all the other neurons too.

```
w = x' * x ;        # initialize the weights using Hebb rule

for l = 1:L         # loop L times

        for i=1:I           #
          w(i,i) = 0 ;      #  ensure the self-weights are zero.
        end                 #

        a  = x * w      ;   # compute all activations
        y  = sigmoid(a) ;   # compute all outputs
        e  = t - y      ;   # compute all errors
        gw = x' * e     ;   # compute the gradients
        gw = gw + gw'   ;   # symmetrize gradients

        w  = w + eta * ( gw - alpha * w ) ;  # make step

endfor
```

Algorithm 42.9. `Octave` source code for optimizing the weights of a Hopfield network, so that it works as an associative memory. cf. algorithm 39.5. The data matrix `x` has I columns and N rows. The matrix `t` is identical to `x` except that -1s are replaced by 0s.

So, just as we defined an objective function (39.11) for the training of a single neuron as a classifier, we can define

$$G(\mathbf{W}) = -\sum_i \sum_n t_i^{(n)} \ln y_i^{(n)} + (1 - t_i^{(n)}) \ln(1 - y_i^{(n)}) \tag{42.28}$$

where

$$t_i^{(n)} = \begin{cases} 1 & x_i^{(n)} = 1 \\ 0 & x_i^{(n)} = -1 \end{cases} \tag{42.29}$$

and

$$y_i^{(n)} = \frac{1}{1 + \exp(-a_i^{(n)})}, \text{ where } a_i^{(n)} = \sum w_{ij} x_j^{(n)} \ . \tag{42.30}$$

We can then steal the algorithm (algorithm 39.5, p.478) which we wrote for the single neuron, to write an algorithm for optimizing a Hopfield network, algorithm 42.9. The convenient syntax of `Octave` requires very few changes; the extra lines enforce the constraints that the self-weights w_{ii} should all be zero and that the weight matrix should be symmetrical ($w_{ij} = w_{ji}$).

As expected, this learning algorithm does a better job than the one-shot Hebbian learning rule. When the six patterns of figure 42.5, which cannot be memorized by the Hebb rule, are learned using algorithm 42.9, all six patterns become stable states.

Exercise 42.8.[4C] Implement this learning rule and investigate empirically its capacity for memorizing random patterns; also compare its avalanche properties with those of the Hebb rule.

▶ 42.9 Hopfield networks for optimization problems

Since a Hopfield network's dynamics minimize an energy function, it is natural to ask whether we can map interesting optimization problems onto Hopfield networks. Biological data processing problems often involve an element of *constraint satisfaction* – in scene interpretation, for example, one might wish to infer the spatial location, orientation, brightness and texture of each visible element, and which visible elements are connected together in objects. These inferences are constrained by the given data and by prior knowledge about continuity of objects.

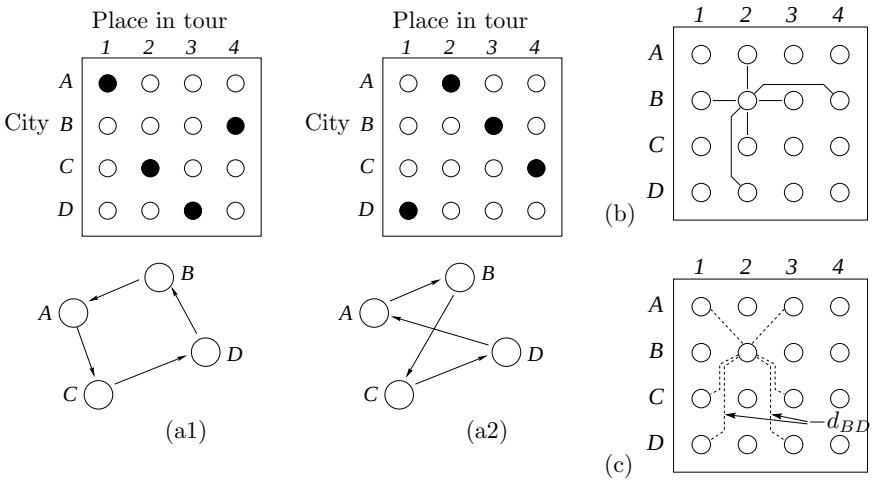


Figure 42.10. Hopfield network for solving a travelling salesman problem with $K = 4$ cities. (a1,2) Two solution states of the 16-neuron network, with activites represented by black = 1, white = 0; and the tours corresponding to these network states. (b) The negative weights between node $B2$ and other nodes; these weights enforce validity of a tour. (c) The negative weights that embody the distance objective function.

Hopfield and Tank (1985) suggested that one might take an interesting constraint satisfaction problem and design the weights of a binary or continuous Hopfield network such that the settling process of the network would minimize the objective function of the problem.

The travelling salesman problem

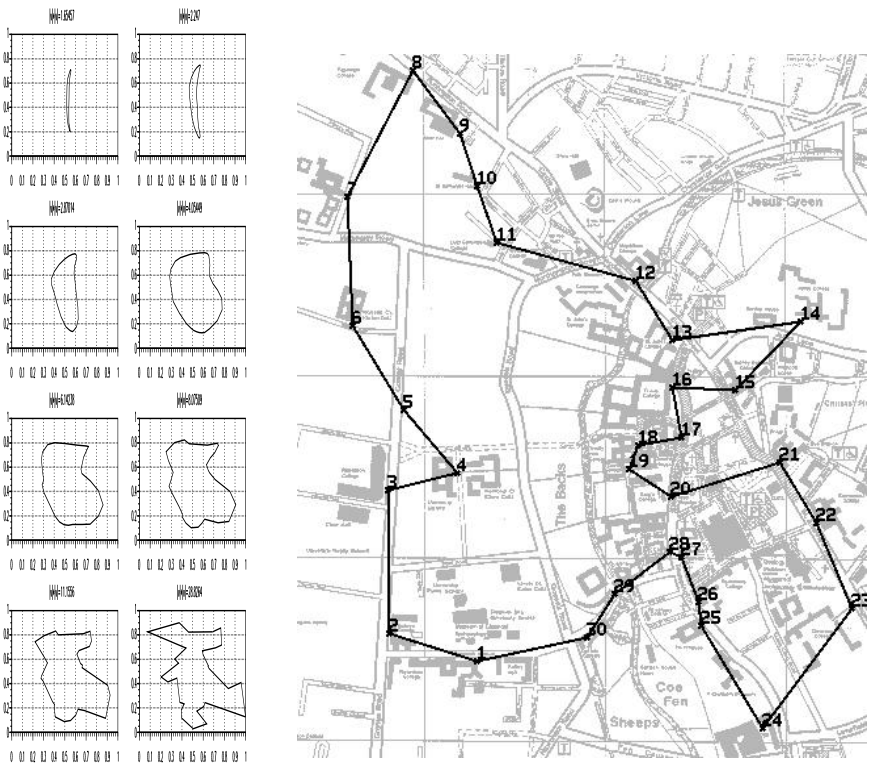
A classic constraint satisfaction problem to which Hopfield networks have been applied is the travelling salesman problem.

A set of K cities is given, and a matrix of the $K(K-1)/2$ distances between those cities. The task is to find a closed tour of the cities, visiting each city once, that has the smallest total distance. The travelling salesman problem is equivalent in difficulty to an NP-complete problem.

The method suggested by Hopfield and Tank is to represent a tentative solution to the problem by the state of a network with $I = K^2$ neurons arranged in a square, with each neuron representing the hypothesis that a particular city comes at a particular point in the tour. It will be convenient to consider the states of the neurons as being between 0 and 1 rather than -1 and 1. Two solution states for a four-city travelling salesman problem are shown in figure 42.10a.

The weights in the Hopfield network play two roles. First, they must define an energy function which is minimized only when the state of the network represents a *valid* tour. A valid state is one that looks like a permutation matrix, having exactly one '1' in every row and one '1' in every column. This rule can be enforced by putting large negative weights between any pair of neurons that are in the same row or the same column, and setting a positive bias for all neurons to ensure that K neurons do turn on. Figure 42.10b shows the negative weights that are connected to one neuron, '$B2$', which represents the statement 'city B comes second in the tour'.

Second, the weights must encode the objective function that we want to minimize – the total distance. This can be done by putting negative weights proportional to the appropriate distances between the nodes in adjacent columns. For example, between the B and D nodes in adjacent columns, the weight would be $-d_{BD}$. The negative weights that are connected to neuron $B2$ are shown in figure 42.10c. The result is that when the network is in a valid state, its total energy will be the total distance of the corresponding



(a) (b)

Figure 42.11. (a) Evolution of the state of a continuous Hopfield network solving a travelling salesman problem using Aiyer's (1991) graduated non-convexity method; the state of the network is projected into the two-dimensional space in which the cities are located by finding the centre of mass for each point in the tour, using the neuron activities as the mass function. (b) The travelling scholar problem. The shortest tour linking the 27 Cambridge Colleges, the Engineering Department, the University Library, and Sree Aiyer's house. From Aiyer (1991).

tour, plus a constant given by the energy associated with the biases.

Now, since a Hopfield network minimizes its energy, it is hoped that the binary or continuous Hopfield network's dynamics will take the state to a minimum that is a valid tour and which might be an optimal tour. This hope is not fulfilled for large travelling salesman problems, however, without some careful modifications. We have not specified the size of the weights that enforce the tour's validity, relative to the size of the distance weights, and setting this scale factor poses difficulties. If 'large' validity-enforcing weights are used, the network's dynamics will rattle into a valid state with little regard for the distances. If 'small' validity-enforcing weights are used, it is possible that the distance weights will cause the network to adopt an *invalid* state that has lower energy than any valid state. Our original formulation of the energy function puts the objective function and the solution's validity in potential conflict with each other. This difficulty has been resolved by the work of Sree Aiyer (1991), who showed how to modify the distance weights so that they would not interfere with the solution's validity, and how to define a continuous Hopfield network whose dynamics are at all times confined to a 'valid subspace'. Aiyer used a *graduated non-convexity* or *deterministic annealing* approach to find good solutions using these Hopfield networks. The deterministic annealing approach involves gradually increasing the gain β of the neurons in the network from 0 to ∞, at which point the state of the network corresponds to a valid tour. A sequence of trajectories generated by applying this method to a thirty-city travelling salesman problem is shown in figure 42.11a.

A solution to the 'travelling scholar problem' found by Aiyer using a continuous Hopfield network is shown in figure 42.11b.

▶ 42.10 Further exercises

▷ Exercise 42.9.[3] Storing two memories.

Two binary memories $\mathbf{m}$ and $\mathbf{n}$ ($m_i, n_i \in \{-1, +1\}$) are stored by Hebbian learning in a Hopfield network using

$$w_{ij} = \begin{cases} m_i m_j + n_i n_j & \text{for } i \neq j \\ 0 & \text{for } i = j. \end{cases} \tag{42.31}$$

The biases b_i are set to zero.

The network is put in the state $\mathbf{x} = \mathbf{m}$. Evaluate the activation a_i of neuron i and show that in can be written in the form

$$a_i = \mu m_i + \nu n_i. \tag{42.32}$$

By comparing the signal strength, μ, with the magnitude of the noise strength, $|\nu|$, show that $\mathbf{x} = \mathbf{m}$ is a stable state of the dynamics of the network.

The network is put in a state $\mathbf{x}$ differing in D places from $\mathbf{m}$,

$$\mathbf{x} = \mathbf{m} + 2\mathbf{d}, \tag{42.33}$$

where the perturbation $\mathbf{d}$ satisfies $d_i \in \{-1, 0, +1\}$. D is the number of components of $\mathbf{d}$ that are non-zero, and for each d_i that is non-zero, $d_i = -m_i$. Defining the overlap between $\mathbf{m}$ and $\mathbf{n}$ to be

$$o_{\mathbf{mn}} = \sum_{i=1}^{I} m_i n_i, \tag{42.34}$$

evaluate the activation a_i of neuron i again and show that the dynamics of the network will restore $\mathbf{x}$ to $\mathbf{m}$ if the number of flipped bits satisfies

$$D < \frac{1}{4}(I - |o_{\mathbf{mn}}| - 2). \tag{42.35}$$

How does this number compare with the maximum number of flipped bits that can be corrected by the optimal decoder, assuming the vector $\mathbf{x}$ is either a noisy version of $\mathbf{m}$ or of $\mathbf{n}$?

Exercise 42.10.[3] Hopfield network as a collection of binary classifiers. This exercise explores the link between unsupervised networks and supervised networks. If a Hopfield network's desired memories are all attracting stable states, then every neuron in the network has weights going to it that solve a classification problem personal to that neuron. Take the set of memories and write them in the form $\mathbf{x}'^{(n)}, x_i^{(n)}$, where $\mathbf{x}'$ denotes all the components $x_{i'}$ for all $i' \neq i$, and let $\mathbf{w}'$ denote the vector of weights $w_{ii'}$, for $i' \neq i$.

Using what we know about the capacity of the single neuron, show that it is almost certainly impossible to store more than $2I$ random memories in a Hopfield network of I neurons.

Lyapunov functions

Exercise 42.11.[3] Erik's puzzle. In a stripped-down version of Conway's game of life, cells are arranged on a square grid. Each cell is either alive or dead. Live cells do not die. Dead cells become alive if two or more of their immediate neighbours are alive. (Neighbours to north, south, east and west.) What is the smallest number of live cells needed in order that these rules lead to an entire $N \times N$ square being alive?

In a d-dimensional version of the same game, the rule is that if d neighbours are alive then you come to life. What is the smallest number of live cells needed in order that an entire $N \times N \times \cdots \times N$ hypercube becomes alive? (And how should those live cells be arranged?)

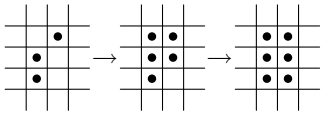


Figure 42.12. Erik's dynamics.

The southeast puzzle

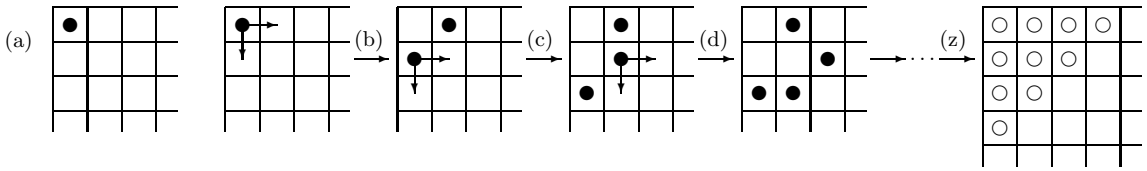


Figure 42.13. The `southeast` puzzle.

The `southeast` puzzle is played on a semi-infinite chess board, starting at its northwest (top left) corner. There are three rules:

1. In the starting position, one piece is placed in the northwest-most square (figure 42.13a).
2. It is not permitted for more than one piece to be on any given square.
3. At each step, you remove one piece from the board, and replace it with two pieces, one in the square immediately to the east, and one in the the square immediately to the south, as illustrated in figure 42.13b. Every such step increases the number of pieces on the board by one.

After move (b) has been made, either piece may be selected for the next move. Figure 42.13c shows the outcome of moving the lower piece. At the next move, either the lowest piece or the middle piece of the three may be selected; the uppermost piece may not be selected, since that would violate rule 2. At move (d) we have selected the middle piece. Now any of the pieces may be moved, except for the leftmost piece.

Now, here is the puzzle:

▷ Exercise 42.12.[4, p.521] Is it possible to obtain a position in which all the ten squares closest to the northwest corner, marked in figure 42.13z, are empty?

[Hint: this puzzle has a connection to data compression.]

▶ 42.11 Solutions

Solution to exercise 42.3 (p.508). Take a binary feedback network with 2 neurons and let $w_{12} = 1$ and $w_{21} = -1$. Then whenever neuron 1 is updated, it will match neuron 2, and whenever neuron 2 is updated, it will flip to the opposite state from neuron 1. There is no stable state.

Solution to exercise 42.4 (p.508). Take a binary Hopfield network with 2 neurons and let $w_{12} = w_{21} = 1$, and let the initial condition be $x_1 = 1$, $x_2 = -1$. Then if the dynamics are synchronous, on every iteration both neurons will flip their state. The dynamics do not converge to a fixed point.

Solution to exercise 42.12 (p.520). The key to this problem is to notice its similarity to the construction of a binary symbol code. Starting from the empty string, we can build a binary tree by repeatedly splitting a codeword into two. Every codeword has an implicit probability 2^{-l}, where l is the depth of the codeword in the binary tree. Whenever we split a codeword in two and create two new codewords whose length is increased by one, the two new codewords each have implicit probability equal to half that of the old codeword. For a complete binary code, the Kraft equality affirms that the sum of these implicit probabilities is 1.

Similarly, in `southeast`, we can associate a 'weight' with each piece on the board. If we assign a weight of 1 to any piece sitting on the top left square; a weight of 1/2 to any piece on a square whose distance from the top left is one; a weight of 1/4 to any piece whose distance from the top left is two; and so forth, with 'distance' being the city-block distance; then every legal move in `southeast` leaves unchanged the total weight of all pieces on the board. Lyapunov functions come in two flavours: the function may be a function of state whose value is known to stay constant; or it may be a function of state that is bounded below, and whose value always decreases or stays constant. The total weight is a Lyapunov function of the second type.

The starting weight is 1, so now we have a powerful tool: a conserved function of the state. Is it possible to find a position in which the ten highest-weight squares are vacant, and the total weight is 1? What is the total weight if *all* the other squares on the board are occupied (figure 42.14)? The total weight would be $\sum_{l=4}^{\infty}(l+1)2^{-l}$, which is equal to 3/4. So it is impossible to empty all ten of those squares.

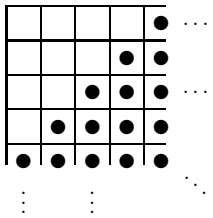


Figure 42.14. A possible position for the `southeast` puzzle?

43

Boltzmann Machines

▶ 43.1 From Hopfield networks to Boltzmann machines

We have noticed that the binary Hopfield network minimizes an energy function

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x} \tag{43.1}$$

and that the continuous Hopfield network with activation function $x_n = \tanh(a_n)$ can be viewed as *approximating* the probability distribution associated with that energy function,

$$P(\mathbf{x}\,|\,\mathbf{W}) = \frac{1}{Z(\mathbf{W})}\exp[-E(\mathbf{x})] = \frac{1}{Z(\mathbf{W})}\exp\left[\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x}\right]. \tag{43.2}$$

These observations motivate the idea of working with a neural network model that actually *implements* the above probability distribution.

The *stochastic Hopfield network* or *Boltzmann machine* (Hinton and Sejnowski, 1986) has the following activity rule:

Activity rule of Boltzmann machine: after computing the activation a_i (42.3),

$$\text{set } x_i = +1 \text{ with probability } \frac{1}{1+e^{-2a_i}} \qquad \text{else set } x_i = -1. \tag{43.3}$$

This rule implements Gibbs sampling for the probability distribution (43.2).

Boltzmann machine learning

Given a set of examples $\{\mathbf{x}^{(n)}\}_1^N$ from the real world, we might be interested in adjusting the weights $\mathbf{W}$ such that the generative model

$$P(\mathbf{x}\,|\,\mathbf{W}) = \frac{1}{Z(\mathbf{W})}\exp\left[\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x}\right] \tag{43.4}$$

is well matched to those examples. We can derive a learning algorithm by writing down Bayes' theorem to obtain the posterior probability of the weights given the data:

$$P(\mathbf{W}\,|\,\{\mathbf{x}^{(n)}\}_1^N\}) = \frac{\left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)}\,|\,\mathbf{W})\right] P(\mathbf{W})}{P(\{\mathbf{x}^{(n)}\}_1^N\})}. \tag{43.5}$$

We concentrate on the first term in the numerator, the likelihood, and derive a maximum likelihood algorithm (though there might be advantages in pursuing a full Bayesian approach as we did in the case of the single neuron). We differentiate the logarithm of the likelihood,

$$\ln\left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)} \,|\, \mathbf{W})\right] = \sum_{n=1}^{N}\left[\frac{1}{2}\mathbf{x}^{(n)\mathsf{T}}\mathbf{W}\mathbf{x}^{(n)} - \ln Z(\mathbf{W})\right], \tag{43.6}$$

with respect to w_{ij}, bearing in mind that $\mathbf{W}$ is defined to be symmetric with $w_{ji} = w_{ij}$.

Exercise 43.1.[2] Show that the derivative of $\ln Z(\mathbf{W})$ with respect to w_{ij} is

$$\frac{\partial}{\partial w_{ij}} \ln Z(\mathbf{W}) = \sum_{\mathbf{x}} x_i x_j P(\mathbf{x} \,|\, \mathbf{W}) = \langle x_i x_j \rangle_{P(\mathbf{x}\,|\,\mathbf{W})}. \tag{43.7}$$

[This exercise is similar to exercise 22.12 (p.307).]

The derivative of the log likelihood is therefore:

$$\frac{\partial}{\partial w_{ij}} \ln P(\{\mathbf{x}^{(n)}\}_1^N \,|\, \mathbf{W}) = \sum_{n=1}^{N}\left[x_i^{(n)} x_j^{(n)} - \langle x_i x_j \rangle_{P(\mathbf{x}\,|\,\mathbf{W})}\right] \tag{43.8}$$

$$= N\left[\langle x_i x_j \rangle_{\text{Data}} - \langle x_i x_j \rangle_{P(\mathbf{x}\,|\,\mathbf{W})}\right]. \tag{43.9}$$

This gradient is proportional to the difference of two terms. The first term is the *empirical* correlation between x_i and x_j,

$$\langle x_i x_j \rangle_{\text{Data}} \equiv \frac{1}{N}\sum_{n=1}^{N}\left[x_i^{(n)} x_j^{(n)}\right], \tag{43.10}$$

and the second term is the correlation between x_i and x_j under the current model,

$$\langle x_i x_j \rangle_{P(\mathbf{x}\,|\,\mathbf{W})} \equiv \sum_{\mathbf{x}} x_i x_j P(\mathbf{x} \,|\, \mathbf{W}). \tag{43.11}$$

The first correlation $\langle x_i x_j \rangle_{\text{Data}}$ is readily evaluated – it is just the empirical correlation between the activities in the real world. The second correlation, $\langle x_i x_j \rangle_{P(\mathbf{x}\,|\,\mathbf{W})}$, is not so easy to evaluate, but it can be estimated by Monte Carlo methods, that is, by observing the average value of $x_i x_j$ while the activity rule of the Boltzmann machine, equation (43.3), is iterated.

In the special case $\mathbf{W} = 0$, we can evaluate the gradient exactly because, by symmetry, the correlation $\langle x_i x_j \rangle_{P(\mathbf{x}\,|\,\mathbf{W})}$ must be zero. If the weights are adjusted by gradient descent with learning rate η, then, after one iteration, the weights will be

$$w_{ij} = \eta \sum_{n=1}^{N}\left[x_i^{(n)} x_j^{(n)}\right], \tag{43.12}$$

precisely the value of the weights given by the Hebb rule, equation (16.5), with which we trained the Hopfield network.

Interpretation of Boltzmann machine learning

One way of viewing the two terms in the gradient (43.9) is as 'waking' and 'sleeping' rules. While the network is 'awake', it measures the correlation between x_i and x_j in the real world, and weights are *increased* in proportion.

While the network is 'asleep', it 'dreams' about the world using the generative model (43.4), and measures the correlations between x_i and x_j in the model world; these correlations determine a proportional *decrease* in the weights. If the second-order correlations in the dream world match the correlations in the real world, then the two terms balance and the weights do not change.

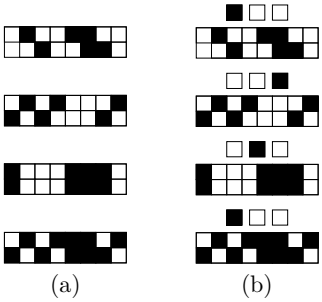


Figure 43.1. The 'shifter' ensembles. (a) Four samples from the plain shifter ensemble. (b) Four corresponding samples from the labelled shifter ensemble.

Criticism of Hopfield networks and simple Boltzmann machines

Up to this point we have discussed Hopfield networks and Boltzmann machines in which all of the neurons correspond to *visible* variables x_i. The result is a probabilistic model that, when optimized, can capture the second-order statistics of the environment. [The second-order statistics of an ensemble $P(\mathbf{x})$ are the expected values $\langle x_i x_j \rangle$ of all the pairwise products $x_i x_j$.] The real world, however, often has higher-order correlations that must be included if our description of it is to be effective. Often the second-order correlations in themselves may carry little or no useful information.

Consider, for example, the ensemble of binary images of chairs. We can imagine images of chairs with various designs – four-legged chairs, comfy chairs, chairs with five legs and wheels, wooden chairs, cushioned chairs, chairs with rockers instead of legs. A child can easily learn to distinguish these images from images of carrots and parrots. But I expect the second-order statistics of the raw data are useless for describing the ensemble. Second-order statistics only capture whether two pixels are likely to be in the same state as each other. Higher-order concepts are needed to make a good generative model of images of chairs.

A simpler ensemble of images in which high-order statistics are important is the 'shifter ensemble', which comes in two flavours. Figure 43.1a shows a few samples from the 'plain shifter ensemble'. In each image, the bottom eight pixels are a copy of the top eight pixels, either shifted one pixel to the left, or unshifted, or shifted one pixel to the right. (The top eight pixels are set at random.) This ensemble is a simple model of the visual signals from the two eyes arriving at early levels of the brain. The signals from the two eyes are similar to each other but may differ by small translations because of the varying depth of the visual world. This ensemble is simple to describe, but its second-order statistics convey no useful information. The correlation between one pixel and any of the three pixels above it is 1/3. The correlation between any other two pixels is zero.

Figure 43.1b shows a few samples from the 'labelled shifter ensemble'. Here, the problem has been made easier by including an extra three neurons that label the visual image as being an instance of either the 'shift left', 'no shift', or 'shift right' sub-ensemble. But with this extra information, the ensemble is still not learnable using second-order statistics alone. The second-order correlation between any label neuron and any image neuron is zero. We need models that can capture higher-order statistics of an environment.

So, how can we develop such models? One idea might be to create models that directly capture higher-order correlations, such as:

$$P'(\mathbf{x} \,|\, \mathbf{W}, \mathbf{V}, \ldots) = \frac{1}{Z'} \exp\left(\frac{1}{2} \sum_{ij} w_{ij} x_i x_j + \frac{1}{6} \sum_{ij} v_{ijk} x_i x_j x_k + \cdots \right). \tag{43.13}$$

Such *higher-order Boltzmann machines* are equally easy to simulate using stochastic updates, and the learning rule for the higher-order parameters v_{ijk} is equivalent to the learning rule for w_{ij}.

▷ Exercise 43.2.[2] Derive the gradient of the log likelihood with respect to v_{ijk}.

It is possible that the spines found on biological neurons are responsible for detecting correlations between small numbers of incoming signals. However, to capture statistics of high enough order to describe the ensemble of images of chairs well would require an unimaginable number of terms. To capture merely the fourth-order statistics in a 128×128 pixel image, we need more than 10^7 parameters.

So measuring moments of images is *not* a good way to describe their underlying structure. Perhaps what we need instead or in addition are *hidden variables*, also known to statisticians as *latent variables*. This is the important innovation introduced by Hinton and Sejnowski (1986). The idea is that the high-order correlations among the visible variables are described by including extra hidden variables and sticking to a model that has only second-order interactions between its variables; the hidden variables induce higher-order correlations between the visible variables.

▶ 43.2 Boltzmann machine with hidden units

We now add *hidden neurons* to our stochastic model. These are neurons that do not correspond to observed variables; they are free to play any role in the probabilistic model defined by equation (43.4). They might actually take on interpretable roles, effectively performing 'feature extraction'.

Learning in Boltzmann machines with hidden units

The activity rule of a Boltzmann machine with hidden units is identical to that of the original Boltzmann machine. The learning rule can again be derived by maximum likelihood, but now we need to take into account the fact that the states of the hidden units are unknown. We will denote the states of the visible units by $\mathbf{x}$, the states of the hidden units by $\mathbf{h}$, and the generic state of a neuron (either visible or hidden) by y_i, with $\mathbf{y} \equiv (\mathbf{x}, \mathbf{h})$. The state of the network when the visible neurons are clamped in state $\mathbf{x}^{(n)}$ is $\mathbf{y}^{(n)} \equiv (\mathbf{x}^{(n)}, \mathbf{h})$. The likelihood of $\mathbf{W}$ given a single data example $\mathbf{x}^{(n)}$ is

$$P(\mathbf{x}^{(n)} \,|\, \mathbf{W}) = \sum_{\mathbf{h}} P(\mathbf{x}^{(n)}, \mathbf{h} \,|\, \mathbf{W}) = \sum_{\mathbf{h}} \frac{1}{Z(\mathbf{W})} \exp\left[\frac{1}{2}[\mathbf{y}^{(n)}]^{\mathsf{T}} \mathbf{W} \mathbf{y}^{(n)}\right], \tag{43.14}$$

where

$$Z(\mathbf{W}) = \sum_{\mathbf{x},\mathbf{h}} \exp\left[\frac{1}{2}\mathbf{y}^{\mathsf{T}} \mathbf{W} \mathbf{y}\right]. \tag{43.15}$$

Equation (43.14) may also be written

$$P(\mathbf{x}^{(n)} \,|\, \mathbf{W}) = \frac{Z_{\mathbf{x}^{(n)}}(\mathbf{W})}{Z(\mathbf{W})} \tag{43.16}$$

where

$$Z_{\mathbf{x}^{(n)}}(\mathbf{W}) = \sum_{\mathbf{h}} \exp\left[\frac{1}{2}[\mathbf{y}^{(n)}]^{\mathsf{T}} \mathbf{W} \mathbf{y}^{(n)}\right]. \tag{43.17}$$

Differentiating the likelihood as before, we find that the derivative with respect to any weight w_{ij} is again the difference between a 'waking' term and a 'sleeping' term,

$$\frac{\partial}{\partial w_{ij}} \ln P(\{\mathbf{x}^{(n)}\}_1^N \,|\, \mathbf{W}) = \sum_n \left\{ \langle y_i y_j \rangle_{P(\mathbf{h} \,|\, \mathbf{x}^{(n)}, \mathbf{W})} - \langle y_i y_j \rangle_{P(\mathbf{x},\mathbf{h} \,|\, \mathbf{W})} \right\}. \tag{43.18}$$

The first term $\langle y_i y_j \rangle_{P(\mathbf{h}|\mathbf{x}^{(n)},\mathbf{W})}$ is the correlation between y_i and y_j if the Boltzmann machine is simulated with the visible variables clamped to $\mathbf{x}^{(n)}$ and the hidden variables freely sampling from their conditional distribution.

The second term $\langle y_i y_j \rangle_{P(\mathbf{x},\mathbf{h}|\mathbf{W})}$ is the correlation between y_i and y_j when the Boltzmann machine generates samples from its model distribution.

Hinton and Sejnowski demonstrated that non-trivial ensembles such as the labelled shifter ensemble can be learned using a Boltzmann machine with hidden units. The hidden units take on the role of feature detectors that spot patterns likely to be associated with one of the three shifts.

The Boltzmann machine is time-consuming to simulate because the computation of the gradient of the log likelihood depends on taking the difference of two gradients, both found by Monte Carlo methods. So Boltzmann machines are not in widespread use. It is an area of active research to create models that embody the same capabilities using more efficient computations (Hinton *et al.*, 1995; Dayan *et al.*, 1995; Hinton and Ghahramani, 1997; Hinton, 2001; Hinton and Teh, 2001).

▶ 43.3 Exercise

▷ Exercise 43.3.[3] Can the 'bars and stripes' ensemble (figure 43.2) be learned by a Boltzmann machine with no hidden units? [You may be surprised!]

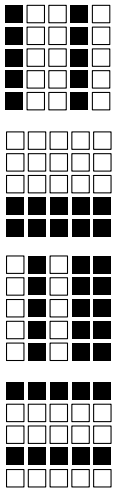


Figure 43.2. Four samples from the 'bars and stripes' ensemble. Each sample is generated by first picking an orientation, horizontal or vertical; then, for each row of spins in that orientation (each bar or stripe respectively), switching all spins on with probability $1/2$.

44

Supervised Learning in Multilayer Networks

▶ 44.1 Multilayer perceptrons

No course on neural networks could be complete without a discussion of supervised multilayer networks, also known as backpropagation networks.

The *multilayer perceptron* is a feedforward network. It has input neurons, hidden neurons and output neurons. The hidden neurons may be arranged in a sequence of layers. The most common multilayer perceptrons have a single hidden layer, and are known as 'two-layer' networks, the number 'two' counting the number of layers of neurons not including the inputs.

Such a feedforward network defines a nonlinear parameterized mapping from an input $\mathbf{x}$ to an output $\mathbf{y} = \mathbf{y}(\mathbf{x}; \mathbf{w}, \mathcal{A})$. The output is a continuous function of the input and of the parameters $\mathbf{w}$; the architecture of the net, i.e., the functional form of the mapping, is denoted by $\mathcal{A}$. Feedforward networks can be 'trained' to perform regression and classification tasks.

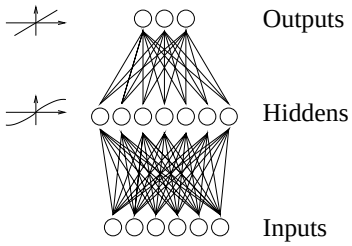


Figure 44.1. A typical two-layer network, with six inputs, seven hidden units, and three outputs. Each line represents one weight.

Regression networks

In the case of a regression problem, the mapping for a network with one hidden layer may have the form:

$$\text{Hidden layer:} \quad a_j^{(1)} = \sum_l w_{jl}^{(1)} x_l + \theta_j^{(1)}; \quad h_j = f^{(1)}(a_j^{(1)}) \tag{44.1}$$

$$\text{Output layer:} \quad a_i^{(2)} = \sum_j w_{ij}^{(2)} h_j + \theta_i^{(2)}; \quad y_i = f^{(2)}(a_i^{(2)}) \tag{44.2}$$

where, for example, $f^{(1)}(a) = \tanh(a)$, and $f^{(2)}(a) = a$. Here l runs over the inputs $x_1, \ldots, x_L$, j runs over the hidden units, and i runs over the outputs. The 'weights' w and 'biases' θ together make up the parameter vector $\mathbf{w}$. The nonlinear sigmoid function $f^{(1)}$ at the hidden layer gives the neural network greater computational flexibility than a standard linear regression model. Graphically, we can represent the neural network as a set of layers of connected neurons (figure 44.1).

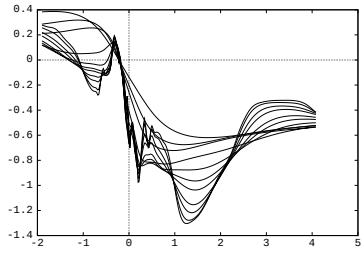


Figure 44.2. Samples from the prior over functions of a one-input network. For each of a sequence of values of $\sigma_{\text{bias}} = 8$, 6, 4, 3, 2, 1.6, 1.2, 0.8, 0.4, 0.3, 0.2, and $\sigma_{\text{in}} = 5\sigma_{\text{bias}}^{w}$, one random function is shown. The other hyperparameters of the network were $H = 400$, $\sigma_{\text{out}}^{w} = 0.05$.

What sorts of functions can these networks implement?

Just as we explored the weight space of the single neuron in Chapter 39, examining the functions it could produce, let us explore the weight space of a multilayer network. In figures 44.2 and 44.3 I take a network with one input and one output and a large number H of hidden units, set the biases

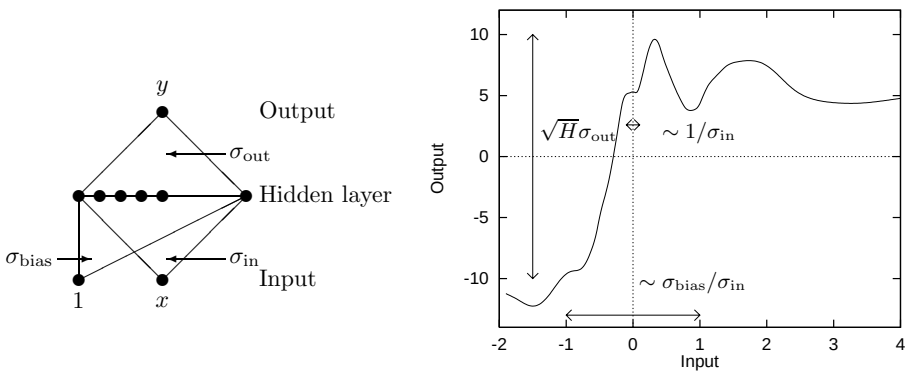


Figure 44.3. Properties of a function produced by a random network. The vertical scale of a typical function produced by the network with random weights is of order $\sqrt{H}\sigma_{\rm out}$; the horizontal range in which the function varies significantly is of order $\sigma_{\rm bias}/\sigma_{\rm in}$; and the shortest horizontal length scale is of order $1/\sigma_{\rm in}$. The function shown was produced by making a random network with $H = 400$ hidden units, and Gaussian weights with $\sigma_{\rm bias} = 4$, $\sigma_{\rm in} = 8$, and $\sigma_{\rm out} = 0.5$.

and weights $\theta_j^{(1)}$, $w_{jl}^{(1)}$, $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values, and plot the resulting function $y(x)$. I set the hidden units' biases $\theta_j^{(1)}$ to random values from a Gaussian with zero mean and standard deviation $\sigma_{\rm bias}$; the input-to-hidden weights $w_{jl}^{(1)}$ to random values with standard deviation $\sigma_{\rm in}$; and the bias and output weights $\theta_i^{(2)}$ and $w_{ij}^{(2)}$ to random values with standard deviation $\sigma_{\rm out}$.

The sort of functions that we obtain depend on the values of $\sigma_{\rm bias}$, $\sigma_{\rm in}$ and $\sigma_{\rm out}$. As the weights and biases are made bigger we obtain more complex functions with more features and a greater sensitivity to the input variable. The vertical scale of a typical function produced by the network with random weights is of order $\sqrt{H}\sigma_{\rm out}$; the horizontal range in which the function varies significantly is of order $\sigma_{\rm bias}/\sigma_{\rm in}$; and the shortest horizontal length scale is of order $1/\sigma_{\rm in}$.

Radford Neal (1996) has also shown that in the limit as $H \to \infty$ the statistical properties of the functions generated by randomizing the weights are independent of the number of hidden units; so, interestingly, the complexity of the functions becomes independent of the number of parameters in the model. What determines the complexity of the typical functions is the characteristic magnitude of the weights. Thus we anticipate that when we fit these models to real data, an important way of controlling the complexity of the fitted function will be to control the characteristic magnitude of the weights.

Figure 44.4 shows one typical function produced by a network with two inputs and one output. This should be contrasted with the function produced by a traditional linear regression model, which is a flat plane. Neural networks can create functions with more complexity than a linear regression.

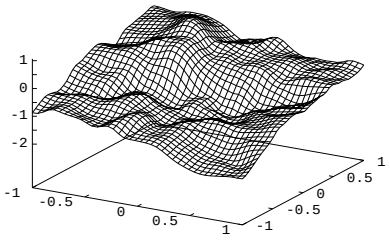


Figure 44.4. One sample from the prior of a two-input network with $\{H, \sigma_{\rm in}^w, \sigma_{\rm bias}^w, \sigma_{\rm out}^w\} = \{400, 8.0, 8.0, 0.05\}$.

▶ 44.2 How a regression network is traditionally trained

This network is trained using a data set $D = \{\mathbf{x}^{(n)}, \mathbf{t}^{(n)}\}$ by adjusting $\mathbf{w}$ so as to minimize an error function, e.g.,

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_n \sum_i \left(t_i^{(n)} - y_i(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 . \tag{44.3}$$

This objective function is a sum of terms, one for each input/target pair $\{\mathbf{x}, \mathbf{t}\}$, measuring how close the output $\mathbf{y}(\mathbf{x}; \mathbf{w})$ is to the target $\mathbf{t}$.

This minimization is based on repeated evaluation of the gradient of E_D. This gradient can be efficiently computed using the *backpropagation* algorithm (Rumelhart *et al.*, 1986), which uses the chain rule to find the derivatives.

Often, regularization (also known as weight decay) is included, modifying the objective function to:

$$M(\mathbf{w}) = \beta E_D + \alpha E_W \tag{44.4}$$

where, for example, $E_W = \frac{1}{2}\sum_i w_i^2$. This additional term favours small values of $\mathbf{w}$ and decreases the tendency of a model to overfit noise in the training data.

Rumelhart *et al.* (1986) showed that multilayer perceptrons can be trained, by gradient descent on $M(\mathbf{w})$, to discover solutions to non-trivial problems such as deciding whether an image is symmetric or not. These networks have been successfully applied to real-world tasks as varied as pronouncing English text (Sejnowski and Rosenberg, 1987) and focussing multiple-mirror telescopes (Angel *et al.*, 1990).

▶ 44.3 Neural network learning as inference

The neural network learning process above can be given the following probabilistic interpretation. [Here we repeat and generalize the discussion of Chapter 41.]

The error function is interpreted as defining a noise model. βE_D is the negative log likelihood:

$$P(D \mid \mathbf{w}, \beta, \mathcal{H}) = \frac{1}{Z_D(\beta)} \exp(-\beta E_D). \tag{44.5}$$

Thus, the use of the sum-squared error E_D (44.3) corresponds to an assumption of Gaussian noise on the target variables, and the parameter β defines a noise level $\sigma_\nu^2 = 1/\beta$.

Similarly the regularizer is interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w} \mid \alpha, \mathcal{H}) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{44.6}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$. The probabilistic model $\mathcal{H}$ specifies the architecture $\mathcal{A}$ of the network, the likelihood (44.5), and the prior (44.6).

The objective function $M(\mathbf{w})$ then corresponds to the *inference* of the parameters $\mathbf{w}$, given the data:

$$P(\mathbf{w} \mid D, \alpha, \beta, \mathcal{H}) = \frac{P(D \mid \mathbf{w}, \beta, \mathcal{H}) P(\mathbf{w} \mid \alpha, \mathcal{H})}{P(D \mid \alpha, \beta, \mathcal{H})} \tag{44.7}$$

$$= \frac{1}{Z_M} \exp(-M(\mathbf{w})). \tag{44.8}$$

The $\mathbf{w}$ found by (locally) minimizing $M(\mathbf{w})$ is then interpreted as the (locally) most probable parameter vector, $\mathbf{w}_{\rm MP}$.

The interpretation of $M(\mathbf{w})$ as a log probability adds little new at this stage. But new tools will emerge when we proceed to other inferences. First, though, let us establish the probabilistic interpretation of classification networks, to which the same tools apply.

Binary classification networks

If the targets t in a data set are binary classification labels $(0, 1)$, it is natural to use a neural network whose output $y(\mathbf{x}; \mathbf{w}, \mathcal{A})$ is bounded between 0 and 1, and is interpreted as a probability $P(t=1 \,|\, \mathbf{x}, \mathbf{w}, \mathcal{A})$. For example, a network with one hidden layer could be described by the feedforward equations (44.1) and (44.2), with $f^{(2)}(a) = 1/(1+e^{-a})$. The error function βE_D is replaced by the negative log likelihood:

$$G(\mathbf{w}) = -\left[\sum_n t^{(n)} \ln y(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \ln(1 - y(\mathbf{x}^{(n)}; \mathbf{w}))\right]. \tag{44.9}$$

The total objective function is then $M = G + \alpha E_W$. Note that this includes no parameter β (because there is no Gaussian noise).

Multi-class classification networks

For a multi-class classification problem, we can represent the targets by a vector, $\mathbf{t}$, in which a single element is set to 1, indicating the correct class, and all other elements are set to 0. In this case it is appropriate to use a 'softmax' network having coupled outputs which sum to one and are interpreted as class probabilities $y_i = P(t_i=1 \,|\, \mathbf{x}, \mathbf{w}, \mathcal{A})$. The last part of equation (44.2) is replaced by:

$$y_i = \frac{e^{a_i}}{\sum_{i'} e^{a_{i'}}}. \tag{44.10}$$

The negative log likelihood in this case is

$$G(\mathbf{w}) = -\sum_n \sum_i t_i^{(n)} \ln y_i(\mathbf{x}^{(n)}; \mathbf{w}). \tag{44.11}$$

As in the case of the regression network, the minimization of the objective function $M(\mathbf{w}) = G + \alpha E_W$ corresponds to an inference of the form (44.8). A variety of useful results can be built on this interpretation.

▶ 44.4 Benefits of the Bayesian approach to supervised feedforward neural networks

From the statistical perspective, supervised neural networks are nothing more than nonlinear curve-fitting devices. Curve fitting is not a trivial task however. The effective complexity of an interpolating model is of crucial importance, as illustrated in figure 44.5. Consider a control parameter that influences the complexity of a model, for example a regularization constant α (weight decay parameter). As the control parameter is varied to increase the complexity of the model (descending from figure 44.5a–c and going from left to right across figure 44.5d), the best fit to the *training* data that the model can achieve becomes increasingly good. However, the empirical performance of the model, the *test error*, first decreases then increases again. *An over-complex model overfits the data and generalizes poorly.* This problem may also complicate the choice of architecture in a multilayer perceptron, the radius of the basis functions in a radial basis function network, and the choice of the input variables themselves in any multidimensional regression problem. Finding values for model control parameters that are appropriate for the data is therefore an important and non-trivial problem.

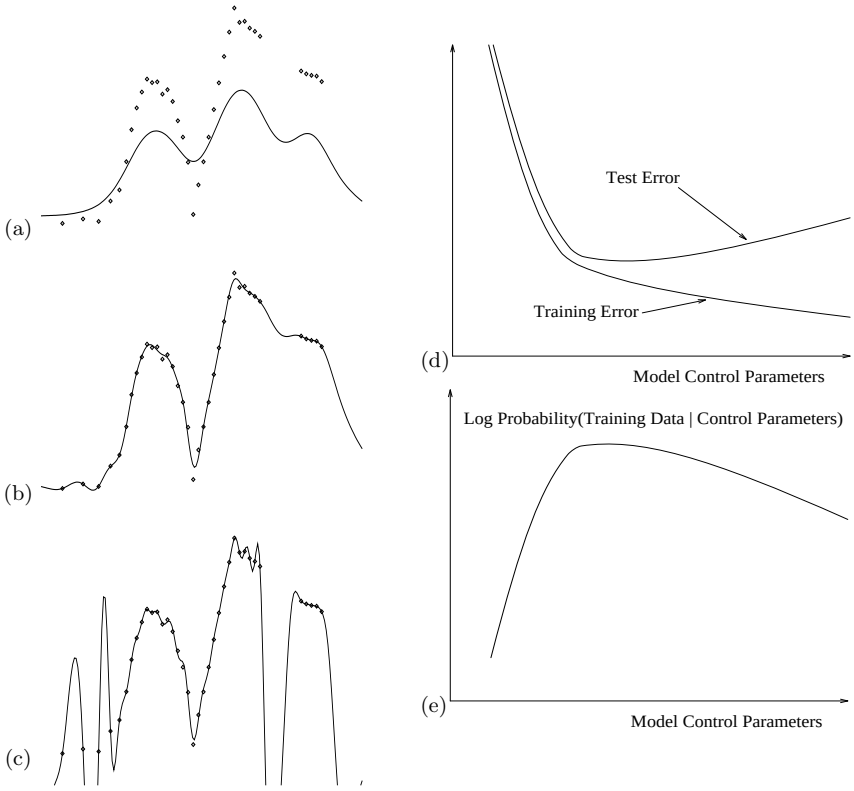


Figure 44.5. Optimization of model complexity. Panels (a–c) show a radial basis function model interpolating a simple data set with one input variable and one output variable. As the regularization constant is varied to increase the complexity of the model (from (a) to (c)), the interpolant is able to fit the training data increasingly well, but beyond a certain point the generalization ability (test error) of the model deteriorates. Probability theory allows us to optimize the control parameters without needing a test set.

The *overfitting problem* can be solved by using a Bayesian approach to control model complexity.

If we give a probabilistic interpretation to the model, then we can evaluate the evidence for alternative values of the control parameters. As was explained in Chapter 28, over-complex models turn out to be less probable, and the evidence $P(\text{Data} \mid \text{Control Parameters})$ can be used as an objective function for optimization of model control parameters (figure 44.5e). The setting of α that maximizes the evidence is displayed in figure 44.5b.

Bayesian optimization of model control parameters has four important advantages. (1) No 'test set' or 'validation set' is involved, so all available training data can be devoted to both model fitting and model comparison. (2) Regularization constants can be optimized on-line, i.e., simultaneously with the optimization of ordinary model parameters. (3) The Bayesian objective function is not noisy, in contrast to a cross-validation measure. (4) The gradient of the evidence with respect to the control parameters can be evaluated, making it possible to simultaneously optimize a large number of control parameters.

