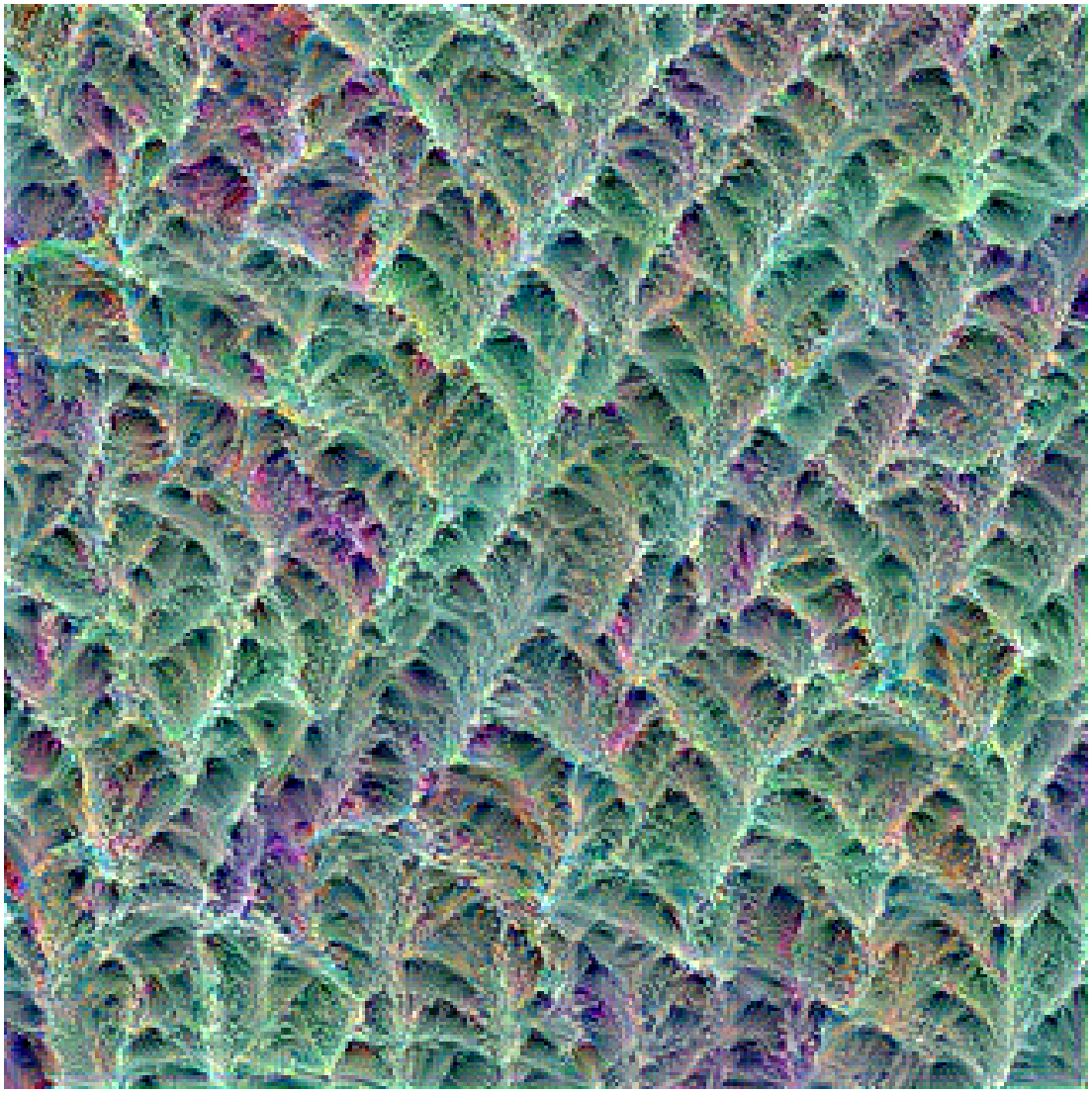


Exercise solutions

Machine learning with neural networks
An introduction for scientists and engineers

BERNHARD MEHLIG

Department of Physics
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden 2022

Exercise solutions
Machine learning with neural networks
An introduction for scientists and engineers

BERNHARD MEHLIG

Department of Physics
University of Gothenburg
Göteborg, Sweden 2022

1 Introduction

This document contains solutions for most of the exercises in *Machine learning with neural networks. An Introduction for scientists and engineers* by B.Mehlig (Cambridge University Press, 2021).

Many exercises solved here derive from exam questions for a course on machine learning with neural networks that I have given at Chalmers and Gothenburg University. I thank Oleksandr Balabanov, Anshuman Dubey, Johan Fries, Jan Meibohm, Marina Rafajlovic, and Ludvig Storm for contributing exam questions. Teaching assistants and colleagues have helped over the years to compile the solutions presented here. I am particularly grateful to Juan Diego Arango, Oleksandr Balabanov, Anshmuan Dubey, Ehsan Ghane, Phillip Gräfensteiner, Hampus Linander, Robert Mehlig, Jan Meibohm, Navid Mousavi, Jan Schiffeler, Ludvig Storm, Marina Rafajlovic, and Arvid Wenzel Wartenberg for contributing solutions. I would also like to thank John Segerstedt who took the initiative to compile solutions to exam questions (a very good exam preparation!), as well as Abraham Deniz for typesetting an early version of this compilation.

The cover image shows an input pattern designed to maximise the output of neurons corresponding to one feature map in a given convolution layer of a deep convolutional neural network. I thank Hampus Linander for making this image.

In the text that follows, references to equations, figures, tables, and citations in *Machine learning with neural networks. An Introduction for scientists and engineers* are typeset in red. A list of *errata* for this book is available from Cambridge Universtify press. Please email me if you find errors or inaccuracies in the solutions.

Gothenburg, October 19 (2022) Bernhard Mehlig

2 Exercises in Chapter 2

Answer (2.1) — For only one pattern, the modified Hebb's rule (2.26) reads: $w_{ij} = \frac{1}{N}(x_i x_j - \delta_{ij})$. Feed pattern $\boldsymbol{x}$ to the network, $\boldsymbol{s}(0) = \boldsymbol{x}$. Now evaluate $\sum_j w_{ij} x_j = \sum_j \frac{1}{N}(x_i x_j x_j - \delta_{ij}) x_j = (1 - \frac{1}{N}) x_i$, using $\sum_j x_j x_j = N$. Now apply the signum function. This gives $\mathrm{sgn}[(1-\frac{1}{N})x_i] = x_i$. It follows that pattern $\boldsymbol{x}$ is reproduced under the network dynamics, so Equation (2.8) is satisfied.

Answer (2.2) — Using Equation (2.13), the weight matrix for Hebb's rule (2.25) can be written as $\mathbb{W} = \frac{1}{N}\sum_\mu \boldsymbol{x}^{(\mu)}\boldsymbol{x}^{(\mu)\mathsf{T}}$. If the patterns are orthogonal ($\boldsymbol{x}^{(\mu)\mathsf{T}}\boldsymbol{x}^{(\nu)} = \delta_{\mu\nu}$), then $\mathbb{W}\boldsymbol{x}^{(\nu)} = \boldsymbol{x}^{(\nu)}$, so $\boldsymbol{x}^{(\nu)}$ is recognised. This means that the cross-talk term vanishes. For the modified Hebb's rule (2.26), the weight matrix takes the form $\mathbb{W} = \frac{1}{N}\sum_\mu(\boldsymbol{x}^{(\mu)}\boldsymbol{x}^{(\mu)\mathsf{T}} - \mathbb{I})$. In this case, $\mathbb{W}\boldsymbol{x}^{(\nu)} = (1-\frac{1}{N})\boldsymbol{x}^{(\nu)}$. This implies that the cross-talk term vanishes as $N \to \infty$.

Answer (2.3) — When we use Hebb's rule (2.25), the local field is obtained as $b_i^{(\nu)} = x_i^{(\nu)} + \frac{1}{N}\sum_{j=1}^N \sum_{\mu\neq\nu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)}$, instead of Equation (2.28). This implies a slightly different definition of the cross-talk term. Equation (2.33) is replaced by:

$$C_i^{(\nu)} = -x_i^{(\nu)} \frac{1}{N} \sum_{j=1}^{N} \sum_{\mu\neq\nu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)}. \tag{2.1}$$

Averaging over random patterns shows that the mean of $C_i^{(\nu)}$ is non-zero, $\langle C_i^{(\nu)}\rangle = -(p-1)/N \approx -p/N$ for large p. This means that the distribution of C is a shifted Gaussian, $P(C) = (2\pi\sigma_C)^{1/2}\exp[-(C-\langle C\rangle)^2/(2\sigma_C^2)]$, instead of Equation (2.36). For small $\alpha = p/N$, the mean tends to zero, so that the new distribution approaches Equation (2.36). For large values of α, the mean $\langle C\rangle$ dominates the error probability. In the limit $\alpha \to \infty$, the mean of $\langle\mathbb{W}\rangle = \frac{p}{N}\mathbb{I}$ dominates the network dynamics. The one-step error probability tends to zero in this limit because all states are reproduced, but the network cannot learn anything meaningful.

Answer (2.4) — Consider $\mathrm{sgn}(\pm x_i^{(1)} \pm x_i^{(2)})$. For $x_i^{(\nu)} = \pm 1$, the argument of the signum function may evaluate to zero, but sgn(0) is not defined.
To show that there are $2^{2n+1}\binom{p}{2n+1}$ mixed states that are superpositions of $2n+1$ out of p patterns, note that there are $\binom{p}{2n+1}$ ways of choosing $2n+1$ out of p patterns, and there are $2^{(2n+1)}$ ways of distributing the signs in Equation (2.52).

Answer (2.5) — Figure 2.1 shows numerical results for the one-step error probability for the mixed state (**??**) It is much larger than the one-step error probability

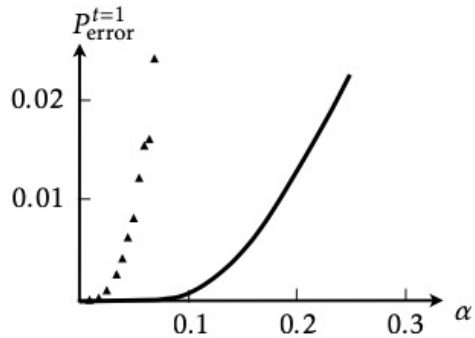


Figure 2.1: One-step error probability for mixed states of the form (**??**), ▲. For comparison, the one-step error probability (**2.53**) for a stored pattern is also shown (solid line). Exercise 2.5.

(**2.39**) for a stored pattern. But like Equation (**2.39**), the error probability tends to zero as $\alpha \to 0$. To show this, we determine whether

$$\operatorname{sgn}\left(\frac{1}{N}\sum_{\mu=1}^{p}\sum_{j=1}^{N} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\text{mix})}\right) = x_i^{(\text{mix})} \tag{2.3}$$

is satisfied in the limit. Following Ref. [**1**], we split the sum in the usual fashion

$$\frac{1}{N}\sum_{\mu=1}^{p}\sum_{j=1}^{N} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\text{mix})} = \sum_{\mu=1}^{3} x_i^{(\mu)} \frac{1}{N}\sum_{j=1}^{N} x_j^{(\mu)} x_j^{(\text{mix})} + \text{cross-talk term}. \tag{2.4}$$

For small values of α, we can ignore the cross-talk term (Section **2.4**). To determine whether the first term on the r.h.s. reproduces $x_i^{(\text{mix})}$, we use that the sum over j on the r.h.s. of Equation (2.4) is an average over $s_\mu = x_j^{(\mu)} x_j^{(\text{mix})}$, for large N. Table 2.1 lists all possible combinations of bits of pattern j and the corresponding values of s_μ. From Equation (**2.29**) we see that $\langle s_\mu \rangle = \frac{1}{2}$. This implies

$$\frac{1}{N}\sum_{\mu=1}^{p}\sum_{j=1}^{N} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\text{mix})} = \frac{1}{2}\sum_{\mu=1}^{3} x_i^{(\mu)}. \tag{2.5}$$

Taking the signum-function, we see that $\boldsymbol{x}^{(\text{mix})}$ is reproduced.

Answer (2.6) — Consider the update $s_2' = \operatorname{sgn}(w_{21} s_1) = -\operatorname{sgn}(s_1)$ because $w_{21} = -1$. Then $H' - H = -\frac{1}{2}(w_{21} + w_{12})(s_1 s_2' - s_1 s_2) = \frac{1}{2}(w_{21} + w_{12}) s_1 (s_1 + s_2)$. Inserting $w_{21} = -1$ and $w_{12} = 2$ gives $H' - H = \frac{1}{2} s_1 (s_1 + s_2) > 0$ if s_1 and s_2 have the same sign.

Answer (2.7) — Assume that the second-order weights $w_{ij}^{(2)}$ are symmetric, and that the diagonal weights are zero. Further, assume that the third-order weights are

Table 2.1: Signs of $s_\mu = x_j^{(\mu)} x_j^{(\text{mix})}$. Exercise 2.5.

$x_j^{(1)}$	$x_j^{(2)}$	$x_j^{(3)}$	$x_j^{(\text{mix})}$	s_1	s_2	s_3
1	1	1	1	1	1	1
1	1	-1	1	1	1	-1
1	-1	1	1	1	-1	1
1	-1	-1	-1	-1	1	1
-1	1	1	1	-1	1	1
-1	1	-1	-1	1	-1	1
-1	-1	1	-1	1	1	-1
-1	-1	-1	-1	1	1	1

symmetric, and that they vanish when at least two indices are the same:

$$\begin{aligned} w^{(3)}_{ijk} &= w^{(3)}_{jik} = w^{(3)}_{ikj} = w^{(3)}_{kji} \\ w^{(3)}_{iik} &= w^{(3)}_{iji} = w^{(3)}_{ijj} = 0\,. \end{aligned} \tag{2.6}$$

Now evaluate the derivative $-\partial H/\partial s_m$ to obtain the asynchronous update rule. One finds

$$s'_m = \text{sgn}(b_m) \quad \text{with} \quad b_m = \sum_j w^{(2)}_{mj} s_j + \sum_{jk} w^{(3)}_{mjk} s_j s_k\,. \tag{2.7}$$

A calculation analogous to that summarised in Section **2.5** shows that H cannot increase under this update rule.

Answer (2.8) — Since $s = \pm 1$ is mapped to $n = 0, 1$ by $s = 2n - 1$, try

$$w_{ij} = \tfrac{1}{N}(2n_i - 1)(2n_j - 1)\,, \quad w_{ii} = 0\,, \tag{2.8}$$

and zero thresholds. Show that this rule works for one stored pattern. The calculation is analogous to Equations (**2.10**) and (**2.11**):

$$\sum_{j\neq i} w_{ij} n_j = \frac{1}{N}\sum_{j\neq i}(2n_i - 1)(2n_j - 1)n_j = (2n_i - 1)\Big(2 - \frac{1}{N}\sum_{j\neq i} n_j\Big), \tag{2.9}$$

Here it was used that $n_j^2 = n_j$ since $n_j = 0, 1$. Since the right factor on the r.h.s. of Eq. (2.9) is positive, one finds:

$$n'_i = \theta_{\text{H}}(2n_i - 1)\,. \tag{2.10}$$

Since $\theta_{\text{H}}(2n_i - 1) = n_i$, we conclude that n_i remains unchanged. So the pattern is recognised.

The answer for the second part is analogous to Equations (2.46) to (2.49). Update neuron m and assume that its state changes, $n'_m \neq n_m$. Assuming that the diagonal weights are set to zero, the resulting change in the energy function is

$$H' - H = -\tfrac{1}{2}\sum_{j\neq m}(w_{mj} + w_{jm})(n'_m n_j - n_m n_j) + \mu_m(n'_m - n_m). \tag{2.11}$$

Assuming that the weights are symmetric, the first term can be evaluated further,

$$H' - H = -(n'_m - n_m)\Big(\sum_{j\neq m} w_{mj} n_j - \mu_m\Big). \tag{2.12}$$

If $n_m = 1$ then $n'_m = 0$, and the factor in the parentheses is negative, so that $H' - H < 0$. If $n_m = 0$, then $n'_m = 1$. In this case the factor in the parentheses is positive, so that again $H' - H < 0$.

Answer (2.9) — Consider the effect of one update following the synchronous rule (1.2). Divide the neurons into two sets: for all neurons in $\mathscr{S}$, $s'_i = s_i$, but for those in the complement $\mathscr{S}^{\mathrm{c}}$ the state changed $s'_i = -s_i$. The corresponding change in the energy function (2.44) reads

$$H' - H = 2\sum_{\substack{i\in\mathscr{S}\\ j\in\mathscr{S}^{\mathrm{c}}}} w_{ij} s_i s_j\,. \tag{2.13}$$

Here it was assumed that the weights are symmetric, and that the diagonal weights vanish. Equation (2.13) simplifies to Equation (2.48) when $\mathscr{S}^{\mathrm{c}}$ contains only one neuron, number m. Now assume that all neurons except the first one change, that is $\mathscr{S} = \{1\}$ and $\mathscr{S}^{\mathrm{c}} = \{2,\ldots,N\}$. Note that this assumption fails for $N = 2$, where it corresponds to $\mathscr{S} = \{1\}$, $\mathscr{S}^{\mathrm{c}} = \{2\}$. For $N = 2$, the update rules are $s'_1 = \mathrm{sgn}(w_{12}s_2)$ and $s'_2 = \mathrm{sgn}(w_{21}s_1)$. Since the weights are symmetric, either both neurons are updated, or none. As consequence we must require that $N > 2$. At any rate, the change in H is

$$H' - H = 2\sum_{j=2}^{N} w_{1j} s_1 s_j = 2\sum_{j=1}^{N} w_{1j} s_1 s_j = 2 s_1 b_1\,. \tag{2.14}$$

In the second equality we used that $w_{11} = 0$. Since the state of the first neuron was assumed not to change, $\mathrm{sgn}(b_1) = s_1$. This means that $s_1 b_1 > 0$. We conclude that the energy function can increase under synchronous updates. Contrast this conclusion to the change of H under asynchronous McCulloc-Pitts updates. In this case $H' - H \leq 0$, as explained in Section 2.5.

Answer (2.10) — We want to show that $\mathrm{d}E/\mathrm{d}t \leq 0$. Differentiation of E w.r.t t yields

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -\sum_{ij} w_{ij} \frac{\mathrm{d}b_i}{\mathrm{d}t} g'(b_i) g(b_j) + \sum_i \theta_i \frac{\mathrm{d}b_i}{\mathrm{d}t} g'(b_i) + \sum_i \frac{\mathrm{d}b_i}{\mathrm{d}t} b_i g'(b_i) \tag{2.15}$$

$$= \sum_i \frac{\mathrm{d}b_i}{\mathrm{d}t} g'(b_i) \Big[-\sum_j w_{ij} g(b_j) + \theta_i + b_i \Big], \tag{2.16}$$

where we used $w_{ij} = w_{ji}$ in the first step. The equation of motion of b_i reads

$$\tau \frac{\mathrm{d}b_i}{\mathrm{d}t} = \tau \sum_j w_{ij} \frac{\mathrm{d}n_i}{\mathrm{d}t} = -b_i - \theta_i + \sum_j w_{ij} g(b_j). \tag{2.17}$$

Combining Equations (2.15) and (2.17) yields

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -\frac{1}{\tau} \sum_i g'(b_i) \Big[b_i + \theta_i - \sum_j w_{ij} g(b_j) \Big]^2. \tag{2.18}$$

This expression cannot be positive because $g'(b) > 0$, and this means that local minima of E are attractors.
The steady states $\boldsymbol{n}^*$ of the dynamics satisfy $n_i^* = g(b_i^*)$, $b_i^* = \sum_j w_{ij} n_i^* - \theta_i$, and $\mathrm{d}E^*/\mathrm{d}t = 0$. But note that they need not be attractors. Consider an example, $w_{11} = w_{22} = 0$, $w_{12} = w_{21} = 10$ and $\theta_1 = \theta_2 = -5$. The steady state $\boldsymbol{n}^* = [\frac{1}{2}, \frac{1}{2}]^\mathsf{T}$ is a saddle point. See Exercise **??** and Equation (**9.18a**).