Probabilistic modelling also handles *uncertainty* in a natural manner. It offers a unique prescription, *marginalization*, for incorporating uncertainty about parameters into predictions; this procedure yields better predictions, as we saw in Chapter 41. Figure 44.6 shows error bars on the predictions of a trained neural network.

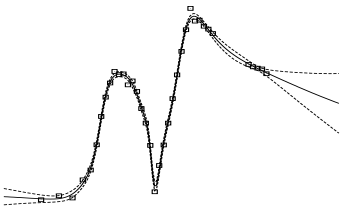


Figure 44.6. Error bars on the predictions of a trained regression network. The solid line gives the predictions of the best-fit parameters of a multilayer perceptron trained on the data points. The error bars (dotted lines) are those produced by the uncertainty of the parameters $\mathbf{w}$. Notice that the error bars become larger where the data are sparse.

Implementation of Bayesian inference

As was mentioned in Chapter 41, Bayesian inference for multilayer networks may be implemented by Monte Carlo sampling, or by deterministic methods employing Gaussian approximations (Neal, 1996; MacKay, 1992c).

Within the Bayesian framework for data modelling, it is easy to improve our probabilistic models. For example, if we believe that some input variables in a problem may be irrelevant to the predicted quantity, but we don't know which, we can define a new model with multiple hyperparameters that captures the idea of uncertain input variable relevance (MacKay, 1994b; Neal, 1996; MacKay, 1995b); these models then infer automatically from the data which are the relevant input variables for a problem.

▶ 44.5 Exercises

Exercise 44.1.[4] How to measure a classifier's quality. You've just written a new classification algorithm and want to measure how well it performs on a test set, and compare it with other classifiers. What performance measure should you use? There are several standard answers. Let's assume the classifier gives an output $y(\mathbf{x})$, where $\mathbf{x}$ is the input, which we won't discuss further, and that the true target value is t. In the simplest discussions of classifiers, both y and t are binary variables, but you might care to consider cases where y and t are more general objects also.

The most widely used measure of performance on a test set is the *error rate* – the fraction of *misclassifications* made by the classifier. This measure forces the classifier to give a 0/1 output and ignores any additional information that the classifier might be able to offer – for example, an indication of the firmness of a prediction. Unfortunately, the error rate does not necessarily measure how *informative* a classifier's output is. Consider frequency tables showing the joint frequency of the 0/1 output of a classifier (horizontal axis), and the true 0/1 variable (vertical axis). The numbers that we'll show are percentages. The error rate e is the sum of the two off-diagonal numbers, which we could call the false positive rate e_+ and the false negative rate e_-.

Of the following three classifiers, A and B have the same error rate of 10% and C has a greater error rate of 12%.

Classifier A

t \ y	0	1
0	90	0
1	10	0

Classifier B

t \ y	0	1
0	80	10
1	0	10

Classifier C

t \ y	0	1
0	78	12
1	0	10

But clearly classifier A, which simply guesses that the outcome is 0 for all cases, is conveying no information at all about t; whereas classifier B has an informative output: if $y = 0$ then we are sure that t really is zero; and if $y = 1$ then there is a 50% chance that $t = 1$, as compared to the prior probability $P(t = 1) = 0.1$. Classifier C is slightly less informative than B, but it is still much more useful than the information-free classifier A.

How common sense ranks the classifiers:
(best) $B > C > A$ (worst).

One way to improve on the error rate as a performance measure is to report the pair (e_+, e_-), the false positive error rate and the false negative error rate, which are $(0, 0.1)$ and $(0.1, 0)$ for classifiers A and B. It is especially important to distinguish between these two error probabilities in applications where the two sorts of error have different associated costs. However, there are a couple of problems with the 'error rate pair':

How error rate ranks the classifiers:
(best) $A = B > C$ (worst).

- First, if I simply told you that classifier A has error rates $(0, 0.1)$ and B has error rates $(0.1, 0)$, it would not be immediately evident that classifier A is actually utterly worthless. Surely we should have a performance measure that gives the worst possible score to A!

- Second, if we turn to a multiple-class classification problem such as digit recognition, then the number of types of error increases from two to $10 \times 9 = 90$ – one for each possible confusion of class t with t'. It would be nice to have some sensible way of collapsing these 90 numbers into a single rankable number that makes more sense than the error rate.

Another reason for not liking the error rate is that it doesn't give a classifier credit for accurately specifying its uncertainty. Consider classifiers that have three outputs available, '0', '1' and a *rejection* class, '?', which indicates that the classifier is not sure. Consider classifiers D and E with the following frequency tables, in percentages:

Classifier D

t \ y	0	?	1
0	74	10	6
1	0	1	9

Classifier E

t \ y	0	?	1
0	78	6	6
1	0	5	5

Both of these classifiers have $(e_+, e_-, r) = (6\%, 0\%, 11\%)$. But are they equally good classifiers? Compare classifier E with C. The two classifiers are equivalent. E is just C in disguise – we could make E by taking the output of C and tossing a coin when C says '1' in order to decide whether to give output '1' or '?'. So E is equal to C and thus inferior to B. Now compare D with B. Can you justify the suggestion that D is a more informative classifier than B, and thus is superior to E? Yet D and E have the same (e_+, e_-, r) scores.

People often plot *error-reject curves* (also known as ROC curves; ROC stands for 'receiver operating characteristic') which show the total $e = (e_+ + e_-)$ versus r as r is allowed to vary from 0 to 1, and use these curves to compare classifiers (figure 44.7). [In the special case of binary classification problems, e_+ may be plotted versus e_- instead.] But as we have seen, error rates can be undiscerning performance measures. Does plotting one error rate as a function of another make this weakness of error rates go away?

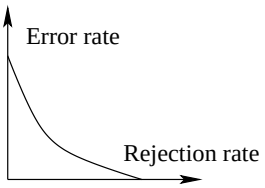


Figure 44.7. An error-reject curve. Some people use the area under this curve as a measure of classifier quality.

For this exercise, either construct an explicit example demonstrating that the error-reject curve, and the area under it, are not necessarily good ways to compare classifiers; or prove that they *are*.

As a suggested alternative method for comparing classifiers, consider the *mutual information* between the output and the target,

$$I(T;Y) \equiv H(T) - H(T \mid Y) = \sum_{y,t} P(y)P(t \mid y) \log \frac{P(t)}{P(t \mid y)}, \tag{44.12}$$

which measures how many *bits* the classifier's output conveys about the target.

Evaluate the mutual information for classifiers A–E above.

Investigate this performance measure and discuss whether it is a useful one. Does it have practical drawbacks?

About Chapter 45

Feedforward neural networks such as multilayer perceptrons are popular tools for nonlinear regression and classification problems. From a Bayesian perspective, a choice of a neural network model can be viewed as defining a prior probability distribution over nonlinear functions, and the neural network's learning process can be interpreted in terms of the posterior probability distribution over the unknown function. (Some learning algorithms search for the function with maximum posterior probability and other Monte Carlo methods draw samples from this posterior probability.)

In the limit of large but otherwise standard networks, Neal (1996) has shown that the prior distribution over nonlinear functions implied by the Bayesian neural network falls in a class of probability distributions known as Gaussian processes. The hyperparameters of the neural network model determine the characteristic lengthscales of the Gaussian process. Neal's observation motivates the idea of discarding parameterized networks and working directly with Gaussian processes. Computations in which the parameters of the network are optimized are then replaced by simple matrix operations using the covariance matrix of the Gaussian process.

In this chapter I will review work on this idea by Williams and Rasmussen (1996), Neal (1997b), Barber and Williams (1997) and Gibbs and MacKay (2000), and will assess whether, for supervised regression and classification tasks, the feedforward network has been superceded.

▷ Exercise 45.1.[3] I regret that this chapter is rather dry. There's no simple explanatory examples in it, and few pictures. This exercise asks you to create interesting pictures to explain to yourself this chapter's ideas.

Source code for computer demonstrations written in the free language `octave` is available at:
`http://www.inference.phy.cam.ac.uk/mackay/itprnn/software.html`.

Radford Neal's software for Gaussian processes is available at:
`http://www.cs.toronto.edu/~radford/`.

45

Gaussian Processes

After the publication of Rumelhart, Hinton and Williams's (1986) paper on supervised learning in neural networks there was a surge of interest in the empirical modelling of relationships in high-dimensional data using nonlinear parametric models such as multilayer perceptrons and radial basis functions. In the Bayesian interpretation of these modelling methods, a nonlinear function $y(\mathbf{x})$ parameterized by parameters $\mathbf{w}$ is assumed to underlie the data $\{\mathbf{x}^{(n)}, t_n\}_{n=1}^N$, and the adaptation of the model to the data corresponds to an *inference* of the function given the data. We will denote the set of input vectors by $\mathbf{X}_N \equiv \{\mathbf{x}^{(n)}\}_{n=1}^N$ and the set of corresponding target values by the vector $\mathbf{t}_N \equiv \{t_n\}_{n=1}^N$. The inference of $y(\mathbf{x})$ is described by the posterior probability distribution

$$P(y(\mathbf{x}) \,|\, \mathbf{t}_N, \mathbf{X}_N) = \frac{P(\mathbf{t}_N \,|\, y(\mathbf{x}), \mathbf{X}_N) P(y(\mathbf{x}))}{P(\mathbf{t}_N \,|\, \mathbf{X}_N)}. \tag{45.1}$$

Of the two terms on the right-hand side, the first, $P(\mathbf{t}_N \,|\, y(\mathbf{x}), \mathbf{X}_N)$, is the probability of the target values given the function $y(\mathbf{x})$, which in the case of regression problems is often assumed to be a separable Gaussian distribution; and the second term, $P(y(\mathbf{x}))$, is the prior distribution on functions assumed by the model. This prior is implicit in the choice of parametric model and the choice of regularizers used during the model fitting. The prior typically specifies that the function $y(\mathbf{x})$ is expected to be continuous and smooth, and has less high frequency power than low frequency power, but the precise meaning of the prior is somewhat obscured by the use of the parametric model.

Now, for the prediction of future values of t, all that matters is the assumed prior $P(y(\mathbf{x}))$ and the assumed noise model $P(\mathbf{t}_N \,|\, y(\mathbf{x}), \mathbf{X}_N)$ – the parameterization of the function $y(\mathbf{x}; \mathbf{w})$ is irrelevant.

The idea of Gaussian process modelling is to place a prior $P(y(\mathbf{x}))$ directly on the space of functions, without parameterizing $y(\mathbf{x})$. The simplest type of prior over functions is called a Gaussian process. It can be thought of as the generalization of a Gaussian distribution over a finite vector space to a function space of infinite dimension. Just as a Gaussian distribution is fully specified by its mean and covariance matrix, a Gaussian process is specified by a mean and a *covariance function*. Here, the mean is a function of $\mathbf{x}$ (which we will often take to be the zero function), and the covariance is a function $C(\mathbf{x}, \mathbf{x}')$ that expresses the expected covariance between the values of the function y at the points $\mathbf{x}$ and $\mathbf{x}'$. The function $y(\mathbf{x})$ in any one data modelling problem is assumed to be a *single* sample from this Gaussian distribution. Gaussian processes are already well established models for various spatial and temporal problems – for example, Brownian motion, Langevin processes and Wiener processes are all examples of Gaussian processes; Kalman filters, widely used

to model speech waveforms, also correspond to Gaussian process models; the method of 'kriging' in geostatistics is a Gaussian process regression method.

Reservations about Gaussian processes

It might be thought that it is not possible to reproduce the interesting properties of neural network interpolation methods with something so simple as a Gaussian distribution, but as we shall now see, many popular nonlinear interpolation methods are equivalent to particular Gaussian processes. (I use the term 'interpolation' to cover both the problem of 'regression' – fitting a curve through noisy data – and the task of fitting an interpolant that passes exactly through the given data points.)

It might also be thought that the computational complexity of inference when we work with priors over infinite-dimensional function spaces might be infinitely large. But by concentrating on the joint probability distribution of the observed data and the quantities we wish to predict, it is possible to make predictions with resources that scale as polynomial functions of N, the number of data points.

▶ 45.1 Standard methods for nonlinear regression

The problem

We are given N data points $\mathbf{X}_N, \mathbf{t}_N = \{\mathbf{x}^{(n)}, t_n\}_{n=1}^N$. The inputs $\mathbf{x}$ are vectors of some fixed input dimension I. The targets t are either real numbers, in which case the task will be a regression or interpolation task, or they are categorical variables, for example $t \in \{0, 1\}$, in which case the task is a classification task. We will concentrate on the case of regression for the time being.

Assuming that a function $y(\mathbf{x})$ underlies the observed data, the task is to infer the function from the given data, and predict the function's value – or the value of the observation t_{N+1} – at a new point $\mathbf{x}^{(N+1)}$.

Parametric approaches to the problem

In a parametric approach to regression we express the unknown function $y(\mathbf{x})$ in terms of a nonlinear function $y(\mathbf{x}; \mathbf{w})$ parameterized by parameters $\mathbf{w}$.

Example 45.2. Fixed basis functions. Using a set of basis functions $\{\phi_h(\mathbf{x})\}_{h=1}^H$, we can write

$$y(\mathbf{x}; \mathbf{w}) = \sum_{h=1}^{H} w_h \phi_h(\mathbf{x}). \tag{45.2}$$

If the basis functions are nonlinear functions of $\mathbf{x}$ such as radial basis functions centred at fixed points $\{\mathbf{c}_h\}_{h=1}^H$,

$$\phi_h(\mathbf{x}) = \exp\left[-\frac{(\mathbf{x} - \mathbf{c}_h)^2}{2r^2}\right], \tag{45.3}$$

then $y(\mathbf{x}; \mathbf{w})$ is a nonlinear function of $\mathbf{x}$; however, since the dependence of y on the parameters $\mathbf{w}$ is linear, we might sometimes refer to this as a 'linear' model. In neural network terms, this model is like a multilayer network whose connections from the input layer to the nonlinear hidden layer are fixed; only the output weights $\mathbf{w}$ are adaptive.

Other possible sets of fixed basis functions include polynomials such as $\phi_h(\mathbf{x}) = x_i^p x_j^q$ where p and q are integer powers that depend on h.

Example 45.3. Adaptive basis functions. Alternatively, we might make a function $y(\mathbf{x})$ from basis functions that depend on additional parameters included in the vector $\mathbf{w}$. In a two-layer feedforward neural network with nonlinear hidden units and a linear output, the function can be written

$$y(\mathbf{x};\mathbf{w}) = \sum_{h=1}^{H} w_h^{(2)} \tanh\left(\sum_{i=1}^{I} w_{hi}^{(1)} x_i + w_{h0}^{(1)}\right) + w_0^{(2)} \tag{45.4}$$

where I is the dimensionality of the input space and the weight vector $\mathbf{w}$ consists of the input weights $\{w_{hi}^{(1)}\}$, the hidden unit biases $\{w_{h0}^{(1)}\}$, the output weights $\{w_h^{(2)}\}$ and the output bias $w_0^{(2)}$. In this model, the dependence of y on $\mathbf{w}$ is nonlinear.

Having chosen the parameterization, we then infer the function $y(\mathbf{x};\mathbf{w})$ by inferring the parameters $\mathbf{w}$. The posterior probability of the parameters is

$$P(\mathbf{w} \mid \mathbf{t}_N, \mathbf{X}_N) = \frac{P(\mathbf{t}_N \mid \mathbf{w}, \mathbf{X}_N) P(\mathbf{w})}{P(\mathbf{t}_N \mid \mathbf{X}_N)}. \tag{45.5}$$

The factor $P(\mathbf{t}_N \mid \mathbf{w}, \mathbf{X}_N)$ states the probability of the observed data points when the parameters $\mathbf{w}$ (and hence, the function y) are known. This probability distribution is often taken to be a separable Gaussian, each data point t_n differing from the underlying value $y(\mathbf{x}^{(n)};\mathbf{w})$ by additive noise. The factor $P(\mathbf{w})$ specifies the prior probability distribution of the parameters. This too is often taken to be a separable Gaussian distribution. If the dependence of y on $\mathbf{w}$ is nonlinear the posterior distribution $P(\mathbf{w} \mid \mathbf{t}_N, \mathbf{X}_N)$ is in general not a Gaussian distribution.

The inference can be implemented in various ways. In the Laplace method, we minimize an objective function

$$M(\mathbf{w}) = -\ln\left[P(\mathbf{t}_N \mid \mathbf{w}, \mathbf{X}_N) P(\mathbf{w})\right] \tag{45.6}$$

with respect to $\mathbf{w}$, locating the locally most probable parameters, then use the curvature of M, $\partial^2 M(\mathbf{w})/\partial w_i \partial w_j$, to define error bars on $\mathbf{w}$. Alternatively we can use more general Markov chain Monte Carlo techniques to create samples from the posterior distribution $P(\mathbf{w} \mid \mathbf{t}_N, \mathbf{X}_N)$.

Having obtained one of these representations of the inference of $\mathbf{w}$ given the data, predictions are then made by marginalizing over the parameters:

$$P(t_{N+1} \mid \mathbf{t}_N, \mathbf{X}_{N+1}) = \int \mathrm{d}^H \mathbf{w}\, P(t_{N+1} \mid \mathbf{w}, \mathbf{x}^{(N+1)}) P(\mathbf{w} \mid \mathbf{t}_N, \mathbf{X}_N). \tag{45.7}$$

If we have a Gaussian representation of the posterior $P(\mathbf{w} \mid \mathbf{t}_N, \mathbf{X}_N)$, then this integral can typically be evaluated directly. In the alternative Monte Carlo approach, which generates R samples $\mathbf{w}^{(r)}$ that are intended to be samples from the posterior distribution $P(\mathbf{w} \mid \mathbf{t}_N, \mathbf{X}_N)$, we approximate the predictive distribution by

$$P(t_{N+1} \mid \mathbf{t}_N, \mathbf{X}_{N+1}) \simeq \frac{1}{R} \sum_{r=1}^{R} P(t_{N+1} \mid \mathbf{w}^{(r)}, \mathbf{x}^{(N+1)}). \tag{45.8}$$

Nonparametric approaches.

In nonparametric methods, predictions are obtained without explicitly parameterizing the unknown function $y(\mathbf{x})$; $y(\mathbf{x})$ lives in the infinite-dimensional space of all continuous functions of $\mathbf{x}$. One well known nonparametric approach to the regression problem is the spline smoothing method (Kimeldorf and Wahba, 1970). A spline solution to a one-dimensional regression problem can be described as follows: we define the estimator of $y(\mathbf{x})$ to be the function $\hat{y}(\mathbf{x})$ that minimizes the functional

$$M(y(x)) = \frac{1}{2}\beta \sum_{n=1}^{N} (y(x^{(n)}) - t_n)^2 + \frac{1}{2}\alpha \int \mathrm{d}x \, [y^{(p)}(x)]^2, \tag{45.9}$$

where $y^{(p)}$ is the pth derivative of y and p is a positive number. If p is set to 2 then the resulting function $\hat{y}(\mathbf{x})$ is a cubic spline, that is, a piecewise cubic function that has 'knots' – discontinuities in its second derivative – at the data points $\{x^{(n)}\}$.

This estimation method can be interpreted as a Bayesian method by identifying the prior for the function $y(x)$ as:

$$\ln P(y(x) \,|\, \alpha) = -\frac{1}{2}\alpha \int \mathrm{d}x \, [y^{(p)}(x)]^2 + \text{const}, \tag{45.10}$$

and the probability of the data measurements $\mathbf{t}_N = \{t_n\}_{n=1}^N$ assuming independent Gaussian noise as:

$$\ln P\left(\mathbf{t}_N \mid y(x), \beta\right) = -\frac{1}{2}\beta \sum_{n=1}^{N} (y(x^{(n)}) - t_n)^2 + \text{const}. \tag{45.11}$$

[The constants in equations (45.10) and (45.11) are functions of α and β respectively. Strictly the prior (45.10) is improper since addition of an arbitrary polynomial of degree $(p-1)$ to $y(x)$ is not constrained. This impropriety is easily rectified by the addition of $(p-1)$ appropriate terms to (45.10).] Given this interpretation of the functions in equation (45.9), $M(y(x))$ is equal to minus the log of the posterior probability $P(y(x) \,|\, \mathbf{t}_N, \alpha, \beta)$, within an additive constant, and the splines estimation procedure can be interpreted as yielding a Bayesian MAP estimate. The Bayesian perspective allows us additionally to put error bars on the splines estimate and to draw typical samples from the posterior distribution, and it gives an automatic method for inferring the hyperparameters α and β.

Comments

Splines priors are Gaussian processes

The prior distribution defined in equation (45.10) is our first example of a Gaussian process. Throwing mathematical precision to the winds, a Gaussian process can be defined as a probability distribution on a space of functions $y(x)$ that can be written in the form

$$P(y(x) \,|\, \mu(x), A) = \frac{1}{Z} \exp\left[-\frac{1}{2}(y(x) - \mu(x))^{\mathsf{T}} A (y(x) - \mu(x))\right], \tag{45.12}$$

where $\mu(x)$ is the mean function and A is a linear operator, and where the inner product of two functions $y(x)^{\mathsf{T}} z(x)$ is defined by, for example, $\int \mathrm{d}x \, y(x) z(x)$.

Here, if we denote by D the linear operator that maps $y(x)$ to the derivative of $y(x)$, we can write equation (45.10) as

$$\ln P(y(x)\,|\,\alpha) = -\frac{1}{2}\,\alpha \int \mathrm{d}x\,[D^p y(x)]^2 + \text{const} = -\frac{1}{2}\,y(x)^{\mathsf{T}} A y(x) + \text{const}, \tag{45.13}$$

which has the same form as equation (45.12) with $\mu(x) = 0$, and $A \equiv [D^p]^{\mathsf{T}} D^p$.

In order for the prior in equation (45.12) to be a proper prior, A must be a positive definite operator, i.e., one satisfying $y(x)^{\mathsf{T}} A y(x) > 0$ for all functions $y(x)$ other than $y(x) = 0$.

Splines can be written as parametric models

Splines may be written in terms of an infinite set of fixed basis functions, as in equation (45.2), as follows. First rescale the x axis so that the interval $(0, 2\pi)$ is much wider than the range of x values of interest. Let the basis functions be a Fourier set $\{\cos hx, \sin hx,\ h = 0, 1, 2, \ldots\}$, so the function is

$$y(x) = \sum_{h=0}^{\infty} w_{h(\cos)} \cos(hx) + \sum_{h=1}^{\infty} w_{h(\sin)} \sin(hx). \tag{45.14}$$

Use the regularizer

$$E_W(\mathbf{w}) = \sum_{h=0}^{\infty} \frac{1}{2} h^{\frac{p}{2}} w_{h(\cos)}^2 + \sum_{h=1}^{\infty} \frac{1}{2} h^{\frac{p}{2}} w_{h(\sin)}^2 \tag{45.15}$$

to define a Gaussian prior on $\mathbf{w}$,

$$P(\mathbf{w}\,|\,\alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{45.16}$$

If $p = 2$ then we have the cubic splines regularizer $E_W(\mathbf{w}) = \int y^{(2)}(x)^2\,\mathrm{d}x$, as in equation (45.9); if $p = 1$ we have the regularizer $E_W(\mathbf{w}) = \int y^{(1)}(x)^2\,\mathrm{d}x$, etc. (To make the prior proper we must add an extra regularizer on the term $w_{0(\cos)}$.) Thus in terms of the prior $P(y(x))$ there is no fundamental difference between the 'nonparametric' splines approach and other parametric approaches.

Representation is irrelevant for prediction

From the point of view of prediction at least, there are two objects of interest. The first is the conditional distribution $P(t_{N+1}\,|\,\mathbf{t}_N, \mathbf{X}_{N+1})$ defined in equation (45.7). The other object of interest, should we wish to compare one model with others, is the joint probability of all the observed data given the model, the evidence $P(\mathbf{t}_N\,|\,\mathbf{X}_N)$, which appeared as the normalizing constant in equation (45.5). Neither of these quantities makes any reference to the representation of the unknown function $y(x)$. So at the end of the day, our choice of representation is irrelevant.

The question we now address is, in the case of popular parametric models, what form do these two quantities take? We will see that for standard models with fixed basis functions and Gaussian distributions on the unknown parameters, the joint probability of all the observed data given the model, $P(\mathbf{t}_N\,|\,\mathbf{X}_N)$, is a multivariate Gaussian distribution with mean zero and with a covariance matrix determined by the basis functions; this implies that the conditional distribution $P(t_{N+1}\,|\,\mathbf{t}_N, \mathbf{X}_{N+1})$ is also a Gaussian distribution, whose mean depends linearly on the values of the targets $\mathbf{t}_N$. Standard parametric models are simple examples of Gaussian processes.

▶ 45.2 From parametric models to Gaussian processes

Linear models

Let us consider a regression problem using H fixed basis functions, for example one-dimensional radial basis functions as defined in equation (45.3).

Let us assume that a list of N input points $\{\mathbf{x}^{(n)}\}$ has been specified and define the $N \times H$ matrix $\mathbf{R}$ to be the matrix of values of the basis functions $\{\phi_h(\mathbf{x})\}_{h=1}^{H}$ at the points $\{\mathbf{x}_n\}$,

$$R_{nh} \equiv \phi_h(\mathbf{x}^{(n)}). \tag{45.17}$$

We define the vector $\mathbf{y}_N$ to be the vector of values of $y(\mathbf{x})$ at the N points,

$$y_n \equiv \sum_h R_{nh} w_h. \tag{45.18}$$

If the prior distribution of $\mathbf{w}$ is Gaussian with zero mean,

$$P(\mathbf{w}) = \text{Normal}(\mathbf{w}; \mathbf{0}, \sigma_w^2 \mathbf{I}), \tag{45.19}$$

then $\mathbf{y}$, being a linear function of $\mathbf{w}$, is also Gaussian distributed, with mean zero. The covariance matrix of $\mathbf{y}$ is

$$\begin{aligned} \mathbf{Q} &= \langle \mathbf{y}\mathbf{y}^{\mathsf{T}} \rangle = \langle \mathbf{R}\mathbf{w}\mathbf{w}^{\mathsf{T}}\mathbf{R}^{\mathsf{T}} \rangle = \mathbf{R} \langle \mathbf{w}\mathbf{w}^{\mathsf{T}} \rangle \mathbf{R}^{\mathsf{T}} & (45.20) \\ &= \sigma_w^2 \mathbf{R}\mathbf{R}^{\mathsf{T}}. & (45.21) \end{aligned}$$

So the prior distribution of $\mathbf{y}$ is:

$$P(\mathbf{y}) = \text{Normal}(\mathbf{y}; \mathbf{0}, \mathbf{Q}) = \text{Normal}(\mathbf{y}; \mathbf{0}, \sigma_w^2 \mathbf{R}\mathbf{R}^{\mathsf{T}}). \tag{45.22}$$

This result, that the vector of N function values $\mathbf{y}$ has a Gaussian distribution, is true for any selected points $\mathbf{X}_N$. This is the defining property of a Gaussian process. *The probability distribution of a function $y(\mathbf{x})$ is a Gaussian process if for any finite selection of points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$, the density $P(y(\mathbf{x}^{(1)}), y(\mathbf{x}^{(2)}), \ldots, y(\mathbf{x}^{(N)}))$ is a Gaussian.*

Now, if the number of basis functions H is smaller than the number of data points N, then the matrix $\mathbf{Q}$ will not have full rank. In this case the probability distribution of $\mathbf{y}$ might be thought of as a flat elliptical pancake confined to an H-dimensional subspace in the N-dimensional space in which $\mathbf{y}$ lives.

What about the target values? If each target t_n is assumed to differ by additive Gaussian noise of variance σ_ν^2 from the corresponding function value y_n then $\mathbf{t}$ also has a Gaussian prior distribution,

$$P(\mathbf{t}) = \text{Normal}(\mathbf{t}; \mathbf{0}, \mathbf{Q} + \sigma_\nu^2 \mathbf{I}). \tag{45.23}$$

We will denote the covariance matrix of $\mathbf{t}$ by $\mathbf{C}$:

$$\mathbf{C} = \mathbf{Q} + \sigma_\nu^2 \mathbf{I} = \sigma_w^2 \mathbf{R}\mathbf{R}^{\mathsf{T}} + \sigma_\nu^2 \mathbf{I}. \tag{45.24}$$

Whether or not $\mathbf{Q}$ has full rank, the covariance matrix $\mathbf{C}$ has full rank since $\sigma_\nu^2 \mathbf{I}$ is full rank.

What does the covariance matrix $\mathbf{Q}$ look like? In general, the (n, n') entry of $\mathbf{Q}$ is

$$Q_{nn'} = [\sigma_w^2 \mathbf{R}\mathbf{R}^{\mathsf{T}}]_{nn'} = \sigma_w^2 \sum_h \phi_h(\mathbf{x}^{(n)}) \phi_h(\mathbf{x}^{(n')}) \tag{45.25}$$

and the (n, n') entry of $\mathbf{C}$ is

$$C_{nn'} = \sigma_w^2 \sum_h \phi_h(\mathbf{x}^{(n)})\phi_h(\mathbf{x}^{(n')}) + \delta_{nn'}\sigma_\nu^2, \tag{45.26}$$

where $\delta_{nn'} = 1$ if $n = n'$ and 0 otherwise.

Example 45.4. Let's take as an example a one-dimensional case, with radial basis functions. The expression for $Q_{nn'}$ becomes simplest if we assume we have uniformly-spaced basis functions with the basis function labelled h centred on the point $x = h$, and take the limit $H \to \infty$, so that the sum over h becomes an integral; to avoid having a covariance that diverges with H, we had better make σ_w^2 scale as $S/(\Delta H)$, where ΔH is the number of basis functions per unit length of the x-axis, and S is a constant; then

$$Q_{nn'} = S \int_{h_{\min}}^{h_{\max}} \mathrm{d}h \; \phi_h(x^{(n)})\phi_h(x^{(n')}) \tag{45.27}$$

$$= S \int_{h_{\min}}^{h_{\max}} \mathrm{d}h \; \exp\left[-\frac{(x^{(n)} - h)^2}{2r^2}\right] \exp\left[-\frac{(x^{(n')} - h)^2}{2r^2}\right]. \tag{45.28}$$

If we let the limits of integration be $\pm\infty$, we can solve this integral:

$$Q_{nn'} = \sqrt{\pi r^2}\, S \exp\left[-\frac{(x^{(n')} - x^{(n)})^2}{4r^2}\right]. \tag{45.29}$$

We are arriving at a new perspective on the interpolation problem. Instead of specifying the prior distribution on functions in terms of basis functions and priors on parameters, the prior can be summarized simply by a covariance function,

$$C(x^{(n)}, x^{(n')}) \equiv \theta_1 \exp\left[-\frac{(x^{(n')} - x^{(n)})^2}{4r^2}\right], \tag{45.30}$$

where we have given a new name, θ_1, to the constant out front.

Generalizing from this particular case, a vista of interpolation methods opens up. Given any valid covariance function $C(\mathbf{x}, \mathbf{x}')$ – we'll discuss in a moment what 'valid' means – we can define the covariance matrix for N function values at locations $\mathbf{X}_N$ to be the matrix $\mathbf{Q}$ given by

$$Q_{nn'} = C(\mathbf{x}^{(n)}, \mathbf{x}^{(n')}) \tag{45.31}$$

and the covariance matrix for N corresponding target values, assuming Gaussian noise, to be the matrix $\mathbf{C}$ given by

$$C_{nn'} = C(\mathbf{x}^{(n)}, \mathbf{x}^{(n')}) + \sigma_\nu^2 \delta_{nn'}. \tag{45.32}$$

In conclusion, the prior probability of the N target values $\mathbf{t}$ in the data set is:

$$P(\mathbf{t}) = \mathrm{Normal}(\mathbf{t}; \mathbf{0}, \mathbf{C}) = \tfrac{1}{Z} e^{-\frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{t}}. \tag{45.33}$$

Samples from this Gaussian process and a few other simple Gaussian processes are displayed in figure 45.1.

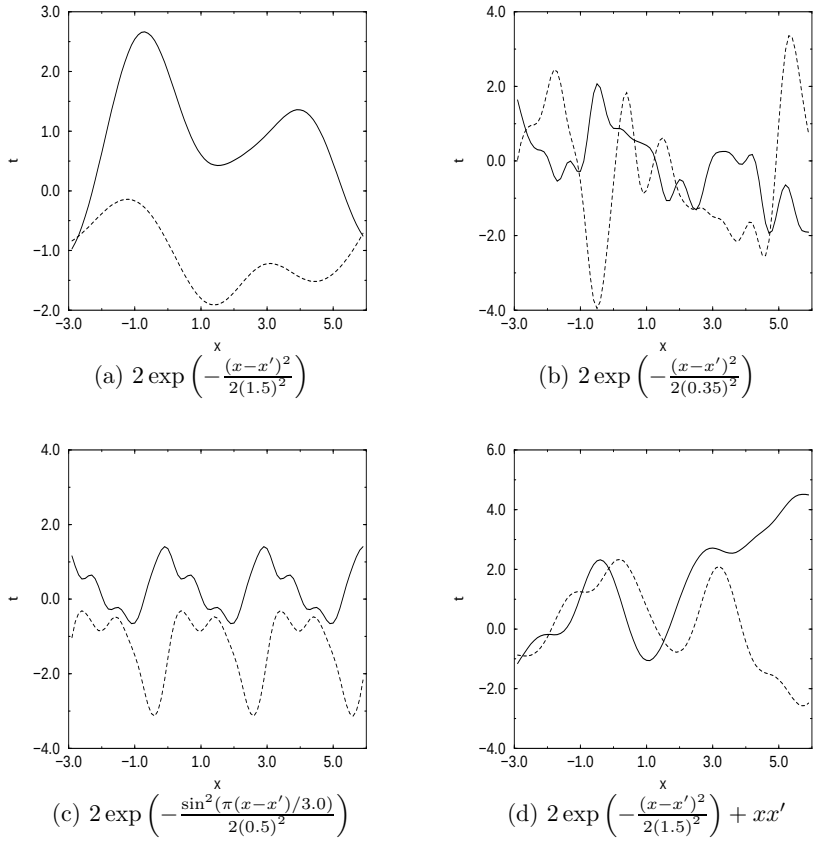


Figure 45.1. Samples drawn from Gaussian process priors. Each panel shows two functions drawn from a Gaussian process prior. The four corresponding covariance functions are given below each plot. The decrease in lengthscale from (a) to (b) produces more rapidly fluctuating functions. The periodic properties of the covariance function in (c) can be seen. The covariance function in (d) contains the non-stationary term xx' corresponding to the covariance of a straight line, so that typical functions include linear trends. From Gibbs (1997).

Multilayer neural networks and Gaussian processes

Figures 44.2 and 44.3 show some random samples from the prior distribution over functions defined by a selection of standard multilayer perceptrons with large numbers of hidden units. Those samples don't seem a million miles away from the Gaussian process samples of figure 45.1. And indeed Neal (1996) showed that the properties of a neural network with one hidden layer (as in equation (45.4)) converge to those of a Gaussian process as the number of hidden neurons tends to infinity, if standard 'weight decay' priors are assumed. The covariance function of this Gaussian process depends on the details of the priors assumed for the weights in the network and the activation functions of the hidden units.

▶ 45.3 Using a given Gaussian process model in regression

We have spent some time talking about priors. We now return to our data and the problem of prediction. How do we make predictions with a Gaussian process?

Having formed the covariance matrix $\mathbf{C}$ defined in equation (45.32) our task is to infer t_{N+1} given the observed vector $\mathbf{t}_N$. The joint density $P(t_{N+1}, \mathbf{t}_N)$ is a Gaussian; so the conditional distribution

$$P(t_{N+1} \,|\, \mathbf{t}_N) = \frac{P(t_{N+1}, \mathbf{t}_N)}{P(\mathbf{t}_N)} \tag{45.34}$$

is also a Gaussian. We now distinguish between different sizes of covariance matrix $\mathbf{C}$ with a subscript, such that $\mathbf{C}_{N+1}$ is the $(N+1)\times(N+1)$ covariance

matrix for the vector $\mathbf{t}_{N+1} \equiv (t_1, \ldots, t_{N+1})^{\mathsf{T}}$. We define submatrices of $\mathbf{C}_{N+1}$ as follows:

$$\mathbf{C}_{N+1} \equiv \begin{bmatrix} \begin{bmatrix} & \mathbf{C}_N & \end{bmatrix} & \begin{bmatrix} \\ \mathbf{k} \\ \\ \end{bmatrix} \\ \begin{bmatrix} & \mathbf{k}^{\mathsf{T}} & \end{bmatrix} & \begin{bmatrix} \kappa \end{bmatrix} \end{bmatrix}. \tag{45.35}$$

The posterior distribution (45.34) is given by

$$P(t_{N+1} \,|\, \mathbf{t}_N) \propto \exp\left[-\frac{1}{2}\left[\mathbf{t}_N \; t_{N+1}\right] \mathbf{C}_{N+1}^{-1} \begin{bmatrix} \mathbf{t}_N \\ t_{N+1} \end{bmatrix}\right]. \tag{45.36}$$

We can evaluate the mean and standard deviation of the posterior distribution of t_{N+1} by brute-force inversion of $\mathbf{C}_{N+1}$. There is a more elegant expression for the predictive distribution, however, which is useful whenever predictions are to be made at a number of new points on the basis of the data set of size N. We can write $\mathbf{C}_{N+1}^{-1}$ in terms of $\mathbf{C}_N$ and $\mathbf{C}_N^{-1}$ using the partitioned inverse equations (Barnett, 1979):

$$\mathbf{C}_{N+1}^{-1} = \begin{bmatrix} \mathbf{M} & \mathbf{m} \\ \mathbf{m}^{\mathsf{T}} & m \end{bmatrix} \tag{45.37}$$

where

$$m = \left(\kappa - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}\right)^{-1} \tag{45.38}$$

$$\mathbf{m} = -m\, \mathbf{C}_N^{-1} \mathbf{k} \tag{45.39}$$

$$\mathbf{M} = \mathbf{C}_N^{-1} + \frac{1}{m} \mathbf{m}\mathbf{m}^T. \tag{45.40}$$

When we substitute this matrix into equation (45.36) we find

$$P(t_{N+1} \,|\, \mathbf{t}_N) = \frac{1}{Z} \exp\left[-\frac{(t_{N+1} - \hat{t}_{N+1})^2}{2\sigma^2_{\hat{t}_{N+1}}}\right] \tag{45.41}$$

where

$$\hat{t}_{N+1} = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}_N \tag{45.42}$$

$$\sigma^2_{\hat{t}_{N+1}} = \kappa - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}. \tag{45.43}$$

The predictive mean at the new point is given by $\hat{t}_{N+1}$ and $\sigma_{\hat{t}_{N+1}}$ defines the error bars on this prediction. Notice that we do not need to invert $\mathbf{C}_{N+1}$ in order to make predictions at $\mathbf{x}^{(N+1)}$. Only $\mathbf{C}_N$ needs to be inverted. Thus Gaussian processes allow one to implement a model with a number of basis functions H much larger than the number of data points N, with the computational requirement being of order N^3, independent of H. [We'll discuss ways of reducing this cost later.]

The predictions produced by a Gaussian process depend entirely on the covariance matrix $\mathbf{C}$. We now discuss the sorts of covariance functions one might choose to define $\mathbf{C}$, and how we can automate the selection of the covariance function in response to data.

▶ 45.4 Examples of covariance functions

The only constraint on our choice of covariance function is that it must generate a non-negative-definite covariance matrix for any set of points $\{\mathbf{x}_n\}_{n=1}^N$.

We will denote the parameters of a covariance function by $\boldsymbol{\theta}$. The covariance matrix of $\mathbf{t}$ has entries given by

$$C_{mn} = C(\mathbf{x}^{(m)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) + \delta_{mn}\mathcal{N}(\mathbf{x}^{(n)}; \boldsymbol{\theta}) \tag{45.44}$$

where C is the covariance function and $\mathcal{N}$ is a noise model which might be stationary or spatially varying, for example,

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\theta}) = \begin{cases} \theta_3 & \text{for input-independent noise} \\ \exp\left(\sum_{j=1}^{J} \beta_j \phi_j(\mathbf{x})\right) & \text{for input-dependent noise.} \end{cases} \tag{45.45}$$

The continuity properties of C determine the continuity properties of typical samples from the Gaussian process prior. An encyclopaedic paper on Gaussian processes giving many valid covariance functions has been written by Abrahamsen (1997).

Stationary covariance functions

A *stationary* covariance function is one that is translation invariant in that it satisfies

$$C(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = D(\mathbf{x} - \mathbf{x}'; \boldsymbol{\theta}) \tag{45.46}$$

for some function D, i.e., the covariance is a function of separation only, also known as the autocovariance function. If additionally C depends only on the *magnitude* of the distance between $\mathbf{x}$ and $\mathbf{x}'$ then the covariance function is said to be *homogeneous*. Stationary covariance functions may also be described in terms of the Fourier transform of the function D, which is known as the power spectrum of the Gaussian process. This Fourier transform is necessarily a positive function of frequency. One way of constructing a valid stationary covariance function is to invent a positive function of frequency and define D to be its inverse Fourier transform.

Example 45.5. Let the power spectrum be a Gaussian function of frequency. Since the Fourier transform of a Gaussian is a Gaussian, the autocovariance function corresponding to this power spectrum is a Gaussian function of separation. This argument rederives the covariance function we derived at equation (45.30).

Generalizing slightly, a popular form for C with hyperparameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \{r_i\})$ is

$$C(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \theta_1 \exp\left[-\frac{1}{2}\sum_{i=1}^{I} \frac{(x_i - x_i')^2}{r_i^2}\right] + \theta_2. \tag{45.47}$$

$\mathbf{x}$ is an I-dimensional vector and r_i is a lengthscale associated with input x_i, the lengthscale in direction i on which y is expected to vary significantly. A very large lengthscale means that y is expected to be essentially a constant function of that input. Such an input could be said to be irrelevant, as in the automatic relevance determination method for neural networks (MacKay, 1994a; Neal, 1996). The θ_1 hyperparameter defines the vertical scale of variations of a typical function. The θ_2 hyperparameter allows the whole function to be offset away from zero by some unknown constant – to understand this term, examine equation (45.25) and consider the basis function $\phi(\mathbf{x}) = 1$.

Another stationary covariance function is

$$C(x, x') = \exp(-|x - x'|^{\nu}) \quad 0 < \nu \leq 2. \tag{45.48}$$

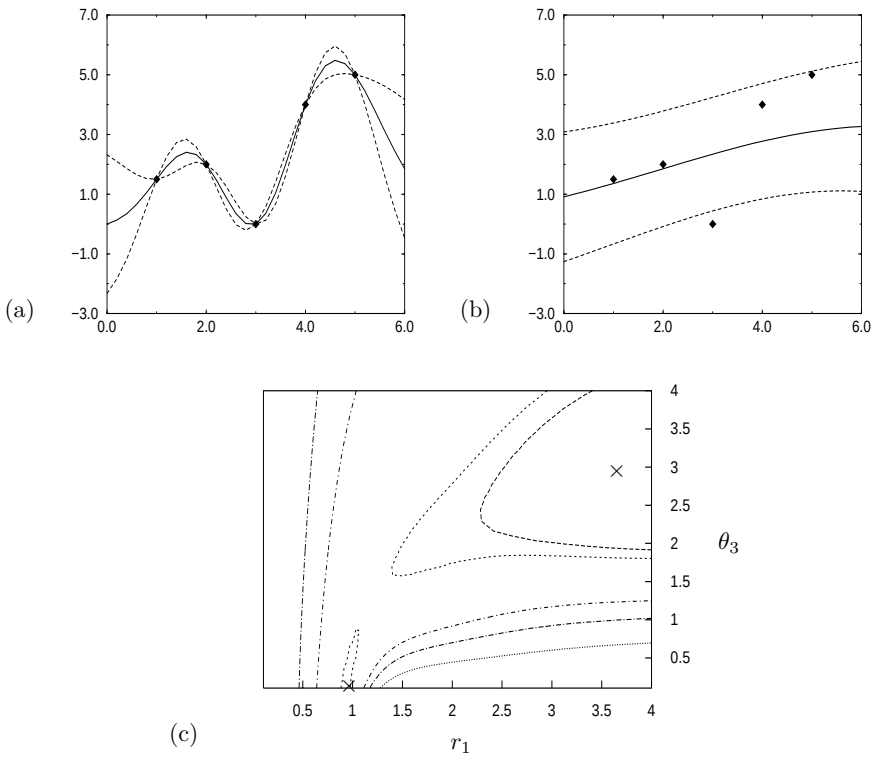


Figure 45.2. Multimodal likelihood functions for Gaussian processes. A data set of five points is modelled with the simple covariance function (45.47), with one hyperparameter θ_3 controlling the noise variance. Panels a and b show the most probable interpolant and its 1σ error bars when the hyperparameters $\boldsymbol{\theta}$ are set to two different values that (locally) maximize the likelihood $P(\mathbf{t}_N \mid \mathbf{X}_N, \boldsymbol{\theta})$: (a) $r_1 = 0.95$, $\theta_3 = 0.0$; (b) $r_1 = 3.5$, $\theta_3 = 3.0$. Panel c shows a contour plot of the likelihood as a function of r_1 and θ_3, with the two maxima shown by crosses. From Gibbs (1997).

For $\nu = 2$, this is a special case of the previous covariance function. For $\nu \in (1, 2)$, the typical functions from this prior are smooth but not analytic functions. For $\nu \leq 1$ typical functions are continuous but not smooth.

A covariance function that models a function that is periodic with known period λ_i in the i^{th} input direction is

$$C(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \theta_1 \exp\left[-\frac{1}{2} \sum_i \left(\frac{\sin\left(\frac{\pi}{\lambda_i}(x_i - x_i')\right)}{r_i} \right)^2 \right]. \tag{45.49}$$

Figure 45.1 shows some random samples drawn from Gaussian processes with a variety of different covariance functions.

Nonstationary covariance functions

The simplest nonstationary covariance function is the one corresponding to a linear trend. Consider the plane $y(\mathbf{x}) = \sum_i w_i x_i + c$. If the $\{w_i\}$ and c have Gaussian distributions with zero mean and variances σ_w^2 and σ_c^2 respectively then the plane has a covariance function

$$C_{\text{lin}}(\mathbf{x}, \mathbf{x}'; \{\sigma_w, \sigma_c\}) = \sum_{i=1}^{I} \sigma_w^2 x_i x_i' + \sigma_c^2. \tag{45.50}$$

An example of random sample functions incorporating the linear term can be seen in figure 45.1d.

▶ 45.5 Adaptation of Gaussian process models

Let us assume that a form of covariance function has been chosen, but that it depends on undetermined hyperparameters $\boldsymbol{\theta}$. We would like to 'learn' these

hyperparameters from the data. This learning process is equivalent to the inference of the hyperparameters of a neural network, for example, weight decay hyperparameters. It is a complexity-control problem, one that is solved nicely by the Bayesian Occam's razor.

Ideally we would like to define a prior distribution on the hyperparameters and integrate over them in order to make our predictions, i.e., we would like to find

$$P(t_{N+1} \mid \mathbf{x}_{N+1}, \mathcal{D}) = \int P(t_{N+1} \mid \mathbf{x}_{N+1}, \boldsymbol{\theta}, \mathcal{D}) P(\boldsymbol{\theta} \mid \mathcal{D}) \, \mathrm{d}\boldsymbol{\theta}. \tag{45.51}$$

But this integral is usually intractable. There are two approaches we can take.

1. We can approximate the integral by using the most probable values of hyperparameters.

$$P(t_{N+1} \mid \mathbf{x}_{N+1}, \mathcal{D}) \simeq P(t_{N+1} \mid \mathbf{x}_{N+1}, \mathcal{D}, \boldsymbol{\theta}_{\mathrm{MP}}) \tag{45.52}$$

2. Or we can perform the integration over $\boldsymbol{\theta}$ numerically using Monte Carlo methods (Williams and Rasmussen, 1996; Neal, 1997b).

Either of these approaches is implemented most efficiently if the gradient of the posterior probability of $\boldsymbol{\theta}$ can be evaluated.