Answer (2.11) — The stored pattern shown in Figure **2.11** is represented by $\boldsymbol{x}^{(1)} = [1, -1, -1, -1]^\mathsf{T}$. The corresponding weight matrix (**2.13**) reads

$$\mathbb{W} = \frac{1}{4} \begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.19}$$

Now evaluate $\boldsymbol{s}(t = 1)$, in two steps. First, apply $\mathbb{W}$ to $\boldsymbol{s}(0)$, where $\boldsymbol{s}(0)$ contains the bits of one of the 16 possible patterns. Second, take the sgn-function. The result is that the stored pattern $\boldsymbol{x}^{(1)}$ is obtained for the five patterns that differ by at most one bit from $\boldsymbol{x}^{(1)}$. For the six patterns that differ by two bits, the network fails because the argument of the signum function is zero. Finally there are five patterns that differ by three or four bits from $\boldsymbol{x}^{(1)}$. In these cases, one obtains the inverted pattern $-\boldsymbol{x}^{(1)}$.

Answer (2.12) — Define

$$Q_{\mu\nu} = \frac{1}{N} \sum_{j=1}^{N} x_j^{(\mu)} x_j^{(\nu)}, \tag{2.20}$$

Table 2.2: Distances and overlaps between the patterns shown in Figure 2.12. Exercise 2.12.

μ	ν	$d_{\mu\nu}$	$Q_{\mu\nu}$
1	1	0	1
1	2	13	$\frac{6}{32}$
1	3	2	$\frac{28}{32}$
1	4	32	-1
1	5	16	0
2	2	0	1
2	3	15	$\frac{2}{32}$
2	4	32	$-\frac{6}{32}$
2	5	19	$-\frac{6}{32}$

as in Equation (3.50). The bit j contributes with $+1$ to $Q_{\mu\nu}$ if $x_j^{(\mu)} = x_j^{(\nu)}$, and with -1 if $x_j^{(\mu)} \neq x_j^{(\nu)}$. Since there are $N = 32$ bits, we have $Q_{\mu\nu} = (32 - 2d_{\mu\nu})/32$, where $d_{\mu\nu}$ is Hamming distance, the number of bits by which the patterns $\boldsymbol{x}^{(\mu)}$ and $\boldsymbol{x}^{(\nu)}$ differ [Equation (2.2)]. Table 2.2 lists the values of $Q_{\mu\nu}$ extracted from Figure 2.12. Hebb's rule implies that

$$b_i^{(\nu)} = \sum_j w_{ij} x^{(\nu)} = x_i^{(1)} Q_{1\nu} + x_i^{(2)} Q_{2\nu}. \tag{2.21}$$

Applying the signum function, one finds

$$\begin{aligned} &\mathrm{sgn}(b_i^{(1)}) = x_i^{(1)}, \quad \mathrm{sgn}(b_i^{(2)}) = x_i^{(2)}, \quad \mathrm{sgn}(b_i^{(3)}) = x_i^{(1)}, \\ &\mathrm{sgn}(b_i^{(4)}) = -x_i^{(1)} = x_i^{(4)}, \quad \mathrm{sgn}(b_i^{(5)}) = -x_i^{(2)}. \end{aligned}$$

We conclude that patterns $\boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(2)}$ and $\boldsymbol{x}^{(4)}$ remain unchanged.

Answer (2.13) — Hebb's rule $w_{ij} = \frac{1}{N}\sum_\mu x_i^{(\mu)} x_j^{(\mu)}$ gives for the weight matrix

$$\mathbb{W} = \frac{4}{3}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.22}$$

Since the weight matrix is proportional to the unit matrix, all patterns are reproduced, not only the stored ones. The network cannot single out the XOR patterns because Hebb's rule is based on two-point correlations of the stored patterns, but three-point correlations are needed to learn the XOR patterns (Chapter 4).

Answer (2.14) — Both s_i and $x_i^{(1)}$ can assume only the values ± 1. It follows that $d = \frac{1}{2} - \frac{1}{2N}\sum_i s_i x_i^{(1)}$. Therefore

$$H = -2N(\tfrac{1}{2} - d)^2 . \tag{2.23}$$

It was shown in Section **2.5** that H cannot increase under the asynchronous McCulloch-Pitts dynamics (**2.5**). For d this is not the case, because it is not invariant under $\boldsymbol{s} \to -\boldsymbol{s}$. The energy function decreases to a local minimum as $\boldsymbol{s} \to -\boldsymbol{x}^{(1)}$, but the distance d increases to unity. It does not account for the fact that the inverted pattern $-\boldsymbol{x}^{(1)}$ is closely related to $\boldsymbol{x}^{(1)}$.

3 Exercises in Chapter 3

Answer (3.1) — See Ref. [33].

Answer (3.2) — First start from Equation (3.32),

$$\beta(1-q)=\beta\int_{-\infty}^{\infty}\frac{\mathrm{d}z}{\sqrt{2\pi}\sigma_z}e^{-z^2/2\sigma_z^2}\big(1-\tanh^2(\beta m_1+\beta z)\big). \tag{3.1}$$

Equation (6.20) shows that

$$1-\tanh^2(\beta m_1+\beta z)=\tfrac{1}{\beta}\tfrac{\mathrm{d}}{\mathrm{d}z}\tanh(\beta m_1+\beta z)\approx\tfrac{1}{2}\delta(m_1+z) \tag{3.2}$$

in the limit of large β. This gives

$$\beta(1-q)\approx\sqrt{\frac{2}{\pi\sigma_z^2}}e^{-m_1^2/2\sigma_z^2}. \tag{3.3}$$

This is Equation (3.39b). We see, in particular, that $\beta(1-q)$ assumes a finite limit as $\beta\to\infty$, so that q must tend to unity in this limit. Therefore it follows from Equation (3.38) that

$$\sigma_z^2=\frac{\alpha}{[1-\beta(1-q)]^2}. \tag{3.4}$$

This is Equation (3.9a). Finally consider Equation (3.26). In the limit $\beta\to\infty$, $\tanh(\beta m_1+\beta z)$ tends to $\mathrm{sgn}(\beta m_1+\beta z)$. Using the definition of the error function [Equation (2.40)], one finds

$$m_1=\mathrm{erf}\big(m_1/\sqrt{2\sigma_z^2}\big). \tag{3.5}$$

This is Equation (3.39c).

Answer (3.3) — Start with Equation (3.39c). Inserting the definition $y=m_1/\sqrt{2\sigma_z^2}$ (page 46), one finds

$$\sqrt{2}\sigma_z\,y=\mathrm{erf}(y). \tag{3.6}$$

Combining Equations (3.39a) and (3.39b) gives

$$\sigma_z-\sqrt{2/\pi}\mathrm{e}^{-y^2}=\sqrt{\alpha}. \tag{3.7}$$

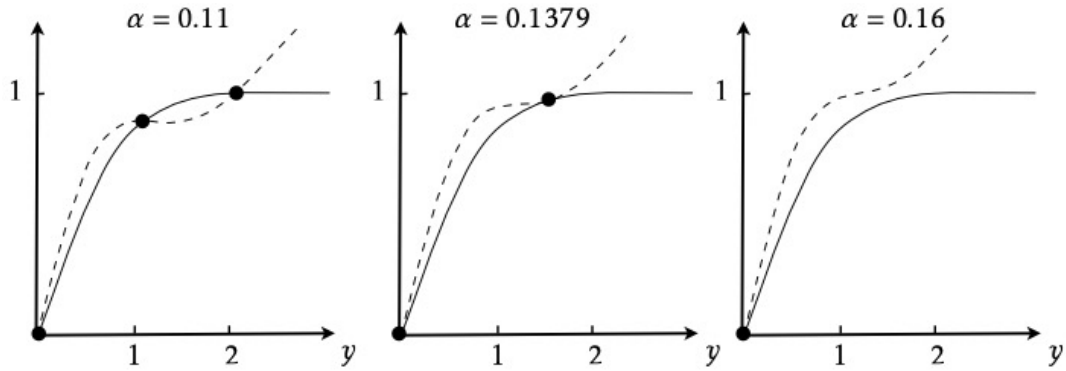


Figure 3.1: Numerical solution of Equation (3.8) for three values of α, $\alpha < \alpha_c$ (left), $\alpha = \alpha_c$ (center), and $\alpha > \alpha_c$ (right). Shown are the l.h.s. of Equation (3.8), dashed lines, and the r.h.s. of Equation (3.8), solid lines. See also Figure 2.16 in Ref. [1]. Exercise 3.3.

Substituting (3.7) into (3.6) gives

$$\sqrt{2}y\left[\sqrt{\alpha}+\sqrt{2/\pi}\mathrm{e}^{-y^2}\right]=\mathrm{erf}(y). \tag{3.8}$$

This expression is equivalent to Equation (3.41). The solutions of Equation (3.8) are illustrated in Figure 3.1. For $\alpha > \alpha_c$, there are no solutions apart from $y = 0$ (which corresponds to $m_1 = 0$). For $\alpha < \alpha_c$, there are two additional solutions. The relevant one is the one for which $m_1 \to 1$ as $\alpha \to 0$, as predicted by the mean-field theory for $\alpha \ll 1$ in the limit of weak noise, Section 3.3. Starting with a large enough value of α ($\alpha > \alpha_c$), one can numerically solve Equation (3.8) to determine the bifurcation value α_c where the two non-zero solutions occur. This yields $\alpha_c \approx 0.1379$, see Equation (3.42).
The phase diagram for finite noise levels is obtained by solving Equations (3.26), (3.33), and (3.38) which are reproduced here in an equivalent form:

$$m_1 = \int \mathrm{d}z P(z)\tanh(\beta m_1 + \beta z), \tag{3.9a}$$

$$q = \int \mathrm{d}z P(z)\tanh^2(\beta m_1 + \beta z), \tag{3.9b}$$

$$P(z) = (2\pi\sigma_z^2)^{-1/2}\exp[-z^2/(2\sigma_z^2)] \quad \text{with} \quad \sigma_z^2 = \alpha q/[1-\beta(1-q)]^2. \tag{3.9c}$$

Equations (3.9) can be solved numerically for m_1, q, and σ_z^2. As in the deterministic limit, there can be multiple solutions. Which one to choose is discussed in Ref. [34]. Figure 3.2 shows numerical solutions of Equations (3.9), namely how the order parameter decreases as the noise level increases, for $\alpha = 0.01$ and $\alpha = 0.05$. The

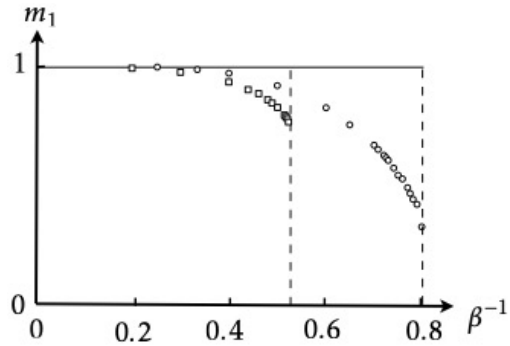


Figure 3.2: Solutions of Equation (3.8) for $\alpha = 0.05$ (□), and $\alpha = 0.01$ (∘). Shown is m_1 as a function of the noise level β^{-1}. The vertical dashed lines indicate the approximate value of the critical noise level above which the network ceases to function. See Fig. 3 in Ref. [34]. Exercise 3.3.

vertical dashed lines in Figure 3.2 show the critical noise levels for $\alpha = 0.01$ and $\alpha = 0.05$, above which the network ceases to function. The values are consistent with the location of the phase boundary, the solid line in Figure 3.5.