Gradient

The posterior probability of $\boldsymbol{\theta}$ is

$$P(\boldsymbol{\theta} \mid \mathcal{D}) \propto P(\mathbf{t}_N \mid \mathbf{X}_N, \boldsymbol{\theta}) P(\boldsymbol{\theta}). \tag{45.53}$$

The log of the first term (the evidence for the hyperparameters) is

$$\ln P(\mathbf{t}_N \mid \mathbf{X}_N, \boldsymbol{\theta}) = -\frac{1}{2} \ln \det \mathbf{C}_N - \frac{1}{2} \mathbf{t}_N^T \mathbf{C}_N^{-1} \mathbf{t}_N - \frac{N}{2} \ln 2\pi, \tag{45.54}$$

and its derivative with respect to a hyperparameter θ is

$$\frac{\partial}{\partial \theta} \ln P(\mathbf{t}_N \mid \mathbf{X}_N, \boldsymbol{\theta}) = -\frac{1}{2} \mathrm{Trace} \left(\mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta} \right) + \frac{1}{2} \mathbf{t}_N^T \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta} \mathbf{C}_N^{-1} \mathbf{t}_N. \tag{45.55}$$

Comments

Assuming that finding the derivatives of the priors is straightforward, we can now search for $\boldsymbol{\theta}_{\mathrm{MP}}$. However there are two problems that we need to be aware of. Firstly, as illustrated in figure 45.2, the evidence may be multimodal. Suitable priors and sensible optimization strategies often eliminate poor optima. Secondly and perhaps most importantly the evaluation of the gradient of the log likelihood requires the evaluation of $\mathbf{C}_N^{-1}$. Any exact inversion method (such as Cholesky decomposition, LU decomposition or Gauss–Jordan elimination) has an associated computational cost that is of order N^3 and so calculating gradients becomes time consuming for large training data sets. Approximate methods for implementing the predictions (equations (45.42) and (45.43)) and gradient computation (equation (45.55)) are an active research area. One approach based on the ideas of Skilling (1993) makes approximations to $\mathbf{C}^{-1}\mathbf{t}$ and $\mathrm{Trace}\, \mathbf{C}^{-1}$ using iterative methods with cost $\mathcal{O}(N^2)$ (Gibbs and MacKay, 1996; Gibbs, 1997). Further references on this topic are given at the end of the chapter.

▶ 45.6 Classification

Gaussian processes can be integrated into classification modelling once we identify a variable that can sensibly be given a Gaussian process prior.

In a binary classification problem, we can define a quantity $a_n \equiv a(\mathbf{x}^{(n)})$ such that the probability that the class is 1 rather than 0 is

$$P(t_n = 1 \,|\, a_n) = \frac{1}{1 + e^{-a_n}}. \tag{45.56}$$

Large positive values of a correspond to probabilities close to one; large negative values of a define probabilities that are close to zero. In a classification problem, we typically intend that the probability $P(t_n = 1)$ should be a smoothly varying function of $\mathbf{x}$. We can embody this prior belief by defining $a(\mathbf{x})$ to have a Gaussian process prior.

Implementation

It is not so easy to perform inferences and adapt the Gaussian process model to data in a classification model as in regression problems because the likelihood function (45.56) is not a Gaussian function of a_n. So the posterior distribution of $\mathbf{a}$ given some observations $\mathbf{t}$ is not Gaussian and the normalization constant $P(\mathbf{t}_N \,|\, \mathbf{X}_N)$ cannot be written down analytically. Barber and Williams (1997) have implemented classifiers based on Gaussian process priors using Laplace approximations (Chapter 27). Neal (1997b) has implemented a Monte Carlo approach to implementing a Gaussian process classifier. Gibbs and MacKay (2000) have implemented another cheap and cheerful approach based on the methods of Jaakkola and Jordan (section 33.8). In this *variational Gaussian process classifier*, we obtain tractable upper and lower bounds for the unnormalized posterior density over $\mathbf{a}$, $P(\mathbf{t}_N \,|\, \mathbf{a})P(\mathbf{a})$. These bounds are parameterized by variational parameters which are adjusted in order to obtain the tightest possible fit. Using normalized versions of the optimized bounds we then compute approximations to the predictive distributions.

Multi-class classification problems can also be solved with Monte Carlo methods (Neal, 1997b) and variational methods (Gibbs, 1997).

▶ 45.7 Discussion

Gaussian processes are moderately simple to implement and use. Because very few parameters of the model need to be determined by hand (generally only the priors on the hyperparameters), Gaussian processes are useful tools for automated tasks where fine tuning for each problem is not possible. We do not appear to sacrifice any performance for this simplicity.

It is easy to construct Gaussian processes that have particular desired properties; for example we can make a straightforward automatic relevance determination model.

One obvious problem with Gaussian processes is the computational cost associated with inverting an $N \times N$ matrix. The cost of direct methods of inversion becomes prohibitive when the number of data points N is greater than about 1000.

Have we thrown the baby out with the bath water?

According to the hype of 1987, neural networks were meant to be intelligent models that discovered features and patterns in data. Gaussian processes in

contrast are simply smoothing devices. How can Gaussian processes possibly replace neural networks? Were neural networks over-hyped, or have we underestimated the power of smoothing methods?

I think both these propositions are true. The success of Gaussian processes shows that many real-world data modelling problems are perfectly well solved by sensible smoothing methods. The most interesting problems, the task of feature discovery for example, are not ones that Gaussian processes will solve. But maybe multilayer perceptrons can't solve them either. Perhaps a fresh start is needed, approaching the problem of machine learning from a paradigm different from the supervised feedforward mapping.

Further reading

The study of Gaussian processes for regression is far from new. Time series analysis was being performed by the astronomer T.N. Thiele using Gaussian processes in 1880 (Lauritzen, 1981). In the 1940s, Wiener–Kolmogorov prediction theory was introduced for prediction of trajectories of military targets (Wiener, 1948). Within the geostatistics field, Matheron (1963) proposed a framework for regression using optimal linear estimators which he called 'kriging' after D.G. Krige, a South African mining engineer. This framework is identical to the Gaussian process approach to regression. Kriging has been developed considerably in the last thirty years (see Cressie (1993) for a review) including several Bayesian treatments (Omre, 1987; Kitanidis, 1986). However the geostatistics approach to the Gaussian process model has concentrated mainly on low-dimensional problems and has largely ignored any probabilistic interpretation of the model. Kalman filters are widely used to implement inferences for stationary one-dimensional Gaussian processes, and are popular models for speech and music modelling (Bar-Shalom and Fortmann, 1988). Generalized radial basis functions (Poggio and Girosi, 1989), ARMA models (Wahba, 1990) and variable metric kernel methods (Lowe, 1995) are all closely related to Gaussian processes. See also O'Hagan (1978).

The idea of replacing supervised neural networks by Gaussian processes was first explored by Williams and Rasmussen (1996) and Neal (1997b). A thorough comparison of Gaussian processes with other methods such as neural networks and MARS was made by Rasmussen (1996). Methods for reducing the complexity of data modelling with Gaussian processes remain an active research area (Poggio and Girosi, 1990; Luo and Wahba, 1997; Tresp, 2000; Williams and Seeger, 2001; Smola and Bartlett, 2001; Rasmussen, 2002; Seeger *et al.*, 2003; Opper and Winther, 2000).

A longer review of Gaussian processes is in (MacKay, 1998b). A review paper on regression with complexity control using hierarchical Bayesian models is (MacKay, 1992a).

Gaussian processes and *support vector learning machines* (Scholkopf *et al.*, 1995; Vapnik, 1995) have a lot in common. Both are kernel-based predictors, the kernel being another name for the covariance function. A Bayesian version of support vectors, exploiting this connection, can be found in (Chu *et al.*, 2001; Chu *et al.*, 2002; Chu *et al.*, 2003b; Chu *et al.*, 2003a).

46

Deconvolution

▶ 46.1 Traditional image reconstruction methods

Optimal linear filters

In many imaging problems, the data measurements $\{d_n\}$ are linearly related to the underlying image $\mathbf{f}$:

$$d_n = \sum_k R_{nk} f_k + n_n. \tag{46.1}$$

The vector $\mathbf{n}$ denotes the inevitable noise that corrupts real data. In the case of a camera which produces a blurred picture, the vector $\mathbf{f}$ denotes the true image, $\mathbf{d}$ denotes the blurred and noisy picture, and the linear operator $\mathbf{R}$ is a convolution defined by the point spread function of the camera. In this special case, the true image and the data vector reside in the same space; but it is important to maintain a distinction between them. We will use the subscript $n = 1, \ldots, N$ to run over data measurements, and the subscripts $k, k' = 1, \ldots, K$ to run over image pixels.

One might speculate that since the blur was created by a linear operation, then perhaps it might be deblurred by another linear operation. We can derive the *optimal linear filter* in two ways.

Bayesian derivation

We assume that the linear operator $\mathbf{R}$ is known, and that the noise $\mathbf{n}$ is Gaussian and independent, with a known standard deviation σ_ν.

$$P(\mathbf{d} \,|\, \mathbf{f}, \sigma_\nu, \mathcal{H}) = \frac{1}{(2\pi\sigma_\nu^2)^{N/2}} \exp\left(- \sum_n \left(d_n - \textstyle\sum_k R_{nk} f_k\right)^2 \Big/ \left(2\sigma_\nu^2\right) \right). \tag{46.2}$$

We assume that the prior probability of the image is also Gaussian, with a scale parameter σ_f.

$$P(\mathbf{f} \,|\, \sigma_f, \mathcal{H}) = \frac{\det^{-\frac{1}{2}} \mathbf{C}}{(2\pi\sigma_f^2)^{K/2}} \exp\left(- \sum_{k,k'} f_k C_{kk'} f'_k \Big/ \left(2\sigma_f^2\right) \right). \tag{46.3}$$

If we assume no correlations among the pixels then the symmetric, full rank matrix $\mathbf{C}$ is equal to the identity matrix $\mathbf{I}$. The more sophisticated 'intrinsic correlation function' model uses $\mathbf{C} = [\mathbf{G}\mathbf{G}^{\mathsf{T}}]^{-1}$, where $\mathbf{G}$ is a convolution that takes us from an imaginary 'hidden' image, which is uncorrelated, to the real correlated image. The intrinsic correlation function should not be confused with the point spread function $\mathbf{R}$ which defines the image-to-data mapping.

A zero-mean Gaussian prior is clearly a poor assumption if it is known that all elements of the image $\mathbf{f}$ are positive, but let us proceed. We can now write down the posterior probability of an image $\mathbf{f}$ given the data $\mathbf{d}$.

$$P(\mathbf{f} \,|\, \mathbf{d}, \sigma_\nu, \sigma_f, \mathcal{H}) = \frac{P(\mathbf{d} \,|\, \mathbf{f}, \sigma_\nu, \mathcal{H}) P(\mathbf{f} \,|\, \sigma_f, \mathcal{H}))}{P(\mathbf{d} \,|\, \sigma_\nu, \sigma_f, \mathcal{H})}. \tag{46.4}$$

In words,

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}. \tag{46.5}$$

The 'evidence' $P(\mathbf{d} \,|\, \sigma_\nu, \sigma_f, \mathcal{H})$ is the normalizing constant for this posterior distribution. Here it is unimportant, but it is used in a more sophisticated analysis to compare, for example, different values of σ_ν and σ_f, or different point spread functions $\mathbf{R}$.

Since the posterior distribution is the product of two Gaussian functions of $\mathbf{f}$, it is also a Gaussian, and can therefore be summarized by its mean, which is also the *most probable image*, $\mathbf{f}_{\rm MP}$, and its covariance matrix:

$$\mathbf{\Sigma}_{\mathbf{f}|\mathbf{d}} \equiv \left[-\nabla\nabla \log P(\mathbf{f} \,|\, \mathbf{d}, \sigma_\nu, \sigma_f, \mathcal{H}) \right]^{-1}, \tag{46.6}$$

which defines the joint error bars on $\mathbf{f}$. In this equation, the symbol ∇ denotes differentiation with respect to the image parameters $\mathbf{f}$. We can find $\mathbf{f}_{\rm MP}$ by differentiating the log of the posterior, and solving for the derivative being zero. We obtain:

$$\mathbf{f}_{\rm MP} = \left[\mathbf{R}^{\mathsf{T}}\mathbf{R} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C} \right]^{-1} \mathbf{R}^{\mathsf{T}}\mathbf{d}. \tag{46.7}$$

The operator $\left[\mathbf{R}^{\mathsf{T}}\mathbf{R} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C} \right]^{-1} \mathbf{R}^{\mathsf{T}}$ is called the optimal linear filter. When the term $\frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}$ can be neglected, the optimal linear filter is the pseudoinverse $\left[\mathbf{R}^{\mathsf{T}}\mathbf{R}\right]^{-1}\mathbf{R}^{\mathsf{T}}$. The term $\frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{C}$ regularizes this ill-conditioned inverse.

The optimal linear filter can also be manipulated into the form:

$$\text{Optimal linear filter} = \mathbf{C}^{-1}\mathbf{R}^{\mathsf{T}} \left[\mathbf{R}\mathbf{C}^{-1}\mathbf{R}^{\mathsf{T}} + \frac{\sigma_\nu^2}{\sigma_f^2}\mathbf{I} \right]^{-1}. \tag{46.8}$$

Minimum square error derivation

The non-Bayesian derivation of the optimal linear filter starts by assuming that we will 'estimate' the true image $\mathbf{f}$ by a linear function of the data:

$$\hat{\mathbf{f}} = \mathbf{W}\mathbf{d}. \tag{46.9}$$

The linear operator $\mathbf{W}$ is then 'optimized' by minimizing the expected sum-squared error between $\hat{\mathbf{f}}$ and the unknown true image . In the following equations, summations over repeated indices k, k', n are implicit. The expectation $\langle \cdot \rangle$ is over both the statistics of the random variables $\{n_n\}$, and the ensemble of images $\mathbf{f}$ which we expect to bump into. We assume that the noise is zero mean and uncorrelated to second order with itself and everything else, with $\langle n_n n_{n'} \rangle = \sigma_\nu^2 \delta_{nn'}$.

$$\langle E \rangle = \frac{1}{2} \left\langle (W_{kn} d_n - f_k)^2 \right\rangle \tag{46.10}$$

$$= \frac{1}{2} \left\langle (W_{kn} R_{nj} f_j - f_k)^2 \right\rangle + \frac{1}{2} W_{kn} W_{kn} \sigma_\nu^2. \tag{46.11}$$

Differentiating with respect to $\mathbf{W}$, and introducing $\mathbf{F} \equiv \left\langle f_{j'} f_j \right\rangle$ (cf. $\sigma_f^2 \mathbf{C}^{-1}$ in the Bayesian derivation above), we find that the optimal linear filter is:

$$\mathbf{W}_{\rm opt} = \mathbf{F}\mathbf{R}^{\mathsf{T}} \left[\mathbf{R}\mathbf{F}\mathbf{R}^{\mathsf{T}} + \sigma_\nu^2 \mathbf{I} \right]^{-1} . \tag{46.12}$$

If we identify $\mathbf{F} = \sigma_f^2 \mathbf{C}^{-1}$, we obtain the optimal linear filter (46.8) of the Bayesian derivation. The ad hoc assumptions made in this derivation were the choice of a quadratic error measure, and the decision to use a linear estimator. It is interesting that without explicit assumptions of Gaussian distributions, this derivation has reproduced the same estimator as the Bayesian posterior mode, $\mathbf{f}_{\rm MP}$.

The advantage of a Bayesian approach is that we can criticize these assumptions and modify them in order to make better reconstructions.

Other image models

The better matched our model of images $P(\mathbf{f} \,|\, \mathcal{H})$ is to the real world, the better our image reconstructions will be, and the less data we will need to answer any given question. The Gaussian models which lead to the optimal linear filter are spectacularly poorly matched to the real world. For example, the Gaussian prior (46.3) fails to specify that all pixel intensities in an image are positive. This omission leads to the most pronounced artefacts where the image under observation has high contrast or large black patches. Optimal linear filters applied to astronomical data give reconstructions with negative areas in them, corresponding to patches of sky that suck energy out of telescopes! The *maximum entropy* model for image deconvolution (Gull and Daniell, 1978) was a great success principally because this model forced the reconstructed image to be positive. The spurious negative areas and complementary spurious positive areas are eliminated, and the quality of the reconstruction is greatly enhanced.

The 'classic maximum entropy' model assigns an entropic prior

$$P(\mathbf{f} \,|\, \alpha, \mathbf{m}, \mathcal{H}_{\rm Classic}) = \exp(\alpha S(\mathbf{f}, \mathbf{m}))/Z, \tag{46.13}$$

where

$$S(\mathbf{f}, \mathbf{m}) = \sum_i (f_i \ln(m_i/f_i) + f_i - m_i) \tag{46.14}$$

(Skilling, 1989). This model enforces positivity; the parameter α defines a characteristic dynamic range by which the pixel values are expected to differ from the default image $\mathbf{m}$.

The 'intrinsic-correlation-function maximum-entropy' model (Gull, 1989) introduces an expectation of spatial correlations into the prior on $\mathbf{f}$ by writing $\mathbf{f} = \mathbf{G}\mathbf{h}$, where $\mathbf{G}$ is a convolution with an intrinsic correlation function, and putting a classic maxent prior on the underlying hidden image $\mathbf{h}$.

Probabilistic movies

Having found not only the most probable image $\mathbf{f}_{\rm MP}$ but also error bars on it, $\mathbf{\Sigma}_{\mathbf{f}|\mathbf{d}}$, one task is to visualize those error bars. Whether or not we use Monte Carlo methods to infer $\mathbf{f}$, a correlated random walk around the posterior distribution can be used to visualize the uncertainties and correlations. For a Gaussian posterior distribution, we can create a correlated sequence of unit normal random vectors $\mathbf{n}$ using

$$\mathbf{n}^{(t+1)} = c\mathbf{n}^{(t)} + s\mathbf{z}, \tag{46.15}$$

where $\mathbf{z}$ is a unit normal random vector and $c^2 + s^2 = 1$ (c controls how persistent the memory of the sequence is). We then render the image sequence defined by

$$\mathbf{f}^{(t)} = \mathbf{f}_{\mathrm{MP}} + \mathbf{\Sigma}_{\mathbf{f}|\mathbf{d}}^{1/2}\mathbf{n}^{(t)} \tag{46.16}$$

where $\mathbf{\Sigma}_{\mathbf{f}|\mathbf{d}}^{1/2}$ is the Cholesky decomposition of $\mathbf{\Sigma}_{\mathbf{f}|\mathbf{d}}$.

▶ 46.2 Supervised neural networks for image deconvolution

Neural network researchers often exploit the following strategy. Given a problem currently solved with a standard algorithm: interpret the computations performed by the algorithm as a parameterized mapping from an input to an output, and call this mapping a neural network; then adapt the parameters to data so as to produce another mapping that solves the task better. By construction, the neural network can reproduce the standard algorithm, so this data-driven adaptation can only make the performance better.

There are several reasons why standard algorithms can be bettered in this way.

1. Algorithms are often not designed to optimize the real objective function. For example, in speech recognition, a hidden Markov model is designed to model the speech signal, and is fitted so as to to maximize the generative probability given the known string of words in the training data; but the real objective is to *discriminate* between different words. If an inadequate model is being used, the neural-net-style training of the model will focus the limited resources of the model on the aspects relevant to the discrimination task. Discriminative training of hidden Markov models for speech recognition does improve their performance.

2. The neural network can be more flexible than the standard model; some of the adaptive parameters might have been viewed as fixed features by the original designers. A flexible network can find properties in the data that were not included in the original model.

▶ 46.3 Deconvolution in humans

A huge fraction of our brain is devoted to vision. One of the neglected features of our visual system is that the raw image falling on the retina is severely blurred: while most people can see with a resolution of about *1 arcminute* (one sixtieth of a degree) under any daylight conditions, bright or dim, *the image on our retina is blurred through a point spread function of width as large as 5 arcminutes* (Wald and Griffin, 1947; Howarth and Bradley, 1986). It is amazing that we are able to resolve pixels that are twenty-five times smaller in area than the blob produced on our retina by any point source.

Isaac Newton was aware of this conundrum. It's hard to make a lens that does not have chromatic aberration, and our cornea and lens, like a lens made of ordinary glass, refract blue light more strongly than red. Typically our eyes focus correctly for the middle of the visible spectrum (green), so if we look at a single white dot made of red, green, and blue light, the image on our retina consists of a sharply focussed green dot surrounded by a broader red blob superposed on an even broader blue blob. The width of the red and blue blobs is proportional to the diameter of the pupil, which is largest under dim lighting conditions. [The blobs are roughly concentric, though most people have a slight bias, such that in one eye the red blob is centred a tiny distance

to the left and the blue is centred a tiny distance to the right, and in the other eye it's the other way round. This slight bias explains why when we look at blue and red writing on a dark background most people perceive the blue writing to be at a slightly greater depth than the red. In a minority of people, this small bias is the other way round and the red/blue depth perception is reversed. But this effect (which many people are aware of, having noticed it in cinemas, for example) is *tiny* compared with the chromatic aberration we are discussing.]

You can vividly demonstrate to yourself how enormous the chromatic aberration in your eye is with the help of a sheet of card and a colour computer screen.

For the most impressive results – I guarantee you will be amazed – use a dim room with no light apart from the computer screen; a pretty strong effect will still be seen even if the room has daylight coming into it, as long as it is not bright sunshine. Cut a slit about 1.5 mm wide in the card. On the screen, display a few small coloured objects on a black background. I especially recommend thin vertical objects coloured pure red, pure blue, magenta (i.e., red plus blue), and white (red plus blue plus green).[1] Include a little black-and-white text on the screen too. Stand or sit sufficiently far away that you can only just read the text – perhaps a distance of four metres or so, if you have normal vision. Now, hold the slit vertically in front of one of your eyes, and close the other eye. Hold the slit near to your eye – brushing your eyelashes – and look through it. Waggle the slit slowly to the left and to the right, so that the slit is alternately in front of the left and right sides of your pupil. What do you see? I see the red objects waggling to and fro, and the blue objects waggling to and fro, through *huge* distances and in opposite directions, while white objects appear to stay still and are negligibly distorted. Thin magenta objects can be seen splitting into their constituent red and blue parts. Measure how large the motion of the red and blue objects is – it's more than 5 minutes of arc for me, in a dim room. Then check how sharply you can see under these conditions – look at the text on the screen, for example: is it not the case that you can see (through your whole pupil) features far smaller than the distance through which the red and blue components were waggling? Yet when you are using the whole pupil, what is falling on your retina must be an image blurred with a blurring diameter equal to the waggling amplitude.

One of the main functions of early visual processing must be to deconvolve this chromatic aberration. Neuroscientists sometimes conjecture that the reason why retinal ganglion cells and cells in the lateral geniculate nucleus (the main brain area to which retinal ganglion cells project) have centre-surround receptive fields with colour opponency (long wavelength in the centre and medium wavelength in the surround, for example) is in order to perform 'feature extraction' or 'edge detection', but I think this view is mistaken. The reason we have centre-surround filters at the first stage of visual processing (in the fovea at least) is for the huge task of deconvolution of chromatic aberration.

I speculate that the *McCollough effect*, an extremely long-lasting association of colours with orientation (McCollough, 1965; MacKay and MacKay, 1974), is produced by the adaptation mechanism that tunes our chromatic-aberration-deconvolution circuits. Our deconvolution circuits need to be rapidly tuneable, because the point spread function of our eye changes with our pupil diameter, which can change within seconds; and indeed the McCollough effect can be induced within 30 seconds. At the same time, the effect is long-lasting

[1] `http://www.inference.phy.cam.ac.uk/mackay/itila/Files.html`

when an eye is covered, because it's in our interests that our deconvolution circuits should stay well-tuned while we sleep, so that we can see sharply the instant we wake up.

I also wonder whether the main reason that we evolved colour vision was not 'in order to see fruit better' but *'so as to be able to see black and white sharper'* – deconvolving chromatic aberration is easier, even in an entirely black and white world, if one has access to chromatic information in the image.

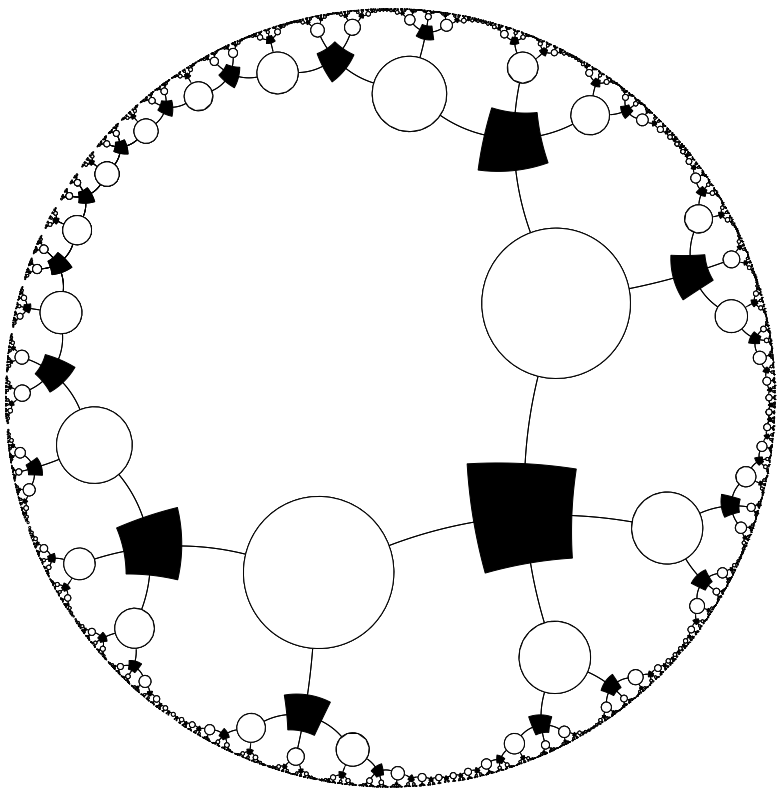
And a final speculation: why do our eyes make micro-saccades when we look at things? These miniature eye-movements are of an angular size bigger than the spacing between the cones in the fovea (which are spaced at roughly 1 minute of arc, the perceived resolution of the eye). The typical size of a microsaccade is 5–10 minutes of arc (Ratliff and Riggs, 1950). Is it a coincidence that this is the same as the size of chromatic aberration? Surely micro-saccades must play an essential role in the deconvolution mechanism that delivers our high-resolution vision.

▶ 46.4 Exercises

Exercise 46.1.[3C] Blur an image with a circular (top hat) point spread function and add noise. Then deconvolve the blurry noisy image using the optimal linear filter. Find error bars and visualize them by making a probabilistic movie.

Part VI

Sparse Graph Codes



About Part VI

The central problem of communication theory is to construct an encoding and a decoding system that make it possible to communicate reliably over a noisy channel. During the 1990s, remarkable progress was made towards the Shannon limit, using codes that are defined in terms of sparse random graphs, and which are decoded by a simple probability-based message-passing algorithm.

In a *sparse-graph code*, the nodes in the graph represent the transmitted bits and the constraints they satisfy. For a linear code with a codeword length N and rate $R = K/N$, the number of constraints is of order $M = N - K$. Any linear code can be described by a graph, but what makes a sparse-graph code special is that each constraint involves only a small number of variables in the graph: so the number of edges in the graph scales roughly linearly with N, rather than quadratically.

In the following four chapters we will look at four families of sparse-graph codes: three families that are excellent for error-correction: *low-density parity-check codes*, *turbo codes*, and *repeat–accumulate codes*; and the family of *digital fountain codes*, which are outstanding for erasure-correction.

All these codes can be decoded by a local message-passing algorithm on the graph, the sum–product algorithm, and, while this algorithm is not a perfect maximum likelihood decoder, the empirical results are record-breaking.

47

Low-Density Parity-Check Codes

A low-density parity-check code (or Gallager code) is a block code that has a parity-check matrix, $\mathbf{H}$, every row and column of which is 'sparse'.

A *regular* Gallager code is a low-density parity-check code in which every column of $\mathbf{H}$ has the same weight j and every row has the same weight k; regular Gallager codes are constructed at random subject to these constraints. A low-density parity-check code with $j = 3$ and $k = 4$ is illustrated in figure 47.1.

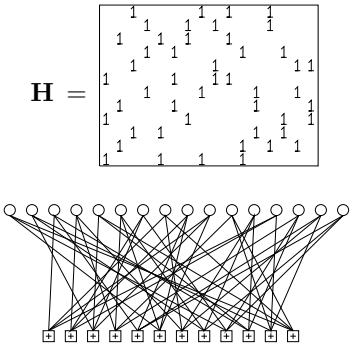


Figure 47.1. A low-density parity-check matrix and the corresponding graph of a rate-$1/4$ low-density parity-check code with blocklength $N = 16$, and $M = 12$ constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by $\boxplus$ squares. Each constraint forces the sum of the $k = 4$ bits to which it is connected to be even.

▶ 47.1 Theoretical properties

Low-density parity-check codes lend themselves to theoretical study. The following results are proved in Gallager (1963) and MacKay (1999b).

Low-density parity-check codes, in spite of their simple construction, are good codes, *given an optimal decoder* (good codes in the sense of section 11.4). Furthermore, they have good distance (in the sense of section 13.2). These two results hold for any column weight $j \geq 3$. Furthermore, there are sequences of low-density parity-check codes in which j increases gradually with N, in such a way that the ratio j/N still goes to zero, that are *very good*, and that have very good distance.

However, we don't have an optimal decoder, and decoding low-density parity-check codes is an NP-complete problem. So what can we do in practice?

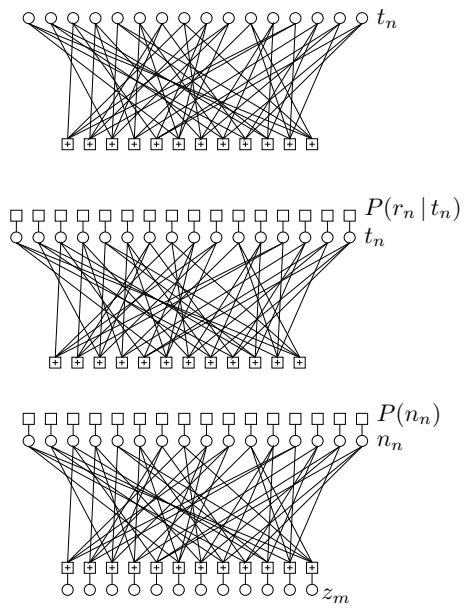
▶ 47.2 Practical decoding

Given a channel output $\mathbf{r}$, we wish to find the codeword $\mathbf{t}$ whose likelihood $P(\mathbf{r} \mid \mathbf{t})$ is biggest. All the effective decoding strategies for low-density parity-check codes are message-passing algorithms. The best algorithm known is the sum–product algorithm, also known as iterative probabilistic decoding or belief propagation.

We'll assume that the channel is a memoryless channel (though more complex channels can easily be handled by running the sum–product algorithm on a more complex graph that represents the expected correlations among the errors (Worthen and Stark, 1998)). For any memoryless channel, there are two approaches to the decoding problem, both of which lead to the generic problem 'find the $\mathbf{x}$ that maximizes

$$P^*(\mathbf{x}) = P(\mathbf{x})\, \mathbb{1}[\mathbf{H}\mathbf{x} = \mathbf{z}]', \tag{47.1}$$

where $P(\mathbf{x})$ is a separable distribution on a binary vector $\mathbf{x}$, and $\mathbf{z}$ is another binary vector. Each of these two approaches represents the decoding problem in terms of a factor graph (Chapter 26).



(a) The prior distribution over codewords

$$P(\mathbf{t}) \propto \mathbb{1}[\mathbf{H}\mathbf{t} = \mathbf{0}].$$

The variable nodes are the transmitted bits $\{t_n\}$.
Each $\boxplus$ node represents the factor $\mathbb{1}[\sum_{n \in \mathcal{N}(m)} t_n = 0 \bmod 2]$.

(b) The posterior distribution over codewords,

$$P(\mathbf{t} \mid \mathbf{r}) \propto P(\mathbf{t}) P(\mathbf{r} \mid \mathbf{t}).$$

Each upper function node represents a likelihood factor $P(r_n \mid t_n)$.

(c) The joint probability of the noise $\mathbf{n}$ and syndrome $\mathbf{z}$,

$$P(\mathbf{n}, \mathbf{z}) = P(\mathbf{n}) \, \mathbb{1}[\mathbf{z} = \mathbf{H}\mathbf{n}].$$

The top variable nodes are now the noise bits $\{n_n\}$.
The added variable nodes at the base are the syndrome values $\{z_m\}$.
Each definition $z_m = \sum_n H_{mn} n_n \bmod 2$ is enforced by a $\boxplus$ factor.

Figure 47.2. Factor graphs associated with a low-density parity-check code.

The codeword decoding viewpoint

First, we note that the prior distribution over codewords,

$$P(\mathbf{t}) \propto \mathbb{1}[\mathbf{H}\mathbf{t} = \mathbf{0} \bmod 2], \tag{47.2}$$

can be represented by a factor graph (figure 47.2a), with the factorization being

$$P(\mathbf{t}) \propto \prod_m \mathbb{1}[\sum_{n \in \mathcal{N}(m)} t_n = 0 \bmod 2]. \tag{47.3}$$

(We'll omit the 'mod 2's from now on.) The posterior distribution over codewords is given by multiplying this prior by the likelihood, which introduces another N factors, one for each received bit.

$$\begin{aligned} P(\mathbf{t} \mid \mathbf{r}) &\propto P(\mathbf{t}) P(\mathbf{r} \mid \mathbf{t}) \\ &\propto \prod_m \mathbb{1}[\sum_{n \in \mathcal{N}(m)} t_n = 0] \prod_n P(r_n \mid t_n) \end{aligned} \tag{47.4}$$

The factor graph corresponding to this function is shown in figure 47.2b. It is the same as the graph for the prior, except for the addition of likelihood 'dongles' to the transmitted bits.

In this viewpoint, the received signal r_n can live in any alphabet; all that matters are the values of $P(r_n \mid t_n)$.

The syndrome decoding viewpoint

Alternatively, we can view the channel output in terms of a binary received vector $\mathbf{r}$ and a noise vector $\mathbf{n}$, with a probability distribution $P(\mathbf{n})$ that can be derived from the channel properties and whatever additional information is available at the channel outputs.

For example, with a binary symmetric channel, we define the noise by $\mathbf{r} = \mathbf{t} + \mathbf{n}$, the syndrome $\mathbf{z} = \mathbf{H}\mathbf{r}$, and noise model $P(n_n = 1) = f$. For other channels such as the Gaussian channel with output $\mathbf{y}$, we may define a received

binary vector $\mathbf{r}$ however we wish and obtain an effective binary noise model $P(\mathbf{n})$ from $\mathbf{y}$ (exercises 9.18 (p.155) and 25.1 (p.325)).

The joint probability of the noise $\mathbf{n}$ and syndrome $\mathbf{z} = \mathbf{Hn}$ can be factored as

$$\begin{aligned} P(\mathbf{n}, \mathbf{z}) &= P(\mathbf{n})\, \mathbb{1}[\mathbf{z} = \mathbf{Hn}] \\ &= \prod_n P(n_n) \prod_m \mathbb{1}[z_m = \sum_{n \in \mathcal{N}(m)} n_n\,]. \end{aligned} \tag{47.5}$$

The factor graph of this function is shown in figure 47.2c. The variables $\mathbf{n}$ and $\mathbf{z}$ can also be drawn in a 'belief network' (also known as a 'Bayesian network', 'causal network', or 'influence diagram') similar to figure 47.2a, but with arrows on the edges from the upper circular nodes (which represent the variables $\mathbf{n}$) to the lower square nodes (which now represent the variables $\mathbf{z}$). We can say that every bit x_n is the parent of j checks z_m, and each check z_m is the child of k bits.

Both decoding viewpoints involve essentially the same graph. Either version of the decoding problem can be expressed as the generic decoding problem 'find the $\mathbf{x}$ that maximizes

$$P^*(\mathbf{x}) = P(\mathbf{x})\, \mathbb{1}[\mathbf{Hx} = \mathbf{z}]'; \tag{47.6}$$

in the codeword decoding viewpoint, $\mathbf{x}$ is the codeword $\mathbf{t}$, and $\mathbf{z}$ is 0; in the syndrome decoding viewpoint, $\mathbf{x}$ is the noise $\mathbf{n}$, and $\mathbf{z}$ is the syndrome.

It doesn't matter which viewpoint we take when we apply the sum–product algorithm. The two decoding algorithms are isomorphic and will give equivalent outcomes (unless numerical errors intervene).

> I tend to use the syndrome decoding viewpoint because it has one advantage: one does not need to implement an *encoder* for a code in order to be able to simulate a decoding problem realistically.

We'll now talk in terms of the generic decoding problem.

▶ 47.3 Decoding with the sum–product algorithm

We aim, given the observed checks, to compute the marginal posterior probabilities $P(x_n = 1 \,|\, \mathbf{z}, \mathbf{H})$ for each n. It is hard to compute these exactly because the graph contains many cycles. However, it is interesting to implement the decoding algorithm that would be appropriate if there were no cycles, on the assumption that the errors introduced might be relatively small. This approach of ignoring cycles has been used in the artificial intelligence literature but is now frowned upon because it produces inaccurate probabilities. However, if we are decoding a good error-correcting code, we don't care about accurate marginal probabilities – we just want the correct codeword. Also, the posterior probability, in the case of a good code communicating at an achievable rate, is expected typically to be hugely concentrated on the most probable decoding; so we are dealing with a distinctive probability distribution to which experience gained in other fields may not apply.

The sum–product algorithm was presented in Chapter 26. We now write out explicitly how it works for solving the decoding problem

$$\mathbf{Hx} = \mathbf{z} \pmod 2.$$

For brevity, we reabsorb the dongles hanging off the x and z nodes in figure 47.2c and modify the sum–product algorithm accordingly. The graph in

which $\mathbf{x}$ and $\mathbf{z}$ live is then the original graph (figure 47.2a) whose edges are defined by the 1s in $\mathbf{H}$. The graph contains nodes of two types, which we'll call checks and bits. The graph connecting the checks and bits is a bipartite graph: bits connect only to checks, and *vice versa*. On each iteration, a probability ratio is propagated along each edge in the graph, and each bit node x_n updates its probability that it should be in state 1.

We denote the set of bits n that participate in check m by $\mathcal{N}(m) \equiv \{n : H_{mn} = 1\}$. Similarly we define the set of checks in which bit n participates, $\mathcal{M}(n) \equiv \{m : H_{mn} = 1\}$. We denote a set $\mathcal{N}(m)$ with bit n excluded by $\mathcal{N}(m)\backslash n$. The algorithm has two alternating parts, in which quantities q_{mn} and r_{mn} associated with each edge in the graph are iteratively updated. The quantity q^x_{mn} is meant to be the probability that bit n of $\mathbf{x}$ has the value x, given the information obtained via checks other than check m. The quantity r^x_{mn} is meant to be the probability of check m being satisfied if bit n of $\mathbf{x}$ is considered fixed at x and the other bits have a separable distribution given by the probabilities $\{q_{mn'} : n' \in \mathcal{N}(m)\backslash n\}$. The algorithm would produce the exact posterior probabilities of all the bits after a fixed number of iterations if the bipartite graph defined by the matrix $\mathbf{H}$ contained no cycles.

Initialization. Let $p^0_n = P(x_n = 0)$ (the prior probability that bit x_n is 0), and let $p^1_n = P(x_n = 1) = 1 - p^0_n$. If we are taking the syndrome decoding viewpoint and the channel is a binary symmetric channel then p^1_n will equal f. If the noise level varies in a known way (for example if the channel is a binary-input Gaussian channel with a real output) then p^1_n is initialized to the appropriate normalized likelihood. For every (n, m) such that $H_{mn} = 1$ the variables q^0_{mn} and q^1_{mn} are initialized to the values p^0_n and p^1_n respectively.

Horizontal step. In the *horizontal* step of the algorithm (horizontal from the point of view of the matrix $\mathbf{H}$), we run through the checks m and compute for each $n \in \mathcal{N}(m)$ two probabilities: first, r^0_{mn}, the probability of the observed value of z_m arising when $x_n = 0$, given that the other bits $\{x_{n'} : n' \neq n\}$ have a separable distribution given by the probabilities $\{q^0_{mn'}, q^1_{mn'}\}$, defined by:

$$r^0_{mn} = \sum_{\{x_{n'}:\, n' \in \mathcal{N}(m)\backslash n\}} P\left(z_m \,|\, x_n = 0,\, \{x_{n'} :\, n' \in \mathcal{N}(m)\backslash n\}\right) \prod_{n' \in \mathcal{N}(m)\backslash n} q^{x_{n'}}_{mn'} \tag{47.7}$$

and second, r^1_{mn}, the probability of the observed value of z_m arising when $x_n = 1$, defined by:

$$r^1_{mn} = \sum_{\{x_{n'}:\, n' \in \mathcal{N}(m)\backslash n\}} P\left(z_m \,|\, x_n = 1,\, \{x_{n'} :\, n' \in \mathcal{N}(m)\backslash n\}\right) \prod_{n' \in \mathcal{N}(m)\backslash n} q^{x_{n'}}_{mn'}. \tag{47.8}$$

The conditional probabilities in these summations are either zero or one, depending on whether the observed z_m matches the hypothesized values for x_n and the $\{x_{n'}\}$.

These probabilities can be computed in various obvious ways based on equation (47.7) and (47.8). The computations may be done most efficiently (if $|\mathcal{N}(m)|$ is large) by regarding $z_m + x_n$ as the final state of a Markov chain with states 0 and 1, this chain being started in state 0, and undergoing transitions corresponding to additions of the various $x_{n'}$, with transition probabilities given by the corresponding $q^0_{mn'}$ and $q^1_{mn'}$. The probabilities for z_m having its observed value given either $x_n = 0$ or $x_n = 1$ can then be found efficiently by use of the forward–backward algorithm (section 25.3).

A particularly convenient implementation of this method uses forward and backward passes in which products of the differences $\delta q_{mn} \equiv q^0_{mn} - q^1_{mn}$ are computed. We obtain $\delta r_{mn} \equiv r^0_{mn} - r^1_{mn}$ from the identity:

$$\delta r_{mn} = (-1)^{z_m} \prod_{n' \in \mathcal{N}(m) \backslash n} \delta q_{mn'}. \tag{47.9}$$

This identity is derived by iterating the following observation: if $\zeta = x_\mu + x_\nu \bmod 2$, and x_μ and x_ν have probabilities q^0_μ, q^0_ν and q^1_μ, q^1_ν of being 0 and 1, then $P(\zeta=1) = q^1_\mu q^0_\nu + q^0_\mu q^1_\nu$ and $P(\zeta=0) = q^0_\mu q^0_\nu + q^1_\mu q^1_\nu$. Thus $P(\zeta=0) - P(\zeta=1) = (q^0_\mu - q^1_\mu)(q^0_\nu - q^1_\nu)$.

We recover r^0_{mn} and r^1_{mn} using

$$r^0_{mn} = {}^1\!/_2(1 + \delta r_{mn}), \quad r^1_{mn} = {}^1\!/_2(1 - \delta r_{mn}). \tag{47.10}$$

The transformations into differences δq and back from δr to $\{r\}$ may be viewed as a Fourier transform and an inverse Fourier transformation.

Vertical step. The *vertical* step takes the computed values of r^0_{mn} and r^1_{mn} and updates the values of the probabilities q^0_{mn} and q^1_{mn}. For each n we compute:

$$q^0_{mn} = \alpha_{mn}\, p^0_n \prod_{m' \in \mathcal{M}(n) \backslash m} r^0_{m'n} \tag{47.11}$$

$$q^1_{mn} = \alpha_{mn}\, p^1_n \prod_{m' \in \mathcal{M}(n) \backslash m} r^1_{m'n} \tag{47.12}$$

where α_{mn} is chosen such that $q^0_{mn} + q^1_{mn} = 1$. These products can be efficiently computed in a downward pass and an upward pass.

We can also compute the 'pseudoposterior probabilities' q^0_n and q^1_n at this iteration, given by:

$$q^0_n = \alpha_n\, p^0_n \prod_{m \in \mathcal{M}(n)} r^0_{mn}, \tag{47.13}$$

$$q^1_n = \alpha_n\, p^1_n \prod_{m \in \mathcal{M}(n)} r^1_{mn}. \tag{47.14}$$

These quantities are used to create a tentative decoding $\hat{\mathbf{x}}$, the consistency of which is used to decide whether the decoding algorithm can halt. (Halt if $\mathbf{H}\hat{\mathbf{x}} = \mathbf{z}$.)

At this point, the algorithm repeats from the horizontal step.

The stop-when-it's-done decoding method. The recommended decoding procedure is to set $\hat{x}_n$ to 1 if $q^1_n > 0.5$ and see if the checks $\mathbf{H}\hat{\mathbf{x}} = \mathbf{z} \bmod 2$ are all satisfied, halting when they are, and declaring a failure if some maximum number of iterations (e.g. 200 or 1000) occurs without successful decoding. In the event of a failure, we may still report $\hat{\mathbf{x}}$, but we flag the whole block as a failure.

We note in passing the difference between this decoding procedure and the widespread practice in the turbo code community, where the decoding algorithm is run for a *fixed* number of iterations (irrespective of whether the decoder finds a consistent state at some earlier time). This practice is wasteful of computer time, and it blurs the distinction between undetected and detected errors. In our procedure, 'undetected' errors occur if the decoder finds an $\hat{\mathbf{x}}$

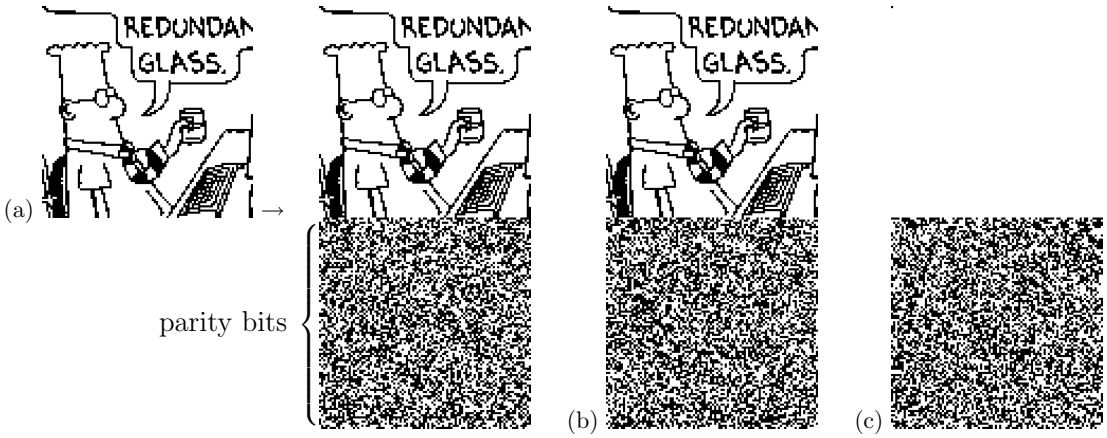


Figure 47.3. Demonstration of encoding with a rate-1/2 Gallager code. The encoder is derived from a very sparse $10\,000 \times 20\,000$ parity-check matrix with three 1s per column (figure 47.4). (a) The code creates transmitted vectors consisting of 10 000 source bits and 10 000 parity-check bits. (b) Here, the source sequence has been altered by changing the first bit. Notice that many of the parity-check bits are changed. Each parity bit depends on about half of the source bits. (c) The transmission for the case $\mathbf{s} = (1, 0, 0, \ldots, 0)$. This vector is the difference (modulo 2) between transmissions (a) and (b). [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

satisfying $\mathbf{H}\hat{\mathbf{x}} = \mathbf{z} \bmod 2$ that is not equal to the true $\mathbf{x}$. 'Detected' errors occur if the algorithm runs for the maximum number of iterations without finding a valid decoding. Undetected errors are of scientific interest because they reveal distance properties of a code. And in engineering practice, it would seem preferable for the blocks that are known to contain detected errors to be so labelled if practically possible.

Cost. In a brute-force approach, the time to create the generator matrix scales as N^3, where N is the block size. The encoding time scales as N^2, but encoding involves only binary arithmetic, so for the block lengths studied here it takes considerably less time than the simulation of the Gaussian channel. Decoding involves approximately $6Nj$ floating-point multiplies per iteration, so the total number of operations per decoded bit (assuming 20 iterations) is about $120t/R$, independent of blocklength. For the codes presented in the next section, this is about 800 operations.

The encoding complexity can be reduced by clever encoding tricks invented by Richardson and Urbanke (2001b) or by specially constructing the parity-check matrix (MacKay *et al.*, 1999).

The decoding complexity can be reduced, with only a small loss in performance, by passing low-precision messages in place of real numbers (Richardson and Urbanke, 2001a).