Answer (3.4) — Start with Equation (3.51):

$$w_{ij} = \frac{1}{N}\sum_{\mu\nu} x_i^{(\mu)}[\mathbb{Q}^{-1}]_{\mu\nu} x_j^{(\nu)} \quad \text{with} \quad Q_{\mu\nu} = \frac{1}{N}\sum_i x_i^{(\mu)} x_i^{(\nu)}. \tag{3.10}$$

To check whether pattern $\boldsymbol{x}^{(\delta)}$ is reproduced, we must evaluate $\sum_j w_{ij} x_j^{(\delta)}$:

$$\frac{1}{N}\sum_j \sum_{\mu,\nu} x_i^{(\mu)}[\mathbb{Q}^{-1}]_{\mu\nu} x_j^{(\nu)} x_j^{(\delta)} = \sum_{\mu,\nu} x_i^{(\mu)}[\mathbb{Q}^{-1}]_{\mu\nu} Q_{\nu\delta} = \sum_\mu x_i^{(\mu)} \delta_{\mu\delta} = x_i^{(\delta)}. \tag{3.11}$$

In other words, all stored patterns are reproduced perfectly. But the learning rule requires that $\mathbb{Q}^{-1}$ exists. This requires that the stored patterns are linearly independent.

Answer (3.5) — The mean-field theory for the Ising model is given by Equations (3.6) and (3.7) with $\beta = (k_B T)^{-1}$ and

$$\langle b_i \rangle = J \sum_{j=\mathrm{nn}(i)} \langle s_j \rangle + h = J z \langle s_i \rangle + h. \tag{3.12}$$

Here z is the coordination number, the number of nearest neighbours in d dimensions (Table 3.1). With the definition of the order parameter $m = \langle \frac{1}{N}\sum_i s_i \rangle$ it follows that

$$m = \tanh(\beta J z m + \beta h) \tag{3.13}$$

Table 3.1: Critical temperature of the d-dimensional Ising model on a hypercubic lattice. Exercise 3.5.

d	1	2	3	4	5
z	2	4	6	8	10
$k_B T_c/J$	0[1]	2.269[2]	4.511[3]	6.680[4]	8.778[5]

For $h = 0$, Equation (3.13) has only the paramagnetic solution $m = 0$ if $\beta J z < 1$. For $\beta J z > 1$, there are three solutions, $m = 0$, as well as $m = \pm|m| \neq 0$. So the critical temperature is

$$k_B T_c = J z\,. \tag{3.14}$$

For $T < T_c$, the system is ferromagnetic, it exhibits a non-zero magnetisation m. Above T_c, the system is paramagnetic, $m = 0$. Expanding $\tanh(x) \sim x - x^3/3$ shows that the magnetisation vanishes as

$$m \sim \pm(3|\delta|)^{1/2} \quad \text{with} \quad \delta = \frac{T - T_c}{T_c} \leq 0\,. \tag{3.15}$$

We say that the magnetisation tends to zero with critical exponent $\frac{1}{2}$ (Figure 3.3). It turns out that this is the correct behaviour for $d \geq 4$. For $d = 1, 2, 3$, the mean-field theory gets the critical exponent wrong because it neglects fluctuations. The mean-field approximation for the critical temperature is compared with the actual value of T_c in Table 3.1. We see that the mean-field prediction for T_c becomes more accurate for larger dimensions. In the limit of $d \to \infty$, the mean-field prediction for T_c becomes exact. In this limit there are enough neighbours to average over, just as in the mean-field theory for the fully-connected Hopfield model, Section **3.4**.

Answer (3.6) — We start from Equation (**3.41**) and show that $y = 1/\sqrt{2\alpha}$ is an approximate solution for $\alpha \ll 1$. Using the asymptotic expansion $\mathrm{erf}(y) \sim 1 - \exp(-y^2)/(\sqrt{\pi}\, y)$ for large y gives that

$$y = 1/\sqrt{2\alpha} \tag{3.16}$$

solves Equation (**3.41**) up to terms $\propto \exp[-1/(2\alpha)]$ that vanish very rapidly as $\alpha \to 0$. This shows that Equation (**3.40**) tends to (**2.39**) in the limit of small α.

[1] Exact solution of one-dimensional Ising model: $k_B T_c = 0$
[2] Exact solution of two-dimensional Ising model: $k_B T_c = 2J/\log(1+\sqrt{2})$
[3] Ferrenberg & Landau, Phys. Rev. B **44** (1991) 5081.
[4] Gaunt, Sykes & McKenzie, J. Phys. A **12** (1979) 871.
[5] Luijten, Binder & Blöte, Eur. Phys. J. B **9** (1999) 289.

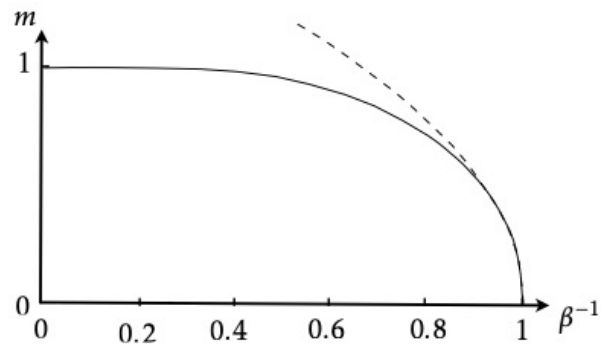


Figure 3.3: Solution $m > 0$ of the mean-field equation (3.13), for $Jz = 1$ and $h = 0$ as a function of $k_B T = \beta^{-1}$ (solid line). Also shown is the approximate mean-field solution near the critical temperature, Equation (3.15), dashed line. Exercise 3.5.

4 Exercises in Chapter 4

Answer (4.1) — Consider the first neuron in the network shown in Figure 2.9, with $w_{21} = -1$, $w_{12} = 2$, and thresholds equal to zero. Using these values, Equation (3.1) yields the following update rule

$$s_1' = \begin{cases} 1 & \text{with probability} \quad [1+\exp(4\beta s_2)]^{-1}, \\ -1 & \text{with probability} \quad [1+\exp(-4\beta s_2)]^{-1}. \end{cases} \tag{4.1}$$

Equation (4.3) with energy function $H = -\frac{w_{12}+w_{21}}{2} s_1 s_2 = -\frac{1}{2} s_1 s_2$ gives, for example,

$$s_1 = -1 \to s_1' = 1 \quad \text{with probability} \quad [1+\exp(-\beta s_2)]^{-1}. \tag{4.2}$$

This is different from Equation (4.1). So Equations (3.1) and (4.3) are not equivalent. As stated in the problem formulation, the reason is that the weights are not symmetric, $w_{21} \neq w_{12}$.

Answer (4.2) — The asynchronous deterministic update rule for 0/1 neurons was derived in Exercise **??**, $n_m' = \theta_H(b_m)$ where $\theta_H(b)$ is the Heaviside function, and $b_m = \sum_j w_{mj} n_j - \mu_m$ is the local field with weights w_{mj} and threshold μ_m. It was shown that the energy function $H = -\frac{1}{2}\sum_{ij} w_{ij} n_i n_j + \sum_i \mu_i n_i$ cannot increase under the dynamics if the weights are symmetric and the diagonal weights vanish.
The following stochastic rule converges to the deterministic dynamics in the limit of weak noise

$$n_m' = \begin{cases} 1 & \text{with probability} \quad p(b_m), \\ 0 & \text{with probability} \quad 1-p(b_m), \end{cases} \tag{4.3}$$

Algorithm Metropolis algorithm

```
initialise s_old = s_0;
for t = 1,...,T do
    draw a random number s_new uniformly distributed in [0, s_max];
    draw a random number r uniformly distributed in [0,1];
    if r < exp(−s_new + s_old) then
        s_old = s_new;
    end if
end for
```

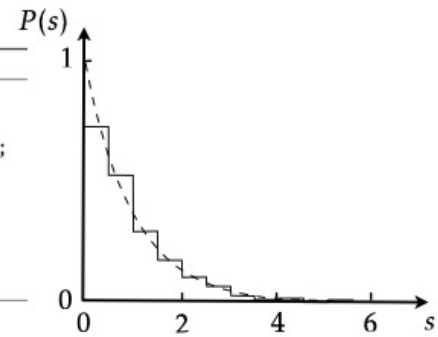


Figure 4.1: Left: algorithm to generate samples from the exponential distribution $\exp(-s)$. Right: histogram obtained from 1000 iterations of the algorithm (solid line). The dashed line shows $\exp(-s)$. Exercise 4.3.

with $p(b) = [1+\exp(-\beta b)]^{-1}$. Note that the argument of the exponential functions lacks a factor of two, compared with Equation (3.1). Now show that this rule is equivalent to Equation (4.3),

$$n_m \to n'_m \neq n_m \quad \text{with probability} \quad [1+\exp(\beta \Delta H_m)]^{-1}, \tag{4.4a}$$

where

$$\begin{aligned}\Delta H_m &= H(\ldots, n'_m, \ldots) - H(\ldots, n_m, \ldots) \\ &= (n'_m - n_m)\Big(-\sum_{j\neq m} \frac{w_{mj}+w_{jm}}{2} n_j + \mu_m\Big) - \tfrac{1}{2} w_{mm}(n'_m n'_m - n_m n_m).\end{aligned} \tag{4.4b}$$

If the weights are symmetric, and if the diagonal weights vanish, then the above expression simplifies to

$$\Delta H_m = -b_m(n'_m - n_m). \tag{4.5}$$

This shows that Equations (4.3) and (4.4) are equivalent if the weights are symmetric and the diagonal weights vanish.

Answer (4.3) — The task is to set up a Markov chain that has the exponential distribution as its steady state. To this end we choose $H = s$. For this case, Algorithm 2 takes the form given in Figure 4.1. The Figure also shows a histogram generated by 1000 iterations of the Markov chain. The dashed line is $\exp(-s)$.

Answer (4.4) — Start by writing down the transition matrix $\mathbb{P}$ with matrix elements $[\mathbb{P}]_{ij} = p(\boldsymbol{s}^{(i)}|\boldsymbol{s}^{(j)})$

$$\mathbb{P} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{4.6}$$

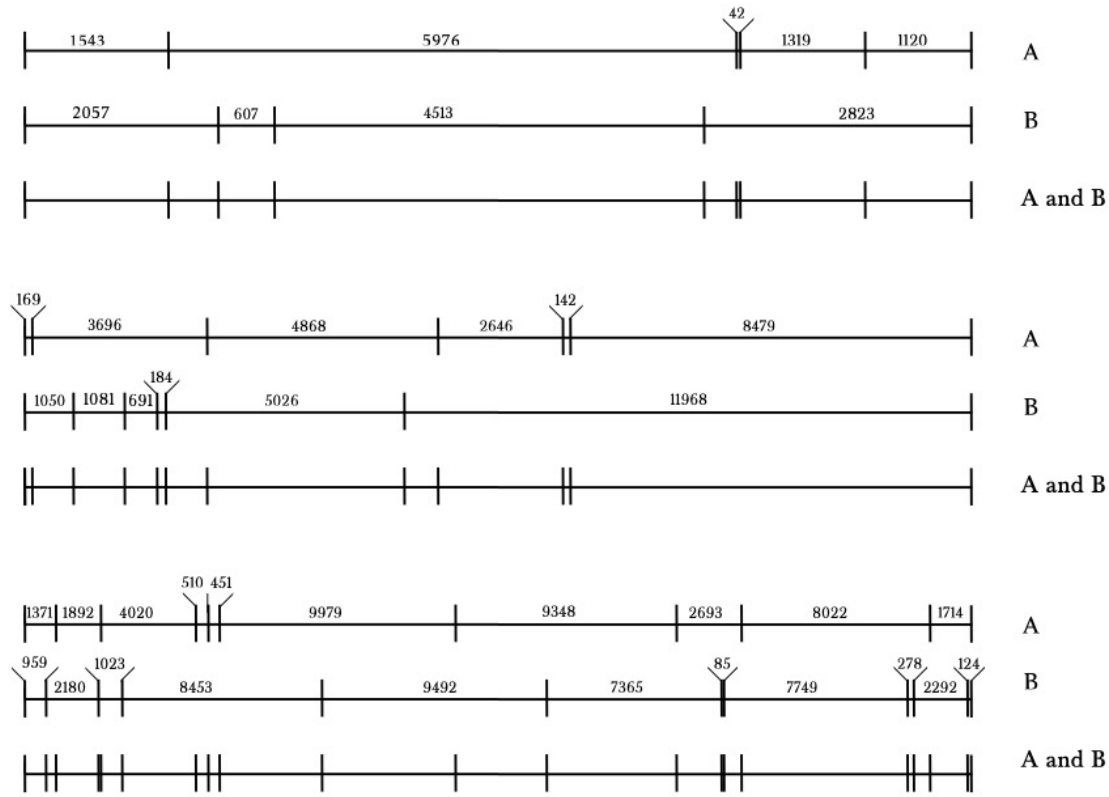


Figure 4.2: Solutions of the three double-digest problems from Table 4.1. Exercise 4.5.

The probability $P_i(t)$ of observing the system in state i at time t is given by

$$\begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix} = \mathbb{P}^t \begin{bmatrix} P_1(0) \\ P_2(0) \\ P_3(0) \end{bmatrix} . \tag{4.7}$$

The probabilities $P_i(t)$ are normalised since the initial probabilities are, $\sum_i P_i(0) = 1$. The steady-state probabilities P_i^* are the components of the normalised eigenvector of $\mathbb{P}$ with eigenvalue unity. This means $P_1^* = P_2^* = P_3^* = \frac{1}{3}$. Evaluating the r.h.s and l.h.s of Equation (4.9) shows that they differ, for example $[\mathbb{P}]_{21} P_1^* = \frac{1}{3}$, while $[\mathbb{P}]_{12} P_2^* = 0$. So detailed balance is not satisified. The reason is that the Markov chain is a cycle. Its steady state is an example of a non-equilibrium steady state, a steady state with a non-zero probability current.

Answer (4.5) — Solutions of the the double-digest problems listed in Table 4.1 are shown in Figure 4.2. The solutions are not uniqe. For the first problem, for example,

there is one b-interval that contains three a-intervals. Any permutation of these a-intervals generates a new solution, 3! solutions in total. In addition there an a interval containing two b-intervals, increasing the degeneracy by a factor of two. Finally, for each solution there is a reflected one. So the first problem has $2\cdot 2\cdot 3! = 24$ distinct solutions. The second problem has $2\cdot 3!\cdot 3! = 72$ solutions. The third problem has only $2\cdot 2 = 4$ solutions.
A simple simulated-annealing algorithm for the double-digest problem suggests new configurations by exchanging a pair of entries in either the permutation σ or μ, such as

$$[5976, 1543, 1319, 1120, 42] \to [5976, 42, 1319, 1120, 1543].$$

This scheme of suggesting new configurations is symmetric (Section **4.1**): the probability of suggestion $[\sigma,\mu]\to[\sigma',\mu']$ is the same as suggesting the move $[\sigma',\mu']\to[\sigma,\mu]$, starting at $[\sigma',\mu']$. For the first two problems, an algorithm with these local moves quickly finds all solutions, using $\beta_t = 10^{-8}\,t$ (where t is the iteration number of the Monte-Carlo algorithm). For the third problem, however, the algorithm arrests in local minima. In this case, more general local moves are required, as described by Goldstein & Waterman, Adv. Appl. Math. **8** (1987) 194.

Answer (4.6) — We start from Equation (**4.18**)

$$\begin{aligned} D_{\mathrm{KL}} &= -\sum_{\mu} P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)}) \log[P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}^{(\mu)})/P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)})] \\ &\geq -\sum_{\mu} P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)})[P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}^{(\mu)})/P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)})-1], \\ &= \sum_{\mu}[P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)})-P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}^{(\mu)})]. \end{aligned} \tag{4.8}$$

For the second step, we used the inequality $\log z \leq z-1$. Finally, we use that the distributions are normalised,

$$\sum_{\mu} P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)}) = 1, \quad \text{and} \quad \sum_{\mu} P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}^{(\mu)}) = 1. \tag{4.9}$$

It follows that $D_{\mathrm{KL}} \geq 0$. One verifies that D_{KL} attains the global minimum $D_{\mathrm{KL}} = 0$ by setting $P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)}) = P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}^{(\mu)})$ in Equation (4.8).
To show that minimusing D_{KL} corresponds to maximising the log-likelihood $\mathscr{L}$, Equation (**4.17**), one starts from

$$D_{\mathrm{KL}} = \sum_{\mu} P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)}) \log[P_{\mathrm{data}}(\boldsymbol{x}^{(\mu)})] - \langle \log P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}^{(\mu)})\rangle_{P_{\mathrm{data}}}. \tag{4.10}$$

The first term is a constant, it does not depend on the weights. The second term equals $-p^{-1}\mathscr{L}$. So minimising D_{KL} corresponds to maximising $\mathscr{L}$.

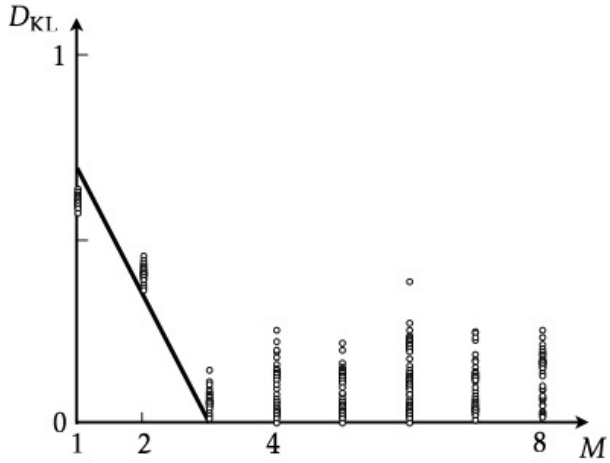


Figure 4.3: Kullback-Leibler divergence versus the number of hidden neurons. Upper bound (**4.40**), solid line, and numerical results (∘) using the CD-k algorithm (Algorithm **3**). Training parameters: $\nu_{\max} = 10^4$, $p_0 = 20$, $k = 200$, and $\eta = 0.005$. Sampling parameters: the Markov chain was iterated for 10^7 iterations. Schematic, adapted from numerical results obtained by Ehsan Ghane. Shown are the results of 100 independent runs for each value of M. Exercise 4.7.

Answer (4.7) — A restricted Boltzmann machine with M hidden neurons was trained with the CD-k algorithm (Algorithm **3**). The weights were initially Gaussian random with mean zero and unit variance, the initial thresholds were set to zero. The remaining parameters are given in the caption. After training, the model (Boltzmann) distribution was sampled using the Markov chain (**4.36**) for 10^7 iterations. Then D_{KL} was computed using Equation (**4.18**). For each M, this process was repeated 100 times, for different initialisations of weights and thresholds. The results are shown in Figure 4.3. We see that there is some spread of D_{KL}-values, likely because the CD-k algorithm fails to explore the global maximum. For $M > 1$, the best results are very close to the upper bound (**4.40**), indicating that the XOR problem is hard to learn. For $M = 1$, the best result is below the upper bound.