▶ 47.4 Pictorial demonstration of Gallager codes

Figures 47.3–47.7 illustrate visually the conditions under which low-density parity-check codes can give reliable communication over binary symmetric channels and Gaussian channels. These demonstrations may be viewed as animations on the world wide web.[1]

[1] `http://www.inference.phy.cam.ac.uk/mackay/codes/gifs/`

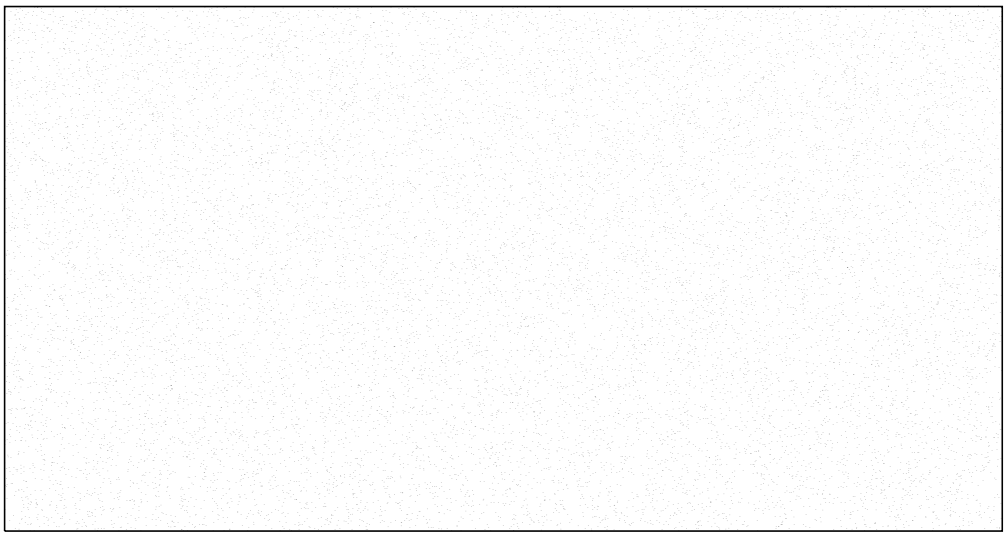
$\mathbf{H} \quad =$ 

Figure 47.4. A low-density parity-check matrix with $N = 20\,000$ columns of weight $j = 3$ and $M = 10\,000$ rows of weight $k = 6$.

Encoding

Figure 47.3 illustrates the encoding operation for the case of a Gallager code whose parity-check matrix is a $10\,000 \times 20\,000$ matrix with three 1s per column (figure 47.4). The high density of the *generator* matrix is illustrated in figure 47.3b and c by showing the change in the transmitted vector when one of the 10 000 source bits is altered. Of course, the source images shown here are highly redundant, and such images should really be compressed before encoding. Redundant images are chosen in these demonstrations to make it easier to see the correction process during the iterative decoding. The decoding algorithm does *not* take advantage of the redundancy of the source vector, and it would work in exactly the same way irrespective of the choice of source vector.

Iterative decoding

The transmission is sent over a channel with noise level $f = 7.5\%$ and the received vector is shown in the upper left of figure 47.5. The subsequent pictures in figure 47.5 show the iterative probabilistic decoding process. The sequence of figures shows the best guess, bit by bit, given by the iterative decoder, after 0, 1, 2, 3, 10, 11, 12, and 13 iterations. The decoder halts after the 13th iteration when the best guess violates no parity checks. This final decoding is error free.

In the case of an unusually noisy transmission, the decoding algorithm fails to find a valid decoding. For this code and a channel with $f = 7.5\%$, such failures happen about once in every 100 000 transmissions. Figure 47.6 shows this error rate compared with the block error rates of classical error-correcting codes.

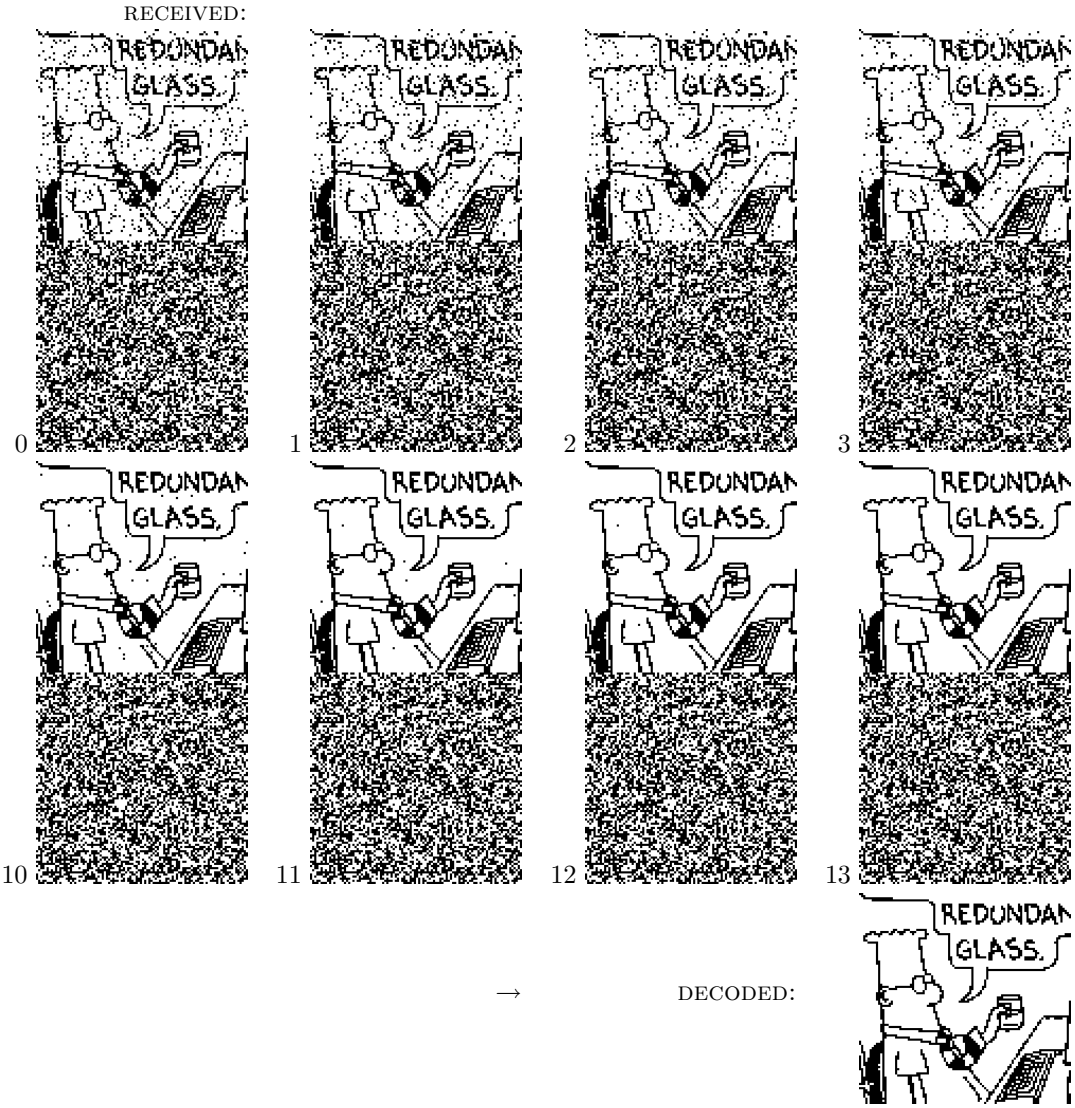


Figure 47.5. Iterative probabilistic decoding of a low-density parity-check code for a transmission received over a channel with noise level $f = 7.5\%$. The sequence of figures shows the best guess, bit by bit, given by the iterative decoder, after 0, 1, 2, 3, 10, 11, 12, and 13 iterations. The decoder halts after the 13th iteration when the best guess violates no parity checks. This final decoding is error free.

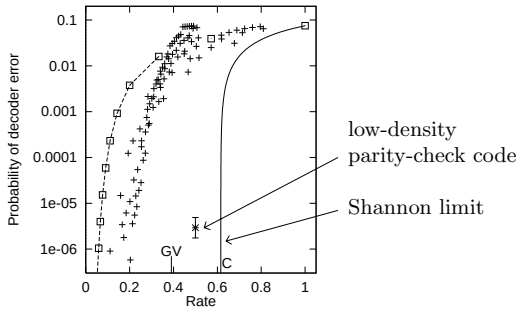


Figure 47.6. Error probability of the low-density parity-check code (with error bars) for binary symmetric channel with $f = 7.5\%$, compared with algebraic codes. Squares: repetition codes and Hamming $(7, 4)$ code; other points: Reed–Muller and BCH codes.

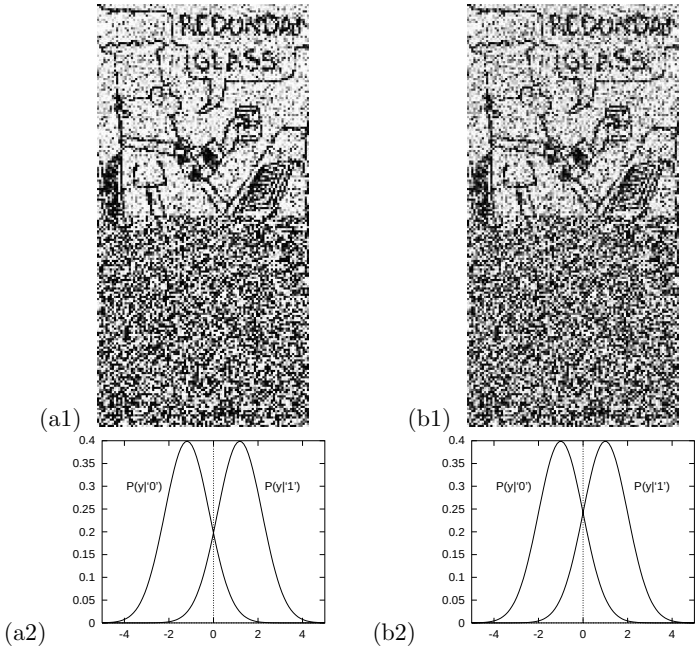


Figure 47.7. Demonstration of a Gallager code for a Gaussian channel. (a1) The received vector after transmission over a Gaussian channel with $x/\sigma = 1.185$ ($E_b/N_0 = 1.47$ dB). The greyscale represents the value of the normalized likelihood. This transmission can be perfectly decoded by the sum–product decoder. The empirical probability of decoding failure is about 10^{-5}. (a2) The probability distribution of the output y of the channel with $x/\sigma = 1.185$ for each of the two possible inputs. (b1) The received transmission over a Gaussian channel with $x/\sigma = 1.0$, which corresponds to the Shannon limit. (b2) The probability distribution of the output y of the channel with $x/\sigma = 1.0$ for each of the two possible inputs.

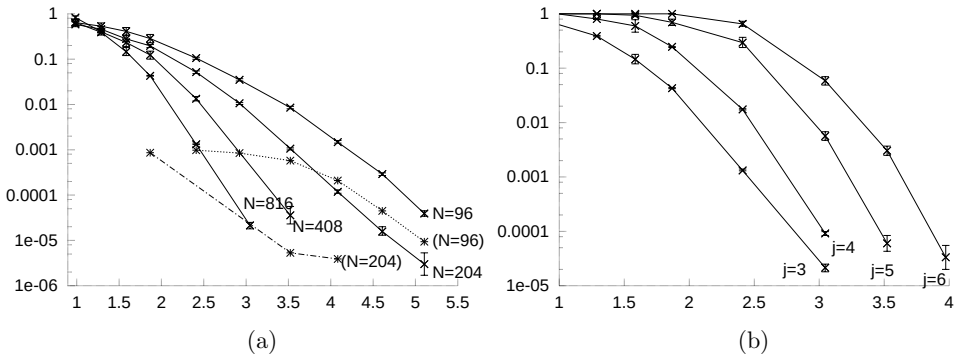


Figure 47.8. Performance of rate-$1/2$ Gallager codes on the Gaussian channel. Vertical axis: block error probability. Horizontal axis: signal-to-noise ratio E_b/N_0. (a) Dependence on blocklength N for $(j, k) = (3, 6)$ codes. From left to right: $N = 816$, $N = 408$, $N = 204$, $N = 96$. The dashed lines show the frequency of undetected errors, which is measurable only when the blocklength is as small as $N = 96$ or $N = 204$. (b) Dependence on column weight j for codes of blocklength $N = 816$.

Gaussian channel

In figure 47.7 the left picture shows the received vector after transmission over a Gaussian channel with $x/\sigma = 1.185$. The greyscale represents the value of the normalized likelihood, $\frac{P(y\,|\,t=1)}{P(y\,|\,t=1)+P(y\,|\,t=0)}$. This signal-to-noise ratio $x/\sigma = 1.185$ is a noise level at which this rate-1/2 Gallager code communicates reliably (the probability of error is $\simeq 10^{-5}$). To show how close we are to the Shannon limit, the right panel shows the received vector when the signal-to-noise ratio is reduced to $x/\sigma = 1.0$, which corresponds to the Shannon limit for codes of rate 1/2.

Variation of performance with code parameters

Figure 47.8 shows how the parameters N and j affect the performance of low-density parity-check codes. As Shannon would predict, increasing the blocklength leads to improved performance. The dependence on j follows a different pattern. Given an *optimal* decoder, the best performance would be obtained for the codes closest to random codes, that is, the codes with largest j. However, the sum–product decoder makes poor progress in dense graphs, so the best performance is obtained for a small value of j. Among the values

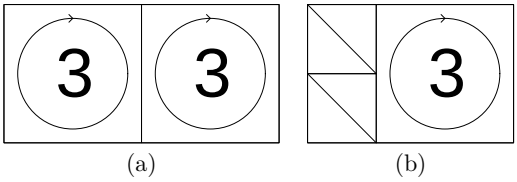


Figure 47.9. Schematic illustration of constructions (a) of a completely regular Gallager code with $j = 3$, $k = 6$ and $R = 1/2$; (b) of a nearly-regular Gallager code with rate 1/3. Notation: an integer represents a number of permutation matrices superposed on the surrounding square. A diagonal line represents an identity matrix.

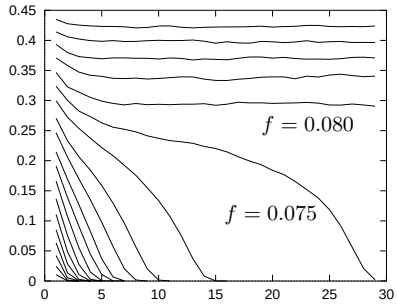


Figure 47.10. Monte Carlo simulation of density evolution, following the decoding process for $j = 4, k = 8$. Each curve shows the average entropy of a bit as a function of number of iterations, as estimated by a Monte Carlo algorithm using 10 000 samples per iteration. The noise level of the binary symmetric channel f increases by steps of 0.005 from bottom graph ($f = 0.010$) to top graph ($f = 0.100$). There is evidently a threshold at about $f = 0.075$, above which the algorithm cannot determine $\mathbf{x}$. From MacKay (1999b).

of j shown in the figure, $j = 3$ is the best, for a blocklength of 816, down to a block error probability of 10^{-5}.

This observation motivates construction of Gallager codes with some columns of weight 2. A construction with $M/2$ columns of weight 2 is shown in figure 47.9b. Too many columns of weight 2, and the code becomes a much poorer code.

As we'll discuss later, we can do even better by making the code even more irregular.

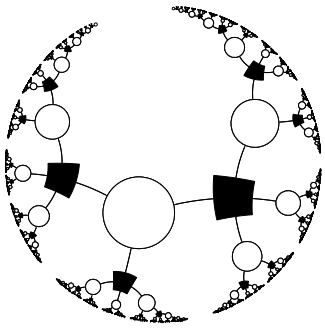


Figure 47.11. Local topology of the graph of a Gallager code with column weight $j = 3$ and row weight $k = 4$. White nodes represent bits, x_l; black nodes represent checks, z_m; each edge corresponds to a 1 in $\mathbf{H}$.

▶ 47.5 Density evolution

One way to study the decoding algorithm is to imagine it running on an infinite tree-like graph with the same local topology as the Gallager code's graph. The larger the matrix $\mathbf{H}$, the closer its decoding properties should approach those of the infinite graph.

Imagine an infinite belief network with no loops, in which every bit x_n connects to j checks and every check z_m connects to k bits (figure 47.11). We consider the iterative flow of information in this network, and examine the average entropy of one bit as a function of number of iterations. At each iteration, a bit has accumulated information from its local network out to a radius equal to the number of iterations. Successful decoding will occur only if the average entropy of a bit decreases to zero as the number of iterations increases.

The iterations of an infinite belief network can be simulated by Monte Carlo methods – a technique first used by Gallager (1963). Imagine a network of radius I (the total number of iterations) centred on one bit. Our aim is to compute the conditional entropy of the central bit x given the state $\mathbf{z}$ of all checks out to radius I. To evaluate the probability that the central bit is 1 given a *particular* syndrome $\mathbf{z}$ involves an I-step propagation from the outside of the network into the centre. At the ith iteration, probabilities r at

radius $I-i+1$ are transformed into qs and then into rs at radius $I-i$ in a way that depends on the states x of the unknown bits at radius $I-i$. In the Monte Carlo method, rather than simulating this network exactly, which would take a time that grows exponentially with I, we create for each iteration a representative sample (of size 100, say) of the values of $\{r, x\}$. In the case of a regular network with parameters j, k, each new pair $\{r, x\}$ in the list at the ith iteration is created by drawing the new x from its distribution and drawing at random with replacement $(j-1)(k-1)$ pairs $\{r, x\}$ from the list at the $(i-1)$th iteration; these are assembled into a tree fragment (figure 47.12) and the sum–product algorithm is run from top to bottom to find the new r value associated with the new node.

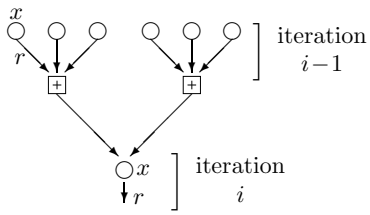


Figure 47.12. A tree-fragment constructed during Monte Carlo simulation of density evolution. This fragment is appropriate for a regular $j=3$, $k=4$ Gallager code.

As an example, the results of runs with $j=4$, $k=8$ and noise densities f between 0.01 and 0.10, using 10 000 samples at each iteration, are shown in figure 47.10. Runs with low enough noise level show a collapse to zero entropy after a small number of iterations, and those with high noise level decrease to a non-zero entropy corresponding to a failure to decode.

The boundary between these two behaviours is called the *threshold* of the decoding algorithm for the binary symmetric channel. Figure 47.10 shows by Monte Carlo simulation that the threshold for regular $(j, k) = (4, 8)$ codes is about 0.075. Richardson and Urbanke (2001a) have derived thresholds for regular codes by a tour de force of direct analytic methods. Some of these thresholds are shown in table 47.13.

(j,k)	$f_{\max}$
(3,6)	0.084
(4,8)	0.076
(5,10)	0.068

Table 47.13. Thresholds $f_{\max}$ for regular low-density parity-check codes, assuming sum–product decoding algorithm, from Richardson and Urbanke (2001a). The Shannon limit for rate-1/2 codes is $f_{\max} = 0.11$.

Approximate density evolution

For practical purposes, the computational cost of density evolution can be reduced by making Gaussian approximations to the probability distributions over the messages in density evolution, and updating only the parameters of these approximations. For further information about these techniques, which produce diagrams known as *EXIT charts*, see (ten Brink, 1999; Chung *et al.*, 2001; ten Brink *et al.*, 2002).

▶ 47.6 Improving Gallager codes

Since the rediscovery of Gallager codes, two methods have been found for enhancing their performance.

$GF(4)$	$\leftrightarrow$	binary
0	$\leftrightarrow$	00
1	$\leftrightarrow$	01
A	$\leftrightarrow$	10
B	$\leftrightarrow$	11

Table 47.14. Translation between $GF(4)$ and binary for message symbols.

Clump bits and checks together

First, we can make Gallager codes in which the variable nodes are grouped together into metavariables consisting of say 3 binary variables, and the check nodes are similarly grouped together into metachecks. As before, a sparse graph can be constructed connecting metavariables to metachecks, with a lot of freedom about the details of how the variables and checks within are wired up. One way to set the wiring is to work in a finite field $GF(q)$ such as $GF(4)$ or $GF(8)$, define low-density parity-check matrices using elements of $GF(q)$, and translate our binary messages into $GF(q)$ using a mapping such as the one for $GF(4)$ given in table 47.14. Now, when messages are passed during decoding, those messages are probabilities and likelihoods over *conjunctions* of binary variables. For example if each clump contains three binary variables then the likelihoods will describe the likelihoods of the eight alternative states of those bits.

$GF(4)$	$\rightarrow$	binary
0	$\rightarrow$	00 00
1	$\rightarrow$	10 01
A	$\rightarrow$	11 10
B	$\rightarrow$	01 11

Table 47.15. Translation between $GF(4)$ and binary for matrix entries. An $M \times N$ parity-check matrix over $GF(4)$ can be turned into a $2M \times 2N$ binary parity-check matrix in this way.

With carefully optimized constructions, the resulting codes over $GF(4)$,

$$\begin{aligned} F^0 &= [f^0 + f^1] + [f^A + f^B] \\ F^1 &= [f^0 - f^1] + [f^A - f^B] \\ F^A &= [f^0 + f^1] - [f^A + f^B] \\ F^B &= [f^0 - f^1] - [f^A - f^B] \end{aligned}$$

Algorithm 47.16. The Fourier transform over $GF(4)$. The Fourier transform F of a function f over $GF(2)$ is given by $F^0 = f^0 + f^1$, $F^1 = f^0 - f^1$. Transforms over $GF(2^k)$ can be viewed as a sequence of binary transforms in each of k dimensions. The inverse transform is identical to the Fourier transform, except that we also divide by 2^k.

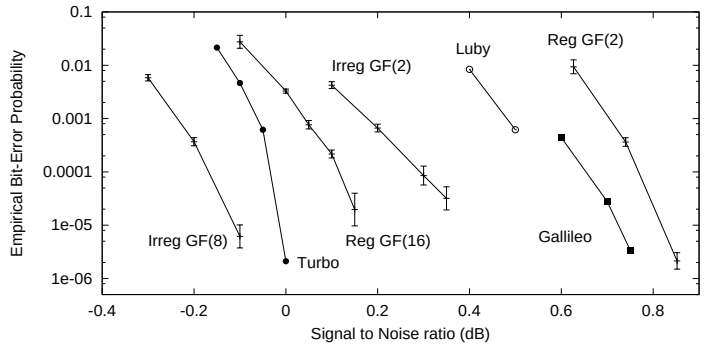


Figure 47.17. Comparison of regular binary Gallager codes with irregular codes, codes over $GF(q)$, and other outstanding codes of rate $1/4$. From left (best performance) to right: Irregular low-density parity-check code over $GF(8)$, blocklength 48 000 bits (Davey, 1999); JPL turbo code (JPL, 1996) blocklength 65 536; Regular low-density parity-check over $GF(16)$, blocklength 24 448 bits (Davey and MacKay, 1998); Irregular binary low-density parity-check code, blocklength 16 000 bits (Davey, 1999); Luby *et al.* (1998) irregular binary low-density parity-check code, blocklength 64 000 bits; JPL code for Galileo (in 1992, this was the best known code of rate 1/4); Regular binary low-density parity-check code: blocklength 40 000 bits (MacKay, 1999b). The Shannon limit is at about -0.79 dB. As of 2003, even better sparse-graph codes have been constructed.

$GF(8)$, and $GF(16)$ perform nearly one decibel better than comparable binary Gallager codes.

The computational cost for decoding in $GF(q)$ scales as $q \log q$, if the appropriate Fourier transform is used in the check nodes: the update rule for the check-to-variable message,

$$r^a_{mn} = \sum_{\mathbf{x}: x_n = a} \mathbb{1} \left[\sum_{n' \in \mathcal{N}(m)} H_{mn'} x_{n'} = z_m \right] \prod_{j \in \mathcal{N}(m) \backslash n} q^{x_j}_{mj}, \tag{47.15}$$

is a convolution of the quantities q^a_{mj}, so the summation can be replaced by a product of the Fourier transforms of q^a_{mj} for $j \in \mathcal{N}(m) \backslash n$, followed by an inverse Fourier transform. The Fourier transform for $GF(4)$ is shown in algorithm 47.16.

Make the graph irregular

The second way of improving Gallager codes, introduced by Luby *et al.* (2001b), is to make their graphs *irregular*. Instead of giving all variable nodes the same degree j, we can have some variable nodes with degree 2, some 3, some 4, and a few with degree 20. Check nodes can also be given unequal degrees – this helps improve performance on erasure channels, but it turns out that for the Gaussian channel, the best graphs have regular check degrees.

Figure 47.17 illustrates the benefits offered by these two methods for improving Gallager codes, focussing on codes of rate $1/4$. Making the binary code irregular gives a win of about 0.4 dB; switching from $GF(2)$ to $GF(16)$ gives

DIFFERENCE SET CYCLIC CODES						
N	7	21	73	273	1057	4161
M	4	**10**	**28**	**82**	**244**	**730**
K	3	11	45	191	813	3431
d	4	**6**	**10**	**18**	34	66
k	3	**5**	**9**	17	33	65

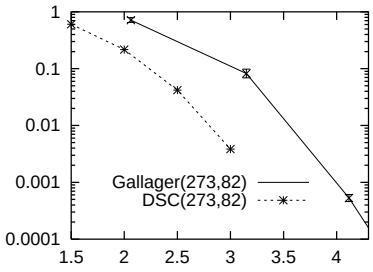


Figure 47.18. An algebraically constructed low-density parity-check code satisfying many redundant constraints outperforms an equivalent random Gallager code. The table shows the N, M, K, distance d, and row weight k of some difference-set cyclic codes, highlighting the codes that have large d/N, small k, and large N/M. In the comparison the Gallager code had $(j, k) = (4, 13)$, and rate identical to the $N = 273$ difference-set cyclic code. Vertical axis: block error probability. Horizontal axis: signal-to-noise ratio E_b/N_0 (dB).

about 0.6 dB; and Matthew Davey's code that combines both these features – it's irregular over $GF(8)$ – gives a win of about 0.9 dB over the regular binary Gallager code.

Methods for optimizing the *profile* of a Gallager code (that is, its number of rows and columns of each degree), have been developed by Richardson *et al.* (2001) and have led to low-density parity-check codes whose performance, when decoded by the sum–product algorithm, is within a hair's breadth of the Shannon limit.

Algebraic constructions of Gallager codes

The performance of regular Gallager codes can be enhanced in a third manner: by designing the code to have *redundant sparse constraints*. There is a *difference-set cyclic code*, for example, that has $N = 273$ and $K = 191$, but the code satisfies not $M = 82$ but N, i.e., *273* low-weight constraints (figure 47.18). It is impossible to make random Gallager codes that have anywhere near this much redundancy among their checks. The difference-set cyclic code performs about 0.7 dB better than an equivalent random Gallager code.

An open problem is to discover codes sharing the remarkable properties of the difference-set cyclic codes but with different blocklengths and rates. I call this task *the Tanner challenge*.

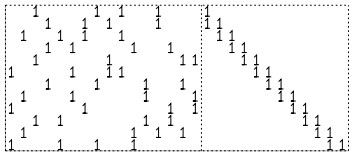
▶ 47.7 Fast encoding of low-density parity-check codes

We now discuss methods for fast encoding of low-density parity-check codes – faster than the standard method, in which a generator matrix $\mathbf{G}$ is found by Gaussian elimination (at a cost of order M^3) and then each block is encoded by multiplying it by $\mathbf{G}$ (at a cost of order M^2).

Staircase codes

Certain low-density parity-check matrices with M columns of weight 2 or less can be encoded easily in linear time. For example, if the matrix has a *staircase* structure as illustrated by the right-hand side of

$$\mathbf{H} = \left[\quad \right], \tag{47.16}$$



and if the data $\mathbf{s}$ are loaded into the first K bits, then the M parity bits $\mathbf{p}$ can be computed from left to right in linear time.

$$\begin{aligned} p_1 &= && \textstyle\sum_{n=1}^{K} H_{1n}s_n \\ p_2 &= p_1 &+ & \textstyle\sum_{n=1}^{K} H_{2n}s_n \\ p_3 &= p_2 &+ & \textstyle\sum_{n=1}^{K} H_{3n}s_n \\ &\vdots \\ p_M &= p_{M-1} &+ & \textstyle\sum_{n=1}^{K} H_{Mn}s_n. \end{aligned} \tag{47.17}$$

If we call two parts of the $\mathbf{H}$ matrix $[\mathbf{H}_s|\mathbf{H}_p]$, we can describe the encoding operation in two steps: first compute an intermediate parity vector $\mathbf{v} = \mathbf{H}_s\mathbf{s}$; then pass $\mathbf{v}$ through an accumulator to create $\mathbf{p}$.

The cost of this encoding method is linear if the sparsity of $\mathbf{H}$ is exploited when computing the sums in (47.17).

Fast encoding of general low-density parity-check codes

Richardson and Urbanke (2001b) demonstrated an elegant method by which the encoding cost of any low-density parity-check code can be reduced from the straightforward method's M^2 to a cost of $N + g^2$, where g, the *gap*, is hopefully a small constant, and in the worst cases scales as a small fraction of N.

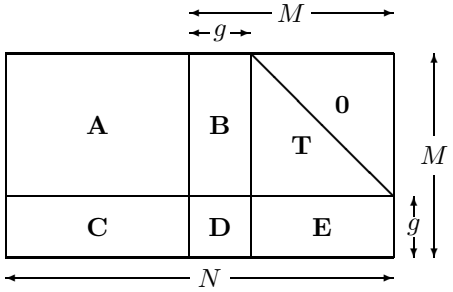


Figure 47.19. The parity-check matrix in approximate lower-triangular form.

In the first step, the parity-check matrix is rearranged, by row-interchange and column-interchange, into the *approximate lower-triangular form* shown in figure 47.19. The original matrix $\mathbf{H}$ was very sparse, so the six matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{T}$, $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$ are also very sparse. The matrix $\mathbf{T}$ is lower triangular and has 1s everywhere on the diagonal.

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix}. \tag{47.18}$$

The source vector $\mathbf{s}$ of length $K = N - M$ is encoded into a transmission $\mathbf{t} = [\mathbf{s}, \mathbf{p}_1, \mathbf{p}_2]$ as follows.

1. Compute the upper syndrome of the source vector,

$$\mathbf{z}_A = \mathbf{A}\mathbf{s}. \tag{47.19}$$

   This can be done in linear time.

2. Find a setting of the second parity bits, $\mathbf{p}_2^A$, such that the upper syndrome is zero.

$$\mathbf{p}_2^A = -\mathbf{T}^{-1}\mathbf{z}_A. \tag{47.20}$$

   This vector can be found in linear time by back-substitution, i.e., computing the first bit of $\mathbf{p}_2^A$, then the second, then the third, and so forth.

3. Compute the lower syndrome of the vector $[\mathbf{s}, \mathbf{0}, \mathbf{p}_2^A]$:

$$\mathbf{z}_B = \mathbf{C}\mathbf{s} - \mathbf{E}\mathbf{p}_2^A. \tag{47.21}$$

   This can be done in linear time.

4. Now we get to the clever bit. Define the matrix

$$\mathbf{F} \equiv -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}, \tag{47.22}$$

   and find its inverse, $\mathbf{F}^{-1}$. This computation needs to be done once only, and its cost is of order g^3. This inverse $\mathbf{F}^{-1}$ is a dense $g \times g$ matrix. [If $\mathbf{F}$ is not invertible then either $\mathbf{H}$ is not of full rank, or else further column permutations of $\mathbf{H}$ can produce an $\mathbf{F}$ that is invertible.]

   Set the first parity bits, $\mathbf{p}_1$, to

$$\mathbf{p}_1 = -\mathbf{F}^{-1}\mathbf{z}_B. \tag{47.23}$$

   This operation has a cost of order g^2.

   Claim: At this point, we have found the correct setting of the first parity bits, $\mathbf{p}_1$.

5. Discard the tentative parity bits $\mathbf{p}_2^A$ and find the new upper syndrome,

$$\mathbf{z}_C = \mathbf{z}_A + \mathbf{B}\mathbf{p}_1. \tag{47.24}$$

   This can be done in linear time.

6. Find a setting of the second parity bits, $\mathbf{p}_2$, such that the upper syndrome is zero,

$$\mathbf{p}_2 = -\mathbf{T}^{-1}\mathbf{z}_C \tag{47.25}$$

   This vector can be found in linear time by back-substitution.

▶ 47.8 Further reading

Low-density parity-check codes codes were first studied in 1962 by Gallager, then were generally forgotten by the coding theory community. Tanner (1981) generalized Gallager's work by introducing more general constraint nodes; the codes that are now called turbo product codes should in fact be called Tanner product codes, since Tanner proposed them, and his colleagues (Karplus and Krit, 1991) implemented them in hardware. Publications on Gallager codes contributing to their 1990s rebirth include (Wiberg *et al.*, 1995; MacKay and Neal, 1995; MacKay and Neal, 1996; Wiberg, 1996; MacKay, 1999b; Spielman, 1996; Sipser and Spielman, 1996). Low-precision decoding algorithms and fast encoding algorithms for Gallager codes are discussed in (Richardson and Urbanke, 2001a; Richardson and Urbanke, 2001b). MacKay and Davey (2000) showed that low-density parity-check codes can outperform Reed–Solomon codes, even on the Reed–Solomon codes' home turf: high rate and short block-lengths. Other important papers include (Luby *et al.*, 2001a; Luby *et al.*, 2001b; Luby *et al.*, 1997; Davey and MacKay, 1998; Richardson *et al.*, 2001; Chung *et al.*, 2001). Useful tools for the design of irregular low-density parity-check codes include (Chung *et al.*, 1999; Urbanke, 2001).

See (Wiberg, 1996; Frey, 1998; McEliece *et al.*, 1998) for further discussion of the sum–product algorithm.

For a view of low-density parity-check code decoding in terms of group theory and coding theory, see (Forney, 2001; Offer and Soljanin, 2000; Offer

and Soljanin, 2001); and for background reading on this topic see (Hartmann and Rudolph, 1976; Terras, 1999). There is a growing literature on the practical design of low-density parity-check codes (Mao and Banihashemi, 2000; Mao and Banihashemi, 2001; ten Brink *et al.*, 2002); they are now being adopted for applications from hard drives to satellite communications.

For low-density parity-check codes applicable to quantum error-correction, see MacKay *et al.* (2004).

▶ 47.9 Exercises

Exercise 47.1.[2] The 'hyperbolic tangent' version of the decoding algorithm. In section 47.3, the sum–product decoding algorithm for low-density parity-check codes was presented first in terms of quantities $q_{mn}^{0/1}$ and $r_{mn}^{0/1}$, then in terms of quantities δq and δr. There is a third description, in which the $\{q\}$ are replaced by log probability-ratios,

$$l_{mn} \equiv \ln \frac{q_{mn}^0}{q_{mn}^1}. \tag{47.26}$$

Show that

$$\delta q_{mn} \equiv q_{mn}^0 - q_{mn}^1 = \tanh(l_{mn}/2). \tag{47.27}$$

Derive the update rules for $\{r\}$ and $\{l\}$.

Exercise 47.2.[2, p.572] I am sometimes asked 'why not decode *other* linear codes, for example algebraic codes, by transforming their parity-check matrices so that they are low-density, and applying the sum–product algorithm?' [Recall that any linear combination of rows of $\mathbf{H}$, $\mathbf{H}' = \mathbf{PH}$, is a valid parity-check matrix for a code, as long as the matrix $\mathbf{P}$ is invertible; so there are many parity check matrices for any one code.]

Explain why a random linear code does not have a low-density parity-check matrix. [Here, low-density means 'having row-weight at most k', where k is some small constant $\ll N$.]

Exercise 47.3.[3] Show that if a low-density parity-check code has more than M columns of weight 2 – say αM columns, where $\alpha > 1$ – then the code will have words with weight of order $\log M$.

Exercise 47.4.[5] In section 13.5 we found the expected value of the weight enumerator function $A(w)$, averaging over the ensemble of all random linear codes. This calculation can also be carried out for the ensemble of low-density parity-check codes (Gallager, 1963; MacKay, 1999b; Litsyn and Shevelev, 2002). It is plausible, however, that the mean value of $A(w)$ is not always a good indicator of the *typical* value of $A(w)$ in the ensemble. For example, if, at a particular value of w, 99% of codes have $A(w) = 0$, and 1% have $A(w) = 100\,000$, then while we might say the typical value of $A(w)$ is zero, the mean is found to be 1000. Find the *typical* weight enumerator function of low-density parity-check codes.

▶ 47.10 Solutions

Solution to exercise 47.2 (p.572). Consider codes of rate R and blocklength N, having $K = RN$ source bits and $M = (1-R)N$ parity-check bits. Let all

the codes have their bits ordered so that the first K bits are independent, so that we could if we wish put the code in systematic form,

$$\mathbf{G} = [\mathbf{1}_K | \mathbf{P}^{\mathsf{T}}]; \;\; \mathbf{H} = [\mathbf{P} | \mathbf{1}_M]. \tag{47.28}$$

The number of *distinct* linear codes is the number of matrices $\mathbf{P}$, which is $\mathcal{N}_1 = 2^{MK} = 2^{N^2 R(1-R)}$. Can these all be expressed as distinct low-density parity-check codes? $\log \mathcal{N}_1 \simeq N^2 R(1-R)$

The number of low-density parity-check matrices with row-weight k is

$$\binom{N}{k}^M \tag{47.29}$$

and the number of distinct codes that they define is at most

$$\mathcal{N}_2 = \binom{N}{k}^M \Big/ M!, \tag{47.30}$$

which is much smaller than $\mathcal{N}_1$, so, by the pigeon-hole principle, it is not possible for every random linear code to map on to a low-density $\mathbf{H}$. $\log \mathcal{N}_2 < Nk \log N$

48

Convolutional Codes and Turbo Codes

This chapter follows tightly on from Chapter 25. It makes use of the ideas of codes and trellises and the forward–backward algorithm.

▶ 48.1 Introduction to convolutional codes

When we studied linear block codes, we described them in three ways:

1. The generator matrix describes how to turn a string of K arbitrary source bits into a transmission of N bits.

2. The parity-check matrix specifies the $M = N - K$ parity-check constraints that a valid codeword satisfies.

3. The trellis of the code describes its valid codewords in terms of paths through a trellis with labelled edges.

A fourth way of describing some block codes, the algebraic approach, is not covered in this book (a) because it has been well covered by numerous other books in coding theory; (b) because, as this part of the book discusses, the state-of-the-art in error-correcting codes makes little use of algebraic coding theory; and (c) because I am not competent to teach this subject.

We will now describe convolutional codes in two ways: first, in terms of mechanisms for generating transmissions $\mathbf{t}$ from source bits $\mathbf{s}$; and second, in terms of trellises that describe the constraints satisfied by valid transmissions.

▶ 48.2 Linear-feedback shift-registers

We generate a transmission with a convolutional code by putting a source stream through a linear filter. This filter makes use of a shift register, linear output functions, and, possibly, linear feedback.

I will draw the shift-register in a right-to-left orientation: bits roll from right to left as time goes on.

Figure 48.1 shows three linear-feedback shift-registers which could be used to define convolutional codes. The rectangular box surrounding the bits $z_1 \ldots z_7$ indicates the *memory* of the filter, also known as its *state*. All three filters have one input and two outputs. On each clock cycle, the source supplies one bit, and the filter outputs two bits $t^{(a)}$ and $t^{(b)}$. By concatenating together these bits we can obtain from our source stream $s_1 s_2 s_3 \ldots$ a transmission stream $t_1^{(a)} t_1^{(b)} t_2^{(a)} t_2^{(b)} t_3^{(a)} t_3^{(b)} \ldots$. Because there are two transmitted bits for every source bit, the codes shown in figure 48.1 have rate $1/2$. Because

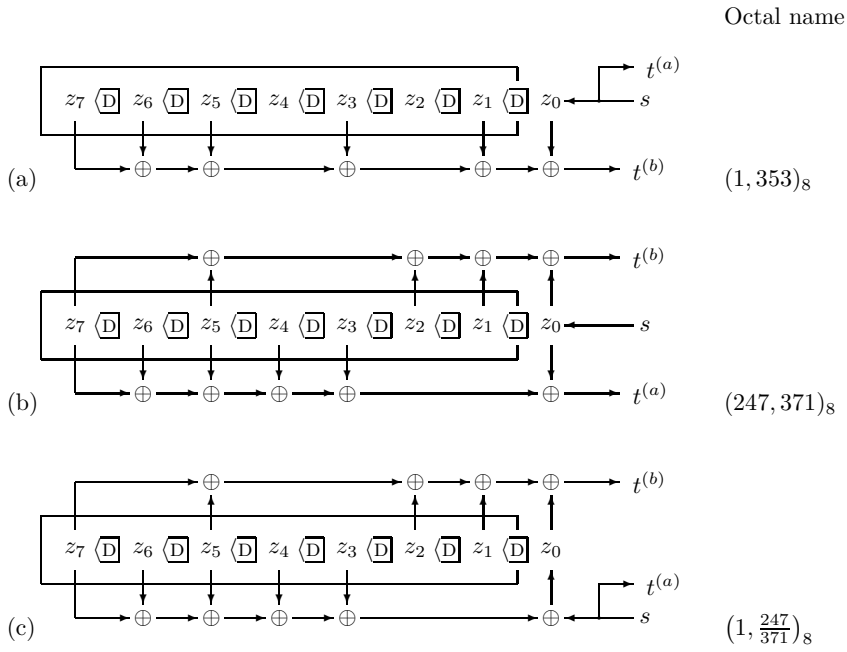


Figure 48.1. Linear-feedback shift-registers for generating convolutional codes with rate 1/2. The symbol ⟨D⟩ indicates a copying with a delay of one clock cycle. The symbol ⊕ denotes linear addition modulo 2 with no delay.
The filters are (a) systematic and nonrecursive; (b) nonsystematic and nonrecursive; (c) systematic and recursive.

these filters require $k = 7$ bits of memory, the codes they define are known as a *constraint-length 7 codes*.

Convolutional codes come in three flavours, corresponding to the three types of filter in figure 48.1.

Systematic nonrecursive

The filter shown in figure 48.1a has no feedback. It also has the property that one of the output bits, $t^{(a)}$, is identical to the source bit s. This encoder is thus called *systematic*, because the source bits are reproduced transparently in the transmitted stream, and *nonrecursive*, because it has no feedback. The other transmitted bit $t^{(b)}$ is a linear function of the state of the filter. One way of describing that function is as a dot product (modulo 2) between two binary vectors of length $k + 1$: a binary vector $\mathbf{g}^{(b)} = (1, 1, 1, 0, 1, 0, 1, 1)$ and the state vector $\mathbf{z} = (z_k, z_{k-1}, \ldots, z_1, z_0)$. We include in the state vector the bit z_0 that will be put into the first bit of the memory on the next cycle. The vector $\mathbf{g}^{(b)}$ has $g^{(b)}_\kappa = 1$ for every κ where there is a tap (a downward pointing arrow) from state bit z_κ into the transmitted bit $t^{(b)}$.

A convenient way to describe these binary tap vectors is in octal. Thus, this filter makes use of the tap vector 353_8. I have drawn the delay lines from right to left to make it easy to relate the diagrams to these octal numbers.

11	101	011
↓	↓	↓
3	5	3

Table 48.2. How taps in the delay line are converted to octal.

Nonsystematic nonrecursive

The filter shown in figure 48.1b also has no feedback, but it is not systematic. It makes use of two tap vectors $\mathbf{g}^{(a)}$ and $\mathbf{g}^{(b)}$ to create its two transmitted bits. This encoder is thus *nonsystematic* and *nonrecursive*. Because of their added complexity, nonsystematic codes can have error-correcting abilities superior to those of systematic nonrecursive codes with the same constraint length.

Systematic recursive

The filter shown in figure 48.1c is similar to the nonsystematic nonrecursive filter shown in figure 48.1b, but it uses the taps that formerly made up $\mathbf{g}^{(a)}$ to make a linear signal that is fed back into the shift register along with the source bit. The output $t^{(b)}$ is a linear function of the state vector as before. The other output is $t^{(a)} = s$, so this filter is systematic.

A recursive code is conventionally identified by an octal ratio, e.g., figure 48.1c's code is denoted by $(247/371)_8$.

Equivalence of systematic recursive and nonsystematic nonrecursive codes

The two filters in figure 48.1b,c are *code-equivalent* in that the *sets* of codewords that they define are identical. For every codeword of the nonsystematic nonrecursive code we can choose a source stream for the other encoder such that its output is identical (and *vice versa*).

To prove this, we denote by p the quantity $\sum_{\kappa=1}^{k} g_\kappa^{(a)} z_\kappa$, as shown in figure 48.3a and b, which shows a pair of smaller but otherwise equivalent filters. If the two transmissions are to be equivalent – that is, the $t^{(a)}$s are equal in both figures and so are the $t^{(b)}$s – then on every cycle the source bit in the systematic code must be $s = t^{(a)}$. So now we must simply confirm that for this choice of s, the systematic code's shift register will follow the same state sequence as that of the nonsystematic code, assuming that the states match initially. In figure 48.3a we have

$$t^{(a)} = p \oplus z_0^{\text{nonrecursive}} \tag{48.1}$$

whereas in figure 48.3b we have

$$z_0^{\text{recursive}} = t^{(a)} \oplus p. \tag{48.2}$$

Substituting for $t^{(a)}$, and using $p \oplus p = 0$ we immediately find

$$z_0^{\text{recursive}} = z_0^{\text{nonrecursive}}. \tag{48.3}$$

Thus, any codeword of a nonsystematic nonrecursive code is a codeword of a systematic recursive code with the same taps – the same taps in the sense that there are vertical arrows in all the same places in figures 48.3(a) and (b), though one of the arrows points up instead of down in (b).

Now, while these two codes are equivalent, the two encoders behave differently. The nonrecursive encoder has a *finite impulse response*, that is, if one puts in a string that is all zeroes except for a single one, the resulting output stream contains a finite number of ones. Once the one bit has passed through all the states of the memory, the delay line returns to the all-zero state. Figure 48.4a shows the state sequence resulting from the source string $\mathbf{s} = (0, 0, 1, 0, 0, 0, 0, 0)$.

Figure 48.4b shows the trellis of the recursive code of figure 48.3b and the response of this filter to the same source string $\mathbf{s} = (0, 0, 1, 0, 0, 0, 0, 0)$. The filter has an *infinite impulse response*. The response settles into a periodic state with period equal to three clock cycles.

▷ Exercise 48.1.[1] What is the input to the recursive filter such that its state sequence and the transmission are the same as those of the nonrecursive filter? (Hint: see figure 48.5.)

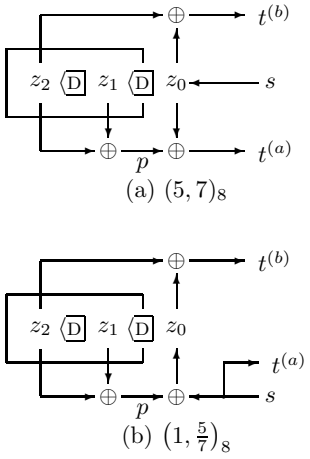
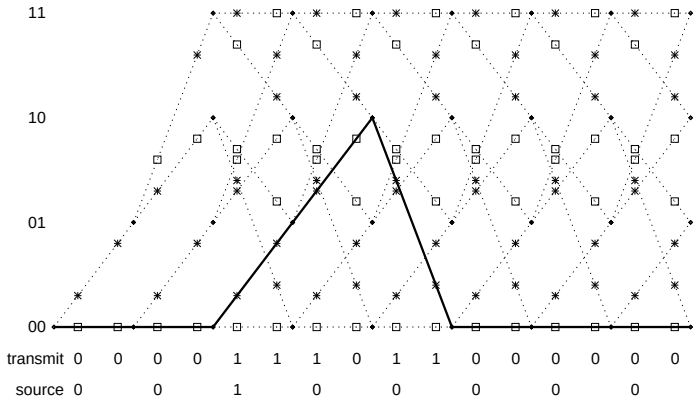


Figure 48.3. Two rate-1/2 convolutional codes with constraint length $k = 2$: (a) non-recursive; (b) recursive. The two codes are equivalent.

(a)



(b)

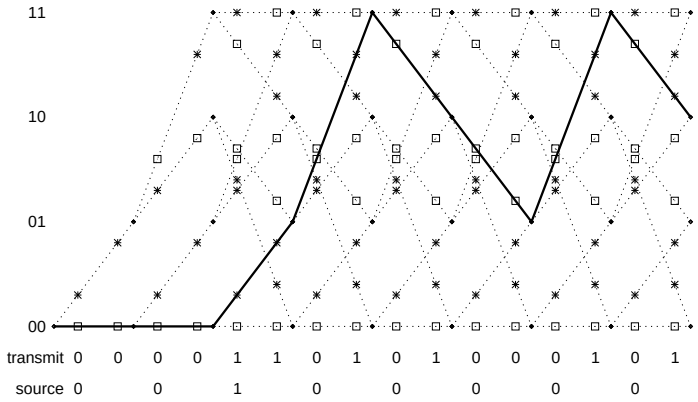


Figure 48.4. Trellises of the rate-1/2 convolutional codes of figure 48.3. It is assumed that the initial state of the filter is $(z_2, z_1) = (0, 0)$. Time is on the horizontal axis and the state of the filter at each time step is the vertical coordinate. On the line segments are shown the emitted symbols $t^{(a)}$ and $t^{(b)}$, with stars for '1' and boxes for '0'. The paths taken through the trellises when the source sequence is 00100000 are highlighted with a solid line. The light dotted lines show the state trajectories that are possible for other source sequences.