Answer (4.8) — The shifter ensemble is illustrated in Figure 4.4, see also Refs. [**15**,**47**]. The patterns are constructed as follows. The bits in the first row are chosen randomly, equal to ±1 with probability $\frac{1}{2}$. The bits in the second row are obtained by copying the bits of the first row. Then the pattern in the second row is shifted to the right with probability $\frac{1}{3}$, to the left with probability $\frac{1}{3}$, and left unchanged with probability $\frac{1}{3}$. Periodic boundary conditions are applied. The three auxiliary indicator bits at the bottom of each pattern help the Boltzmann machine to learn: they indicate whether the second row is shifted to the right, left, or whether it remains

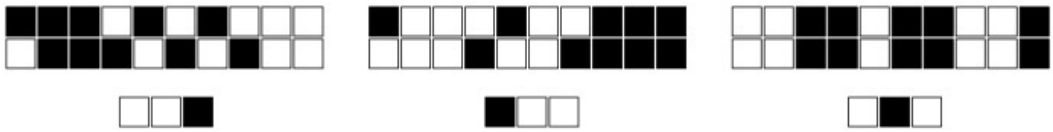


Figure 4.4: Illustration of shifter ensemble. Exercise 4.8.

unchanged. All patterns obtained in this way occur with the same probability P_{data}. All other patterns are assigned $P_{\text{data}} = 0$.
This ensemble cannot be represented with a Boltzmann machine that relies only on two-point correlations [15,47], because the three-point correlations of the bits are required (between a bit in the first row, in the second row, and an indicator bit).

Answer (4.9) — The energy function of the restricted Boltzmann machine is given by Equation (4.29),

$$H = -\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij} h_i v_j + \sum_{j=1}^{N} \theta_j^{(\text{v})} v_j + \sum_{i=1}^{M} \theta_i^{(\text{h})} h_i \,. \tag{4.11}$$

The determnistic update rule follows from Equation (4.30),

$$h'_m = \text{sgn}(b_m^{(\text{h})}), \quad \text{and} \quad v'_n = \text{sgn}(b_n^{(\text{v})}), \tag{4.12}$$

with $b_i^{(\text{h})} = \sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\text{h})}$ and $b_j^{(\text{v})} = \sum_{i=1}^{M} h_i w_{ij} - \theta_j^{(\text{v})}$. Consider first the changes in H when updating the hidden neurons, keeping the states of the visible neurons unchanged (constant). We write

$$H = -\sum_{i=1}^{M} h_i \Big(\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\text{h})} \Big) + \text{const}. \tag{4.13}$$

This allows us to express the change in H as

$$H' - H = -\sum_{i=1}^{M} (h'_i - h_i) \Big(\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\text{h})} \Big). \tag{4.14}$$

Suppose that $h_i = 1$ and $h'_i = -1$, so that $h'_i - h_i < 0$. It follows from Equation (4.12) that the sign of $\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\text{h})}$ equals $h'_i < 0$. Therefore $H' - H < 0$. Now assume that $h_i = -1$ and $h'_i = 1$. In this case $h'_i - h_i > 0$ and $\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\text{h})} > 0$. Again $H' - H < 0$. When $h'_i = h_i$, the energy function does not change. In summary, H cannot increase when updating the hidden neurons (keeping the states of the visible neurons fixed). Here the argument works for synchronous updates of the hidden

neurons because there are no interactions between them. For the Hopfield model, the energy function can increase under synchronous updates (Exercise **??**). In a similar fashion one shows that H cannot increase under synchronous updates of the visible neurons, if one keeps the states of the hidden neurons constant.

Answer (4.10) — Consider first a Boltzmann machine without hidden neurons, but with thresholds. In this case, Equation (**4.16**) takes the form

$$P_{\mathrm{B}}(\boldsymbol{s}=\boldsymbol{x}) = Z^{-1}\exp\Big(\tfrac{1}{2}\sum_{i\neq j} w_{ij}x_i x_j - \sum_i \theta_i x_i\Big). \tag{4.15}$$

To derive the learning rule for the thresholds, we need to evaluate the gradient of

$$\frac{\partial \log \mathscr{L}}{\partial \theta_m} = \frac{\partial}{\partial \theta_m}\sum_{\mu}\Big(-\log Z + \tfrac{1}{2}\sum_{i\neq j} w_{ij}x_i^{(\mu)}x_j^{(\mu)} - \sum_i \theta_i x_i^{(\mu)}\Big), \tag{4.16}$$

with

$$\log Z = \sum_{s_1=\pm1,\dots,s_N=\pm1} \exp\Big(\tfrac{1}{2}\sum_{i\neq j} w_{ij}s_i s_j - \sum_i \theta_i s_i\Big). \tag{4.17}$$

The derivative of $\log Z$ evaluates to

$$\frac{\partial \log Z}{\partial \theta_m} = -\sum_{s_1=\pm1,\dots,s_N=\pm1} s_m P_{\mathrm{B}}(\boldsymbol{s}) = -\langle s_m\rangle_{\mathrm{model}} \tag{4.18}$$

Evaluating the derivative of the second term in Equation (4.16) in a similar way, one obtains

$$\frac{\partial \log \mathscr{L}}{\partial \theta_m} = -p\big(\langle x_m\rangle_{\mathrm{data}} - \langle s_m\rangle_{\mathrm{model}}\big). \tag{4.19}$$

Comparing with Equation (**4.26**), we see that the same rule of thumb applies as described in Chapter **6.1**: the learning rule for the thresholds is obtained from that of the weights by replacing the the state of the neuron in the weight-update formula by -1.

Now consider a restricted Boltzmann machine with hidden neurons. There are two thresholds in Equation (**4.29**), for the visible and for the hidden neurons. The derivatives of the likelihood w.r.t. the thresholds are computed following the steps outlined in Chapter **4.4**. We begin with the learning rule for $\theta_n^{(\mathrm{v})}$. For a single pattern $\boldsymbol{x}^{(\mu)}$, one finds

$$\frac{\partial \log \mathscr{L}}{\partial \theta_n^{(\mathrm{v})}} = -\big(x_n^{(\mu)} - \langle v_n\rangle_{\mathrm{model}}\big). \tag{4.20}$$

The model average of v_n over the steady-state Boltzmann distribution can be generated using the Markov chain (4.36), with $\boldsymbol{v}_{t=0} = \boldsymbol{x}^{(\mu)}$. In practice one approximates the average by the k-th iterate, $\langle v_n \rangle_{\text{model}} \approx v_{n,t=k}$ (CD-k algorithm). This gives

$$\delta\theta_n^{(\text{v})} = \eta \frac{\partial \log \mathscr{L}}{\partial \theta_n^{(\text{v})}} \approx -\eta(v_{n,t=0} - v_{n,t=k}), \tag{4.21}$$

Equation (4.39a). Now consider the learning rule for $\theta_m^{(\text{h})}$. For a single pattern $\boldsymbol{x}^{(\mu)}$, one finds

$$\frac{\partial \log \mathscr{L}}{\partial \theta_m^{(\text{h})}} = -\big(\langle h_m \rangle_{\text{data}} - \langle h_m \rangle_{\text{model}}\big), \tag{4.22}$$

where $\langle h_m \rangle_{\text{data}}$ is the average of h_m over the Boltzmann distribution conditional on $\boldsymbol{v} = \boldsymbol{x}^{(\mu)}$. Using (4.33) and (4.34), one has $\langle h_m \rangle_{\text{data}} = \tanh\big(\sum_{j=1}^N w_{mj} x_j^{(\mu)} - \theta_m^{(\text{h})}\big)$. The average $\langle h_m \rangle_{\text{model}}$ is is computed in two steps. First, one calculates the average of h_m conditional on a given state $\boldsymbol{v}$ of the visible neurons. This yields $\tanh\big(\sum_{j=1}^N w_{mj} v_j - \theta_m^{(\text{h})}\big)$. Then one averages $\tanh\big(\sum_{j=1}^N w_{mj} v_j - \theta_m^{(\text{h})}\big)$ over the Boltzmann distribution using a Markov chain with $\boldsymbol{v}_{t=0} = \boldsymbol{x}^{(\mu)}$. Using the CD-$k$ approximation gives

$$\begin{aligned}\delta\theta_m^{(\text{h})} &= \eta \frac{\partial \log \mathscr{L}}{\partial \theta_m^{(\text{h})}} \\ &\approx -\Big[\tanh\Big(\sum_{j=1}^N w_{mj} v_{j,t=0} - \theta_m^{(h)}\Big) - \tanh\Big(\sum_{j=1}^N w_{mj} v_{j,t=k} \theta_m^{(h)}\Big)\Big].\end{aligned} \tag{4.23}$$

where the factor of p can be dropped, since it only affects the value of the learning rate (μ).

Answer (4.11) — We start from Equation (4.32),

$$\delta w_{mn}^{(\mu)} = \eta\big(\langle h_m x_n^{(\mu)} \rangle_{\text{data}} - \langle h_m v_n \rangle_{\text{model}}\big). \tag{4.24}$$

The term $\langle h_m x_n^{(\mu)} \rangle_{\text{data}}$ is computed by averaging over all states of the hidden neurons when the pattern $\boldsymbol{x}^{(\mu)}$ is clamped to the visible neurons. So

$$\langle h_m x_n^{(\mu)} \rangle_{\text{data}} = \sum_{h_1=0,1,\ldots,h_M=0,1} h_m x_n^{(\mu)} \Big[\prod_{i=1}^M P(h_i | \boldsymbol{v} = \boldsymbol{x}^{(\mu)})\Big]. \tag{4.25}$$

Using normalisation, $\sum_{h_j=0,1} P(h_j | \boldsymbol{v} = \boldsymbol{x}^{(\mu)}) = 1$, one finds

$$\langle h_m x_n^{(\mu)} \rangle_{\text{data}} = \sum_{h_m=0,1} h_m x_n^{(\mu)} P(h_m | \boldsymbol{v} = \boldsymbol{x}^{(\mu)}). \tag{4.26}$$

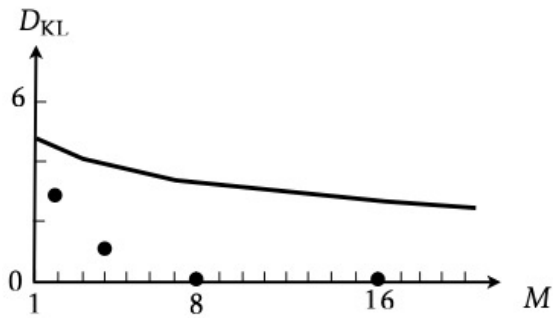


Figure 4.5: Kullback-Leibler divergence versus the number of hidden neurons. Theoretical upper bound (**4.40**), solid line, and results of numerical simulations with Algorithm **3**, symbols. Exercise 4.12.

For 0/1 neurons, the stochastic update rule (**4.30**) is replaced by

$$h'_m = \begin{cases} 1 & \text{with probability} \quad p(b_m^{(\mathrm{h})}), \\ 0 & \text{with probability} \quad 1 - p(b_m^{(\mathrm{h})}), \end{cases} \tag{4.27}$$

with $b_m^{(\mathrm{h})} = \sum_j w_{ij} v_j - \theta_i^{(\mathrm{h})}$ and $p(b_m^{(\mathrm{h})}) = [1 + \exp(-b_m^{(\mathrm{h})})]^{-1}$. Note that the argument of the exponential functions lacks a factor of two, compared with Equation (**3.1**), see Exercise **??**.
We use (4.27) to evaluate the average in Equation (4.26):

$$\langle h_m x_n^{(\mu)} \rangle_{\mathrm{data}} = p(b_m^{(\mathrm{h})}). \tag{4.28}$$

The second average in (4.24) is evaluated in an analogous fashion

$$\langle h_m v_n \rangle_{\mathrm{model}} = \langle p(b_m^{(\mathrm{h})}) v_n \rangle_{\mathrm{model}}. \tag{4.29}$$

Contrast Equations (4.28) and (4.29) with Equations (**4.34**) and (**4.35**). For ±1-neurons, the dependence b on the local field is $\tanh(b)$, just as in Equation (**3.7**). But for 0/1-neurons this is replaced by the sigmoid dependence $p(b)$.

Answer (4.12) — Figure 4.5 shows a comparison between the Kullback-Leibler entropy obtained from computer simulations with the upper bound (**4.40**), as a function of the number of hidden neurons. We see that hidden neurons are needed to represent the bars-and-stripes ensemble, but the numerical results for D_{KL} are significantly below the upper bound. We conclude that the bars-and-stripes ensemble is easier to learn than a general binary distribution with with $N = 9$ bits.

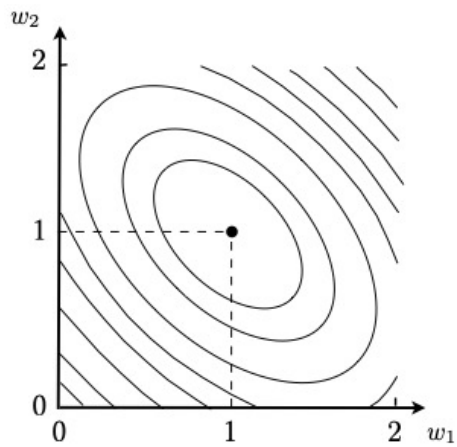


Figure 5.1: Energy function $H(w_1, w_2, \theta = \frac{3}{2})$ for the Boolean AND problem. Exercise 5.1.

5 Exercises in Chapter 5

Answer (5.1) — The energy function for a linear unit with threshold θ is given by Equation (**5.23**), $H = \frac{1}{2}\sum_\mu (t^{(\mu)} - \boldsymbol{w}\cdot\boldsymbol{x}^{(\mu)} + \theta)^2$. The derivatives of H with respect to $\boldsymbol{w}$ and θ are

$$\frac{\partial H}{\partial \boldsymbol{w}} = p\big(\langle t\boldsymbol{x}\rangle - \langle \boldsymbol{x}\boldsymbol{x}^\mathsf{T}\rangle \boldsymbol{w} + \theta\langle \boldsymbol{x}\rangle\big), \quad \frac{\partial H}{\partial \theta} = p\big(\langle t\rangle - \boldsymbol{w}^\mathsf{T}\langle \boldsymbol{x}\rangle + \theta\big). \tag{5.1}$$

For the AND problem (Figure),

$$\langle t\boldsymbol{x}\rangle = \begin{bmatrix}0\\0\end{bmatrix}, \quad \langle \boldsymbol{x}\boldsymbol{x}^\mathsf{T}\rangle = \tfrac{1}{4}\begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}, \quad \langle \boldsymbol{x}\rangle = \tfrac{1}{2}\begin{bmatrix}1\\1\end{bmatrix}, \quad \langle t\rangle = -\tfrac{1}{2}. \tag{5.2}$$

Set the derivatives to zero to determine $\boldsymbol{w}$ and θ. This gives $\boldsymbol{w} = [1,1]^\mathsf{T}$ and $\theta = \frac{3}{2}$, and we find $O^{(1)} = \boldsymbol{w}\cdot\boldsymbol{x}^{(1)} - \theta = -\frac{3}{2}$, $O^{(2)} = -\frac{1}{2}$, $O^{(3)} = -\frac{1}{2}$, and $O^{(4)} = \frac{1}{2}$. So $O^{(\mu)} \neq t^{(\mu)}$. However, the values obtained for $\boldsymbol{w}$ and θ correspond to a local minimum of H at $\boldsymbol{w} = [1,1]^\mathsf{T}$ and $\theta = \frac{3}{2}$, where $H = \frac{1}{2}$. Since H is non-zero at the minimum, the optimal solution is only approximate. Figure 5.1 shows how the energy function depends on w_1 and w_2 for $\theta = \frac{3}{2}$.

An alternative way to minimise the quadratic function H is to use a singular-value decomposition to evaluate the formal solution Equation (**5.21**). Here we have a non-zero threshold, and only one output. In this case, Equation (**5.21**) becomes

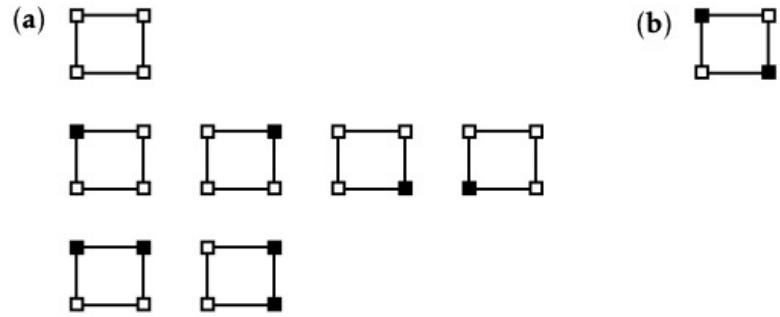


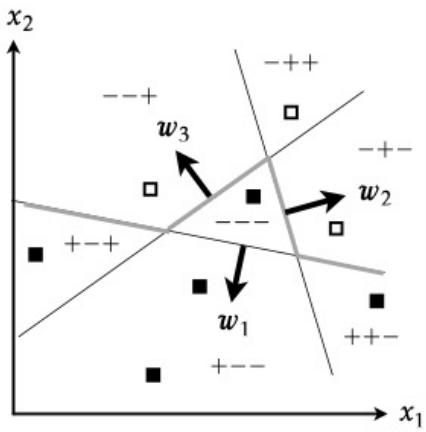
Figure 5.2: Half of all Boolean functions with two-dimensional inputs. The other half is obtained by conjugation. (**a**) These functions are linearly separable. (**b**) The XOR function is not linearly separable. Exercise 5.2.

$w_k = \frac{1}{4}\sum_{\mu\nu}(t^{(\mu)}+\theta)[\mathbb{Q}^{-1}]_{\mu\nu}x_k^{(\mu)}$. The matrix $\mathbb{Q}$ [Equation (**5.22**)] read:

$$\mathbb{Q} = \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}. \tag{5.3}$$

Since it is not invertible, we compute its pseudo-inverse using its singular-value decomposition $\mathbb{Q} = \mathbb{U}\mathbb{S}\mathbb{V}^\mathsf{T}$ where $\mathbb{S}$ is the diagonal matrix with the singular values of $\mathbb{Q}$. See also Exercise **??**. The pseudo-inverse is obtained in two steps. Construct the diagonal matrix $\mathbb{S}'$ by taking the reciprocal of all non-zero singular values. Then compute $\mathbb{V}\mathbb{S}'\mathbb{U}^\mathsf{T}$. Computing the weights with this matrix instead of $\mathbb{Q}^{-1}$, we find $\boldsymbol{w} = [1,1]^\mathsf{T}$ for $\theta = \frac{3}{2}$, the same as above.

Answer (5.2) — For two-dimensional inputs, there are $2^{2^2} = 16$ functions in total. Eight of them are shown in Figure 5.2. The other eight are obtained by conjugation. We conclude that 14 Boolean functions with two-dimensional inputs are linearly separable, two are not: the XOR and the XNOR function.
Boolean functions with three-dimensional inputs can be represented by colouring the corners of a cube, either ■ or □. Since the cube has eight corners, there are $2^8 = 256$ distinct ways of colouring, corresponding to 256 Boolean functions.
The function without any ■ is linearly separable. The eight functions with only one ■ are linearly separable. The cube has 12 edges. The 12 functions with two ■ on an edge are linearly separable. The cube has six faces. There are 4 functions with three ■ per face, this yields $6 \cdot 4 = 24$ linearly separable functions. The following functions with four ■ are linearly separable: six functions with all corners of a face black plus 8 functions with for ■ connecting to a corner. Conjugation gives more



V_1	V_2	V_3	target
−	−	−	+1
+	−	−	+1
−	+	−	−1
−	−	+	−1
+	+	−	+1
+	−	+	+1
−	+	+	−1
+	+	+	-

Figure 5.3: (**a**) Decision boundaries in input plane for Exercise **??**. (**b**) Encoding of the different regions and corresponding target values for Exercise 5.4.

linearly separable functions, in total $2(1+8+12+24)+14 = 104$. The remaining functions are not linearly separable.

Answer (5.3) — The particular problem in $\boldsymbol{V}$-space shown in Figure **5.16** is linearly separable. This is true in general because the decision boundary in $\boldsymbol{x}$-space separates ■ from □. This implies that the cluster of ■-corners of the hypercube in $\boldsymbol{V}$-space is connected.