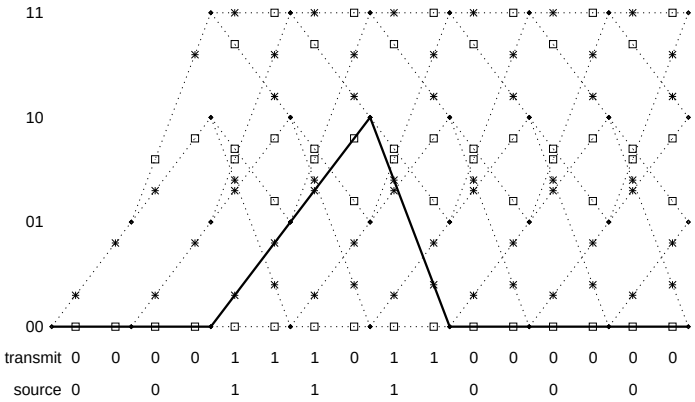


Figure 48.5. The source sequence for the systematic recursive code 00111000 produces the same path through the trellis as 00100000 does in the nonsystematic nonrecursive case.

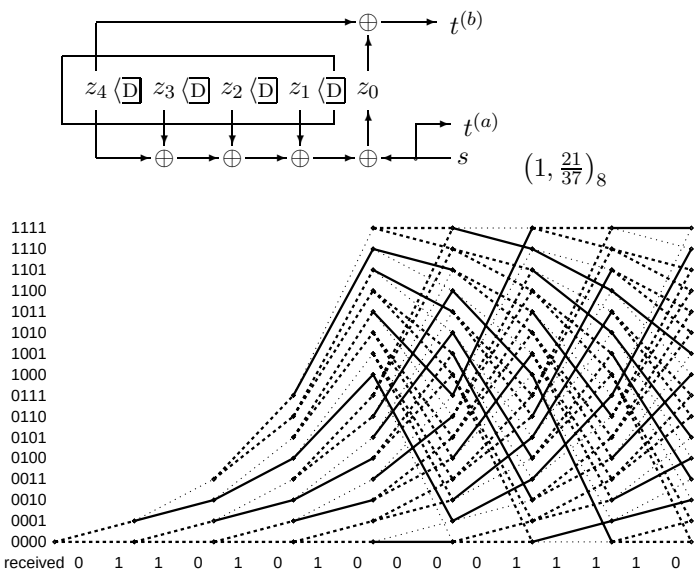


Figure 48.6. The trellis for a $k = 4$ code painted with the likelihood function when the received vector is equal to a codeword with just one bit flipped. There are three line styles, depending on the value of the likelihood: thick solid lines show the edges in the trellis that match the corresponding two bits of the received string exactly; thick dotted lines show edges that match one bit but mismatch the other; and thin dotted lines show the edges that mismatch both bits.

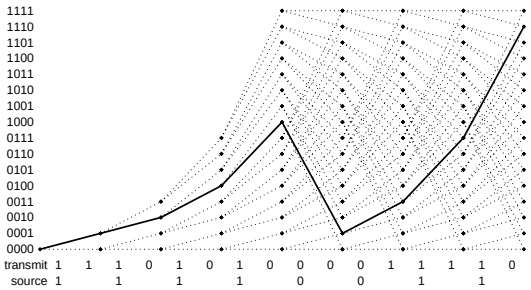
In general a linear-feedback shift-register with k bits of memory has an impulse response that is periodic with a period that is at most $2^k - 1$, corresponding to the filter visiting every non-zero state in its state space.

Incidentally, cheap pseudorandom number generators and cheap cryptographic products make use of exactly these periodic sequences, though with larger values of k than 7; the random number seed or cryptographic key selects the initial state of the memory. There is thus a close connection between certain cryptanalysis problems and the decoding of convolutional codes.

▶ 48.3 Decoding convolutional codes

The receiver receives a bit stream, and wishes to infer the state sequence and thence the source stream. The posterior probability of each bit can be found by the sum–product algorithm (also known as the forward–backward or BCJR algorithm), which was introduced in section 25.3. The most probable state sequence can be found using the min–sum algorithm of section 25.3 (also known as the Viterbi algorithm). The nature of this task is illustrated in figure 48.6, which shows the cost associated with each edge in the trellis for the case of a sixteen-state code; the channel is assumed to be a binary symmetric channel and the received vector is equal to a codeword except that one bit has been flipped. There are three line styles, depending on the value of the likelihood: thick solid lines show the edges in the trellis that match the corresponding two bits of the received string exactly; thick dotted lines show edges that match one bit but mismatch the other; and thin dotted lines show the edges that mismatch both bits. The min–sum algorithm seeks the path through the trellis that uses as many solid lines as possible; more precisely, it minimizes the cost of the path, where the cost is zero for a solid line, one for a thick dotted line, and two for a thin dotted line.

▷ Exercise 48.2.[1, p.581] Can you spot the most probable path and the flipped bit?



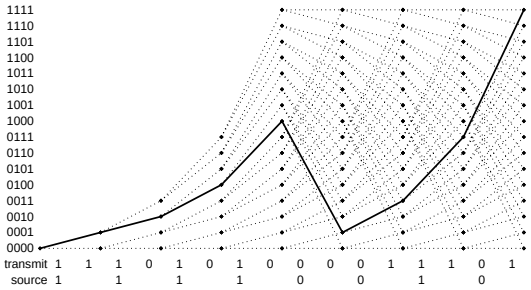


Figure 48.7. Two paths that differ in two transmitted bits only.

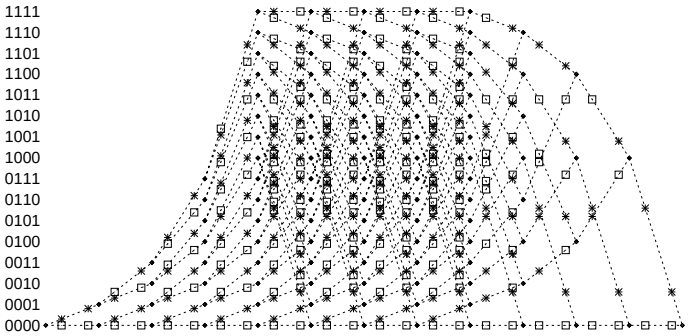


Figure 48.8. A terminated trellis. When any codeword is completed, the filter state is 0000.

Unequal protection

A defect of the convolutional codes presented thus far is that they offer unequal protection to the source bits. Figure 48.7 shows two paths through the trellis that differ in only two transmitted bits. The last source bit is less well protected than the other source bits. This unequal protection of bits motivates the *termination* of the trellis.

A terminated trellis is shown in figure 48.8. Termination slightly reduces the number of source bits used per codeword. Here, four source bits are turned into parity bits because the $k = 4$ memory bits must be returned to zero.

▶ 48.4 Turbo codes

An (N, K) turbo code is defined by a number of constituent convolutional encoders (often, two) and an equal number of *interleavers* which are $K \times K$ permutation matrices. Without loss of generality, we take the first interleaver to be the identity matrix. A string of K source bits is encoded by feeding them into each constituent encoder in the order defined by the associated interleaver, and transmitting the bits that come out of each constituent encoder. Often the first constituent encoder is chosen to be a systematic encoder, just like the recursive filter shown in figure 48.6, and the second is a non-systematic one of rate 1 that emits parity bits only. The transmitted codeword then consists of

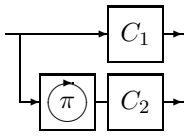


Figure 48.10. The encoder of a turbo code. Each box C_1, C_2, contains a convolutional code. The source bits are reordered using a permutation π before they are fed to C_2. The transmitted codeword is obtained by concatenating or interleaving the outputs of the two convolutional codes.

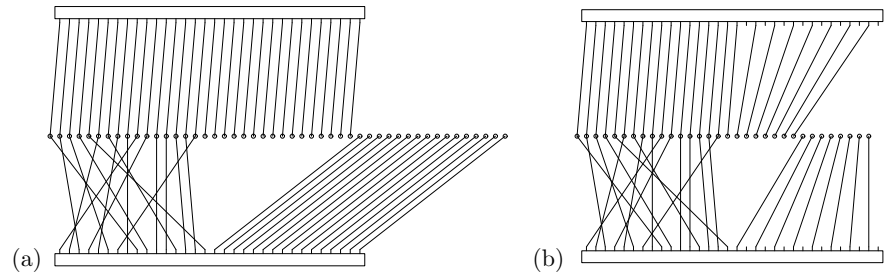


Figure 48.9. Rate-1/3 (a) and rate-1/2 (b) turbo codes represented as factor graphs. The circles represent the codeword bits. The two rectangles represent trellises of rate-1/2 convolutional codes, with the systematic bits occupying the left half of the rectangle and the parity bits occupying the right half. The puncturing of these constituent codes in the rate-1/2 turbo code is represented by the lack of connections to half of the parity bits in each trellis.

K source bits followed by M_1 parity bits generated by the first convolutional code and M_2 parity bits from the second. The resulting turbo code has rate $1/3$.

The turbo code can be represented by a factor graph in which the two trellises are represented by two large rectangular nodes (figure 48.9a); the K source bits and the first M_1 parity bits participate in the first trellis and the K source bits and the last M_2 parity bits participate in the second trellis. Each codeword bit participates in either one or two trellises, depending on whether it is a parity bit or a source bit. Each trellis node contains a trellis exactly like the terminated trellis shown in figure 48.8, except one thousand times as long. [There are other factor graph representations for turbo codes that make use of more elementary nodes, but the factor graph given here yields the standard version of the sum–product algorithm used for turbo codes.]

If a turbo code of smaller rate such as $1/2$ is required, a standard modification to the rate-$1/3$ code is to *puncture* some of the parity bits (figure 48.9b).

Turbo codes are decoded using the sum–product algorithm described in Chapter 26. On the first iteration, each trellis receives the channel likelihoods, and runs the forward–backward algorithm to compute, for each bit, the relative likelihood of its being `1` or `0`, given the information about the other bits. These likelihoods are then passed across from each trellis to the other, and multiplied by the channel likelihoods on the way. We are then ready for the second iteration: the forward–backward algorithm is run again in each trellis using the updated probabilities. After about ten or twenty such iterations, it's hoped that the correct decoding will be found. It is common practice to stop after some fixed number of iterations, but we can do better.

As a stopping criterion, the following procedure can be used at every iteration. For each time-step in each trellis, we identify the most probable edge, according to the local messages. If these most probable edges join up into two valid paths, one in each trellis, and if these two paths are consistent with each other, it is reasonable to stop, as subsequent iterations are unlikely to take the decoder away from this codeword. If a maximum number of iterations is reached without this stopping criterion being satisfied, a decoding error can be reported. This stopping procedure is recommended for several reasons: it allows a big saving in decoding time with no loss in error probability; it allows decoding failures that are detected by the decoder to be so identified – knowing that a particular block is definitely corrupted is surely useful information for the receiver! And when we distinguish between detected and undetected errors, the undetected errors give helpful insights into the low-weight codewords

of the code, which may improve the process of code design.

Turbo codes as described here have excellent performance down to decoded error probabilities of about 10^{-5}, but randomly-constructed turbo codes tend to have an *error floor* starting at that level. This error floor is caused by low-weight codewords. To reduce the height of the error floor, one can attempt to modify the random construction to increase the weight of these low-weight codewords. The tweaking of turbo codes is a black art, and it never succeeds in totalling eliminating low-weight codewords; more precisely, the low-weight codewords can be eliminated only by sacrificing the turbo code's excellent performance. In contrast, low-density parity-check codes rarely have error floors, as long as their number of weight–2 columns is not too large (cf. exercise 47.3, p.572).

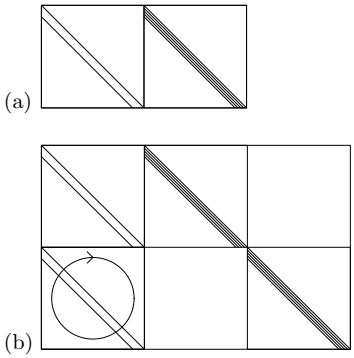


Figure 48.11. Schematic pictures of the parity-check matrices of (a) a convolutional code, rate 1/2, and (b) a turbo code, rate 1/3. Notation: A diagonal line represents an identity matrix. A band of diagonal lines represent a band of diagonal 1s. A circle inside a square represents the random permutation of all the columns in that square. A number inside a square represents the number of random permutation matrices superposed in that square. Horizontal and vertical lines indicate the boundaries of the blocks within the matrix.

▶ 48.5 Parity-check matrices of convolutional codes and turbo codes

We close by discussing the parity-check matrix of a rate-$1/2$ convolutional code viewed as a linear block code. We adopt the convention that the N bits of one block are made up of the $N/2$ bits $t^{(a)}$ followed by the $N/2$ bits $t^{(b)}$.

▷ Exercise 48.3.[2] Prove that a convolutional code has a low-density parity-check matrix as shown schematically in figure 48.11a.

Hint: It's easiest to figure out the parity constraints satisfied by a convolutional code by thinking about the nonsystematic nonrecursive encoder (figure 48.1b). Consider putting through filter a a stream that's been through convolutional filter b, and *vice versa*; compare the two resulting streams. Ignore termination of the trellises.

The parity-check matrix of a turbo code can be written down by listing the constraints satisfied by the two constituent trellises (figure 48.11b). So turbo codes are also special cases of low-density parity-check codes. If a turbo code is punctured, it no longer necessarily has a low-density parity-check matrix, but it always has a *generalized parity-check matrix* that is sparse, as explained in the next chapter.

Further reading

For further reading about convolutional codes, Johannesson and Zigangirov (1999) is highly recommended. One topic I would have liked to include is *sequential decoding*. Sequential decoding explores only the most promising paths in the trellis, and backtracks when evidence accumulates that a wrong turning has been taken. Sequential decoding is used when the trellis is too big for us to be able to apply the maximum likelihood algorithm, the min–sum algorithm. You can read about sequential decoding in Johannesson and Zigangirov (1999).

For further information about the use of the sum–product algorithm in turbo codes, and the rarely-used but highly recommended stopping criteria for halting their decoding, Frey (1998) is essential reading. (And there's lots more good stuff in the same book!)

▶ 48.6 Solutions

Solution to exercise 48.2 (p.578). The first bit was flipped. The most probable path is the upper one in figure 48.7.

49

Repeat–Accumulate Codes

In Chapter 1 we discussed a very simple and not very effective method for communicating over a noisy channel: the repetition code. We now discuss a code that is almost as simple, and whose performance is outstandingly good.

Repeat–accumulate codes were studied by Divsalar *et al.* (1998) for theoretical purposes, as simple turbo-like codes that might be more amenable to analysis than messy turbo codes. Their practical performance turned out to be just as good as other sparse-graph codes.

▶ 49.1 The encoder

1. Take K source bits.
$$s_1 s_2 s_3 \ldots s_K$$
2. Repeat each bit three times, giving $N = 3K$ bits.
$$s_1 s_1 s_1 s_2 s_2 s_2 s_3 s_3 s_3 \ldots s_K s_K s_K$$
3. Permute these N bits using a random permutation (a fixed random permutation – the same one for every codeword). Call the permuted string $\mathbf{u}$.
$$u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 \ldots u_N$$
4. Transmit the *accumulated sum*.
$$\begin{aligned} t_1 &= u_1 \\ t_2 &= t_1 + u_2 \ (\mathrm{mod}\, 2) \\ \ldots \quad t_n &= t_{n-1} + u_n \ (\mathrm{mod}\, 2) \quad \ldots \\ t_N &= t_{N-1} + u_N \ (\mathrm{mod}\, 2). \end{aligned} \tag{49.1}$$
5. That's it!

▶ 49.2 Graph

Figure 49.1a shows the graph of a repeat–accumulate code, using four types of node: equality constraints ⊟, intermediate binary variables (black circles), parity constraints ⊞, and the transmitted bits (white circles).

The source sets the values of the black bits at the bottom, three at a time, and the accumulator computes the transmitted bits along the top.

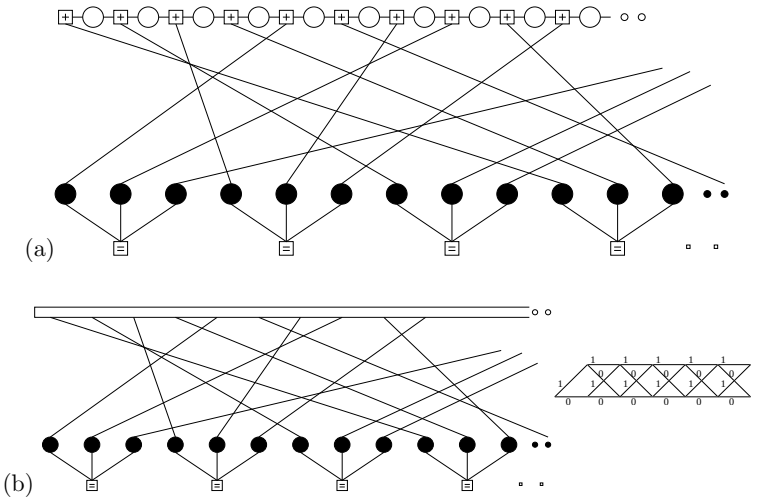


Figure 49.1. Factor graphs for a repeat–accumulate code with rate 1/3. (a) Using elementary nodes. Each white circle represents a transmitted bit. Each ⊞ constraint forces the sum of the 3 bits to which it is connected to be even. Each black circle represents an intermediate binary variable. Each ⊟ constraint forces the three variables to which it is connected to be equal.
(b) Factor graph normally used for decoding. The top rectangle represents the trellis of the accumulator, shown in the inset.

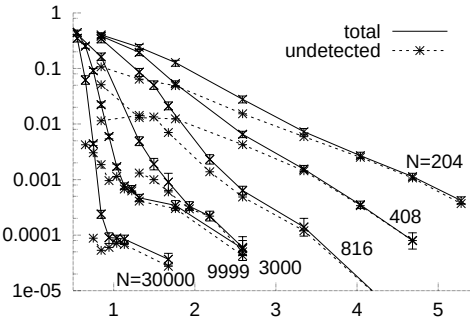


Figure 49.2. Performance of six rate-1/3 repeat–accumulate codes on the Gaussian channel. The blocklengths range from $N = 204$ to $N = 30\,000$. Vertical axis: block error probability; horizontal axis: E_b/N_0. The dotted lines show the frequency of undetected errors.

This graph is a factor graph for the prior probability over codewords, with the circles being binary variable nodes, and the squares representing two types of factor nodes. As usual, each ⊞ contributes a factor of the form $\mathbb{1}[\sum x = 0 \bmod 2]$; each ⊟ contributes a factor of the form $\mathbb{1}[x_1 = x_2 = x_3]$.

▶ 49.3 Decoding

The repeat–accumulate code is normally decoded using the sum–product algorithm on the factor graph depicted in figure 49.1b. The top box represents the trellis of the accumulator, including the channel likelihoods. In the first half of each iteration, the top trellis receives likelihoods for every transition in the trellis, and runs the forward–backward algorithm so as to produce likelihoods for each variable node. In the second half of the iteration, these likelihoods are multiplied together at the ⊟ nodes to produce new likelihood messages to send back to the trellis.

As with Gallager codes and turbo codes, the stop-when-it's-done decoding method can be applied, so it is possible to distinguish between undetected errors (which are caused by low-weight codewords in the code) and detected errors (where the decoder gets stuck and knows that it has failed to find a valid answer).

Figure 49.2 shows the performance of six randomly-constructed repeat–accumulate codes on the Gaussian channel. If one does not mind the error floor which kicks in at about a block error probability of 10^{-4}, the performance is staggeringly good for such a simple code (cf. figure 47.17).

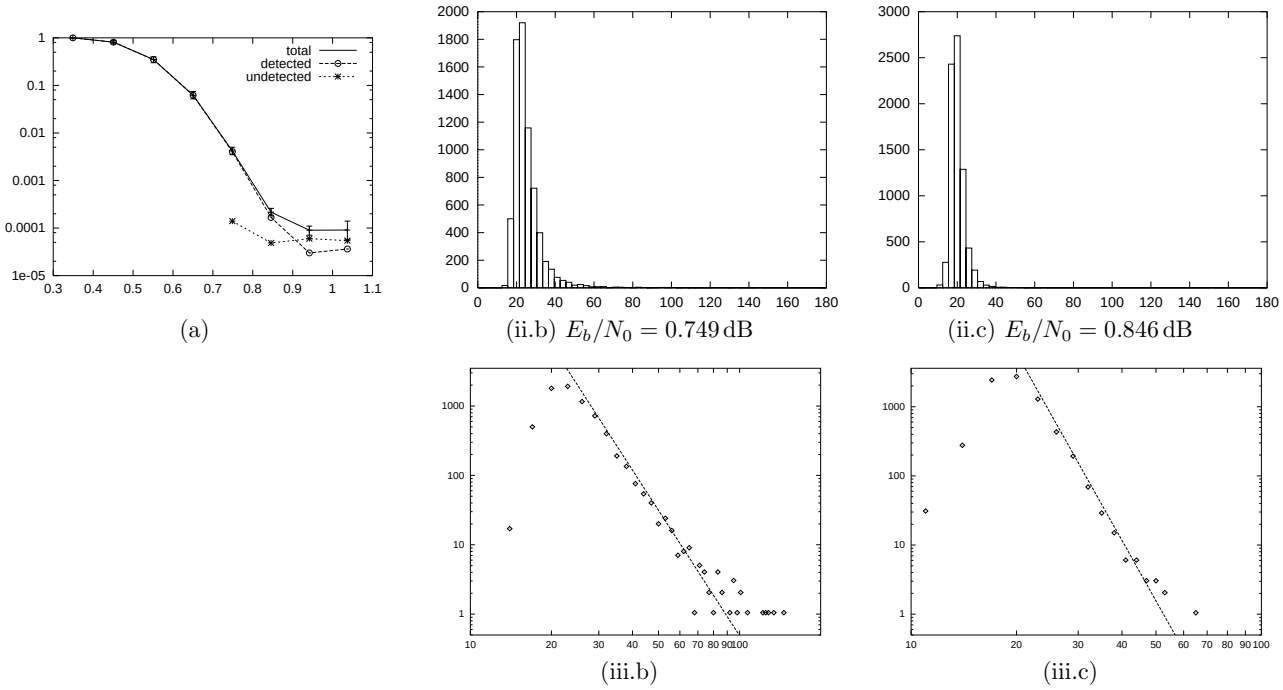


Figure 49.3. Histograms of number of iterations to find a valid decoding for a repeat–accumulate code with source block length $K = 10\,000$ and transmitted blocklength $N = 30\,000$. (a) Block error probability versus signal-to-noise ratio for the RA code. (ii.b) Histogram for $x/\sigma = 0.89$, $E_b/N_0 = 0.749\,\mathrm{dB}$. (ii.c) $x/\sigma = 0.90$, $E_b/N_0 = 0.846\,\mathrm{dB}$. (iii.b, iii.c) Fits of power laws to (ii.b) $(1/\tau^6)$ and (ii.c) $(1/\tau^9)$.

▶ 49.4 Empirical distribution of decoding times

It is interesting to study the number of iterations τ of the sum–product algorithm required to decode a sparse-graph code. Given one code and a set of channel conditions, the decoding time varies randomly from trial to trial. We find that the histogram of decoding times follows a power law, $P(\tau) \propto \tau^{-p}$, for large τ. The power p depends on the signal-to-noise ratio and becomes smaller (so that the distribution is more heavy-tailed) as the signal-to-noise ratio decreases. We have observed power laws in repeat–accumulate codes and in irregular and regular Gallager codes. Figures 49.3(ii) and (iii) show the distribution of decoding times of a repeat–accumulate code at two different signal-to-noise ratios. The power laws extend over several orders of magnitude.

Exercise 49.1.[5] Investigate these power laws. Does density evolution predict them? Can the design of a code be used to manipulate the power law in a useful way?

▶ 49.5 Generalized parity-check matrices

I find that it is helpful when relating sparse-graph codes to each other to use a common representation for them all. Forney (2001) introduced the idea of a *normal graph* in which the only nodes are ⊞ and ⊟ and all variable nodes have degree one or two; variable nodes with degree two can be represented on edges that connect a ⊞ node to a ⊟ node. The *generalized parity-check matrix* is a graphical way of representing normal graphs. In a parity-check matrix, the columns are transmitted bits, and the rows are linear constraints. In a generalized parity-check matrix, additional columns may be included, which represent state variables that are not transmitted. One way of thinking of these state variables is that they are punctured from the code before transmission.

State variables are indicated by a horizontal line above the corresponding columns. The other pieces of diagrammatic notation for generalized parity-

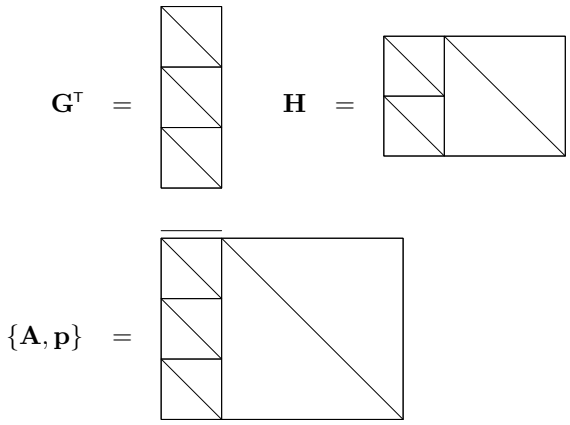


Figure 49.4. The generator matrix, parity-check matrix, and a generalized parity-check matrix of a repetition code with rate $1/3$.

check matrices are, as in (MacKay, 1999b; MacKay *et al.*, 1998):

- A diagonal line in a square indicates that that part of the matrix contains an identity matrix.
- Two or more parallel diagonal lines indicate a band-diagonal matrix with a corresponding number of 1s per row.
- A horizontal ellipse with an arrow on it indicates that the corresponding columns in a block are randomly permuted.
- A vertical ellipse with an arrow on it indicates that the corresponding rows in a block are randomly permuted.
- An integer surrounded by a circle represents that number of superposed random permutation matrices.

Definition. A generalized parity-check matrix is a pair $\{\mathbf{A}, \mathbf{p}\}$, where $\mathbf{A}$ is a binary matrix and $\mathbf{p}$ is a list of the punctured bits. The matrix defines a set of *valid vectors* $\mathbf{x}$, satisfying

$$\mathbf{A}\mathbf{x} = 0; \tag{49.2}$$

for each valid vector there is a codeword $\mathbf{t}(\mathbf{x})$ that is obtained by puncturing from $\mathbf{x}$ the bits indicated by $\mathbf{p}$. For any one code there are many generalized parity-check matrices.

The *rate* of a code with generalized parity-check matrix $\{\mathbf{A}, \mathbf{p}\}$ can be estimated as follows. If $\mathbf{A}$ is $L \times M'$, and $\mathbf{p}$ punctures S bits and selects N bits for transmission ($L = N + S$), then the effective number of constraints on the codeword, M, is

$$M = M' - S, \tag{49.3}$$

the number of source bits is

$$K = N - M = L - M', \tag{49.4}$$

and the rate is greater than or equal to

$$R = 1 - \frac{M}{N} = 1 - \frac{M' - S}{L - S}. \tag{49.5}$$

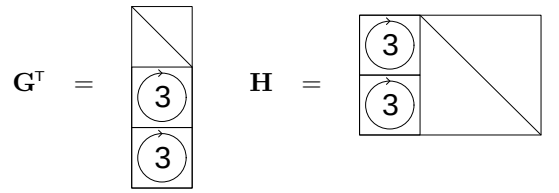


Figure 49.5. The generator matrix and parity-check matrix of a systematic low-density generator-matrix code. The code has rate $1/3$.

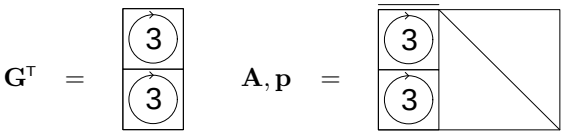


Figure 49.6. The generator matrix and generalized parity-check matrix of a *non-systematic* low-density generator-matrix code. The code has rate $1/2$.

Examples

Repetition code. The generator matrix, parity-check matrix, and generalized parity-check matrix of a simple rate-$1/3$ repetition code are shown in figure 49.4.

Systematic low-density generator-matrix code. In an (N, K) systematic low-density generator-matrix code, there are no state variables. A transmitted codeword $\mathbf{t}$ of length N is given by

$$\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{49.6}$$

where

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P} \end{bmatrix}, \tag{49.7}$$

with $\mathbf{I}_K$ denoting the $K \times K$ identity matrix, and $\mathbf{P}$ being a very sparse $M \times K$ matrix, where $M = N - K$. The parity-check matrix of this code is

$$\mathbf{H} = [\mathbf{P}|\mathbf{I}_M]. \tag{49.8}$$

In the case of a rate-$1/3$ code, this parity-check matrix might be represented as shown in figure 49.5.

Non-systematic low-density generator-matrix code. In an (N, K) non-systematic low-density generator-matrix code, a transmitted codeword $\mathbf{t}$ of length N is given by

$$\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{49.9}$$

where $\mathbf{G}^{\mathsf{T}}$ is a very sparse $N \times K$ matrix. The generalized parity-check matrix of this code is

$$\mathbf{A} = \left[\overline{\mathbf{G}^{\mathsf{T}}}|\mathbf{I}_N\right], \tag{49.10}$$

and the corresponding generalized parity-check equation is

$$\mathbf{A}\mathbf{x} = 0, \quad \text{where } \mathbf{x} = \begin{bmatrix} \mathbf{s} \\ \mathbf{t} \end{bmatrix}. \tag{49.11}$$

Whereas the parity-check matrix of this simple code is typically a complex, dense matrix, the generalized parity-check matrix retains the underlying simplicity of the code.

In the case of a rate-$1/2$ code, this generalized parity-check matrix might be represented as shown in figure 49.6.

Low-density parity-check codes and linear MN codes. The parity-check matrix of a rate-1/3 low-density parity-check code is shown in figure 49.7a.

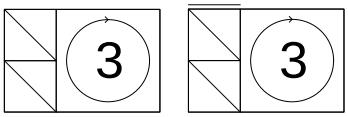
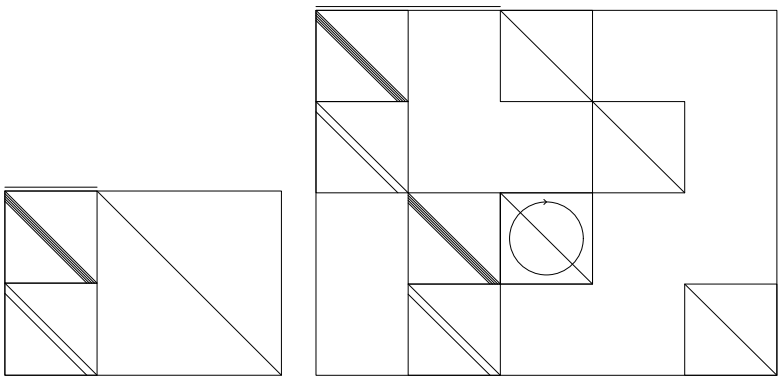


Figure 49.7. The generalized parity-check matrices of (a) a rate-$1/3$ Gallager code with $M/2$ columns of weight 2; (b) a rate-$1/2$ linear MN code.



(a) (b)

Figure 49.8. The generalized parity-check matrices of (a) a convolutional code with rate $1/2$. (b) a rate-$1/3$ turbo code built by parallel concatenation of two convolutional codes.

A linear MN code is a non-systematic low-density parity-check code. The K state bits of an MN code are the source bits. Figure 49.7b shows the generalized parity-check matrix of a rate-1/2 linear MN code.

Convolutional codes. In a non-systematic, non-recursive convolutional code, the source bits, which play the role of state bits, are fed into a delay-line and two linear functions of the delay-line are transmitted. In figure 49.8a, these two parity streams are shown as two successive vectors of length K. [It is common to interleave these two parity streams, a bit-reordering that is not relevant here, and is not illustrated.]

Concatenation. 'Parallel concatenation' of two codes is represented in one of these diagrams by aligning the matrices of two codes in such a way that the 'source bits' line up, and by adding blocks of zero-entries to the matrix such that the state bits and parity bits of the two codes occupy separate columns. An example is given by the turbo code that follows. In 'serial concatenation', the columns corresponding to the *transmitted* bits of the first code are aligned with the columns corresponding to the *source* bits of the second code.

Turbo codes. A turbo code is the parallel concatenation of two convolutional codes. The generalized parity-check matrix of a rate-1/3 turbo code is shown in figure 49.8b.

Repeat–accumulate codes. The generalized parity-check matrices of a rate-1/3 repeat–accumulate code is shown in figure 49.9. Repeat-accumulate codes are equivalent to staircase codes (section 47.7, p.569).

Intersection. The generalized parity-check matrix of the intersection of two codes is made by stacking their generalized parity-check matrices on top of each other in such a way that all the transmitted bits' columns are correctly aligned, and any punctured bits associated with the two component codes occupy separate columns.

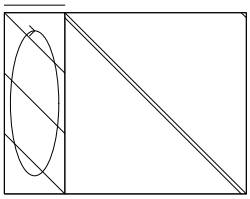


Figure 49.9. The generalized parity-check matrix of a repeat–accumulate code with rate $1/3$.

About Chapter 50

The following exercise provides a helpful background for digital fountain codes.

▷ Exercise 50.1.[3] An author proofreads his $K = 700$-page book by inspecting random pages. He makes N page-inspections, and does not take any precautions to avoid inspecting the same page twice.

(a) After $N = K$ page-inspections, what fraction of pages do you expect have never been inspected?

(b) After $N > K$ page-inspections, what is the probability that one or more pages have never been inspected?

(c) Show that in order for the probability that all K pages have been inspected to be $1 - \delta$, we require $N \simeq K \ln(K/\delta)$ page-inspections.

[This problem is commonly presented in terms of throwing N balls at random into K bins; what's the probability that every bin gets at least one ball?]

50

Digital Fountain Codes

Digital fountain codes are record-breaking sparse-graph codes for channels with erasures.

Channels with erasures are of great importance. For example, files sent over the internet are chopped into packets, and each packet is either received without error or not received. A simple channel model describing this situation is a q-ary erasure channel, which has (for all inputs in the input alphabet $\{0, 1, 2, \ldots, q{-}1\}$) a probability $1 - f$ of transmitting the input without error, and probability f of delivering the output '?'. The alphabet size q is 2^l, where l is the number of bits in a packet.

Common methods for communicating over such channels employ a feedback channel from receiver to sender that is used to control the retransmission of erased packets. For example, the receiver might send back messages that identify the *missing* packets, which are then retransmitted. Alternatively, the receiver might send back messages that acknowledge each *received* packet; the sender keeps track of which packets have been acknowledged and retransmits the others until all packets have been acknowledged.

These simple retransmission protocols have the advantage that they will work regardless of the erasure probability f, but purists who have learned their Shannon theory will feel that these retransmission protocols are wasteful. If the erasure probability f is large, the number of feedback messages sent by the first protocol will be large. Under the second protocol, it's likely that the receiver will end up receiving multiple redundant copies of some packets, and heavy use is made of the feedback channel. According to Shannon, there is no need for the feedback channel: the capacity of the forward channel is $(1 - f)l$ bits, whether or not we have feedback.

The wastefulness of the simple retransmission protocols is especially evident in the case of a broadcast channel with erasures – channels where one sender broadcasts to many receivers, and each receiver receives a random fraction $(1 - f)$ of the packets. If every packet that is missed by one or more receivers has to be retransmitted, those retransmissions will be terribly redundant. Every receiver will have already received most of the retransmitted packets.

So, we would like to make erasure-correcting codes that require no feedback or almost no feedback. The classic block codes for erasure correction are called Reed–Solomon codes. An (N, K) Reed–Solomon code (over an alphabet of size $q = 2^l$) has the ideal property that if any K of the N transmitted symbols are received then the original K source symbols can be recovered. [See Berlekamp (1968) or Lin and Costello (1983) for further information; Reed–Solomon codes exist for $N < q$.] But Reed–Solomon codes have the disadvantage that they are practical only for small K, N, and q: standard im-

plementations of encoding and decoding have a cost of order $K(N-K)\log_2 N$ packet operations. Furthermore, with a Reed–Solomon code, as with any block code, one must estimate the erasure probability f and choose the code rate $R = K/N$ *before* transmission. If we are unlucky and f is larger than expected and the receiver receives fewer than K symbols, what are we to do? We'd like a simple way to extend the code on the fly to create a lower-rate (N', K) code. For Reed–Solomon codes, no such on-the-fly method exists.

There is a better way, pioneered by Michael Luby (2002) at his company Digital Fountain, the first company whose business is based on sparse-graph codes.

The digital fountain codes I describe here, *LT codes*, were invented by Luby in 1998. The idea of a digital fountain code is as follows. The encoder is a fountain that produces an endless supply of water drops (encoded packets); let's say the original source file has a size of Kl bits, and each drop contains l encoded bits. Now, anyone who wishes to receive the encoded file holds a bucket under the fountain and collects drops until the number of drops in the bucket is a little larger than K. They can then recover the original file.

LT stands for 'Luby transform'.

Digital fountain codes are *rateless* in the sense that the number of encoded packets that can be generated from the source message is potentially limitless; and the number of encoded packets generated can be determined on the fly. Regardless of the statistics of the erasure events on the channel, we can send as many encoded packets as are needed in order for the decoder to recover the source data. The source data can be decoded from any set of K' encoded packets, for K' slightly larger than K (in practice, about 5% larger).

Digital fountain codes also have fantastically small encoding and decoding complexities. With probability $1 - \delta$, K packets can be communicated with average encoding and decoding costs both of order $K \ln(K/\delta)$ packet operations.

Luby calls these codes *universal* because they are simultaneously near-optimal for every erasure channel, and they are very efficient as the file length K grows. The overhead $K' - K$ is of order $\sqrt{K}(\ln(K/\delta))^2$.

▶ 50.1 A digital fountain's encoder

Each encoded packet t_n is produced from the source file $s_1 s_2 s_3 \ldots s_K$ as follows:

1. Randomly choose the degree d_n of the packet from a degree distribution $\rho(d)$; the appropriate choice of ρ depends on the source file size K, as we'll discuss later.

2. Choose, uniformly at random, d_n distinct input packets, and set t_n equal to the bitwise sum, modulo 2 of those d_n packets. This sum can be done by successively exclusive-or-ing the packets together.

This encoding operation defines a graph connecting encoded packets to source packets. If the mean degree $\bar{d}$ is significantly smaller than K then the graph is sparse. We can think of the resulting code as an irregular low-density generator-matrix code.

The decoder needs to know the degree of each packet that is received, and which source packets it is connected to in the graph. This information can be communicated to the decoder in various ways. For example, if the sender and receiver have synchronized clocks, they could use identical pseudo-random

number generators, seeded by the clock, to choose each random degree and each set of connections. Alternatively, the sender could pick a random key, κ_n, given which the degree and the connections are determined by a pseudo-random process, and send that key in the header of the packet. As long as the packet size l is much bigger than the key size (which need only be 32 bits or so), this key introduces only a small overhead cost.

▶ 50.2 The decoder

Decoding a sparse-graph code is especially easy in the case of an erasure channel. The decoder's task is to recover $\mathbf{s}$ from $\mathbf{t} = \mathbf{G}\mathbf{s}$, where $\mathbf{G}$ is the matrix associated with the graph. The simple way to attempt to solve this problem is by message-passing. We can think of the decoding algorithm as the sum–product algorithm if we wish, but all messages are either *completely uncertain* messages or *completely certain* messages. Uncertain messages assert that a message packet s_k could have any value, with equal probability; certain messages assert that s_k has a particular value, with probability one.

This simplicity of the messages allows a simple description of the decoding process. We'll call the encoded packets $\{t_n\}$ check nodes.

1. Find a check node t_n that is connected to *only one* source packet s_k. (If there is no such check node, this decoding algorithm halts at this point, and fails to recover all the source packets.)
   (a) Set $s_k = t_n$.
   (b) Add s_k to all checks $t_{n'}$ that are connected to s_k:
   $$t_{n'} := t_{n'} + s_k \quad \text{for all } n' \text{ such that } G_{n'k} = 1. \tag{50.1}$$
   (c) Remove all the edges connected to the source packet s_k.
2. Repeat (1) until all $\{s_k\}$ are determined.

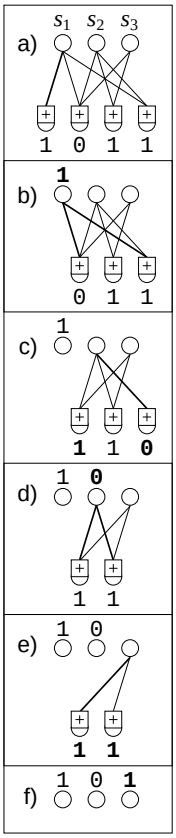


Figure 50.1. Example decoding for a digital fountain code with $K = 3$ source bits and $N = 4$ encoded bits.

This decoding process is illustrated in figure 50.1 for a toy case where each packet is just one bit. There are three source packets (shown by the upper circles) and four received packets (shown by the lower check symbols), which have the values $t_1t_2t_3t_4 = \texttt{1011}$ at the start of the algorithm.

At the first iteration, the only check node that is connected to a sole source bit is the first check node (panel a). We set that source bit s_1 accordingly (panel b), discard the check node, then add the value of s_1 (1) to the checks to which it is connected (panel c), disconnecting s_1 from the graph. At the start of the second iteration (panel c), the fourth check node is connected to a sole source bit, s_2. We set s_2 to t_4 (0, in panel d), and add s_2 to the two checks it is connected to (panel e). Finally, we find that two check nodes are both connected to s_3, and they agree about the value of s_3 (as we would hope!), which is restored in panel f.

▶ 50.3 Designing the degree distribution

The probability distribution $\rho(d)$ of the degree is a critical part of the design: occasional encoded packets must have high degree (i.e., d similar to K) in order to ensure that there are not some source packets that are connected to no-one. Many packets must have low degree, so that the decoding process

can get started, and keep going, and so that the total number of addition operations involved in the encoding and decoding is kept small. For a given degree distribution $\rho(d)$, the statistics of the decoding process can be predicted by an appropriate version of density evolution.

Ideally, to avoid redundancy, we'd like the received graph to have the property that just one check node has degree one at each iteration. At each iteration, when this check node is processed, the degrees in the graph are reduced in such a way that one new degree-one check node appears. *In expectation*, this ideal behaviour is achieved by the *ideal soliton distribution*,

$$\begin{aligned} \rho(1) &= 1/K \\ \rho(d) &= \frac{1}{d(d-1)} \quad \text{for } d = 2, 3, \ldots, K. \end{aligned} \tag{50.2}$$

The expected degree under this distribution is roughly $\ln K$.

▷ Exercise 50.2.[2] Derive the ideal soliton distribution. At the first iteration ($t = 0$) let the number of packets of degree d be $h_0(d)$; show that (for $d > 1$) the expected number of packets of degree d that have their degree reduced to $d - 1$ is $h_0(d)d/K$; and at the tth iteration, when t of the K packets have been recovered and the number of packets of degree d is $h_t(d)$, the expected number of packets of degree d that have their degree reduced to $d - 1$ is $h_t(d)d/(K - t)$. Hence show that in order to have the expected number of packets of degree 1 satisfy $h_t(1) = 1$ for all $t \in \{0, \ldots K - 1\}$, we must to start with have $h_0(1) = 1$ and $h_0(2) = K/2$; and more generally, $h_t(2) = (K - t)/2$; then by recursion solve for $h_0(d)$ for $d = 3$ upwards.

This degree distribution works poorly in practice, because fluctuations around the expected behaviour make it very likely that at some point in the decoding process there will be no degree-one check nodes; and, furthermore, a few source nodes will receive no connections at all. A small modification fixes these problems.

The *robust soliton distribution* has two extra parameters, c and δ; it is designed to ensure that the expected number of degree-one checks is about

$$S \equiv c \ln(K/\delta)\sqrt{K}, \tag{50.3}$$

rather than 1, throughout the decoding process. The parameter δ is a bound on the probability that the decoding fails to run to completion after a certain number K' of packets have been received. The parameter c is a constant of order 1, if our aim is to prove Luby's main theorem about LT codes; in practice however it can be viewed as a free parameter, with a value somewhat smaller than 1 giving good results. We define a positive function

$$\tau(d) = \begin{cases} \frac{S}{K}\frac{1}{d} & \text{for } d = 1, 2, \ldots (K/S)-1 \\ \frac{S}{K}\ln(S/\delta) & \text{for } d = K/S \\ 0 & \text{for } d > K/S \end{cases} \tag{50.4}$$

(see figure 50.2 and exercise 50.4 (p.594)) then add the ideal soliton distribution ρ to τ and normalize to obtain the robust soliton distribution, μ:

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z}, \tag{50.5}$$

where $Z = \sum_d \rho(d) + \tau(d)$. The number of encoded packets required at the receiving end to ensure that the decoding can run to completion, with probability at least $1 - \delta$, is $K' = KZ$.

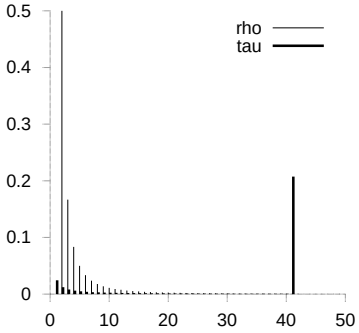


Figure 50.2. The distributions $\rho(d)$ and $\tau(d)$ for the case $K = 10\,000$, $c = 0.2$, $\delta = 0.05$, which gives $S = 244$, $K/S = 41$, and $Z \simeq 1.3$. The distribution τ is largest at $d = 1$ and $d = K/S$.

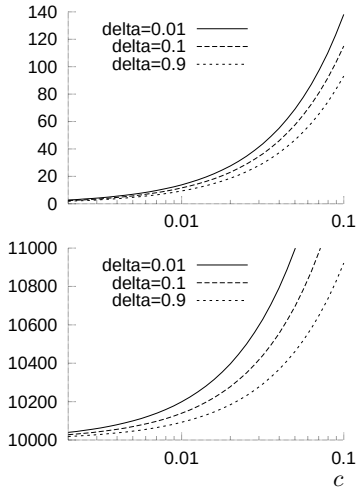


Figure 50.3. The number of degree-one checks S (upper figure) and the quantity K' (lower figure) as a function of the two parameters c and δ, for $K = 10\,000$. Luby's main theorem proves that there exists a value of c such that, given K' received packets, the decoding algorithm will recover the K source packets with probability $1 - \delta$.

Luby's (2002) analysis explains how the small-d end of τ has the role of ensuring that the decoding process gets started, and the spike in τ at $d = K/S$ is included to ensure that every source packet is likely to be connected to a check at least once. Luby's key result is that (for an appropriate value of the constant c) receiving $K' = K + 2\ln(S/\delta)S$ checks ensures that all packets can be recovered with probability at least $1 - \delta$. In the illustrative figures I have set the allowable decoder failure probability δ quite large, because the actual failure probability is much smaller than is suggested by Luby's conservative analysis.

In practice, LT codes can be tuned so that a file of original size $K \simeq 10\,000$ packets is recovered with an overhead of about 5%. Figure 50.4 shows histograms of the actual number of packets required for a couple of settings of the parameters, achieving mean overheads smaller than 5% and 10% respectively.

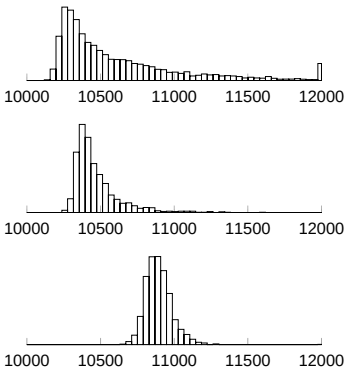


Figure 50.4. Histograms of the actual number of packets N required in order to recover a file of size $K = 10\,000$ packets. The parameters were as follows: top histogram: $c = 0.01$, $\delta = 0.5$ ($S = 10$, $K/S = 1010$, and $Z \simeq 1.01$); middle: $c = 0.03$, $\delta = 0.5$ ($S = 30$, $K/S = 337$, and $Z \simeq 1.03$); bottom: $c = 0.1$, $\delta = 0.5$ ($S = 99$, $K/S = 101$, and $Z \simeq 1.1$).

▶ 50.4 Applications

Digital fountain codes are an excellent solution in a wide variety of situations. Let's mention two.

Storage

You wish to make a backup of a large file, but you are aware that your magnetic tapes and hard drives are all unreliable in the sense that catastrophic failures, in which some stored packets are permanently lost within one device, occur at a rate of something like 10^{-3} per day. How should you store your file?

A digital fountain can be used to spray encoded packets all over the place, on every storage device available. Then to recover the backup file, whose size was K packets, one simply needs to find $K' \simeq K$ packets from anywhere. Corrupted packets do not matter; we simply skip over them and find more packets elsewhere.

This method of storage also has advantages in terms of *speed* of file recovery. In a hard drive, it is standard practice to store a file in successive sectors of a hard drive, to allow rapid reading of the file; but if, as occasionally happens, a packet is lost (owing to the reading head being off track for a moment, giving a burst of errors that cannot be corrected by the packet's error-correcting code), a whole revolution of the drive must be performed to bring back the packet to the head for a second read. The time taken for one revolution produces an undesirable delay in the file system.

If files were instead stored using the digital fountain principle, with the digital drops stored in one or more consecutive sectors on the drive, then one would never need to endure the delay of re-reading a packet; packet loss would become less important, and the hard drive could consequently be operated faster, with higher noise level, and with fewer resources devoted to noisy-channel coding.

▷ Exercise 50.3.[2] Compare the digital fountain method of robust storage on multiple hard drives with RAID (the redundant array of independent disks).

Broadcast

Imagine that ten thousand subscribers in an area wish to receive a digital movie from a broadcaster. The broadcaster can send the movie in packets

over a broadcast network – for example, by a wide-bandwidth phone line, or by satellite.

Imagine that not all packets are received at all the houses. Let's say $f = 0.1\%$ of them are lost at each house. In a standard approach in which the file is transmitted as a plain sequence of packets with no encoding, each house would have to notify the broadcaster of the fK missing packets, and request that they be retransmitted. And with ten thousand subscribers all requesting such retransmissions, there would be a retransmission request for almost every packet. Thus the broadcaster would have to repeat the entire broadcast twice in order to ensure that most subscribers have received the whole movie, and most users would have to wait roughly twice as long as the ideal time before the download was complete.

If the broadcaster uses a digital fountain to encode the movie, each subscriber can recover the movie from *any* $K' \simeq K$ packets. So the broadcast needs to last for only, say, $1.1K$ packets, and every house is very likely to have successfully recovered the whole file.

Another application is broadcasting data to cars. Imagine that we want to send updates to in-car navigation databases by satellite. There are hundreds of thousands of vehicles, and they can receive data only when they are out on the open road; there are no feedback channels. A standard method for sending the data is to put it in a *carousel*, broadcasting the packets in a fixed periodic sequence. 'Yes, a car may go through a tunnel, and miss out on a few hundred packets, but it will be able to collect those missed packets an hour later when the carousel has gone through a full revolution (we hope); or maybe the following day...'

If instead the satellite uses a digital fountain, each car needs to receive only an amount of data equal to the original file size (plus 5%).

Further reading

The encoders and decoders sold by Digital Fountain have even higher efficiency than the LT codes described here, and they work well for all blocklengths, not only large lengths such as $K \gtrsim 10\,000$. Shokrollahi (2003) presents *raptor codes*, which are an extension of LT codes with linear-time encoding and decoding.

▶ 50.5 Further exercises

▷ Exercise 50.4.[2] Understanding the robust soliton distribution.

Repeat the analysis of exercise 50.2 (p.592) but now aim to have the expected number of packets of degree 1 be $h_t(1) = 1 + S$ for all t, instead of 1. Show that the initial required number of packets is

$$h_0(d) = \frac{K}{d(d-1)} + \frac{S}{d} \quad \text{for } d > 1. \tag{50.6}$$

The reason for truncating the second term beyond $d = K/S$ and replacing it by the spike at $d = K/S$ (see equation (50.4)) is to ensure that the decoding complexity does not grow larger than $O(K \ln K)$.

Estimate the expected number of packets $\sum_d h_0(d)$ and the expected number of edges in the sparse graph $\sum_d h_0(d)d$ (which determines the decoding complexity) if the histogram of packets is as given in (50.6). Compare with the expected numbers of packets and edges when the robust soliton distribution (50.4) is used.

Exercise 50.5.[4] Show that the spike at $d = K/S$ (equation (50.4)) is an adequate replacement for the tail of high-weight packets in (50.6).

Exercise 50.6.[3C] Investigate experimentally how necessary the spike at $d = K/S$ (equation (50.4)) is for successful decoding. Investigate also whether the tail of $\rho(d)$ beyond $d = K/S$ is necessary. What happens if all high-weight degrees are removed, both the spike at $d = K/S$ and the tail of $\rho(d)$ beyond $d = K/S$?

Exercise 50.7.[4] Fill in the details in the proof of Luby's main theorem, that receiving $K' = K + 2\ln(S/\delta)S$ checks ensures that all the source packets can be recovered with probability at least $1 - \delta$.

Exercise 50.8.[4C] Optimize the degree distribution of a digital fountain code for a file of $K = 10\,000$ packets. Pick a sensible objective function for your optimization, such as minimizing the mean of N, the number of packets required for complete decoding, or the 95th percentile of the histogram of N (figure 50.4).

▷ Exercise 50.9.[3] Make a model of the situation where a data stream is broadcast to cars, and quantify the advantage that the digital fountain has over the carousel method.

Exercise 50.10.[2] Construct a simple example to illustrate the fact that the digital fountain decoder of section 50.2 is suboptimal – it sometimes gives up even though the information available is sufficient to decode the whole file. How does the cost of the optimal decoder compare?

▷ Exercise 50.11.[2] If every transmitted packet were created by adding together source packets at random with probability $1/2$ of each source packet's being included, show that the probability that $K' = K$ received packets suffice for the optimal decoder to be able to recover the K source packets is just a little below $1/2$. [To put it another way, what is the probability that a random $K \times K$ matrix has full rank?]

Show that if $K' = K + \Delta$ packets are received, the probability that they will not suffice for the optimal decoder is roughly $2^{-\Delta}$.

▷ Exercise 50.12.[4C] Implement an optimal digital fountain decoder that uses the method of Richardson and Urbanke (2001b) derived for fast *encoding* of sparse-graph codes (section 47.7) to handle the matrix inversion required for optimal decoding. Now that you have changed the decoder, you can reoptimize the degree distribution, using higher-weight packets. By how much can you reduce the overhead? Confirm the assertion that this approach makes digital fountain codes viable as erasure-correcting codes for all blocklengths, not just the large blocklengths for which LT codes are excellent.

▷ Exercise 50.13.[5] Digital fountain codes are excellent rateless codes for erasure channels. Make a rateless code for a channel that has both erasures and *noise*.

▶ 50.6 Summary of sparse-graph codes

A simple method for designing error-correcting codes for noisy channels, first pioneered by Gallager (1962), has recently been rediscovered and generalized, and communication theory has been transformed. The practical performance of Gallager's low-density parity-check codes and their modern cousins is vastly better than the performance of the codes with which textbooks have been filled in the intervening years.

Which sparse-graph code is 'best' for a noisy channel depends on the chosen rate and blocklength, the permitted encoding and decoding complexity, and the question of whether occasional undetected errors are acceptable. Low-density parity-check codes are the most versatile; it's easy to make a competitive low-density parity-check code with almost any rate and blocklength, and low-density parity-check codes virtually never make undetected errors.

For the special case of the erasure channel, the sparse-graph codes that are best are digital fountain codes.

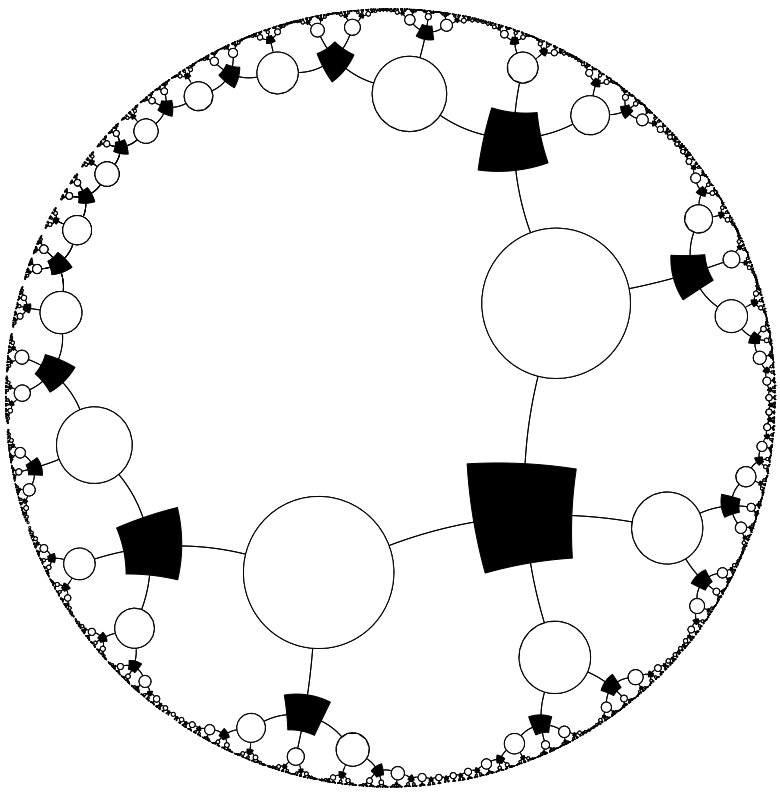
▶ 50.7 Conclusion

The best solution to the communication problem is:

> Combine a simple, pseudo-random code
> with a message-passing decoder.

Part VII

Appendices



A

Notation

What does $P(A \mid B, C)$ mean? $P(A \mid B, C)$ is pronounced 'the probability that A is true *given that* B is true *and* C is true'. Or, more briefly, 'the probability of A given B and C'. (See Chapter 2, p.22.)

What do log **and** ln **mean?** In this book, $\log x$ means the base-two logarithm, $\log_2 x$; $\ln x$ means the natural logarithm, $\log_e x$.

What does $\hat{s}$ mean? Usually, a 'hat' over a variable denotes a guess or estimator. So $\hat{s}$ is a guess at the value of s.

Integrals. There is no difference between $\int f(u)\,\mathrm{d}u$ and $\int \mathrm{d}u\, f(u)$. The integrand is $f(u)$ in both cases.

What does $\prod_{n=1}^{N}$ mean? This is like the summation $\sum_{n=1}^{N}$ but it denotes a product. It's pronounced 'product over n from 1 to N'. So, for example,

$$\prod_{n=1}^{N} n = 1 \times 2 \times 3 \times \cdots \times N = N! = \exp\left[\sum_{n=1}^{N} \ln n\right]. \tag{A.1}$$

I like to choose the name of the free variable in a sum or a product – here, n – to be the lower case version of the range of the sum. So n usually runs from 1 to N, and m usually runs from 1 to M. This is a habit I learnt from Yaser Abu-Mostafa, and I think it makes formulae easier to understand.

What does $\binom{N}{n}$ mean? This is pronounced 'N choose n', and it is the number of ways of selecting an unordered set of n objects from a set of size N.

$$\binom{N}{n} = \frac{N!}{(N-n)!\,n!}. \tag{A.2}$$

This function is known as the combination function.

What is $\Gamma(x)$? The *gamma function* is defined by $\Gamma(x) \equiv \int_0^\infty \mathrm{d}u\; u^{x-1} e^{-u}$, for $x > 0$. The gamma function is an extension of the factorial function to real number arguments. In general, $\Gamma(x+1) = x\Gamma(x)$, and for integer arguments, $\Gamma(x+1) = x!$. The digamma function is defined by $\Psi(x) \equiv \frac{\mathrm{d}}{\mathrm{d}x} \ln \Gamma(x)$.

For large x (for practical purposes, $0.1 \leq x \leq \infty$),

$$\ln \Gamma(x) \simeq \left(x - \tfrac{1}{2}\right) \ln(x) - x + \tfrac{1}{2} \ln 2\pi + O(1/x); \tag{A.3}$$

and for small x (for practical purposes, $0 \leq x \leq 0.5$):

$$\ln \Gamma(x) \simeq \ln \frac{1}{x} - \gamma_e x + O(x^2) \tag{A.4}$$

where γ_e is Euler's constant.

What does $H_2^{-1}(1 - R/C)$ mean? Just as $\sin^{-1}(s)$ denotes the inverse function to $s = \sin(x)$, so $H_2^{-1}(h)$ is the inverse function to $h = H_2(x)$.

There is potential confusion when people use $\sin^2 x$ to denote $(\sin x)^2$, since then we might expect $\sin^{-1} s$ to denote $1/\sin(s)$; I therefore like to avoid using the notation $\sin^2 x$.

What does $f'(x)$ mean? The answer depends on the context. Often, a 'prime' is used to denote differentiation:

$$f'(x) \equiv \frac{\mathrm{d}}{\mathrm{d}x} f(x); \tag{A.5}$$

similarly, a dot denotes differentiation with respect to time, t:

$$\dot{x} \equiv \frac{\mathrm{d}}{\mathrm{d}t} x. \tag{A.6}$$

However, the prime is also a useful indicator for 'another variable', for example 'a new value for a variable'. So, for example, x' might denote 'the new value of x'. Also, if there are two integers that both range from 1 to N, I will often name those integers n and n'.

So my rule is: if a prime occurs in an expression that could be a function, such as $f'(x)$ or $h'(y)$, then it denotes differentiation; otherwise it indicates 'another variable'.

What is the error function? Definitions of this function vary. I define it to be the cumulative probability of a standard (variance = 1) normal distribution,

$$\Phi(z) \equiv \int_{-\infty}^{z} \exp(-z^2/2)/\sqrt{2\pi}\, \mathrm{d}z. \tag{A.7}$$

What does $\mathcal{E}(r)$ mean? $\mathcal{E}[r]$ is pronounced 'the expected value of r' or 'the expectation of r', and it is the mean value of r. Another symbol for 'expected value' is the pair of angle-brackets, $\langle r \rangle$.

What does $|x|$ mean? The vertical bars '$|\cdot|$' have two meanings. If $\mathcal{A}$ is a set, then $|\mathcal{A}|$ denotes the number of elements in the set; if x is a number, then $|x|$ is the absolute value of x.

What does $[\mathbf{A}|\mathbf{P}]$ mean? Here, $\mathbf{A}$ and $\mathbf{P}$ are matrices with the same number of rows. $[\mathbf{A}|\mathbf{P}]$ denotes the double-width matrix obtained by putting $\mathbf{A}$ alongside $\mathbf{P}$. The vertical bar is used to avoid confusion with the product $\mathbf{AP}$.

What does $\mathbf{x}^{\mathsf{T}}$ mean? The superscript T is pronounced 'transpose'. Transposing a row-vector turns it into a column vector:

$$(1, 2, 3)^{\mathsf{T}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \tag{A.8}$$

and *vice versa*. [Normally my vectors, indicated by bold face type ($\mathbf{x}$), are column vectors.]

Similarly, matrices can be transposed. If M_{ij} is the entry in row i and column j of matrix $\mathbf{M}$, and $\mathbf{N} = \mathbf{M}^{\mathsf{T}}$, then $N_{ji} = M_{ij}$.

What are Trace $\mathbf{M}$ **and** det $\mathbf{M}$**?** The trace of a matrix is the sum of its diagonal elements,

$$\text{Trace}\,\mathbf{M} = \sum_i M_{ii}. \tag{A.9}$$

The determinant of $\mathbf{M}$ is denoted $\det \mathbf{M}$.

What does δ_{mn} mean? The δ matrix is the identity matrix.

$$\delta_{mn} = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n. \end{cases}$$

Another name for the identity matrix is $\mathbf{I}$ or $\mathbf{1}$. Sometimes I include a subscript on this symbol – $\mathbf{1}_K$ – which indicates the size of the matrix $(K \times K)$.

What does $\delta(x)$ mean? The delta function has the property

$$\int \mathrm{d}x\, f(x)\delta(x) = f(0). \tag{A.10}$$

Another possible meaning for $\delta(S)$ is the truth function, which is 1 if the proposition S is true but I have adopted another notation for that. After all, the symbol δ is quite busy already, with the two roles mentioned above in addition to its role as a small real number δ and an increment operator (as in δx)!

What does $\mathbb{1}[S]$ mean? $\mathbb{1}[S]$ is the truth function, which is 1 if the proposition S is true and 0 otherwise. For example, the number of positive numbers in the set $T = \{-2, 1, 3\}$ can be written

$$\sum_{x \in T} \mathbb{1}[x > 0]. \tag{A.11}$$

What is the difference between ':=' and '='? In an algorithm, $x := y$ means that the variable x is updated by assigning it the value of y.

In contrast, $x = y$ is a proposition, a statement that x is equal to y.

See Chapters 23 and 29 for further definitions and notation relating to probability distributions.

B

Some Physics

B.1 About phase transitions

A system with states $\mathbf{x}$ in contact with a heat bath at temperature $T = 1/\beta$ has probability distribution

$$P(\mathbf{x} \,|\, \beta) = \frac{1}{Z(\beta)} \exp(-\beta E(\mathbf{x})). \tag{B.1}$$

The partition function is

$$Z(\beta) = \sum_{\mathbf{x}} \exp(-\beta E(\mathbf{x})). \tag{B.2}$$

The inverse temperature β can be interpreted as defining an exchange rate between entropy and energy. $(1/\beta)$ is the amount of energy that must be given to a heat bath to increase its entropy by one nat.

Often, the system will be affected by some other parameters such as the volume of the box it is in, V, in which case Z is a function of V too, $Z(\beta, V)$.

For any system with a finite number of states, the function $Z(\beta)$ is evidently a continuous function of β, since it is simply a sum of exponentials. Moreover, all the derivatives of $Z(\beta)$ with respect to β are continuous too.

What phase transitions are all about, however, is this: phase transitions correspond to values of β and V (called critical points) at which the derivatives of Z have discontinuities or divergences.

Immediately we can deduce:

> Only systems with an infinite number of states can show phase transitions.

Often, we include a parameter N describing the size of the system. Phase transitions may appear in the limit $N \to \infty$. Real systems may have a value of N like 10^{23}.

If we make the system large by simply grouping together N independent systems whose partition function is $Z_{(1)}(\beta)$, then nothing interesting happens. The partition function for N independent identical systems is simply

$$Z_{(N)}(\beta) = [Z_{(1)}(\beta)]^N. \tag{B.3}$$

Now, while this function $Z_{(N)}(\beta)$ may be a very rapidly varying function of β, that doesn't mean it is showing phase transitions. The natural way to look at the partition function is in the logarithm

$$\ln Z_{(N)}(\beta) = N \ln Z_{(1)}(\beta). \tag{B.4}$$

Duplicating the original system N times simply scales up all properties like the energy and heat capacity of the system by a factor of N. So if the original system showed no phase transitions then the scaled up system won't have any either.

Only systems with long-range correlations show phase transitions.

Long-range correlations do not require long-range energetic couplings; for example, a magnet has only short-range couplings (between adjacent spins) but these are sufficient to create long-range order.

Why are points at which derivatives diverge interesting?

The derivatives of $\ln Z$ describe properties like the heat capacity of the system (that's the second derivative) or its fluctuations in energy. If the second derivative of $\ln Z$ diverges at a temperature $1/\beta$, then the heat capacity of the system diverges there, which means it can absorb or release energy without changing temperature (think of ice melting in ice water); when the system is at equilibrium at that temperature, its energy fluctuates a lot, in contrast to the normal law-of-large-numbers behaviour, where the energy only varies by one part in $\sqrt{N}$.

A toy system that shows a phase transition

Imagine a collection of N coupled spins that have the following energy as a function of their state $\mathbf{x} \in \{0,1\}^N$.

$$E(\mathbf{x}) = \begin{cases} -N\epsilon & \mathbf{x} = (0,0,0,\ldots,0) \\ 0 & \text{otherwise.} \end{cases} \tag{B.5}$$

This energy function describes a ground state in which all the spins are aligned in the zero direction; the energy per spin in this state is $-\epsilon$. if any spin changes state then the energy is zero. This model is like an extreme version of a magnetic interaction, which encourages pairs of spins to be aligned.

We can contrast it with an ordinary system of N *independent* spins whose energy is:

$$E^0(\mathbf{x}) = \epsilon \sum_n (2x_n - 1). \tag{B.6}$$

Like the first system, the system of independent spins has a single ground state $(0,0,0,\ldots,0)$ with energy $-N\epsilon$, and it has roughly 2^N states with energy very close to 0, so the low-temperature and high-temperature properties of the independent-spin system and the coupled-spin system are virtually identical.

The partition function of the coupled-spin system is

$$Z(\beta) = e^{\beta N\epsilon} + 2^N - 1. \tag{B.7}$$

The function

$$\ln Z(\beta) = \ln \left(e^{\beta N\epsilon} + 2^N - 1 \right) \tag{B.8}$$

is sketched in figure B.1a along with its low temperature behaviour,

$$\ln Z(\beta) \simeq N\beta\epsilon, \quad \beta \to \infty, \tag{B.9}$$

and its high temperature behaviour,

$$\ln Z(\beta) \simeq N \ln 2, \quad \beta \to 0. \tag{B.10}$$

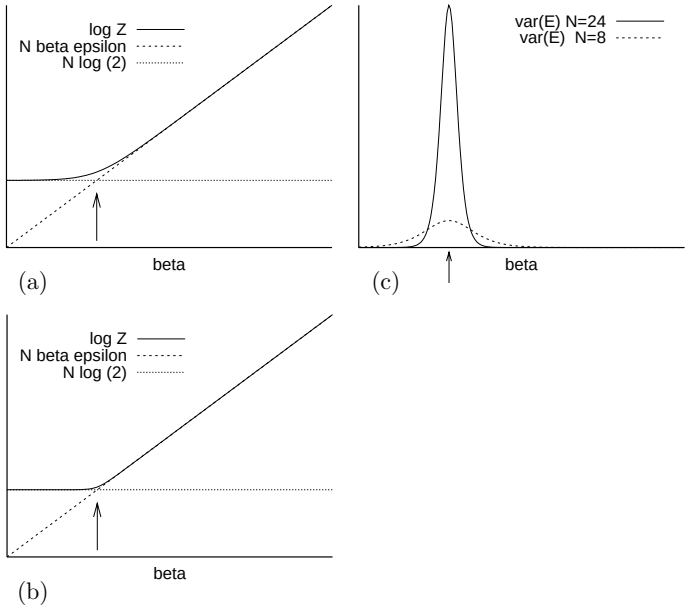


Figure B.1. (a) Partition function of toy system which shows a phase transition for large N. The arrow marks the point $\beta_c = \log 2/\epsilon$. (b) The same, for larger N. (c) The variance of the energy of the system as a function of β for two system sizes. As N increases the variance has an increasingly sharp peak at the critical point β_c. Contrast with figure B.2.

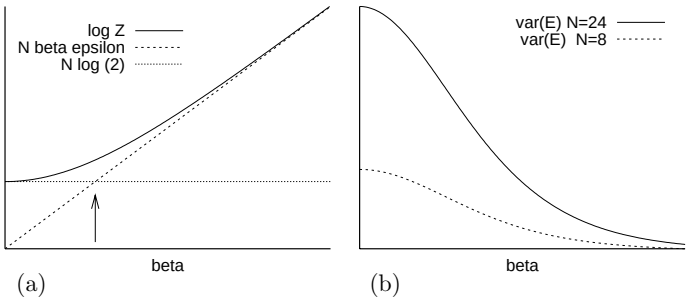


Figure B.2. The partition function (a) and energy-variance (b) of a system consisting of N independent spins. The partition function changes gradually from one asymptote to the other, regardless of how large N is; the variance of the energy does not have a peak. The fluctuations are largest at high temperature (small β) and scale linearly with system size N.

The arrow marks the point

$$\beta = \frac{\ln 2}{\epsilon} \tag{B.11}$$

at which these two asymptotes intersect. In the limit $N \to \infty$, the graph of $\ln Z(\beta)$ becomes more and more sharply bent at this point (figure B.1b).

The second derivative of $\ln Z$, which describes the variance of the energy of the system, has a peak value, at $\beta = \ln 2/\epsilon$, roughly equal to

$$\frac{N^2\epsilon^2}{4}, \tag{B.12}$$

which corresponds to the system spending half of its time in the ground state and half its time in the other states.

At this critical point, the heat capacity of this system is thus proportional to N^2; the heat capacity per spin is proportional to N, which, for infinite N, is infinite, in contrast to the behaviour of systems away from phase transitions, whose capacity per atom is a finite number.

For comparison, figure B.2 shows the partition function and energy-variance of the ordinary independent-spin system.

More generally

Phase transitions can be categorized into 'first-order' and 'continuous' transitions. In a first-order phase transition, there is a discontinuous change of one

or more order-parameters; in a continuous transition, all order-parameters change continuously. [What's an order-parameter? – a scalar function of the state of the system; or, to be precise, the expectation of such a function.]

In the vicinity of a critical point, the concept of 'typicality' defined in Chapter 4 does not hold. For example, our toy system, at its critical point, has a 50% chance of being in a state with energy $-N\epsilon$, and roughly a $1/2^{N+1}$ chance of being in each of the other states that have energy zero. It is thus not the case that $\ln 1/P(\mathbf{x})$ is very likely to be close to the entropy of the system at this point, unlike a system with N i.i.d. components.

Remember that information content ($\ln 1/P(\mathbf{x})$) and energy are very closely related. If typicality holds, then the system's energy has negligible fluctuations, and *vice versa*.

C

Some Mathematics

C.1 Finite field theory

Most linear codes are expressed in the language of Galois theory

Why are Galois fields an appropriate language for linear codes? First, a definition and some examples.

A field F is a set $F = \{0, F'\}$ such that

1. F forms an Abelian group under an addition operation '+', with 0 being the identity; [Abelian means all elements commute, i.e., satisfy $a + b = b + a$.]
2. F' forms an Abelian group under a multiplication operation '$\cdot$'; multiplication of any element by 0 yields 0;
3. these operations satisfy the distributive rule $(a + b) \cdot c = a \cdot c + b \cdot c$.

For example, the real numbers form a field, with '+' and '$\cdot$' denoting ordinary addition and multiplication.

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Table C.1. Addition and multiplication tables for $GF(2)$.

A Galois field $GF(q)$ is a field with a finite number of elements q.

A unique Galois field exists for any $q = p^m$, where p is a prime number and m is a positive integer; there are no other finite fields.

$GF(2)$. The addition and multiplication tables for $GF(2)$ are shown in table C.1. These are the rules of addition and multiplication modulo 2.

$GF(p)$. For any prime number p, the addition and multiplication rules are those for ordinary addition and multiplication, modulo p.

$GF(4)$. The rules for $GF(p^m)$, with $m > 1$, are *not* those of ordinary addition and multiplication. For example the tables for $GF(4)$ (table C.2) are *not* the rules of addition and multiplication modulo 4. Notice that $1 + 1 = 0$, for example. So how can $GF(4)$ be described? It turns out that the elements can be related to *polynomials*. Consider polynomial functions of x of degree 1 and with coefficients that are elements of $GF(2)$. The polynomials shown in table C.3 obey the addition and multiplication rules of $GF(4)$ *if* addition and multiplication are modulo the polynomial $x^2 + x + 1$, and the coefficients of the polynomials are from $GF(2)$. For example, $B \cdot B = x^2 + (1 + 1)x + 1 = x = A$. Each element may also be represented as a bit pattern as shown in table C.3, with addition being bitwise modulo 2, and multiplication defined with an appropriate carry operation.

$+$	0	1	A	B
0	0	1	A	B
1	1	0	B	A
A	A	B	0	1
B	B	A	1	0

$\cdot$	0	1	A	B
0	0	0	0	0
1	0	1	A	B
A	0	A	B	1
B	0	B	1	A

Table C.2. Addition and multiplication tables for $GF(4)$.

Element	Polynomial	Bit pattern
0	0	00
1	1	01
A	x	10
B	$x + 1$	11

Table C.3. Representations of the elements of $GF(4)$.

$GF(8)$. We can denote the elements of $GF(8)$ by $\{0, 1, A, B, C, D, E, F\}$. Each element can be mapped onto a polynomial over $GF(2)$. The multiplication and addition operations are given by multiplication and addition of the polynomials, modulo $x^3 + x + 1$. The multiplication table is given below.

element	polynomial	binary representation
0	0	000
1	1	001
A	x	010
B	$x+1$	011
C	x^2	100
D	x^2+1	101
E	x^2+x	110
F	x^2+x+1	111

$\cdot$	0	1	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0
1	0	1	A	B	C	D	E	F
A	0	A	C	E	B	1	F	D
B	0	B	E	D	F	C	1	A
C	0	C	B	F	E	A	D	1
D	0	D	1	C	A	F	B	E
E	0	E	F	1	D	B	A	C
F	0	F	D	A	1	E	C	B

Why are Galois fields relevant to linear codes? Imagine generalizing a binary generator matrix $\mathbf{G}$ and binary vector $\mathbf{s}$ to a matrix and vector with elements from a larger set, and generalizing the addition and multiplication operations that define the product $\mathbf{Gs}$. In order to produce an appropriate input for a symmetric channel, it would be convenient if, for random $\mathbf{s}$, the product $\mathbf{Gs}$ produced all elements in the enlarged set with equal probability. This uniform distribution is easiest to guarantee if these elements form a group under both addition and multiplication, because then these operations do not break the symmetry among the elements. When two random elements of a multiplicative group are multiplied together, all elements are produced with equal probability. This is not true of other sets such as the integers, for which the multiplication operation is more likely to give rise to some elements (the composite numbers) than others. Galois fields, by their definition, avoid such symmetry-breaking effects.

▶ C.2 Eigenvectors and eigenvalues

A *right-eigenvector* of a square matrix $\mathbf{A}$ is a non-zero vector $\mathbf{e}_{\mathsf{R}}$ that satisfies

$$\mathbf{A}\mathbf{e}_{\mathsf{R}} = \lambda \mathbf{e}_{\mathsf{R}}, \tag{C.1}$$

where λ is the eigenvalue associated with that eigenvector. The eigenvalue may be a real number or complex number and it may be zero. Eigenvectors may be real or complex.

A *left-eigenvector* of a matrix $\mathbf{A}$ is a vector $\mathbf{e}_{\mathsf{L}}$ that satisfies

$$\mathbf{e}_{\mathsf{L}}^{\mathsf{T}}\mathbf{A} = \lambda \mathbf{e}_{\mathsf{L}}^{\mathsf{T}}. \tag{C.2}$$

The following statements for right-eigenvectors also apply to left-eigenvectors.

- If a matrix has two or more linearly independent right-eigenvectors with the same eigenvalue then that eigenvalue is called a degenerate eigenvalue of the matrix, or a repeated eigenvalue. Any linear combination of those eigenvectors is another right-eigenvector with the same eigenvalue.
- The principal right-eigenvector of a matrix is, by definition, the right-eigenvector with the largest associated eigenvalue.
- If a real matrix has a right-eigenvector with complex eigenvalue $\lambda = x + yi$ then it also has a right-eigenvector with the conjugate eigenvalue $\lambda^* = x - yi$.

Symmetric matrices

If $\mathbf{A}$ is a real symmetric $N \times N$ matrix then

1. all the eigenvalues and eigenvectors of $\mathbf{A}$ are real;
2. every left-eigenvector of $\mathbf{A}$ is also a right-eigenvector of $\mathbf{A}$ with the same eigenvalue, and *vice versa*;
3. a set of N eigenvectors and eigenvalues $\{\mathbf{e}^{(a)}, \lambda_a\}_{a=1}^{N}$ can be found that are orthonormal, that is,

$$\mathbf{e}^{(a)} \cdot \mathbf{e}^{(b)} = \delta_{ab}; \tag{C.3}$$

the matrix can be expressed as a weighted sum of outer products of the eigenvectors:

$$\mathbf{A} = \sum_{a=1}^{N} \lambda_a [\mathbf{e}^{(a)}][\mathbf{e}^{(a)}]^{\mathsf{T}}. \tag{C.4}$$

(Whereas I often use i and n as indices for sets of size I and N, I will use the indices a and b to run over eigenvectors, even if there are N of them. This is to avoid confusion with the components of the eigenvectors, which are indexed by n, e.g. $e_n^{(a)}$.)

General square matrices

An $N \times N$ matrix can have up to N distinct eigenvalues. Generically, there are N eigenvalues, all distinct, and each has one left-eigenvector and one right-eigenvector. In cases where two or more eigenvalues coincide, for each distinct eigenvalue that is non-zero there is at least one left-eigenvector and one right-eigenvector.

Left- and right-eigenvectors that have different eigenvalue are orthogonal, that is,

$$\text{if } \lambda_a \neq \lambda_b \text{ then } \mathbf{e}_{\mathsf{L}}^{(a)} \cdot \mathbf{e}_{\mathsf{R}}^{(b)} = 0. \tag{C.5}$$

Non-negative matrices

Definition. If all the elements of a non-zero matrix $\mathbf{C}$ satisfy $C_{mn} \geq 0$ then $\mathbf{C}$ is a non-negative matrix. Similarly, if all the elements of a non-zero vector $\mathbf{c}$ satisfy $c_n \geq 0$ then $\mathbf{c}$ is a non-negative vector.

Properties. A non-negative matrix has a principal eigenvector that is non-negative. It may also have other eigenvectors with the same eigenvalue that are not non-negative. But if the principal eigenvalue of a non-negative matrix is not degenerate, then the matrix has only one principal eigenvector $\mathbf{e}^{(1)}$, and it is non-negative.

Generically, all the other eigenvalues are smaller in absolute magnitude. [There can be several eigenvalues of identical magnitude in special cases.]

Transition probability matrices

An important example of a non-negative matrix is a transition probability matrix $\mathbf{Q}$.

Definition. A transition probability matrix $\mathbf{Q}$ has columns that are probability vectors, that is, it satisfies $\mathbf{Q} \geq 0$ and

$$\sum_i Q_{ij} = 1 \text{ for all } j. \tag{C.6}$$

Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$									
$\begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	2.41		1		−0.41					
	$\begin{bmatrix} .58 \\ .82 \\ 0 \end{bmatrix}$	$\begin{bmatrix} .82 \\ .58 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -.58 \\ .82 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -.82 \\ .58 \\ 0 \end{bmatrix}$				
$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$	1.62		−0.62							
	$\begin{bmatrix} .53 \\ .85 \end{bmatrix}$	$\begin{bmatrix} .53 \\ .85 \end{bmatrix}$	$\begin{bmatrix} .85 \\ -.53 \end{bmatrix}$	$\begin{bmatrix} .85 \\ -.53 \end{bmatrix}$						
$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	1.62		0.5+0.9i		0.5−0.9i		−0.62			
	$\begin{bmatrix} .60 \\ .37 \\ .37 \\ .60 \end{bmatrix}$	$\begin{bmatrix} .60 \\ .37 \\ .37 \\ .60 \end{bmatrix}$	$\begin{bmatrix} .1-.5i \\ -.3-.4i \\ .3+.4i \\ -.1+.5i \end{bmatrix}$	$\begin{bmatrix} .1-.5i \\ .3+.4i \\ -.3-.4i \\ -.1+.5i \end{bmatrix}$	$\begin{bmatrix} .1+.5i \\ -.3+.4i \\ .3-.4i \\ -.1-.5i \end{bmatrix}$	$\begin{bmatrix} .1+.5i \\ .3-.4i \\ -.3+.4i \\ -.1-.5i \end{bmatrix}$	$\begin{bmatrix} .37 \\ -.60 \\ -.60 \\ .37 \end{bmatrix}$	$\begin{bmatrix} .37 \\ -.60 \\ -.60 \\ .37 \end{bmatrix}$		

Table C.4. Some matrices and their eigenvectors.

Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$									
$\begin{bmatrix} 0 & .38 \\ 1 & .62 \end{bmatrix}$	1		−0.38							
	$\begin{bmatrix} .71 \\ .71 \end{bmatrix}$	$\begin{bmatrix} .36 \\ .93 \end{bmatrix}$	$\begin{bmatrix} -.93 \\ .36 \end{bmatrix}$	$\begin{bmatrix} -.71 \\ .71 \end{bmatrix}$						
$\begin{bmatrix} 0 & .35 & 0 \\ 0 & 0 & .46 \\ 1 & .65 & .54 \end{bmatrix}$	1		−0.2−0.3i		−0.2+0.3i					
	$\begin{bmatrix} .58 \\ .58 \\ .58 \end{bmatrix}$	$\begin{bmatrix} .14 \\ .41 \\ .90 \end{bmatrix}$	$\begin{bmatrix} -.8+.1i \\ -.2-.5i \\ .2+.2i \end{bmatrix}$	$\begin{bmatrix} .2-.5i \\ -.6+.2i \\ .4+.3i \end{bmatrix}$	$\begin{bmatrix} -.8-.1i \\ -.2+.5i \\ .2-.2i \end{bmatrix}$	$\begin{bmatrix} .2+.5i \\ -.6-.2i \\ .4-.3i \end{bmatrix}$				

Table C.5. Transition probability matrices for generating random paths through trellises.

This property can be rewritten in terms of the all-ones vector $\mathbf{n} = (1, 1, \ldots, 1)^{\mathsf{T}}$:

$$\mathbf{n}^{\mathsf{T}}\mathbf{Q} = \mathbf{n}^{\mathsf{T}}. \tag{C.7}$$

So $\mathbf{n}$ is the principal left-eigenvector of $\mathbf{Q}$ with eigenvalue $\lambda_1 = 1$.

$$\mathbf{e}_{\mathsf{L}}^{(1)} = \mathbf{n}. \tag{C.8}$$

Because it is a non-negative matrix, $\mathbf{Q}$ has a principal right-eigenvector that is non-negative, $\mathbf{e}_{\mathsf{R}}^{(1)}$. Generically, for Markov processes that are ergodic, this eigenvector is the only right-eigenvector with eigenvalue of magnitude 1 (see table C.6 for illustrative exceptions). This vector, if we normalize it such that $\mathbf{e}_{\mathsf{R}}^{(1)} \cdot \mathbf{n} = 1$, is called the invariant distribution of the transition probability matrix. It is the probability density that is left unchanged under $\mathbf{Q}$. Unlike the principal left-eigenvector, which we explicitly identified above, we can't usually identify the principal right-eigenvector without computation.

The matrix may have up to $N - 1$ other right-eigenvectors all of which are orthogonal to the left-eigenvector $\mathbf{n}$, that is, they are zero-sum vectors.

▶ C.3 Perturbation theory

Perturbation theory is not used in this book, but it is useful in this book's fields. In this section we derive first-order perturbation theory for the eigenvectors and eigenvalues of square, *not necessarily symmetric*, matrices. Most presentations of perturbation theory focus on symmetric matrices, but non-symmetric matrices (such as transition matrices) also deserve to be perturbed!

	Matrix	Eigenvalues and eigenvectors $\mathbf{e}_{\mathsf{L}}, \mathbf{e}_{\mathsf{R}}$							
		1		1		0.70		0.70	
(a)	$\begin{bmatrix} .90 & .20 & 0 & 0 \\ .10 & .80 & 0 & 0 \\ 0 & 0 & .90 & .20 \\ 0 & 0 & .10 & .80 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ .71 \\ .71 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ .89 \\ .45 \end{bmatrix}$	$\begin{bmatrix} .71 \\ .71 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} .89 \\ .45 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} .45 \\ -.89 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} .71 \\ -.71 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ -.45 \\ .89 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ -.71 \\ .71 \end{bmatrix}$
		1		0.98		0.70		0.69	
(a′)	$\begin{bmatrix} .90 & .20 & 0 & 0 \\ .10 & .79 & .02 & 0 \\ 0 & .01 & .88 & .20 \\ 0 & 0 & .10 & .80 \end{bmatrix}$	$\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$	$\begin{bmatrix} .87 \\ .43 \\ .22 \\ .11 \end{bmatrix}$	$\begin{bmatrix} -.18 \\ -.15 \\ .66 \\ .72 \end{bmatrix}$	$\begin{bmatrix} -.66 \\ -.28 \\ .61 \\ .33 \end{bmatrix}$	$\begin{bmatrix} .20 \\ -.40 \\ -.40 \\ .80 \end{bmatrix}$	$\begin{bmatrix} .63 \\ -.63 \\ -.32 \\ .32 \end{bmatrix}$	$\begin{bmatrix} -.19 \\ .41 \\ -.44 \\ .77 \end{bmatrix}$	$\begin{bmatrix} -.61 \\ .65 \\ -.35 \\ .30 \end{bmatrix}$
		1		0.70		−0.70		−1	
(b)	$\begin{bmatrix} 0 & 0 & .90 & .20 \\ 0 & 0 & .10 & .80 \\ .90 & .20 & 0 & 0 \\ .10 & .80 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} .50 \\ .50 \\ .50 \\ .50 \end{bmatrix}$	$\begin{bmatrix} .63 \\ .32 \\ .63 \\ .32 \end{bmatrix}$	$\begin{bmatrix} -.32 \\ .63 \\ -.32 \\ .63 \end{bmatrix}$	$\begin{bmatrix} .50 \\ -.50 \\ .50 \\ -.50 \end{bmatrix}$	$\begin{bmatrix} .32 \\ -.63 \\ -.32 \\ .63 \end{bmatrix}$	$\begin{bmatrix} -.50 \\ .50 \\ .50 \\ -.50 \end{bmatrix}$	$\begin{bmatrix} .50 \\ .50 \\ -.50 \\ -.50 \end{bmatrix}$	$\begin{bmatrix} .63 \\ .32 \\ -.63 \\ -.32 \end{bmatrix}$

Table C.6. Illustrative transition probability matrices and their eigenvectors showing the two ways of being non-ergodic. (a) More than one principal eigenvector with eigenvalue 1 because the state space falls into two unconnected pieces. (a′) A small perturbation breaks the degeneracy of the principal eigenvectors. (b) Under this chain, the density may oscillate between two parts of the state space. In addition to the invariant distribution, there is another right-eigenvector with eigenvalue −1. In general such circulating densities correspond to complex eigenvalues with magnitude 1.

We assume that we have an $N \times N$ matrix $\mathbf{H}$ that is a function $\mathbf{H}(\epsilon)$ of a real parameter ϵ, with $\epsilon = 0$ being our starting point. We assume that a Taylor expansion of $\mathbf{H}(\epsilon)$ is appropriate:

$$\mathbf{H}(\epsilon) = \mathbf{H}(0) + \epsilon \mathbf{V} + \cdots \tag{C.9}$$

where

$$\mathbf{V} \equiv \frac{\partial \mathbf{H}}{\partial \epsilon}. \tag{C.10}$$

We assume that for all ϵ of interest, $\mathbf{H}(\epsilon)$ has a complete set of N right-eigenvectors and left-eigenvectors, and that these eigenvectors and their eigenvalues are continuous functions of ϵ. This last assumption is not necessarily a good one: if $\mathbf{H}(0)$ has degenerate eigenvalues then it is possible for the eigenvectors to be discontinuous in ϵ; in such cases, degenerate perturbation theory is needed. That's a fun topic, but let's stick with the non-degenerate case here.

We write the eigenvectors and eigenvalues as follows:

$$\mathbf{H}(\epsilon)\mathbf{e}_{\mathsf{R}}^{(a)}(\epsilon) = \lambda^{(a)}(\epsilon)\mathbf{e}_{\mathsf{R}}^{(a)}(\epsilon), \tag{C.11}$$

and we Taylor-expand

$$\lambda^{(a)}(\epsilon) = \lambda^{(a)}(0) + \epsilon\mu^{(a)} + \cdots \tag{C.12}$$

with

$$\mu^{(a)} \equiv \frac{\partial \lambda^{(a)}(\epsilon)}{\partial \epsilon} \tag{C.13}$$

and

$$\mathbf{e}_{\mathsf{R}}^{(a)}(\epsilon) = \mathbf{e}_{\mathsf{R}}^{(a)}(0) + \epsilon \mathbf{f}_{\mathsf{R}}^{(a)} + \cdots \tag{C.14}$$

with

$$\mathbf{f}_{\mathsf{R}}^{(a)} \equiv \frac{\partial \mathbf{e}_{\mathsf{R}}^{(a)}}{\partial \epsilon}, \tag{C.15}$$

and similar definitions for $\mathbf{e}_{\mathsf{L}}^{(a)}$ and $\mathbf{f}_{\mathsf{L}}^{(a)}$. We define these left-vectors to be row vectors, so that the 'transpose' operation is not needed and can be banished.

We are free to constrain the magnitudes of the eigenvectors in whatever way we please. Each left-eigenvector and each right-eigenvector has an arbitrary magnitude. The natural constraints to use are as follows. First, we constrain the inner products with:

$$\mathbf{e}_{\mathsf{L}}^{(a)}(\epsilon)\mathbf{e}_{\mathsf{R}}^{(a)}(\epsilon) = 1, \quad \text{for all } a. \tag{C.16}$$

Expanding the eigenvectors in ϵ, equation (C.19) implies

$$(\mathbf{e}_{\mathsf{L}}^{(a)}(0) + \epsilon \mathbf{f}_{\mathsf{L}}^{(a)} + \cdots)(\mathbf{e}_{\mathsf{R}}^{(a)}(0) + \epsilon \mathbf{f}_{\mathsf{R}}^{(a)} + \cdots) = 1, \tag{C.17}$$

from which we can extract the terms in ϵ, which say:

$$\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mathbf{f}_{\mathsf{L}}^{(a)}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = 0 \tag{C.18}$$

We are now free to choose the two constraints:

$$\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} = 0, \quad \mathbf{f}_{\mathsf{L}}^{(a)}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = 0, \tag{C.19}$$

which in the special case of a symmetric matrix correspond to constraining the eigenvectors to be of constant length, as defined by the Euclidean norm.

OK, now that we have defined our cast of characters, what do the defining equations (C.11) and (C.9) tell us about our Taylor expansions (C.13) and (C.15)? We expand equation (C.11) in ϵ.

$$(\mathbf{H}(0)+\epsilon\mathbf{V}+\cdots)(\mathbf{e}_{\mathsf{R}}^{(a)}(0)+\epsilon\mathbf{f}_{\mathsf{R}}^{(a)}+\cdots) = (\lambda^{(a)}(0)+\epsilon\mu^{(a)}+\cdots)(\mathbf{e}_{\mathsf{R}}^{(a)}(0)+\epsilon\mathbf{f}_{\mathsf{R}}^{(a)}+\cdots). \tag{C.20}$$

Identifying the terms of order ϵ, we have:

$$\mathbf{H}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = \lambda^{(a)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mu^{(a)}\mathbf{e}_{\mathsf{R}}^{(a)}(0). \tag{C.21}$$

We can extract interesting results from this equation by hitting it with $\mathbf{e}_{\mathsf{L}}^{(b)}(0)$:

$$\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{H}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = \mathbf{e}_{\mathsf{L}}^{(b)}(0)\lambda^{(a)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mu^{(a)}\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{e}_{\mathsf{R}}^{(a)}(0).$$

$$\Rightarrow \ \lambda^{(b)}\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = \lambda^{(a)}(0)\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} + \mu^{(a)}\delta_{ab}. \tag{C.22}$$

Setting $b = a$ we obtain

$$\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = \mu^{(a)}. \tag{C.23}$$

Alternatively, choosing $b \neq a$, we obtain:

$$\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0) = \left[\lambda^{(a)}(0) - \lambda^{(b)}(0)\right] \mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} \tag{C.24}$$

$$\Rightarrow \mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{f}_{\mathsf{R}}^{(a)} = \frac{1}{\lambda^{(a)}(0) - \lambda^{(b)}(0)}\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0). \tag{C.25}$$

Now, assuming that the right-eigenvectors $\{\mathbf{e}_{\mathsf{R}}^{(b)}(0)\}_{b=1}^{N}$ form a complete basis, we must be able to write

$$\mathbf{f}_{\mathsf{R}}^{(a)} = \sum_b w_b \mathbf{e}_{\mathsf{R}}^{(b)}(0), \tag{C.26}$$

where

$$w_b = \mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{f}_{\mathsf{R}}^{(a)}, \tag{C.27}$$

so, comparing (C.25) and (C.27), we have:

$$\mathbf{f}_{\mathsf{R}}^{(a)} = \sum_{b \neq a} \frac{\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0)}{\lambda^{(a)}(0) - \lambda^{(b)}(0)} \mathbf{e}_{\mathsf{R}}^{(b)}(0). \tag{C.28}$$

Equations (C.23) and (C.28) are the solution to the first-order perturbation theory problem, giving respectively the first derivative of the eigenvalue and the eigenvectors.

Second-order perturbation theory

If we expand the eigenvector equation (C.11) to second order in ϵ, and assume that the equation

$$\mathbf{H}(\epsilon) = \mathbf{H}(0) + \epsilon\mathbf{V} \tag{C.29}$$

is exact, that is, $\mathbf{H}$ is a purely linear function of ϵ, then we have:

$$\begin{aligned} &(\mathbf{H}(0) + \epsilon\mathbf{V})(\mathbf{e}_{\mathsf{R}}^{(a)}(0) + \epsilon\mathbf{f}_{\mathsf{R}}^{(a)} + \frac{1}{2}\epsilon^2\mathbf{g}_{\mathsf{R}}^{(a)} + \cdots) \\ = \;& (\lambda^{(a)}(0) + \epsilon\mu^{(a)} + \tfrac{1}{2}\epsilon^2\nu^{(a)} + \cdots)(\mathbf{e}_{\mathsf{R}}^{(a)}(0) + \epsilon\mathbf{f}_{\mathsf{R}}^{(a)} + \tfrac{1}{2}\epsilon^2\mathbf{g}_{\mathsf{R}}^{(a)} + \cdots) \end{aligned} \tag{C.30}$$

where $\mathbf{g}_{\mathsf{R}}^{(a)}$ and $\nu^{(a)}$ are the second derivatives of the eigenvector and eigenvalue. Equating the second-order terms in ϵ in equation (C.30),

$$\mathbf{V}\mathbf{f}_{\mathsf{R}}^{(a)} + \frac{1}{2}\mathbf{H}(0)\mathbf{g}_{\mathsf{R}}^{(a)} = \frac{1}{2}\lambda^{(a)}(0)\mathbf{g}_{\mathsf{R}}^{(a)} + \frac{1}{2}\nu^{(a)}\mathbf{e}_{\mathsf{R}}^{(a)}(0) + \mu^{(a)}\mathbf{f}_{\mathsf{R}}^{(a)}. \tag{C.31}$$

Hitting this equation on the left with $\mathbf{e}_{\mathsf{L}}^{(a)}(0)$, we obtain:

$$\begin{aligned} &\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{V}\mathbf{f}_{\mathsf{R}}^{(a)} + \frac{1}{2}\lambda^{(a)}\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{g}_{\mathsf{R}}^{(a)} \\ = \;& \tfrac{1}{2}\lambda^{(a)}(0)\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{g}_{\mathsf{R}}^{(a)} + \tfrac{1}{2}\nu^{(a)}\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{e}_{\mathsf{R}}^{(a)}(0) + \mu^{(a)}\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{f}_{\mathsf{R}}^{(a)}. \end{aligned} \tag{C.32}$$

The term $\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{f}_{\mathsf{R}}^{(a)}$ is equal to zero because of our constraints (C.19), so

$$\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{V}\mathbf{f}_{\mathsf{R}}^{(a)} = \frac{1}{2}\nu^{(a)}, \tag{C.33}$$

so the second derivative of the eigenvalue with respect to ϵ is given by

$$\frac{1}{2}\nu^{(a)} = \mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{V}\sum_{b \neq a} \frac{\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0)}{\lambda^{(a)}(0) - \lambda^{(b)}(0)} \mathbf{e}_{\mathsf{R}}^{(b)}(0) \tag{C.34}$$

$$= \sum_{b \neq a} \frac{[\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0)][\mathbf{e}_{\mathsf{L}}^{(a)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(b)}(0)]}{\lambda^{(a)}(0) - \lambda^{(b)}(0)}. \tag{C.35}$$

This is as far as we will take the perturbation expansion.

Summary

If we introduce the abbreviation V_{ba} for $\mathbf{e}_{\mathsf{L}}^{(b)}(0)\mathbf{V}\mathbf{e}_{\mathsf{R}}^{(a)}(0)$, we can write the eigenvectors of $\mathbf{H}(\epsilon) = \mathbf{H}(0) + \epsilon\mathbf{V}$ to first order as

$$\mathbf{e}_{\mathsf{R}}^{(a)}(\epsilon) = \mathbf{e}_{\mathsf{R}}^{(a)}(0) + \epsilon\sum_{b \neq a} \frac{V_{ba}}{\lambda^{(a)}(0) - \lambda^{(b)}(0)} \mathbf{e}_{\mathsf{R}}^{(b)}(0) + \cdots \tag{C.36}$$

and the eigenvalues to second order as

$$\lambda^{(a)}(\epsilon) = \lambda^{(a)}(0) + \epsilon V_{aa} + \epsilon^2 \sum_{b \neq a} \frac{V_{ba}V_{ab}}{\lambda^{(a)}(0) - \lambda^{(b)}(0)} + \cdots. \tag{C.37}$$

C.4 Some numbers

	2^{8192}	10^{2466}	Number of distinct 1-kilobyte files
	2^{1024}	10^{308}	Number of states of a 2D Ising model with 32×32 spins
2^{1000}		10^{301}	Number of binary strings of length 1000
2^{500}		3×10^{150}	
	2^{469}	10^{141}	Number of binary strings of length 1000 having 100 1s and 900 0s
	2^{266}	10^{80}	Number of electrons in universe
2^{200}		1.6×10^{60}	
	2^{190}	10^{57}	Number of electrons in solar system
	2^{171}	3×10^{51}	Number of electrons in the earth
2^{100}		10^{30}	
	2^{98}	3×10^{29}	Age of universe/picoseconds
	2^{58}	3×10^{17}	Age of universe/seconds
2^{50}		10^{15}	
2^{40}		10^{12}	
		10^{11}	Number of neurons in human brain
		10^{11}	Number of bits stored on a DVD
		3×10^{10}	Number of bits in the wheat genome
		6×10^{9}	Number of bits in the human genome
	2^{32}	6×10^{9}	Population of earth
2^{30}		10^{9}	
		2.5×10^{8}	Number of fibres in the corpus callosum
		2×10^{8}	Number of bits in *C. Elegans* (a worm) genome
		2×10^{8}	Number of bits in *Arabidopsis thaliana* (a flowering plant related to broccoli) genome
	2^{25}	3×10^{7}	One year/seconds
		2×10^{7}	Number of bits in the compressed PostScript file that is this book
		2×10^{7}	Number of bits in `unix` kernel
		10^{7}	Number of bits in the *E. Coli* genome, or in a floppy disk
		4×10^{6}	Number of years since human/chimpanzee divergence
2^{20}		10^{6}	1 048 576
		2×10^{5}	Number of generations since human/chimpanzee divergence
		3×10^{4}	Number of genes in human genome
		3×10^{4}	Number of genes in *Arabidopsis thaliana* genome
		1.5×10^{3}	Number of base pairs in a gene
2^{10}	e^{7}	10^{3}	$2^{10} = 1024$; $e^{7} = 1096$
2^{0}		10^{0}	1
	2^{-2}	2.5×10^{-1}	Lifetime probability of dying from smoking one pack of cigarettes per day.
		10^{-2}	Lifetime probability of dying in a motor vehicle accident
2^{-10}		10^{-3}	
		10^{-5}	Lifetime probability of developing cancer because of drinking 2 litres per day of water containing 12 p.p.b. benzene
2^{-20}		10^{-6}	
		3×10^{-8}	Probability of error in transmission of coding DNA, per nucleotide, per generation
2^{-30}		10^{-9}	
2^{-60}		10^{-18}	Probability of undetected error in a hard disk drive, after error correction

Bibliography

Abrahamsen, P. (1997) A review of Gaussian random fields and correlation functions. Technical Report 917, Norwegian Computing Center, Blindern, N-0314 Oslo, Norway. 2nd edition.

Abramson, N. (1963) *Information Theory and Coding*. McGraw-Hill.

Adler, S. L. (1981) Over-relaxation method for the Monte-Carlo evaluation of the partition function for multiquadratic actions. *Physical Review D – Particles and Fields* **23** (12): 2901–2904.

Aiyer, S. V. B. (1991) *Solving Combinatorial Optimization Problems Using Neural Networks*. Cambridge Univ. Engineering Dept. PhD dissertation. CUED/F-INFENG/TR 89.

Aji, S., Jin, H., Khandekar, A., McEliece, R. J., and MacKay, D. J. C. (2000) BSC thresholds for code ensembles based on 'typical pairs' decoding. In *Codes, Systems and Graphical Models*, ed. by B. Marcus and J. Rosenthal, volume 123 of *IMA Volumes in Mathematics and its Applications*, pp. 195–210. Springer.

Amari, S., Cichocki, A., and Yang, H. H. (1996) A new learning algorithm for blind signal separation. In *Advances in Neural Information Processing Systems*, ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, volume 8, pp. 757–763. MIT Press.

Amit, D. J., Gutfreund, H., and Sompolinsky, H. (1985) Storing infinite numbers of patterns in a spin glass model of neural networks. *Phys. Rev. Lett.* **55**: 1530–1533.

Angel, J. R. P., Wizinowich, P., Lloyd-Hart, M., and Sandler, D. (1990) Adaptive optics for array telescopes using neural-network techniques. *Nature* **348**: 221–224.

Bahl, L. R., Cocke, J., Jelinek, F., and Raviv, J. (1974) Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Info. Theory* **IT-20**: 284–287.

Baldwin, J. (1896) A new factor in evolution. *American Naturalist* **30**: 441–451.

Bar-Shalom, Y., and Fortmann, T. (1988) *Tracking and Data Association*. Academic Press.

Barber, D., and Williams, C. K. I. (1997) Gaussian processes for Bayesian classification via hybrid Monte Carlo. In *Neural Information Processing Systems 9*, ed. by M. C. Mozer, M. I. Jordan, and T. Petsche, pp. 340–346. MIT Press.

Barnett, S. (1979) *Matrix Methods for Engineers and Scientists*. McGraw-Hill.

Battail, G. (1993) We can think of good codes, and even decode them. In *Eurocode '92. Udine, Italy, 26-30 October*, ed. by P. Camion, P. Charpin, and S. Harari, number 339 in CISM Courses and Lectures, pp. 353–368. Springer.

Baum, E., Boneh, D., and Garrett, C. (1995) On genetic algorithms. In *Proc. Eighth Annual Conf. on Computational Learning Theory*, pp. 230–239. ACM.

Baum, E. B., and Smith, W. D. (1993) Best play for imperfect players and game tree search. Technical report, NEC, Princeton, NJ.

Baum, E. B., and Smith, W. D. (1997) A Bayesian approach to relevance in game playing. *Artificial Intelligence* **97** (1-2): 195–242.

Baum, L. E., and Petrie, T. (1966) Statistical inference for probabilistic functions of finite-state Markov chains. *Ann. Math. Stat.* **37**: 1559–1563.

Beal, M. J., Ghahramani, Z., and Rasmussen, C. E. (2002) The infinite hidden Markov model. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Bell, A. J., and Sejnowski, T. J. (1995) An information maximization approach to blind separation and blind deconvolution. *Neural Computation* **7** (6): 1129–1159.

Bentley, J. (2000) *Programming Pearls*. Addison-Wesley, second edition.

Berger, J. (1985) *Statistical Decision theory and Bayesian Analysis*. Springer.

Berlekamp, E. R. (1968) *Algebraic Coding Theory*. McGraw-Hill.

Berlekamp, E. R. (1980) The technology of error-correcting codes. *IEEE Trans. Info. Theory* **68**: 564–593.

Berlekamp, E. R., McEliece, R. J., and van Tilborg, H. C. A. (1978) On the intractability of certain coding problems. *IEEE Trans. Info. Theory* **24** (3): 384–386.

Berrou, C., and Glavieux, A. (1996) Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Trans. on Communications* **44**: 1261–1271.

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993) Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *Proc. 1993 IEEE International Conf. on Communications, Geneva, Switzerland*, pp. 1064–1070.

Berzuini, C., Best, N. G., Gilks, W. R., and Larizza, C. (1997) Dynamic conditional independence models and Markov chain Monte Carlo methods. *J. American Statistical Assoc.* **92** (440): 1403–1412.

Berzuini, C., and Gilks, W. R. (2001) Following a moving target – Monte Carlo inference for dynamic Bayesian models. *J. Royal Statistical Society Series B – Statistical Methodology* **63** (1): 127–146.

Bhattacharyya, A. (1943) On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.* **35**: 99–110.

Bishop, C. M. (1992) Exact calculation of the Hessian matrix for the multilayer perceptron. *Neural Computation* **4** (4): 494–501.

Bishop, C. M. (1995) *Neural Networks for Pattern Recognition*. Oxford Univ. Press.

Bishop, C. M., Winn, J. M., and Spiegelhalter, D. (2002) VIBES: A variational inference engine for Bayesian networks. In *Advances in Neural Information Processing Systems XV*, ed. by S. Becker, S. Thrun, and K. Obermayer.

Blahut, R. E. (1987) *Principles and Practice of Information Theory*. Addison-Wesley.

Bottou, L., Howard, P. G., and Bengio, Y. (1998) The Z-coder adaptive binary coder. In *Proc. Data Compression Conf., Snowbird, Utah, March 1998*, pp. 13–22.

Box, G. E. P., and Tiao, G. C. (1973) *Bayesian Inference in Statistical Analysis*. Addison–Wesley.

Braunstein, A., Mézard, M., and Zecchina, R., (2003) Survey propagation: an algorithm for satisfiability. `cs.CC/0212002`.

Bretthorst, G. (1988) *Bayesian Spectrum Analysis and Parameter Estimation*. Springer. Also available at `bayes.wustl.edu`.

Bridle, J. S. (1989) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neuro-computing: Algorithms, Architectures and Applications*, ed. by F. Fougelman-Soulie and J. Hérault. Springer–Verlag.

Bulmer, M. (1985) *The Mathematical Theory of Quantitative Genetics*. Oxford Univ. Press.

Burrows, M., and Wheeler, D. J. (1994) A block-sorting lossless data compression algorithm. Technical Report 124, Digital SRC.

Byers, J., Luby, M., Mitzenmacher, M., and Rege, A. (1998) A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM '98, September 2–4, 1998*.

Cairns-Smith, A. G. (1985) *Seven Clues to the Origin of Life*. Cambridge Univ. Press.

Calderbank, A. R., and Shor, P. W. (1996) Good quantum error-correcting codes exist. *Phys. Rev. A* **54**: 1098. `quant-ph/9512032`.

Carroll, L. (1998) *Alice's Adventures in Wonderland; and, Through the Looking-glass: and what Alice Found There*. Macmillan Children's Books.

Childs, A. M., Patterson, R. B., and MacKay, D. J. C. (2001) Exact sampling from non-attractive distributions using summary states. *Physical Review E* **63**: 036113.

Chu, W., Keerthi, S. S., and Ong, C. J. (2001) A unified loss function in Bayesian framework for support vector regression. In *Proc. 18th International Conf. on Machine Learning*, pp. 51–58.

Chu, W., Keerthi, S. S., and Ong, C. J. (2002) A new Bayesian design method for support vector classification. In *Special Section on Support Vector Machines of the 9th International Conf. on Neural Information Processing*.

Chu, W., Keerthi, S. S., and Ong, C. J. (2003a) Bayesian support vector regression using a unified loss function. *IEEE Trans. on Neural Networks*. Submitted.

Chu, W., Keerthi, S. S., and Ong, C. J. (2003b) Bayesian trigonometric support vector classifier. *Neural Computation*.

Chung, S.-Y., Richardson, T. J., and Urbanke, R. L. (2001) Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. *IEEE Trans. Info. Theory* **47** (2): 657–670.

Chung, S.-Y., Urbanke, R. L., and Richardson, T. J., (1999) LDPC code design applet. `lids.mit.edu/~sychung/gaopt.html`.

Comon, P., Jutten, C., and Herault, J. (1991) Blind separation of sources. 2. Problems statement. *Signal Processing* **24** (1): 11–20.

Copas, J. B. (1983) Regression, prediction and shrinkage (with discussion). *J. R. Statist. Soc. B* **45** (3): 311–354.

Cover, T. M. (1965) Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. on Electronic Computers* **14**: 326–334.

Cover, T. M., and Thomas, J. A. (1991) *Elements of Information Theory*. Wiley.

Cowles, M. K., and Carlin, B. P. (1996) Markov-chain Monte-Carlo convergence diagnostics – a comparative review. *J. American Statistical Assoc.* **91** (434): 883–904.

Cox, R. (1946) Probability, frequency, and reasonable expectation. *Am. J. Physics* **14**: 1–13.

Cressie, N. (1993) *Statistics for Spatial Data*. Wiley.

Davey, M. C. (1999) *Error-correction using Low-Density Parity-Check Codes*. Univ. of Cambridge PhD dissertation.

Davey, M. C., and MacKay, D. J. C. (1998) Low density parity check codes over GF(q). *IEEE Communications Letters* **2** (6): 165–167.

Davey, M. C., and MacKay, D. J. C. (2000) Watermark codes: Reliable communication over insertion/deletion channels. In *Proc. 2000 IEEE International Symposium on Info. Theory*, p. 477.

Davey, M. C., and MacKay, D. J. C. (2001) Reliable communication over channels with insertions, deletions and substitutions. *IEEE Trans. Info. Theory* **47** (2): 687–698.

Dawid, A., Stone, M., and Zidek, J. (1996) Critique of E.T Jaynes's 'paradoxes of probability theory'. Technical Report 172, Dept. of Statistical Science, Univ. College London.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995) The Helmholtz machine. *Neural Computation* **7** (5): 889–904.

Divsalar, D., Jin, H., and McEliece, R. J. (1998) Coding theorems for 'turbo-like' codes. In *Proc. 36th Allerton Conf. on Communication, Control, and Computing, Sept. 1998*, pp. 201–210. Allerton House.

Doucet, A., de Freitas, J., and Gordon, N. eds. (2001) *Sequential Monte Carlo Methods in Practice*. Springer.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987) Hybrid Monte Carlo. *Physics Letters B* **195**: 216–222.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998) *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ. Press.

Dyson, F. J. (1985) *Origins of Life*. Cambridge Univ. Press.

Elias, P. (1975) Universal codeword sets and representations of the integers. *IEEE Trans. Info. Theory* **21** (2): 194–203.

Eyre-Walker, A., and Keightley, P. (1999) High genomic deleterious mutation rates in hominids. *Nature* **397**: 344–347.

Felsenstein, J. (1985) Recombination and sex: is Maynard Smith necessary? In *Evolution. Essays in Honour of John Maynard Smith*, ed. by P. J. Greenwood, P. H. Harvey, and M. Slatkin, pp. 209–220. Cambridge Univ. Press.

Ferreira, H., Clarke, W., Helberg, A., Abdel-Ghaffar, K. S., and Vinck, A. H. (1997) Insertion/deletion correction with spectral nulls. *IEEE Trans. Info. Theory* **43** (2): 722–732.

Feynman, R. P. (1972) *Statistical Mechanics*. Addison–Wesley.

Forney, Jr., G. D. (1966) *Concatenated Codes*. MIT Press.

Forney, Jr., G. D. (2001) Codes on graphs: Normal realizations. *IEEE Trans. Info. Theory* **47** (2): 520–548.

Frey, B. J. (1998) *Graphical Models for Machine Learning and Digital Communication*. MIT Press.

Gallager, R. G. (1962) Low density parity check codes. *IRE Trans. Info. Theory* **IT-8**: 21–28.

Gallager, R. G. (1963) *Low Density Parity Check Codes*. Number 21 in MIT Research monograph series. MIT Press. Available from `www.inference.phy.cam.ac.uk/mackay/gallager/papers/`.

Gallager, R. G. (1968) *Information Theory and Reliable Communication*. Wiley.

Gallager, R. G. (1978) Variations on a theme by Huffman. *IEEE Trans. Info. Theory* **IT-24** (6): 668–674.

Gibbs, M. N. (1997) *Bayesian Gaussian Processes for Regression and Classification*. Cambridge Univ. PhD dissertation. `www.inference.phy.cam.ac.uk/mng10/`.

Gibbs, M. N., and MacKay, D. J. C., (1996) Efficient implementation of Gaussian processes for interpolation. `www.inference.phy.cam.ac.uk/mackay/abstracts/gpros.html`.

Gibbs, M. N., and MacKay, D. J. C. (2000) Variational Gaussian process classifiers. *IEEE Trans. on Neural Networks* **11** (6): 1458–1464.

Gilks, W., Roberts, G., and George, E. (1994) Adaptive direction sampling. *Statistician* **43**: 179–189.

Gilks, W., and Wild, P. (1992) Adaptive rejection sampling for Gibbs sampling. *Applied Statistics* **41**: 337–348.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996) *Markov Chain Monte Carlo in Practice*. Chapman and Hall.

Goldie, C. M., and Pinch, R. G. E. (1991) *Communication theory*. Cambridge Univ. Press.

Golomb, S. W., Peile, R. E., and Scholtz, R. A. (1994) *Basic Concepts in Information Theory and Coding: The Adventures of Secret Agent 00111*. Plenum Press.

Good, I. J. (1979) Studies in the history of probability and statistics. XXXVII. A.M. Turing's statistical work in World War II. *Biometrika* **66** (2): 393–396.

Graham, R. L. (1966) On partitions of a finite set. *Journal of Combinatorial Theory* **1**: 215–223.

Graham, R. L., and Knowlton, K. C., (1968) Method of identifying conductors in a cable by establishing conductor connection groupings at both ends of the cable. U.S. Patent 3,369,177.

Green, P. J. (1995) Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **82**: 711–732.

Gregory, P. C., and Loredo, T. J. (1992) A new method for the detection of a periodic signal of unknown shape and period. In *Maximum Entropy and Bayesian Methods,*, ed. by G. Erickson and C. Smith. Kluwer. Also in *Astrophysical Journal*, **398**, pp. 146–168, Oct 10, 1992.

Gull, S. F. (1988) Bayesian inductive inference and maximum entropy. In *Maximum Entropy and Bayesian Methods in Science and Engineering, vol. 1: Foundations*, ed. by G. Erickson and C. Smith, pp. 53–74. Kluwer.

Gull, S. F. (1989) Developments in maximum entropy data analysis. In *Maximum Entropy and Bayesian Methods, Cambridge 1988*, ed. by J. Skilling, pp. 53–71. Kluwer.

Gull, S. F., and Daniell, G. (1978) Image reconstruction from incomplete and noisy data. *Nature* **272**: 686–690.

Hamilton, W. D. (2002) *Narrow Roads of Gene Land, Volume 2: Evolution of Sex*. Oxford Univ. Press.

Hanson, R., Stutz, J., and Cheeseman, P. (1991a) Bayesian classification theory. Technical Report FIA–90-12-7-01, NASA Ames.

Hanson, R., Stutz, J., and Cheeseman, P. (1991b) Bayesian classification with correlation and inheritance. In *Proc. 12th Intern. Joint Conf. on Artificial Intelligence, Sydney, Australia*, volume 2, pp. 692–698. Morgan Kaufmann.

Hartmann, C. R. P., and Rudolph, L. D. (1976) An optimum symbol by symbol decoding rule for linear codes. *IEEE Trans. Info. Theory* **IT-22**: 514–517.

Harvey, M., and Neal, R. M. (2000) Inference for belief networks using coupling from the past. In *Uncertainty in Artificial Intelligence: Proc. Sixteenth Conf.*, pp. 256–263.

Hebb, D. O. (1949) *The Organization of Behavior*. Wiley.

Hendin, O., Horn, D., and Hopfield, J. J. (1994) Decomposition of a mixture of signals in a model of the olfactory bulb. *Proc. Natl. Acad. Sci. USA* **91** (13): 5942–5946.

Hertz, J., Krogh, A., and Palmer, R. G. (1991) *Introduction to the Theory of Neural Computation*. Addison-Wesley.

Hinton, G. (2001) Training products of experts by minimizing contrastive divergence. Technical Report 2000-004, Gatsby Computational Neuroscience Unit, Univ. College London.

Hinton, G., and Nowlan, S. (1987) How learning can guide evolution. *Complex Systems* **1**: 495–502.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995) The wake-sleep algorithm for unsupervised neural networks. *Science* **268** (5214): 1158–1161.

Hinton, G. E., and Ghahramani, Z. (1997) Generative models for discovering sparse distributed representations. *Philosophical Trans. Royal Society B*.

Hinton, G. E., and Sejnowski, T. J. (1986) Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, ed. by D. E. Rumelhart and J. E. McClelland, pp. 282–317. MIT Press.

Hinton, G. E., and Teh, Y. W. (2001) Discovering multiple constraints that are frequently approximately satisfied. In *Uncertainty in Artificial Intelligence: Proc. Seventeenth Conf. (UAI-2001)*, pp. 227–234. Morgan Kaufmann.

Hinton, G. E., and van Camp, D. (1993) Keeping neural networks simple by minimizing the description length of the weights. In *Proc. 6th Annual Workshop on Comput. Learning Theory*, pp. 5–13. ACM Press, New York, NY.

Hinton, G. E., Welling, M., Teh, Y. W., and Osindero, S. (2001) A new view of ICA. In *Proc. International Conf. on Independent Component Analysis and Blind Signal Separation*, volume 3.

Hinton, G. E., and Zemel, R. S. (1994) Autoencoders, minimum description length and Helmholtz free energy. In *Advances in Neural Information Processing Systems 6*, ed. by J. D. Cowan, G. Tesauro, and J. Alspector. Morgan Kaufmann.

Hodges, A. (1983) *Alan Turing: The Enigma*. Simon and Schuster.

Hojen-Sorensen, P. A., Winther, O., and Hansen, L. K. (2002) Mean field approaches to independent component analysis. *Neural Computation* **14**: 889–918.

Holmes, C., and Denison, D. (2002) Perfect sampling for wavelet reconstruction of signals. *IEEE Trans. Signal Processing* **50**: 237–244.

Holmes, C., and Mallick, B. (1998) Perfect simulation for orthogonal model mixing. Technical report, Imperial College, London.

Hopfield, J. J. (1974) Kinetic proofreading: A new mechanism for reducing errors in biosynthetic processes requiring high specificity. *Proc. Natl. Acad. Sci. USA* **71** (10): 4135–4139.

Hopfield, J. J. (1978) Origin of the genetic code: A testable hypothesis based on tRNA structure, sequence, and kinetic proofreading. *Proc. Natl. Acad. Sci. USA* **75** (9): 4334–4338.

Hopfield, J. J. (1980) The energy relay: A proofreading scheme based on dynamic cooperativity and lacking all characteristic symptoms of kinetic proofreading in DNA replication and protein synthesis. *Proc. Natl. Acad. Sci. USA* **77** (9): 5248–5252.

Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA* **79**: 2554–8.

Hopfield, J. J. (1984) Neurons with graded response properties have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA* **81**: 3088–92.

Hopfield, J. J. (1987) Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proc. Natl. Acad. Sci. USA* **84**: 8429–33.

Hopfield, J. J., and Brody, C. D. (2000) What is a moment? "Cortical" sensory integration over a brief interval. *Proc. Natl. Acad. Sci* **97**: 13919–13924.

Hopfield, J. J., and Brody, C. D. (2001) What is a moment? Transient synchrony as a collective mechanism for spatiotemporal integration. *Proc. Natl. Acad. Sci* **98**: 1282–1287.

Hopfield, J. J., and Tank, D. W. (1985) Neural computation of decisions in optimization problems. *Biol. Cybernetics* **52**: 1–25.

Howarth, P., and Bradley, A. (1986) The longitudinal aberration of the human eye and its correction. *Vision Res.* **26**: 361–366.

Huber, M. (1998) Exact sampling and approximate counting techniques. In *Proc. 30th ACM Symposium on the Theory of Computing*, pp. 31–40.

Huffman, D. (1952) A method for construction of minimum-redundancy codes. *Proc. of IRE* **40** (9): 1098–1101.

Ichikawa, K., Bhadeshia, H. K. D. H., and MacKay, D. J. C. (1996) Model for hot cracking in low-alloy steel weld metals. *Science and Technology of Welding and Joining* **1**: 43–50.

Isard, M., and Blake, A. (1996) Visual tracking by stochastic propagation of conditional density. In *Proc. Fourth European Conf. Computer Vision*, pp. 343–356.

Isard, M., and Blake, A. (1998) Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision* **29** (1): 5–28.

Jaakkola, T. S., and Jordan, M. I. (1996) Computing upper and lower bounds on likelihoods in intractable networks. In *Proc. Twelfth Conf. on Uncertainty in AI*. Morgan Kaufman.

Jaakkola, T. S., and Jordan, M. I. (2000a) Bayesian logistic regression: a variational approach. *Statistics and Computing* **10**: 25–37.

Jaakkola, T. S., and Jordan, M. I. (2000b) Bayesian parameter estimation via variational methods. *Statistics and Computing* **10** (1): 25–37.

Jaynes, E. T. (1983) Bayesian intervals versus confidence intervals. In *E.T. Jaynes. Papers on Probability, Statistics and Statistical Physics*, ed. by R. D. Rosenkrantz, p. 151. Kluwer.

Jaynes, E. T. (2003) *Probability Theory: The Logic of Science*. Cambridge Univ. Press. Edited by G. Larry Bretthorst.

Jensen, F. V. (1996) *An Introduction to Bayesian Networks*. UCL press.

Johannesson, R., and Zigangirov, K. S. (1999) *Fundamentals of Convolutional Coding*. IEEE Press.

Jordan, M. I. ed. (1998) *Learning in Graphical Models*. NATO Science Series. Kluwer Academic Publishers.

JPL, (1996) Turbo codes performance. Available from `www331.jpl.nasa.gov/public/TurboPerf.html`.

Jutten, C., and Herault, J. (1991) Blind separation of sources. 1. An adaptive algorithm based on neuromimetic architecture. *Signal Processing* **24** (1): 1–10.

Karplus, K., and Krit, H. (1991) A semi-systolic decoder for the PDSC–73 error-correcting code. *Discrete Applied Mathematics* **33**: 109–128.

Kepler, T., and Oprea, M. (2001) Improved inference of mutation rates: I. An integral representation of the Luria-Delbrück distribution. *Theoretical Population Biology* **59**: 41–48.

Kimeldorf, G. S., and Wahba, G. (1970) A correspondence between Bayesian estimation of stochastic processes and smoothing by splines. *Annals of Math. Statistics* **41** (2): 495–502.

Kitanidis, P. K. (1986) Parameter uncertainty in estimation of spatial functions: Bayesian analysis. *Water Resources Research* **22**: 499–507.

Knuth, D. E. (1968) *The Art of Computer Programming*. Addison Wesley.

Kondrashov, A. S. (1988) Deleterious mutations and the evolution of sexual reproduction. *Nature* **336** (6198): 435–440.

Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001) Factor graphs and the sum-product algorithm. *IEEE Trans. Info. Theory* **47** (2): 498–519.

Kschischang, F. R., and Sorokine, V. (1995) On the trellis structure of block codes. *IEEE Trans. Info. Theory* **41** (6): 1924–1937.

Lauritzen, S. L. (1981) Time series analysis in 1880, a discussion of contributions made by T. N. Thiele. *ISI Review* **49**: 319–333.

Lauritzen, S. L. (1996) *Graphical Models*. Number 17 in Oxford Statistical Science Series. Clarendon Press.

Lauritzen, S. L., and Spiegelhalter, D. J. (1988) Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society B* **50**: 157–224.

Levenshtein, V. I. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady* **10** (8): 707–710.

Lin, S., and Costello, Jr., D. J. (1983) *Error Control Coding: Fundamentals and Applications*. Prentice-Hall.

Litsyn, S., and Shevelev, V. (2002) On ensembles of low-density parity-check codes: asymptotic distance distributions. *IEEE Trans. Info. Theory* **48** (4): 887–908.

Loredo, T. J. (1990) From Laplace to supernova SN 1987A: Bayesian inference in astrophysics. In *Maximum Entropy and Bayesian Methods, Dartmouth, U.S.A., 1989*, ed. by P. Fougere, pp. 81–142. Kluwer.

Lowe, D. G. (1995) Similarity metric learning for a variable kernel classifier. *Neural Computation* **7**: 72–85.

Luby, M. (2002) LT codes. In *Proc. The 43rd Annual IEEE Symposium on Foundations of Computer Science, November 16–19 2002*, pp. 271–282.

Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (1998) Improved low-density parity-check codes using irregular graphs and belief propagation. In *Proc. IEEE International Symposium on Info. Theory*, p. 117.

Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (2001a) Efficient erasure correcting codes. *IEEE Trans. Info. Theory* **47** (2): 569–584.

Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (2001b) Improved low-density parity-check codes using irregular graphs and belief propagation. *IEEE Trans. Info. Theory* **47** (2): 585–598.

Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., Spielman, D. A., and Stemann, V. (1997) Practical loss-resilient codes. In *Proc. Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*.

Luo, Z., and Wahba, G. (1997) Hybrid adaptive splines. *J. Amer. Statist. Assoc.* **92**: 107–116.

Luria, S. E., and Delbrück, M. (1943) Mutations of bacteria from virus sensitivity to virus resistance. *Genetics* **28**: 491–511. Reprinted in *Microbiology: A Centenary Perspective*, Wolfgang K. Joklik, ed., 1999, ASM Press, and available from `www.esp.org/`.

Luttrell, S. P. (1989) Hierarchical vector quantisation. *Proc. IEE Part I* **136**: 405–413.

Luttrell, S. P. (1990) Derivation of a class of training algorithms. *IEEE Trans. on Neural Networks* **1** (2): 229–232.

MacKay, D. J. C. (1991) *Bayesian Methods for Adaptive Models*. California Institute of Technology PhD dissertation.

MacKay, D. J. C. (1992a) Bayesian interpolation. *Neural Computation* **4** (3): 415–447.

MacKay, D. J. C. (1992b) The evidence framework applied to classification networks. *Neural Computation* **4** (5): 698–714.

MacKay, D. J. C. (1992c) A practical Bayesian framework for backpropagation networks. *Neural Computation* **4** (3): 448–472.

MacKay, D. J. C. (1994a) Bayesian methods for backpropagation networks. In *Models of Neural Networks III*, ed. by E. Domany, J. L. van Hemmen, and K. Schulten, chapter 6, pp. 211–254. Springer.

MacKay, D. J. C. (1994b) Bayesian non-linear modelling for the prediction competition. In *ASHRAE Trans., V.100, Pt.2*, pp. 1053–1062. American Society of Heating, Refrigeration, and Air-conditioning Engineers.

MacKay, D. J. C. (1995a) Free energy minimization algorithm for decoding and cryptanalysis. *Electronics Letters* **31** (6): 446–447.

MacKay, D. J. C. (1995b) Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems* **6**: 469–505.

MacKay, D. J. C., (1997a) Ensemble learning for hidden Markov models. `www.inference.phy.cam.ac.uk/mackay/abstracts/ensemblePaper.html`.

MacKay, D. J. C., (1997b) Iterative probabilistic decoding of low density parity check codes. Animations available on world wide web. `www.inference.phy.cam.ac.uk/mackay/codes/gifs/`.

MacKay, D. J. C. (1998a) Choice of basis for Laplace approximation. *Machine Learning* **33** (1): 77–86.

MacKay, D. J. C. (1998b) Introduction to Gaussian processes. In *Neural Networks and Machine Learning*, ed. by C. M. Bishop, NATO ASI Series, pp. 133–166. Kluwer.

MacKay, D. J. C. (1999a) Comparison of approximate methods for handling hyperparameters. *Neural Computation* **11** (5): 1035–1068.

MacKay, D. J. C. (1999b) Good error correcting codes based on very sparse matrices. *IEEE Trans. Info. Theory* **45** (2): 399–431.

MacKay, D. J. C., (2000) An alternative to runlength-limiting codes: Turn timing errors into substitution errors. Available from `www.inference.phy.cam.ac.uk/mackay/`.

MacKay, D. J. C., (2001) A problem with variational free energy minimization. `www.inference.phy.cam.ac.uk/mackay/abstracts/minima.html`.

MacKay, D. J. C., and Davey, M. C. (2000) Evaluation of Gallager codes for short block length and high rate applications. In *Codes, Systems and Graphical Models*, ed. by B. Marcus and J. Rosenthal, volume 123 of *IMA Volumes in Mathematics and its Applications*, pp. 113–130. Springer.

MacKay, D. J. C., Mitchison, G. J., and McFadden, P. L. (2004) Sparse-graph codes for quantum error-correction. *IEEE Trans. Info. Theory* **50** (10): 2315–2330.

MacKay, D. J. C., and Neal, R. M. (1995) Good codes based on very sparse matrices. In *Cryptography and Coding. 5th IMA Conf., LNCS 1025*, ed. by C. Boyd, pp. 100–111. Springer.

MacKay, D. J. C., and Neal, R. M. (1996) Near Shannon limit performance of low density parity check codes. *Electronics Letters* **32** (18): 1645–1646. Reprinted *Electronics Letters*, **33**(6):457–458, March 1997.

MacKay, D. J. C., and Peto, L. (1995) A hierarchical Dirichlet language model. *Natural Language Engineering* **1** (3): 1–19.

MacKay, D. J. C., Wilson, S. T., and Davey, M. C. (1998) Comparison of constructions of irregular Gallager codes. In *Proc. 36th Allerton Conf. on Communication, Control, and Computing, Sept. 1998*, pp. 220–229. Allerton House.

MacKay, D. J. C., Wilson, S. T., and Davey, M. C. (1999) Comparison of constructions of irregular Gallager codes. *IEEE Trans. on Communications* **47** (10): 1449–1454.

MacKay, D. M., and MacKay, V. (1974) The time course of the McCollough effect and its physiological implications. *J. Physiol.* **237**: 38–39.

MacKay, D. M., and McCulloch, W. S. (1952) The limiting information capacity of a neuronal link. *Bull. Math. Biophys.* **14**: 127–135.

MacWilliams, F. J., and Sloane, N. J. A. (1977) *The Theory of Error-correcting Codes*. North-Holland.

Mandelbrot, B. (1982) *The Fractal Geometry of Nature*. W.H. Freeman.

Mao, Y., and Banihashemi, A. (2000) Design of good LDPC codes using girth distribution. In *IEEE International Symposium on Info. Theory, Italy, June, 2000*.

Mao, Y., and Banihashemi, A. (2001) A heuristic search for good LDPC codes at short block lengths. In *IEEE International Conf. on Communications*.

Marinari, E., and Parisi, G. (1992) Simulated tempering – a new Monte-Carlo scheme. *Europhysics Letters* **19** (6): 451–458.

Matheron, G. (1963) Principles of geostatistics. *Economic Geology* **58**: 1246–1266.

Maynard Smith, J. (1968) 'Haldane's dilemma' and the rate of evolution. *Nature* **219** (5159): 1114–1116.

Maynard Smith, J. (1978) *The Evolution of Sex*. Cambridge Univ. Press.

Maynard Smith, J. (1988) *Games, Sex and Evolution*. Harvester–Wheatsheaf.

Maynard Smith, J., and Szathmary, E. (1995) *The Major Transitions in Evolution*. Freeman.

Maynard Smith, J., and Szathmary, E. (1999) *The Origins of Life*. Oxford Univ. Press.

McCollough, C. (1965) Color adaptation of edge-detectors in the human visual system. *Science* **149**: 1115–1116.

McEliece, R. J. (2002) *The Theory of Information and Coding*. Cambridge Univ. Press, second edition.

McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998) Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communications* **16** (2): 140–152.

McMillan, B. (1956) Two inequalities implied by unique decipherability. *IRE Trans. Inform. Theory* **2**: 115–116.

Minka, T. (2001) *A family of algorithms for approximate Bayesian inference*. MIT PhD dissertation.

Miskin, J. W. (2001) *Ensemble Learning for Independent Component Analysis*. Dept. of Physics, Univ. of Cambridge PhD dissertation.

Miskin, J. W., and MacKay, D. J. C. (2000) Ensemble learning for blind image separation and deconvolution. In *Advances in Independent Component Analysis*, ed. by M. Girolami. Springer.

Miskin, J. W., and MacKay, D. J. C. (2001) Ensemble learning for blind source separation. In *ICA: Principles and Practice*, ed. by S. Roberts and R. Everson. Cambridge Univ. Press.

Mosteller, F., and Wallace, D. L. (1984) *Applied Bayesian and Classical Inference. The case of* The Federalist *papers*. Springer.

Neal, R. M. (1991) Bayesian mixture modelling by Monte Carlo simulation. Technical Report CRG–TR–91–2, Computer Science, Univ. of Toronto.

Neal, R. M. (1993a) Bayesian learning via stochastic dynamics. In *Advances in Neural Information Processing Systems 5*, ed. by C. L. Giles, S. J. Hanson, and J. D. Cowan, pp. 475–482. Morgan Kaufmann.

Neal, R. M. (1993b) Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG–TR–93–1, Dept. of Computer Science, Univ. of Toronto.

Neal, R. M. (1995) Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation. Technical Report 9508, Dept. of Statistics, Univ. of Toronto.

Neal, R. M. (1996) *Bayesian Learning for Neural Networks*. Springer.

Neal, R. M. (1997a) Markov chain Monte Carlo methods based on 'slicing' the density function. Technical Report 9722, Dept. of Statistics, Univ. of Toronto.

Neal, R. M. (1997b) Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report CRG–TR–97–2, Dept. of Computer Science, Univ. of Toronto.

Neal, R. M. (1998) Annealed importance sampling. Technical Report 9805, Dept. of Statistics, Univ. of Toronto.

Neal, R. M. (2001) Defining priors for distributions using Dirichlet diffusion trees. Technical Report 0104, Dept. of Statistics, Univ. of Toronto.

Neal, R. M. (2003) Slice sampling. *Annals of Statistics* **31** (3): 705–767.

Neal, R. M., and Hinton, G. E. (1998) A new view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, ed. by M. I. Jordan, NATO Science Series, pp. 355–368. Kluwer.

Nielsen, M., and Chuang, I. (2000) *Quantum Computation and Quantum Information*. Cambridge Univ. Press.

Offer, E., and Soljanin, E. (2000) An algebraic description of iterative decoding schemes. In *Codes, Systems and Graphical Models*, ed. by B. Marcus and J. Rosenthal, volume 123 of *IMA Volumes in Mathematics and its Applications*, pp. 283–298. Springer.

Offer, E., and Soljanin, E. (2001) LDPC codes: a group algebra formulation. In *Proc. Internat. Workshop on Coding and Cryptography WCC 2001, 8-12 Jan. 2001, Paris*.

O'Hagan, A. (1978) On curve fitting and optimal design for regression. *J. Royal Statistical Society, B* **40**: 1–42.

O'Hagan, A. (1987) Monte Carlo is fundamentally unsound. *The Statistician* **36**: 247–249.

O'Hagan, A. (1994) *Bayesian Inference*, volume 2B of *Kendall's Advanced Theory of Statistics*. Edward Arnold.

Omre, H. (1987) Bayesian kriging – merging observations and qualified guesses in kriging. *Mathematical Geology* **19**: 25–39.

Opper, M., and Winther, O. (2000) Gaussian processes for classification: Mean-field algorithms. *Neural Computation* **12** (11): 2655–2684.

Patrick, J. D., and Wallace, C. S. (1982) Stone circle geometries: an information theory approach. In *Archaeoastronomy in the Old World*, ed. by D. C. Heggie, pp. 231–264. Cambridge Univ. Press.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. (2000) *Causality*. Cambridge Univ. Press.

Pearlmutter, B. A., and Parra, L. C. (1996) A context-sensitive generalization of ICA. In *International Conf. on Neural Information Processing, Hong Kong*, pp. 151–157.

Pearlmutter, B. A., and Parra, L. C. (1997) Maximum likelihood blind source separation: A context-sensitive generalization of ICA. In *Advances in Neural Information Processing Systems*, ed. by M. C. Mozer, M. I. Jordan, and T. Petsche, volume 9, p. 613. MIT Press.

Pinto, R. L., and Neal, R. M. (2001) Improving Markov chain Monte Carlo estimators by coupling to an approximating chain. Technical Report 0101, Dept. of Statistics, Univ. of Toronto.

Poggio, T., and Girosi, F. (1989) A theory of networks for approximation and learning. Technical Report A.I. 1140, MIT.

Poggio, T., and Girosi, F. (1990) Networks for approximation and learning. *Proc. IEEE* **78**: 1481–1497.

Polya, G. (1954) *Induction and Analogy in Mathematics*. Princeton Univ. Press.

Propp, J. G., and Wilson, D. B. (1996) Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms* **9** (1-2): 223–252.

Rabiner, L. R., and Juang, B. H. (1986) An introduction to hidden Markov models. *IEEE ASSP Magazine* pp. 4–16.

Rasmussen, C. E. (1996) *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*. Univ. of Toronto PhD dissertation.

Rasmussen, C. E. (2000) The infinite Gaussian mixture model. In *Advances in Neural Information Processing Systems 12*, ed. by S. Solla, T. Leen, and K.-R. Müller, pp. 554–560. MIT Press.

Rasmussen, C. E., (2002) Reduced rank Gaussian process learning. Unpublished manuscript.

Rasmussen, C. E., and Ghahramani, Z. (2002) Infinite mixtures of Gaussian process experts. In *Advances in Neural Information Processing Systems 14*, ed. by T. G. Diettrich, S. Becker, and Z. Ghahramani. MIT Press.

Rasmussen, C. E., and Ghahramani, Z. (2003) Bayesian Monte Carlo. In *Advances in Neural Information Processing Systems XV*, ed. by S. Becker, S. Thrun, and K. Obermayer.

Ratliff, F., and Riggs, L. A. (1950) Involuntary motions of the eye during monocular fixation. *J. Exptl. Psychol.* **40**: 687–701.

Ratzer, E. A., and MacKay, D. J. C. (2003) Sparse low-density parity-check codes for channels with cross-talk. In *Proc. 2003 IEEE Info. Theory Workshop, Paris*.

Reif, F. (1965) *Fundamentals of Statistical and Thermal Physics*. McGraw–Hill.

Richardson, T., Shokrollahi, M. A., and Urbanke, R. (2001) Design of capacity-approaching irregular low-density parity check codes. *IEEE Trans. Info. Theory* **47** (2): 619–637.

Richardson, T., and Urbanke, R. (2001a) The capacity of low-density parity check codes under message-passing decoding. *IEEE Trans. Info. Theory* **47** (2): 599–618.

Richardson, T., and Urbanke, R. (2001b) Efficient encoding of low-density parity-check codes. *IEEE Trans. Info. Theory* **47** (2): 638–656.

Ridley, M. (2000) *Mendel's Demon: gene justice and the complexity of life*. Phoenix.

Ripley, B. D. (1991) *Statistical Inference for Spatial Processes*. Cambridge Univ. Press.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge Univ. Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back-propagating errors. *Nature* **323**: 533–536.

Russell, S., and Wefald, E. (1991) *Do the Right Thing: Studies in Limited Rationality*. MIT Press.

Schneier, B. (1996) *Applied Cryptography*. Wiley.

Scholkopf, B., Burges, C., and Vapnik, V. (1995) Extracting support data for a given task. In *Proc. First International Conf. on Knowledge Discovery and Data Mining*, ed. by U. M. Fayyad and R. Uthurusamy. AAAI Press.

Scholtz, R. A. (1982) The origins of spread-spectrum communications. *IEEE Trans. on Communications* **30** (5): 822–854.

Seeger, M., Williams, C. K. I., and Lawrence, N. (2003) Fast forward selection to speed up sparse Gaussian process regression. In *Proc. Ninth International Workshop on Artificial Intelligence and Statistics*, ed. by C. Bishop and B. J. Frey. Society for Artificial Intelligence and Statistics.

Sejnowski, T. J. (1986) Higher order Boltzmann machines. In *Neural networks for computing*, ed. by J. Denker, pp. 398–403. American Institute of Physics.

Sejnowski, T. J., and Rosenberg, C. R. (1987) Parallel networks that learn to pronounce English text. *Journal of Complex Systems* **1** (1): 145–168.

Shannon, C. E. (1948) A mathematical theory of communication. *Bell Sys. Tech. J.* **27**: 379–423, 623–656.

Shannon, C. E. (1993) The best detection of pulses. In *Collected Papers of Claude Shannon*, ed. by N. J. A. Sloane and A. D. Wyner, pp. 148–150. IEEE Press.

Shannon, C. E., and Weaver, W. (1949) *The Mathematical Theory of Communication*. Univ. of Illinois Press.

Shokrollahi, A. (2003) Raptor codes. Technical report, Laboratoire d'algorithmique, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. Available from `algo.epfl.ch/`.

Sipser, M., and Spielman, D. A. (1996) Expander codes. *IEEE Trans. Info. Theory* **42** (6.1): 1710–1722.

Skilling, J. (1989) Classic maximum entropy. In *Maximum Entropy and Bayesian Methods, Cambridge 1988*, ed. by J. Skilling. Kluwer.

Skilling, J. (1993) Bayesian numerical analysis. In *Physics and Probability*, ed. by W. T. Grandy, Jr. and P. Milonni. Cambridge Univ. Press.

Skilling, J., and MacKay, D. J. C. (2003) Slice sampling – a binary implementation. *Annals of Statistics* **31** (3): 753–755. Discussion of *Slice Sampling* by Radford M. Neal.

Slepian, D., and Wolf, J. (1973) Noiseless coding of correlated information sources. *IEEE Trans. Info. Theory* **19**: 471–480.

Smola, A. J., and Bartlett, P. (2001) Sparse Greedy Gaussian Process Regression. In *Advances in Neural Information Processing Systems 13*, ed. by T. K. Leen, T. G. Diettrich, and V. Tresp, pp. 619–625. MIT Press.

Spiegel, M. R. (1988) *Statistics*. Schaum's outline series. McGraw-Hill, 2nd edition.

Spielman, D. A. (1996) Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Info. Theory* **42** (6.1): 1723–1731.

Sutton, R. S., and Barto, A. G. (1998) *Reinforcement Learning: An Introduction*. MIT Press.

Swanson, L. (1988) A new code for Galileo. In *Proc. 1988 IEEE International Symposium Info. Theory*, pp. 94–95.

Tanner, M. A. (1996) *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions*. Springer Series in Statistics. Springer, 3rd edition.

Tanner, R. M. (1981) A recursive approach to low complexity codes. *IEEE Trans. Info. Theory* **27** (5): 533–547.

Teahan, W. J. (1995) Probability estimation for PPM. In *Proc. NZCSRSC'95*. Available from `citeseer.nj.nec.com/teahan95probability.html`.

ten Brink, S. (1999) Convergence of iterative decoding. *Electronics Letters* **35** (10): 806–808.

ten Brink, S., Kramer, G., and Ashikhmin, A., (2002) Design of low-density parity-check codes for multi-antenna modulation and detection. Submitted to *IEEE Trans. on Communications*.

Terras, A. (1999) *Fourier Analysis on Finite Groups and Applications*. Cambridge Univ. Press.

Thomas, A., Spiegelhalter, D. J., and Gilks, W. R. (1992) BUGS: A program to perform Bayesian inference using Gibbs sampling. In *Bayesian Statistics 4*, ed. by J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, pp. 837–842. Clarendon Press.

Tresp, V. (2000) A Bayesian committee machine. *Neural Computation* **12** (11): 2719–2741.

Urbanke, R., (2001) LdpcOpt – a fast and accurate degree distribution optimizer for LDPC code ensembles. `lthcwww.epfl.ch/research/ldpcopt/`.

Vapnik, V. (1995) *The Nature of Statistical Learning Theory*. Springer.

Viterbi, A. J. (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Info. Theory* **IT-13**: 260–269.

Wahba, G. (1990) *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics. CBMS-NSF Regional Conf. series in applied mathematics.

Wainwright, M. J., Jaakkola, T., and Willsky, A. S. (2003) Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Trans. Info. Theory* **45** (9): 1120–1146.

Wald, G., and Griffin, D. (1947) The change in refractive power of the eye in bright and dim light. *J. Opt. Soc. Am.* **37**: 321–336.

Wallace, C., and Boulton, D. (1968) An information measure for classification. *Comput. J.* **11** (2): 185–194.

Wallace, C. S., and Freeman, P. R. (1987) Estimation and inference by compact coding. *J. R. Statist. Soc. B* **49** (3): 240–265.

Ward, D. J., Blackwell, A. F., and MacKay, D. J. C. (2000) Dasher – A data entry interface using continuous gestures and language models. In *Proc. User Interface Software and Technology 2000*, pp. 129–137.

Ward, D. J., and MacKay, D. J. C. (2002) Fast hands-free writing by gaze direction. *Nature* **418** (6900): 838.

Welch, T. A. (1984) A technique for high-performance data compression. *IEEE Computer* **17** (6): 8–19.

Welling, M., and Teh, Y. W. (2001) Belief optimization for binary networks: A stable alternative to loopy belief propagation. In *Uncertainty in Artificial Intelligence: Proc. Seventeenth Conf. (UAI-2001)*, pp. 554–561. Morgan Kaufmann.

Wiberg, N. (1996) *Codes and Decoding on General Graphs*. Dept. of Elec. Eng., Linköping, Sweden PhD dissertation. Linköping Studies in Science and Technology No. 440.

Wiberg, N., Loeliger, H.-A., and Kötter, R. (1995) Codes and iterative decoding on general graphs. *European Trans. on Telecommunications* **6**: 513–525.

Wiener, N. (1948) *Cybernetics*. Wiley.

Williams, C. K. I., and Rasmussen, C. E. (1996) Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*, ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press.

Williams, C. K. I., and Seeger, M. (2001) Using the Nyström Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems 13*, ed. by T. K. Leen, T. G. Diettrich, and V. Tresp, pp. 682–688. MIT Press.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987) Arithmetic coding for data compression. *Communications of the ACM* **30** (6): 520–540.

Wolf, J. K., and Siegel, P. (1998) On two-dimensional arrays and crossword puzzles. In *Proc. 36th Allerton Conf. on Communication, Control, and Computing, Sept. 1998*, pp. 366–371. Allerton House.

Worthen, A. P., and Stark, W. E. (1998) Low-density parity check codes for fading channels with memory. In *Proc. 36th Allerton Conf. on Communication, Control, and Computing, Sept. 1998*, pp. 117–125.

Yedidia, J. S. (2000) An idiosyncratic journey beyond mean field theory. Technical report, Mitsubishi Electric Res. Labs. TR-2000-27.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2000) Generalized belief propagation. Technical report, Mitsubishi Electric Res. Labs. TR-2000-26.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2001a) Bethe free energy, Kikuchi approximations and belief propagation algorithms. Technical report, Mitsubishi Electric Res. Labs. TR-2001-16.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2001b) Characterization of belief propagation and its generalizations. Technical report, Mitsubishi Electric Res. Labs. TR-2001-15.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2002) Constructing free energy approximations and generalized belief propagation algorithms. Technical report, Mitsubishi Electric Res. Labs. TR-2002-35.

Yeung, R. W. (1991) A new outlook on Shannon-information measures. *IEEE Trans. Info. Theory* **37** (3.1): 466–474.

Yuille, A. L. (2001) A double-loop algorithm to minimize the Bethe and Kikuchi free energies. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, ed. by M. Figueiredo, J. Zerubia, and A. Jain, number 2134 in LNCS, pp. 3–18. Springer.

Zipf, G. K. (1949) *Human Behavior and the Principle of Least Effort*. Addison-Wesley.

Ziv, J., and Lempel, A. (1977) A universal algorithm for sequential data compression. *IEEE Trans. Info. Theory* **23** (3): 337–343.

Ziv, J., and Lempel, A. (1978) Compression of individual sequences via variable-rate coding. *IEEE Trans. Info. Theory* **24** (5): 530–536.

Index

Γ, 598
$\Phi(z)$, 514
χ^2, 40, 323, 458, 459
λ, 119
σ_N and σ_{N-1}, 320
:=, 600
?, 419
2s, 156

Abu-Mostafa, Yaser, 482
acceptance rate, 365, 369, 394
acceptance ratio method, 379
accumulator, 254, 570, 582
activation function, 471
activity rule, 470, 471
adaptive direction sampling, 393
adaptive models, 101
adaptive rejection sampling, 370
address, 201, 468
Aiyer, Sree, 518
Alberto, 56
alchemists, 74
algebraic coding theory, 19, 574
algorithm, *see* learning algorithms
  BCJR, 330
  belief propagation, 330, 336
  covariant, 442
  EM, 432
  exact sampling, 413
  expectation–maximization, 432
  function minimization, 473
  genetic, 395, 396
  Hamiltonian Monte Carlo, 387, 496
  independent component analysis, 443
  Langevin Monte Carlo, 496
  leapfrog, 389
  max–product, 339
  message-passing, 330
  min–sum, 339
  Monte Carlo, *see* Monte Carlo methods
  Newton–Raphson, 303, 441
  perfect simulation, 413
  sum–product, 336
  Viterbi, 340
Alice, 199
Allais paradox, 454
alphabetical ordering, 194
America, 354
American, 238, 260
amino acid, 201, 204, 279, 362
anagram, 200
annealing, 379, 392, 397
  deterministic, 518
  importance sampling, 379
antiferromagnetic, 400
ape, 269
approximation
  by Gaussian, 2, 301, 341, 350, 496
  Laplace, 341, 547
  of complex distribution, 185, 282, 364, 422, 433
  of density evolution, 567
  saddle-point, 341
  Stirling, 1
  variational, 422
arabic, 127
architecture, 470, 529
arithmetic coding, 101, **110**, 111
  decoder, 118
  software, 121
  uses beyond compression, 118, 250, 255
arithmetic progression, 344
arms race, 278
artificial intelligence, 121, 129
associative memory, 468, 505, 507
assumptions, 26, 50
astronomy, 551
asymptotic equipartition, 80, 384
  why it is a misleading term, 83
Atlantic, 173
AutoClass, 306
automatic relevance determination, 532, 544
automobile data reception, 594
average, 26, *see* expectation
AWGN, 177

background rate, 307
backpropagation, 473, 475, 528, 535
backward pass, 244
bad, *see* error-correcting code
Balakrishnan, Sree, 518
balance, 66
Baldwin effect, 279
ban (unit), 264
Banburismus, 265
band-limited signal, 178
bandwidth, 178, 182
bar-code, 262, 399
base transitions, 373
base-pairing, 280
basis dependence, 306, 342
bat, 213, 214
battleships, 71
Bayes' theorem, 6, 24, 25, 27, 28, 48–50, 53, 148, 324, 344, 347, 446, 493, 522
Bayes, Rev. Thomas, 51
Bayesian belief networks, 293
Bayesian inference, 26, 346, 457
BCH codes, 13
BCJR algorithm, 330, 578
bearing, 307
Belarusian, 238
belief, 57
belief propagation, 330, 557, *see* message passing *and* sum–product algorithm
Benford's law, 446
bent coin, 1, 30, 38, 51, 76, 113, 307
Berlekamp, Elwyn, 172, 213
Bernoulli distribution, 117
Berrou, C., 186
bet, 200, 209, 455
beta distribution, 316
beta function, 316
beta integral, 30
Bethe free energy, 434
Bhattacharyya parameter, 215
bias, 345, 506
  in neural net, 471
  in statistics, 306, 307, 321
biexponential, 88, 313, 448
bifurcation, 89, 291, 426
binary entropy function, 2, 15
binary erasure channel, **148**, 151
binary images, 399
binary representations, 132
binary symmetric channel, 4, **148**, 151, 211, 215, 229
binding DNA, 201
binomial distribution, 1, 311
bipartite graph, 19
birthday, 156, 157, 160, 198, 200
bit, 3, 73
bit (unit), 264
bits back, 104, 108, 353
bivariate Gaussian, 388
black, 355
Bletchley Park, 265
Blind Watchmaker, 269, 396
block code, 9, *see* source code or error-correcting code
block-sorting, 121
blood group, 55
blow up, 306
blur, 549

Bob, 199
Boltzmann entropy, 85
Boltzmann machine, 522
bombes, 265
book ISBN, 235
bookies, 456
Bottou, Leon, 121
bound, 85
  union, 166, 216, 230
bounded-distance decoder, 207, 212
bounding chain, 419
box, 343, 351
boyish matters, 58
brain, 468
Bridge, 126
British, 260
broadcast channel, 237, 239, 594
Brody, Carlos, 246
Brownian motion, 280, 316, 535
BSC, *see* channel, binary symmetric
budget, 94, 96
Buffon's needle, 38
`BUGS`, 371, 431
buoy, 307
burglar alarm and earthquake, 293
Burrows–Wheeler transform, 121
burst errors, 185, 186
bus-stop paradox, 39, 46, 107
byte, 134, 265

cable labelling, 175
calculator, 320
camera, 549
canonical, 88
capacity, 14, **146**, **150**, 151, 183, 484
  channel with synchronization errors, 187
  constrained channel, 251
  Gaussian channel, 182
  Hopfield network, 514
  neural network, 483
  neuron, 483
  symmetry argument, 151
car data reception, 594
card, 233
casting out nines, 198
Cauchy distribution, 85, 88, 313, 362
caution, *see* sermon
  equipartition, 83
  Gaussian distribution, 312
  importance sampling, 362, 382
  sampling theory, 64
cave, 214
caveat, *see* caution *and* sermon
cellphone, *see* mobile phone
cellular automaton, 130
central-limit theorem, 36, 41, 88, 131, *see* law of large numbers
centre of gravity, 35
chain rule, 528
challenges, 246
  Tanner, 569
channel
  AWGN, 177
  binary erasure, **148**, 151
  binary symmetric, 4, 146, **148**, 151, 206, 211, 215, 229
  broadcast, 237, 239, 594
  bursty, 185, 557
  capacity, 14, 146, 150, 250
    connection with physics, 257
  coding theorem, *see* noisy-channel coding theorem
  complex, 184, 557
  constrained, 248, 255, 256
  continuous, 178
  discrete memoryless, 147
  erasure, 188, 219, 589
  extended, 153
  fading, 186
  Gaussian, 155, 177, 186
  input ensemble, 150
  multiple access, 237
  multiterminal, 239
  noiseless, 248
  noisy, 3, 146
  noisy typewriter, **148**, 152
  symmetric, 171
  two-dimensional, 262
  unknown noise level, 238
  variable symbol durations, 256
  with dependent sources, 236
  with memory, 557
  Z channel, **148**, 149, 150, 172
cheat, 200
Chebyshev inequality, 81, 85
checkerboard, 404, 520
Chernoff bound, 85
chess, 451
chess board, 406, 520
chi-squared, 27, 40, 323, 458
Cholesky decomposition, 552
chromatic aberration, 552
cinema, 187
circle, 316
classical statistics, 64
  criticisms, 32, 50, 457
classifier, 532
Clockville, 39
clustering, **284**, 303
coalescence, 413
cocked hat, 307
code, *see* error-correcting code, source code (for data compression), symbol code, arithmetic coding, linear code, random code or hash code
  dual, *see* error-correcting code, dual
  for constrained channel, 249
    variable-length, 255
code-equivalent, 576
codebreakers, 265
codeword, *see* source code, symbol code, or error-correcting code
coding theory, 4, 19, 205, 215, 574
coin, 1, 30, 38, 63, 76, 307, 464
coincidence, 267, 343, 351
collective, 403
collision, 200
coloured noise, 179
combination, 2, 490, 598
commander, 241
communication, v, 3, 16, 138, 146, 156, 162, 167, 178, 182, 186, 192, 205, 210, 215, 394, 556, 562, 596
  broadcast, 237
  of dependent information, 236
  over noiseless channels, 248
  perspective on learning, 483, 512
competitive learning, 285
complexity, 531, 548
complexity control, 289, 346, 347, 349
`compress`, 119
compression, *see* source code
  future methods, 129
  lossless, 74
  lossy, 74, 284, 285
  of already-compressed files, 74
  of *any* file, 74
  universal, 121
computer, 370
concatenation, 185, 214, 220
  error-correcting codes, 16, 21, 184, 185, 579
  in compression, 92
  in Markov chains, 373
concave $\frown$, 35
conditional entropy, 138, 146
cones, 554
confidence interval, 457, 464
confidence level, 464
confused gameshow host, 57
conjugate gradient, 479
conjugate prior, 319
conjuror, 233
connection between
  channel capacity and physics, 257
  error correcting code and latent variable model, 437
  pattern recognition and error-correction, 481
  supervised and unsupervised learning, 515
  vector quantization and error-correction, 285
connection matrix, 253, 257
constrained channel, 248, 257, 260, 399
  variable-length code, 249
constraint satisfaction, 516
content-addressable memory, 192, 193, 469, 505
continuous channel, 178
control treatment, 458
conventions, *see* notation
convex hull, 102
convex $\smile$, 35
convexity, 370
convolution, 568
convolutional code, 184, 186, **574**, 587
  equivalence, 576

Conway, John H., 86, 520
Copernicus, 346
correlated sources, 138, 237
correlations, 505
  among errors, 557
  and phase transitions, 602
  high-order, 524
  in images, 549
cost function, 180, 451
cost of males, 277
counting, 241
counting argument, 21, 222
coupling from the past, 413
covariance, 440
covariance function, 535
covariance matrix, 176
covariant algorithm, 442
Cover, Thomas, 456, 482
Cox axioms, 26
crib, 265, 268
critical fluctuations, 403
critical path, 246
cross-validation, 353, 531
crossover, 396
crossword, 260
cryptanalysis, 265, 578
cryptography, 200, 578
  digital signatures, 199
  tamper detection, 199
cumulative probability function, 156
cycles in graphs, 242
cyclic, 19

Dasher, 119
data compression, 73, *see* source code *and* compression
data entry, 118
data modelling, *see* modelling
data set, 288
Davey, Matthew C., 569
death penalty, 354, 355
deciban (unit), 264
decibel, 178, 186
decision theory, 346, **451**
decoder, 4, 146, 152
  bitwise, 220, 324
  bounded-distance, 207
  codeword, 220, 324
  maximum *a posteriori*, 325
  probability of error, 221
deconvolution, 551
degree, 568
degree sequence, *see* profile
degrees of belief, 26
degrees of freedom, 322, 459
déjà vu, 121
delay line, 575
Delbrück, Max, 446
deletions, 187
delta function, 438, 600
density evolution, 566, 567, 592
density modelling, 284, 303
dependent sources, 138, 237
depth of lake, 359
design theory, 209
detailed balance, 374, 391
detection of forgery, 199
deterministic annealing, 518
dictionary, 72, 119
die, rolling, 38
difference-set cyclic code, 569
differentiator, 254
diffusion, 316
digamma function, 598
digital cinema, 187
digital fountain, 590
digital signature, 199, 200
digital video broadcast, 593
dimensions, 180
dimer, 204
directory, 193
Dirichlet distribution, 316
Dirichlet model, 117
discriminant function, 179
discriminative training, 552
disease, 25, 458
disk drive, 3, 188, 215, 248, 255
distance, 205
  D_{KL}, 34
  bad, 207, 214
  distance distribution, 206
  entropy distance, 140
  Gilbert–Varshamov, 212, 221
  good, 207
  Hamming, 206
  isn't everything, 215
  of code, 206, 214, 220
    good/bad, 207
  of concatenated code, 214
  of product code, 214
  relative entropy, 34
  very bad, 207
distribution, 311
  beta, 316
  biexponential, 313
  binomial, 311
  Cauchy, 85, 312
  Dirichlet, 316
  exponential, 311, 313
  gamma, 313
  Gaussian, 312
    sample from, 312
  inverse-cosh, 313
  log-normal, 315
  Luria–Delbrück, 446
  normal, 312
  over periodic variables, 315
  Poisson, 175, 311, 315
  Student-t, 312
  Von Mises, 315
divergence, 34
DjVu, 121
DNA, 3, 55, 201, 204, 257, 421
  replication, 279, 280
do the right thing, 451
dodecahedron code, 20, 206, 207
dongle, 558
doors, on game show, 57
Dr. Bloggs, 462
draw straws, 233
dream, 524
DSC, *see* difference-set cyclic code
dual, 216
dumb Metropolis, 394, 496

E_{b}/N_0, 177, 178, 223
earthquake and burglar alarm, 293
earthquake, during game show, 57
Ebert, Todd, 222
edge, 251
eigenvalue, 254, 342, 372, 409, 606
Elias, Peter, 111, 135
EM algorithm, 283, 432
email, 201
empty string, 119
encoder, 4
energy, 291, 401, 601
English, 72, 110, 260
Enigma, 265, 268
ensemble, **67**
  extended, 76
ensemble learning, 429
entropic distribution, 318, 551
entropy, 2, 32, 67, 601
  Boltzmann, 85
  conditional, 138
  Gibbs, 85
  joint, 138
  marginal, 139
  mutual information, 139
  of continuous variable, 180
  relative, 34
entropy distance, 140
epicycles, 346
equipartition, 80
erasure channel, 219, 589
erasure correction, 188, 190, 220
erf, 156, *see* error function
ergodic, 120, 373
error bars, 301, 501
error correction, *see* error-correcting code
  in DNA replication, 280
  in protein synthesis, 280
error detection, 198, 199, 203
error floor, 581
error function, 156, 473, 490, 514, 529, 599
error probability
  and distance, 215, 221
  block, 152
  in compression, 74
error-correcting code, 188, 203
  bad, 183, 207
  block code, 9, **151**, 183
  concatenated, 184–186, 214, 579
  convolutional, 184, **574**, 587
  cyclic, 19
  decoding, 184
  density evolution, 566
  difference-set cyclic, 569
  distance, *see* distance
  dodecahedron, 20, 206, 207
  dual, 216, 218
  equivalence, 576
  erasure channel, 589
  error probability, 171, 215, 221

fountain code, 589
Gallager, **557**
Golay, 209
good, 183, 184, 207, 214, 218
Hamming, 19, 214
in DNA replication, 280
in protein synthesis, 280
interleaving, 186
linear, **9**, 171, 183, 184, 229
coding theorem, 229
low-density generator-matrix, 218, 590
low-density parity-check, 20, 187, 218, **557**, 596
fast encoding, 569
profile, 569
staircase, 569
LT code, 590
maximum distance separable, 220
nonlinear, 187
P_3, 218
parity-check code, 220
pentagonful, 221
perfect, 208, 211, 212, 219, 589
practical, 183, 187
product code, 184, 214
quantum, 572
random, 184
random linear, 211, 212
raptor code, 594
rate, 152, 229
rateless, 590
rectangular, 184
Reed–Solomon code, 571, 589
repeat–accumulate, **582**
repetition, 183
simple parity, 218
sparse graph, 556
density evolution, 566
syndrome decoding, 11, 371
variable rate, 238, 590
very bad, 207
very good, 183
weight enumerator, 206
with varying level of protection, 239
error-reject curves, 533
errors, *see* channel
estimator, 48, 307, 320, 446, 459
eugenics, 273
euro, 63
evidence, 29, 53, 298, 322, 347, 531
typical behaviour of, 54, 60
evolution, 269, 279
as learning, 277
Baldwin effect, 279
colour vision, 554
of the genetic code, 279
evolutionary computing, 394, 395
exact sampling, 413
exchange rate, 601
exchangeability, 263
exclusive or, 590
EXIT chart, 567
expectation, 27, 35, 37
expectation propagation, 340
expectation–maximization algorithm, 283, 432
experimental design, 463
experimental skill, 309
explaining away, 293, 295
exploit, 453
explore, 453
exponential distribution, 45, 313
on integers, 311
exponential-family, 307, 308
expurgation, 167, 171
extended channel, 153, 159
extended code, **92**
extended ensemble, 76
extra bit, **98**, 101
extreme value, 446
eye movements, 554

factor analysis, 437, 444
factor graph, 334–336, 434, 556, 557, 580, 583
factorial, 2
fading channel, 186
feedback, 506, 589
female, 277
ferromagnetic, 400
Feynman, Richard, 422
Fibonacci, 253
field, 605, *see* Galois field
file storage, 188
finger, 119
finite field theory, *see* Galois field
fitness, 269, 279
fixed point, 508
Florida, 355
fluctuation analysis, 446
fluctuations, 401, 404, 427, 602
focus, 529
football pools, 209
forensic, 47, 421
forgery, 199, 200
forward pass, 244
forward probability, 27
forward–backward algorithm, 326, 330
Fotherington–Thomas, 241
fountain code, 589
Fourier transform, 88, 219, 339, 544, 568
fovea, 554
free energy, 257, 407, 409, 410, *see* partition function
minimization, 423
variational, 423
frequency, 26
frequentist, 320, *see* sampling theory
Frey, Brendan J., 353
Frobenius–Perron theorem, 410
frustration, 406
full probabilistic model, 156
function minimization, 473
functions, 246

gain, 507
Galileo code, 186
Gallager code, **557**
Gallager, Robert G., 170, 187, 557
Galois field, 185, 224, 567, 568, 605
gambling, 455
game, *see* puzzle
Bridge, 126
chess, 451
guess that tune, 204
guessing, 110
life, 520
`sixty-three`, 70
`submarine`, 71
three doors, 57, 60, 454
twenty questions, 70
game show, 57, 454
game-playing, 451
gamma distribution, 313, 319
gamma function, 598
ganglion cells, 491
Gaussian channel, 155, **177**
Gaussian distribution, 2, 36, **176**, 312, 321, 398, 549
N–dimensional, 124
approximation, 501
parameters, 319
sample from, 312
Gaussian processes, 535
variational classifier, 547
general position, 484
generalization, 483
generalized parity-check matrix, 581
generating function, 88
generative model, 27, 156
generator matrix, 9, 183
genes, 201
genetic algorithm, 269, 395, 396
genetic code, 279
genome, 201, 280
geometric progression, 258
geostatistics, 536, 548
GF(q), *see* Galois field
Gibbs entropy, 85
Gibbs sampling, **370**, 391, 418, *see* Monte Carlo methods
Gibbs' inequality, 34, 37, 44
Gilbert–Varshamov conjecture, 212
Gilbert–Varshamov distance, 212, 221
Gilbert–Varshamov rate, 212
Gilks, Wally R., 393
girlie stuff, 58
Glauber dynamics, 370
Glavieux, A., 186
Golay code, 209
golden ratio, 253
good, *see* error-correcting code
Good, Jack, 265
gradient descent, 476, 479, 498, 529
natural, 443
graduated non-convexity, 518
Graham, Ronald L., 175
grain size, 180
graph, 251
factor graph, 334
of code, 19, 20, 556
graphs and cycles, 242
guerilla, 242

guessing decoder, 224
guessing game, 110, 111, 115
Gull, Steve, 48, 61, 551
`gzip`, 119

Haldane, J.B.S., 278
Hamilton, William D., 278
Hamiltonian Monte Carlo, **387**, 397, 496, 497
Hamming code, **8**, 17, 183, 184, 190, 208, 209, 214, 219
  graph, 19
Hamming distance, 206
handwritten digits, 156
hard drive, 593
hash code, 193, 231
hash function, 195, 200, 228
  linear, 231
  one-way, 200
hat puzzle, 222
heat bath, 370, 601
heat capacity, 401, 404
Hebb, Donald, 505
Hebbian learning, 505, 507
Hertz, 178
Hessian, 501
hidden Markov model, 437
hidden neurons, 525
hierarchical clustering, 284
hierarchical model, 379, 548
high dimensions, life in, 37, 124
hint for computing mutual information, 149
Hinton, Geoffrey E., 353, 429, 432, 522
hitchhiker, 280
homogeneous, 544
Hooke, Robert, 200
Hopfield network, 283, **505**, 506, 517
  capacity, 514
Hopfield, John J., 246, 280, 517
horse race, 455
hot-spot, 275
Huffman code, 91, **99**, 103
  'optimality', 99, 101
  disadvantages, 100, 115
  general alphabet, 104, 107
human, 269
human–machine interfaces, 119, 127
hybrid Monte Carlo, 387, *see* Hamiltonian Monte Carlo
hydrogen bond, 280
hyperparameter, 64, 309, 318, 319, 379, 479
hypersphere, 42
hypothesis testing, *see* model comparison, sampling theory

i.i.d., 80
ICA, *see* independent component analysis
ICF (intrinsic correlation function), 551
identical twin, 111
identity matrix, 600
ignorance, 446
ill-posed problem, 309, 310
image, 549
  integral, 246
image analysis, 343, 351
image compression, 74, 284
image models, 399
image processing, 246
image reconstruction, 551
implicit assumptions, 186
implicit probabilities, **97**, 98, 102
importance sampling, **361**, 379
  weakness of, 382
improper, 314, 316, 319, 320, 342, 353
in-car navigation, 594
independence, 138
independent component analysis, 313, **437**, 443
indicator function, 600
inequality, 35, 81
inference, 27, 529
  and learning, 493
information content, 32, 72, 73, 91, 97, 115, 349
  how to measure, 67
  Shannon, 67
information maximization, 443
information retrieval, 193
information theory, 4
inner code, 184
Inquisition, 346
insertions, 187
instantaneous, 92
integral image, 246
interleaving, 184, 186, 579
internet, 188, 589
intersection, 66, 222
intrinsic correlation function, 549, 551
invariance, 445
invariant distribution, 372
inverse probability, 27
inverse-arithmetic-coder, 118
inverse-cosh distribution, 313
inverse-gamma distribution, 314
inversion of hash function, 199
investment portfolio, 455
irregular, 568
ISBN, 235
Ising model, 130, 283, 399, 400
iterative probabilistic decoding, 557

Jaakkola, Tommi S., 433, 547
Jacobian, 320
janitor, 464
Jeffreys prior, 316
Jensen's inequality, 35, 44
Jet Propulsion Laboratory, 186
Johnson noise, 177
joint ensemble, 138
joint entropy, 138
joint typicality, 162
joint typicality theorem, 163
Jordan, Michael I., 433, 547
journal publication policy, 463
judge, 55
juggling, 15
junction tree algorithm, 340
jury, 26, 55

K-means clustering, **285**, **303**
  derivation, 303
  soft, 289
kaboom, 306, 433
Kalman filter, 535
kernel, 548
key points
  communication, 596
  how much data needed, 53
  likelihood principle, 32
  model comparison, 53
  Monte Carlo, 358, 367
  solving probability problems, 61
keyboard, 119
Kikuchi free energy, 434
KL distance, 34
Knowlton–Graham partitions, 175
Knuth, Donald, xii
Kolmogorov, Andrei Nikolaevich, 548
Kraft inequality, **94**, 521
Kraft, L.G., 95
kriging, 536
Kullback–Leibler divergence, 34, *see* relative entropy

Lagrange multiplier, 174
lake, 359
Langevin method, **496**, 498
Langevin process, 535
language model, 119
Laplace approximation, *see* Laplace's method
Laplace model, 117
Laplace prior, 316
Laplace's method, 341, 354, 496, 501, 537, 547
Laplace's rule, 52
latent variable, 432, 437
latent variable model, 283
  compression, 353
law of large numbers, 36, 81, 82, 85
lawyer, 55, 58, 61
Le Cun, Yann, 121
leaf, 336
leapfrog algorithm, 389
learning, 471
  as communication, 483
  as inference, 492, 493
  Hebbian, 505, 507
  in evolution, 277
learning algorithms, 468, *see* algorithm
  backpropagation, 528
  Boltzmann machine, 522
  classification, 475
  competitive learning, 285
  Hopfield network, 505
  K-means clustering, **286**, **289**, 303
  multilayer perceptron, 528
  single neuron, 475
learning rule, 470
Lempel–Ziv coding, 110, 119–122
  criticisms, 128
life, 520

life in high dimensions, 37, 124
likelihood, 6, 28, 49, 152, 324, 529, 558
  contrasted with probability, 28
  subjectivity, 30
likelihood equivalence, 447
likelihood principle, 32, 61, 464
limit cycle, 508
linear block code, 9, 11, 19, 171, 183, 186, 206, 229
  coding theorem, 229
  decoding, 184
linear regression, 342, 527
linear-feedback shift-register, 184, 574
Litsyn, Simon, 572
little 'n' large data set, 288
log-normal, 315
logarithms, 2
logit, 307, 316
long thin strip, 409
loopy belief propagation, 434
loopy message-passing, 338, 340, 556
loss function, 451
lossy compression, 168, 284, 285
low-density generator-matrix code, 207, 590
low-density parity-check code, **557**, *see* error-correcting code
LT code, 590
Luby, Michael G., 568, 590
Luria, Salvador, 446
Lyapunov function, 287, 291, **508**, 520, 521

machine learning, 246
macho, 319
MacKay, David J.C., 187, 496, 557
magician, 233
magnet, 602
magnetic recording, 593
majority vote, 5
male, 277
Mandelbrot, Benoit, 262
MAP, *see* maximum *a posteriori*
mapping, 92
marginal entropy, 139, 140
marginal likelihood, 29, 298, 322, *see* evidence
marginal probability, 23, 147
marginalization, 29, 295, 319
Markov chain, 141, 168
  construction, 373
Markov chain Monte Carlo, *see* Monte Carlo methods
Markov model, 111, 437, *see* Markov chain
marriage, 454
matrix, 409
matrix identities, 438
max–product, 339
maxent, 308, *see* maximum entropy
maximum distance separable, 219
maximum entropy, 308, 551
maximum likelihood, 6, 152, 300, 347
maximum *a posteriori*, 6, 307, 325, 538
McCollough effect, 553
MCMC (Markov chain Monte Carlo), *see* Monte Carlo methods
McMillan, B., 95
MD5, 200
MDL, *see* minimum description length
MDS, 220
mean, 1
mean field theory, 422, 425
melody, 201, 203
memory, 468
  address-based, 468
  associative, 468, 505
  content-addressable, 192, 469
MemSys, 551
message passing, 187, 241, 248, 283, 324, 407, 556, 591
  BCJR, 330
  belief propagation, 330
  forward–backward, 330
  in graphs with cycles, 338
  loopy, 338, 340, 434
  sum–product algorithm, 336
  Viterbi, 329
metacode, 104, 108
metric, 512
Metropolis method, 496, *see* Monte Carlo methods
Mézard, Marc, 340
micro-saccades, 554
microcanonical, 87
microsoftus, 458
microwave oven, 127
min–sum algorithm, 245, 325, 329, 339, 578, 581
mine (hole in ground), 451
minimax, 455
minimization, 473, *see* optimization
minimum description length, **352**
minimum distance, 206, 214, *see* distance
Minka, Thomas, 340
mirror, 529
Mitzenmacher, Michael, 568
mixing coefficients, 298, 312
mixture modelling, 282, 284, 303, 437
mixture of Gaussians, 312
mixtures in Markov chains, 373
ML, *see* maximum likelihood
MLP, *see* multilayer perceptron
MML, *see* minimum description length
mobile phone, 182, 186
model, 111, 120
model comparison, 198, 346, 347, 349
  typical evidence, 54, 60
modelling, 285
  density modelling, 284, 303
  images, 524
  latent variable models, 353, 432, 437
  nonparametric, 538
moderation, 29, 498, *see* marginalization
molecules, 201
Molesworth, 241
momentum, 387, 479
Monte Carlo methods, 357, 498
  acceptance rate, 394
  acceptance ratio method, 379
  and communication, 394
  annealed importance sampling, 379
  coalescence, 413
  dependence on dimension, 358
  exact sampling, 413
  for visualization, 551
  Gibbs sampling, **370**, 391, 418
  Hamiltonian Monte Carlo, 387, 496
  hybrid Monte Carlo, *see* Hamiltonian Monte Carlo
  importance sampling, **361**, 379
    weakness of, 382
  Langevin method, 498
  Markov chain Monte Carlo, **365**
  Metropolis method, **365**
    dumb Metropolis, 394, 496
  Metropolis–Hastings, **365**
  multi-state, 392, 395, 398
  overrelaxation, 390, 391
  perfect simulation, 413
  random walk suppression, 370, 387
  random-walk Metropolis, 388
  rejection sampling, **364**
    adaptive, 370
  reversible jump, 379
  simulated annealing, 379, 392
  slice sampling, 374
  thermodynamic integration, 379
  umbrella sampling, 379
Monty Hall problem, 57
Morse, 256
motorcycle, 110
movie, 551
multilayer perceptron, 529, 535
multiple access channel, 237
multiterminal networks, 239
multivariate Gaussian, 176
Munro–Robbins theorem, 441
murder, 26, 58, 61, 354
music, 201, 203
mutation rate, 446
mutual information, 139, 146, 150, 151
  how to compute, 149
myth, 347
  compression, 74

nat (unit), 264, 601
natural gradient, 443
natural selection, 269
navigation, 594
Neal, Radford M., 111, 121, 187, 374, 379, 391, 392, 397, 419, 420, 429, 432, 496
needle, Buffon's, 38
network, 529
neural network, 468, 470
  capacity, 483
  learning as communication, 483
  learning as inference, 492

neuron, 471
  capacity, 483
Newton, Isaac, 200, 552
Newton–Raphson method, 303, 441
nines, 198
noise, 3, *see* channel
  coloured, 179
  spectral density, 177
  white, 177, 179
noisy channel, *see* channel
noisy typewriter, **148**, 152, 154
noisy-channel coding theorem, **15**, 152, 162, 171, 229
  Gaussian channel, 181
  linear codes, 229
  poor man's version, 216
noisy-or, 294
non-confusable inputs, 152
noninformative, 319
nonlinear, 535
nonlinear code, 20, 187
nonparametric data modelling, 538
nonrecursive, 575
noodle, Buffon's, 38
normal, 312, *see* Gaussian
normal graph, 219, 584
normalizing constant, *see* partition function
not-sum, 335
notation, **598**
  absolute value, 33, 599
  conventions of this book, 147
  convex/concave, 35
  entropy, 33
  error function, 156
  expectation, 37
  intervals, 90
  logarithms, 2
  matrices, 147
  probability, 22, 30
  set size, 33, 599
  transition probability, 147
  vectors, 147
NP-complete, **184**, 325, 517
nucleotide, 201, 204
nuisance parameters, 319
numerology, 208
Nyquist sampling theorem, 178

objective function, 473
Occam factor, 322, 345, **348**, 350, 352
Occam's razor, **343**
octal, **575**
`octave`, 478
odds, 456
Ode to Joy, 203
officer for whimsical departmental rules, 464
Oliver, 56
one-way hash function, 200
optic nerve, 491
optimal decoder, 152
optimal input distribution, **150**, 162
optimal linear filter, 549
optimal stopping, 454
optimization, 169, 392, 429, 479, 505, 516, 531
  gradient descent, 476
  Newton algorithm, 441
  of model complexity, 531
order parameter, 604
ordered overrelaxation, 391
orthodox statistics, 320, *see* sampling theory
outer code, 184
overfitting, 306, 322, 529, 531
overrelaxation, 390

p-value, 64, 457, 462
packet, 188, 589
paradox, 107
  Allais, 454
  bus-stop, 39
  heat capacity, 401
  Simpson's, 355
  waiting for a six, 38
paranormal, 233
parasite, 278
parent, 559
parity, 9
parity-check bits, 9, 199, 203
parity-check code, 220
parity-check constraints, 20
parity-check matrix, 12, 183, 229, 332
  generalized, 581
parity-check nodes, 19, 219, 567, 568, 583
parse, 119, 448
Parsons code, 204
parthenogenesis, 273
partial order, 418
partial partition functions, 407
particle filter, 396
partition, 174
partition function, 401, 407, 409, 422, 423, 601, 603
  analogy with lake, 360
  partial, 407
partitioned inverse, 543
path-counting, 244
pattern recognition, 156, 179, 201
pentagonful code, 21, 221
perfect code, 208, 210, 211, 219, 589
perfect simulation, 413
periodic variable, 315
permutation, 19, 268
Petersen graph, 221
phase transition, 361, 403, 601
philosophy, 26, 119, 384
phone, 125, 594
  cellular, *see* mobile phone
phone directory, 193
phone number, 58, 129
photon counter, 307, 342, 448
physics, 80, 85, 257, 357, 401, 422, 514, 601
pigeon-hole principle, 86, 573
pitchfork bifurcation, 291, 426
plaintext, 265
plankton, 359
point estimate, 432
point spread function, 549
pointer, 119
poisoned glass, 103
Poisson distribution, 2, 175, 307, 311, 342
Poisson process, 39, 46, 448
Poissonville, 39, 313
polymer, 257
poor man's coding theorem, 216
porridge, 280
portfolio, 455
positive definite, 539
positivity, 551
posterior probability, 6, 152
power cost, 180
power law, 584
practical, 183, *see* error-correcting code
precision, **176**, 181, 312, 320, 383
precisions add, 181
prediction, 29, 52
predictive distribution, 111
prefix code, **92**, 95
prior, 6, 308, 529
  assigning, 308
  improper, 353
  Jeffreys, 316
  subjectivity, 30
prior equivalence, 447
priority of bits in a message, 239
prize, on game show, 57
probabilistic model, 111, 120
probabilistic movie, 551
probability, 26, 38
  Bayesian, 50
  contrasted with likelihood, 28
  density, 30, 33
probability distributions, 311, *see* distribution
probability of block error, 152
probability propagation, *see* sum–product algorithm
product code, 184, 214
profile, of random graph, 568
pronunciation, 34
proper, 539
proposal density, **364**, 365
Propp, Jim G., 413, 418
prosecutor's fallacy, 25
prospecting, 451
protein, 204, 269
  regulatory, 201, 204
  synthesis, 280
protocol, 589
pseudoinverse, 550
Punch, 448
puncturing, 222, 580
pupil, 553
puzzle, *see* game
  cable labelling, 173
  chessboard, 520
  fidelity of DNA replication, 280
  hat, 222, 223
  life, 520
  magic trick, 233, 234

poisoned glass, 103
secretary, 454
`southeast`, 520
transatlantic cable, 173
weighing 12 balls, 68

quantum error-correction, 572
queue, 454
QWERTY, 119

R_3, *see* repetition code
race, 354
radial basis function, 535, 536
radio, 186
radix, 104
RAID, 188, 190, 219, 593
random, 26, 357
random cluster model, 418
random code, 156, 161, 164, 165, 184, 192, 195, 214, 565
for compression, 231
random number generator, 578
random variable, 26, 463
random walk, 367
suppression, 370, 387, 390, 395
random-coding exponent, 171
random-walk Metropolis method, 388
rant, *see* sermon
raptor codes, 594
rate, **152**
rate-distortion theory, 167
rateless code, 590
reading aloud, 529
receiver operating characteristic, 533
recognition, 204
record breaking, 446
rectangular code, 184
reducible, 373
redundancy, 4, 33
in channel code, 146
redundant array of independent disks, 188, 190, 219, 593
redundant constraints in code, 20
Reed–Solomon code, 185, 571, 589
regression, 342, 536
regret, 455
regular, 557
regularization, 529, 550
regularization constant, 309, 479
reinforcement learning, 453
rejection, 364, 366, 533
rejection sampling, **364**
adaptive, 370
relative entropy, 34, 98, 102, 142, 422, 429, 435, 475
reliability function, 171
repeat–accumulate code, **582**
repetition code, 5, 13, 15, 16, 46, 183
responsibility, 289
retransmission, 589
reverse, 110
reversible, 374
reversible jump, 379
Richardson, Thomas J., 570, 595
Rissanen, Jorma, 111
ROC, 533
rolling die, 38
roman, 127
rule of thumb, 380
runlength, 256
runlength-limited channel, 249

saccades, 554
saddle-point approximation, 341
sailor, 307
sample, 312, 356
from Gaussian, 312
sampler density, 362
sampling distribution, 459
sampling theory, 38, 320
criticisms, 32, 64
sandwiching method, 419
satellite communications, 186, 594
scaling, 203
Schönberg, 203
Schottky anomaly, 404
scientists, 309
secret, 200
secretary problem, 454
security, 199, 201
seek time, 593
Sejnowski, Terry J., 522
self-delimiting, 132
self-dual, 218
self-orthogonal, 218
self-punctuating, 92
separation, 242, 246
sequence, 344
sequential decoding, 581
sequential probability ratio test, 464
sermon, *see* caution
classical statistics, 64
confidence level, 465
dimensions, 180
gradient descent, 441
illegal integral, 180
importance sampling, 382
interleaving, 189
MAP method, 283, 306
maximum entropy, 308
maximum likelihood, 306
most probable is atypical, 283
p-value, 463
sampling theory, 64
sphere-packing, 209, 212
stopping rule, 463
turbo codes, 581
unbiased estimator, 307
worst-case-ism, 207
set, 66
shannon (unit), 265
Shannon, Claude, 3, 14, 15, 152, 164, 212, 215, 262, *see* noisy-channel coding theorem, source coding theorem, information content
shattering, 485
shifter ensemble, 524
Shokrollahi, M. Amin, 568
shortening, 222
Siegel, Paul, 262
sigmoid, 473, 527
signal-to-noise ratio, 177, 178, 223
significance level, 51, 64, 457, 463
simplex, 173, 316
Simpson's paradox, 355
Simpson, O.J., *see* wife-beaters
simulated annealing, 379, 392, *see* annealing
six, waiting for, 38
Skilling, John, 392
sleep, 524, 554
Slepian–Wolf, *see* dependent sources
slice sampling, 374
multi-dimensional, 378
soft K-means clustering, 289
softmax, softmin, 289, 316, 339
software, xi
arithmetic coding, 121
`BUGS`, 371
Dasher, 119
free, xii
Gaussian processes, 534
hash function, 200
`VIBES`, 431
solar system, 346
soldier, 241
soliton distribution, 592
sound, 187
source code, 73, *see* compression, symbol code, arithmetic coding, Lempel–Ziv
algorithms, 119, 121
block code, 76
block-sorting compression, 121
Burrows–Wheeler transform, 121
for complex sources, 353
for constrained channel, 249, 255
for integers, 132
Huffman, *see* Huffman code
implicit probabilities, 102
optimal lengths, 97, 102
prefix code, 95
software, 121
stream codes, 110–130
supermarket, 96, 104, 112
symbol code, 91
uniquely decodeable, 94
variable symbol durations, 125, 256
source coding theorem, 78, 91, 229, 231
`southeast` puzzle, 520
span, 331
sparse-graph code, 338, 556
density evolution, 566
profile, 569
sparsifier, 255
species, 269
`spell`, 201
sphere packing, 182, 205
sphere-packing exponent, 172
Spielman, Daniel A., 568
spin system, 400
spines, 525
spline, 538

spread spectrum, 182, 188
spring, 291
spy, 464
square, 38
staircase, 569, 587
stalactite, 214
standard deviation, 320
stars, 307
state diagram, 251
statistic, 458
  sufficient, 300
statistical physics, *see* physics
statistical test, 51, 458
steepest descents, 441
stereoscopic vision, 524
stiffness, 289
Stirling's approximation, 1, 8
stochastic, 472
stochastic dynamics, *see* Hamiltonian Monte Carlo
stochastic gradient, 476
stop-when-it's-done, 561, 583
stopping rule, 463
straws, drawing, 233
stream codes, 110–130
student, 125
Student-t distribution, 312, 323
subjective probability, 26, 30
`submarine`, 71
subscriber, 593
subset, 66
substring, 119
sufficient statistics, 300
sum rule, 39, 46
sum–product algorithm, 187, 245, 326, **336**, 407, 434, 556, 572, 578
summary, 335
summary state, 418
summation convention, 438
super-channel, 184
supermarket (codewords), 96, 104, 112
support vector, 548
surprise value, 264
survey propagation, 340
suspicious coincidences, 351
symbol code, **91**
  budget, 94, **96**
  codeword, **92**
  disadvantages, 100
  optimal, 91
  self-delimiting, 132
  supermarket, 112
symmetric channel, 171
symmetry argument, 151
synchronization, 249
synchronization errors, 187
syndrome, 10, 11, 20
syndrome decoding, 11, 216, 229, 371
systematic, 575

t-distribution, *see* Student-t
tail, 85, 312, 313, 440, 446, 503, 584
tamper detection, 199
Tank, David W., 517
Tanner challenge, 569
Tanner product code, 571
Tanner, Michael, 569
Tanzanite, 451
tap, 575
telephone, *see* phone
telescope, 529
temperature, 392, 601
termination, 579
terminology, 598, *see* notation
  Monte Carlo methods, 372
test
  fluctuation, 446
  statistical, 51, 458
text entry, 118
thermal distribution, 88
thermodynamic integration, 379
thermodynamics, 404, 601
  third law, 406
Thiele, T.N., 548
thin shell, 37, 125
third law of thermodynamics, 406
Thitimajshima, P., 186
three cards, 142
three doors, 57
threshold, 567
tiling, 420
time-division, 237
timing, 187
training data, 529
transatlantic, 173
transfer matrix method, 407
transition, 251
transition probability, 147, 356, 607
translation-invariant, 409
travelling salesman problem, 246, 517
tree, 242, 336, 343, 351
trellis, 251, **326**, 574, 577, 580, 583, 608
  section, 251, 257
  termination, 579
triangle, 307
truth function, 211, 600
tube, 257
turbo code, 186, 556
turbo product code, 571
Turing, Alan, 265
twenty questions, 70, 103
twin, 111
twos, 156
typical evidence, 54, 60
typical set, **80**, 154, 363
  for compression, 80
  for noisy channel, 154
typical-set decoder, 165, 230
typicality, **78**, 80, 162

umbrella sampling, 379
unbiased estimator, 307, 321, 449
uncompression, 231
union, 66
union bound, 166, 216, 230
uniquely decodeable, **93**, 94
units, 264
universal, 110, 120, 121, 135, 590
universality, in physics, 400
Urbanke, Rüdiger, 570, 595
urn, 31
user interfaces, 118
utility, 451

vaccination, 458
Vapnik–Chervonenkis dimension, 489
variable-length code, 249, 255
variable-rate error-correcting codes, 238, 590
variance, 1, 27, 88, 321
variance–covariance matrix, 176
variances add, 1, 181
variational Bayes, 429
variational free energy, 422, 423
variational methods, 422, 433, 496, 508
  typical properties, 435
  variational Gaussian process, 547
VC dimension, 489
vector quantization, 284, 290
very good, *see* error-correcting code
`VIBES`, 431
Virtakallio, Juhani, 209
vision, 554
visualization, 551
Viterbi algorithm, 245, 329, 340, 578
volume, 42, 90
Von Mises distribution, 315

Wainwright, Martin, 340
waiting for a bus, 39, 46
warning, *see* caution *and* sermon
Watson–Crick base pairing, 280
weather collator, 236
weighing babies, 164
weighing problem, 66, 68
weight
  importance sampling, 362
  in neural net, 471
  of binary vector, 20
weight decay, 479, 529
weight enumerator, 206, 211, 214, 216
  typical, 572
weight space, 473, 474, 487
Wenglish, 72, 260
what number comes next?, 344
white, 355
white noise, 177, 179
Wiberg, Niclas, 187
widget, 309
Wiener process, 535
Wiener, Norbert, 548
wife-beater, 58, 61
Wilson, David B., 413, 418
window, 307
Winfree, Erik, 520
wodge, 309
Wolf, Jack, 262
word-English, 260
world record, 446
worst-case-ism, 207, 213
writing, 118

Yedidia, Jonathan, 340

Z channel, **148**, 149–151, 155
Zipf plot, 262, 263, 317
Zipf's law, 40, 262, 263
Zipf, George K., 262