Answer (5.4) — A possible solution is shown in Figure 5.3. Panel (**a**) shows how the decision boundaries of the three hidden neurons divide up the input plane. Panel (**b**) shows the target values for the different regions. A corresponding solution for the output unit is $O = (V_1 - V_2 - V_3)$.

Answer (5.5) — The three-dimensional parity function is not linearly separable, Figure 5.4. The proposed solution uses eight hidden neurons $V_j^{(\mu)}$, all with the same threshold $\theta_j = 2$. Their weight vectors are chosen to equal the pattern vectors, $\boldsymbol{w}_j = \boldsymbol{x}^{(j)}$ (see also page **196**). This means that

$$-\theta_j + \sum_k w_{jk} x_k^{\mu)} = -2 + \sum_k x_k^{(j)} x_k^{\mu)} \begin{cases} > 0 & \text{if}\,, j = \mu\,, \\ < 0 & \text{if}\; j \neq \mu\,. \end{cases} \tag{5.5}$$

It follows from Equation (**??**) that $V_j^{(\mu)} = 1$ if $j = \mu$, and equal to zero otherwise. So the hidden neurons are *winning* neurons (page **183**). The weights of the output neuron depend on how the inputs are labeled. Consider the choice shown in Figure 5.4. Inputs with odd values of μ have $t^{(\mu)} = +1$, inputs with odd values of μ have

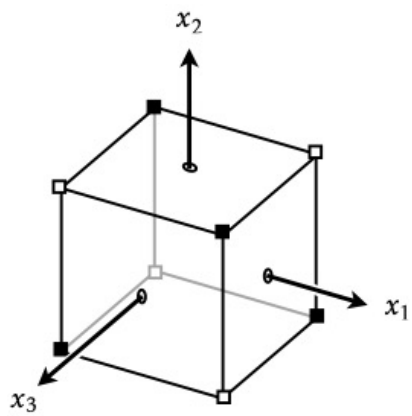
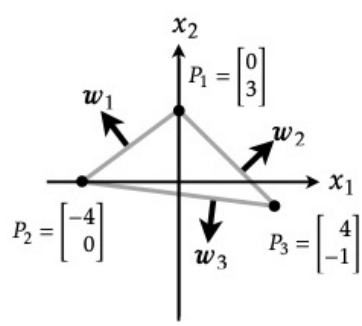


Figure 5.4: Three-dimensional parity function. Exercise 5.5.



V_1	V_2	V_3	target
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	1	1	0
1	0	1	0
1	1	0	0
1	1	1	-

Figure 5.5: (**a**) Input plane for Exercise 5.6. (**b**) Value table for Exercise 5.6.

$t^{(\mu)} = -1$. We obtain the correct outputs with $O^{(\mu)} = \boldsymbol{W} \cdot \boldsymbol{V}^{(\mu)}$ if we choose $\boldsymbol{W} = [-1, 1, -1, 1, -1, 1, -1, 1]^{\mathsf{T}}$.

Answer (5.6) — A possible solution for the hidden neurons is shown in Figure 5.5. The weight vectors are

$$\boldsymbol{w}_1 = \begin{bmatrix} -3 \\ 4 \end{bmatrix}, \quad \boldsymbol{w}_2 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \quad \boldsymbol{w}_3 = \begin{bmatrix} -1 \\ -8 \end{bmatrix}. \tag{5.6}$$

The thresholds are read off from the intersections of the decision boundary with the x_2-axis [Equation (5.13)]: $\theta_1 = 12$, $\theta_2 = 12$, $\theta_3 = 4$. A possible choice for the weights and threshold of the output neuron can be read off from the value table in Figure 5.5: $\boldsymbol{W} = [-1, -1, -1]^{\mathsf{T}}$ and $\Theta = \frac{1}{2}$, so that $O = \theta_{\mathrm{H}}(-V_1 - V_2 - V_3 + 1)$.

Answer (5.7) — Points $\boldsymbol{x}$ on the decision boundary i satisfy Equation (5.13), $\boldsymbol{w}_i \cdot$

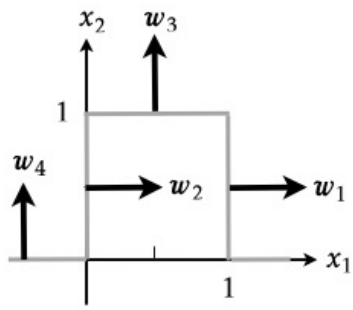


Figure 5.6: Decision boundaries and corresponding weights for Exercise 5.7.

$\boldsymbol{x} - \theta_i = 0$. For $\boldsymbol{w}_1$, for example, we have

$$\begin{bmatrix} w_{11} & w_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \theta_1 = 0 . \tag{5.9}$$

This gives $w_{11} = \theta_1$ and $w_{22} = 0$. Choosing $w_{11} = 1$ gives $\theta_1 = 1$. The other weight vectors and thresholds are determined analogously:

$$\boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \theta_1 = 1 , \quad \boldsymbol{w}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \theta_2 = 0 , \quad \boldsymbol{w}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \theta_3 = 1 , \quad \boldsymbol{w}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \theta_4 = 0 . \tag{5.10}$$

See Figure 5.6.
For the second part, compare the right panel in Figure **5.22** with the left one: the codes 1101 and 1111 were interchanged. As a consequence the third hidden neuron assumes different values on the weight-vector side of the decision boundary corresponding to $\boldsymbol{w}_3$. This is makes it impossible to solve the classification problem shown in the right panel with the decision boundaries drawn.

Answer (5.8) — Weight vectors for the three decision boundaries in Figure **5.23** are shown in Figure 5.7. From Equation (**5.13**) we infer

$$\boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \theta_1 = 1 , \quad \boldsymbol{w}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \theta_2 = \tfrac{1}{2} , \quad \boldsymbol{w}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \theta_3 = \tfrac{4}{5} . \tag{5.11}$$

The resulting output problem is shown on the right in 5.7. It can be solved by a decision boundary that contains the points $\boldsymbol{x}_1 = [\frac{1}{2}, 1, 0]^\mathsf{T}$, $\boldsymbol{x}_2 = [1, \frac{1}{2}, , 0]^\mathsf{T}$, and $\boldsymbol{x}_3 = [0, 0, 1]^\mathsf{T}$. Equation (**5.13**) gives three conditions for these three points:

$$W_1 + \tfrac{1}{2} W_2 = \Theta , \quad W_2 + \tfrac{1}{2} W_1 = \Theta , \quad \text{and} \quad W_3 = \Theta . \tag{5.12}$$

The solution is $\boldsymbol{W} = [\frac{2}{3}\Theta, \frac{2}{3}\Theta, \Theta]^\mathsf{T}$. To map the origin $\boldsymbol{V} = [0, 0, 0]^\mathsf{T}$ to output $O = 1$, we must choose a negative threshold, for example $\Theta = -1$. In this case, the output neuron calculates $O = \theta_\mathrm{H}(-\frac{2}{3} V_1 - \frac{2}{3} V_2 - V_3 - 1)$.

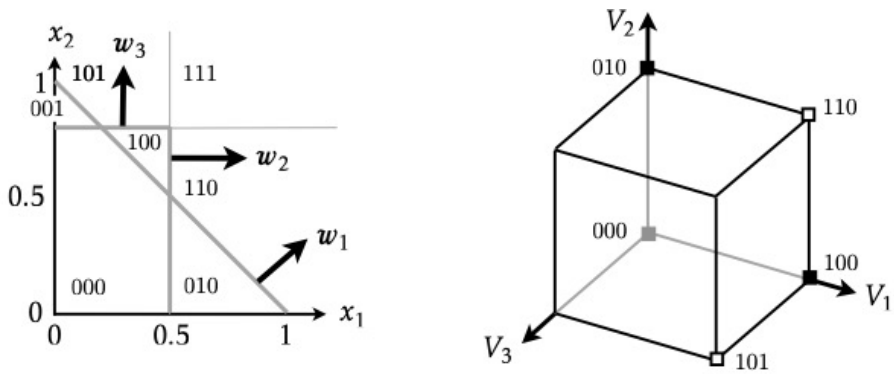


Figure 5.7: Left: weights and decision boundaries in the input plane for Exercise 5.8. Right: output problem for Exercise 5.8.

Answer (5.9) — The probability p_n is defined on page **85**, $p_n = (\frac{1}{2})^n \binom{n-1}{N-1}$. The binomial coefficient equals $\binom{n-1}{N-1} = (n-1)!/[(N-1)!(n-N)!]$ for $n \geq N$, and zero otherwise. Note that p_n is normalised, $\sum_n p_n = 1$. To evaluate the mean, use that $n\binom{n-1}{N-1} = N\binom{n}{N}$. This gives:

$$\sum_n n p_n = 2N \sum_n (\tfrac{1}{2})^{n+1} \binom{n}{N} = 2N\,, \tag{5.13}$$

because the sum on the r.h.s evaluates to unity (normalisation of p_n).

Answer (5.10) — Figure 5.8 shows all problems obtained by randomly colouring three random points in two dimensions that are in general position [Figure **5.10**]. Out of eight patterns, there are six that are linearly separable. So $P(3,2) = \frac{3}{4}$ as Equation (**5.29**) predicts for $p = 3$ patterns in $N = 2$ dimensions. Note: which problems are homogeneously separable depends on the location of the points w.r.t the origin and each other. But for given locations it is always the case that $P(3,2) = \frac{3}{4}$.

Answer (5.11) — A *linear* unit can solve a classification problem $O_i^{(\mu)} = t_i^{(\mu)}$ ($i = 1,\ldots,N$ and $\mu = 1,\ldots,p$) if the inverse of the overlap matrix (**5.22**) exists, if its columns are linearly independent. This requires linearly independent patterns $\boldsymbol{x}^{(\mu)}$, and therefore $p \leq N$. Introducing a nonlinear, monotonically increasing activation function $g(b)$, such as the sigmoid function, does not help. Since the activation function is monotonically increasing, it can be inverted to map the targets $g^{-1}(t_i^{(\mu)})$. Applying g^{-1} to the network output results in a linear function. So solving the classification problem requires that there are at most N patterns.

Answer (5.12) — Equation (**??**) is quoted by Hertz, Krogh & Palmer [**1**]. They ob-

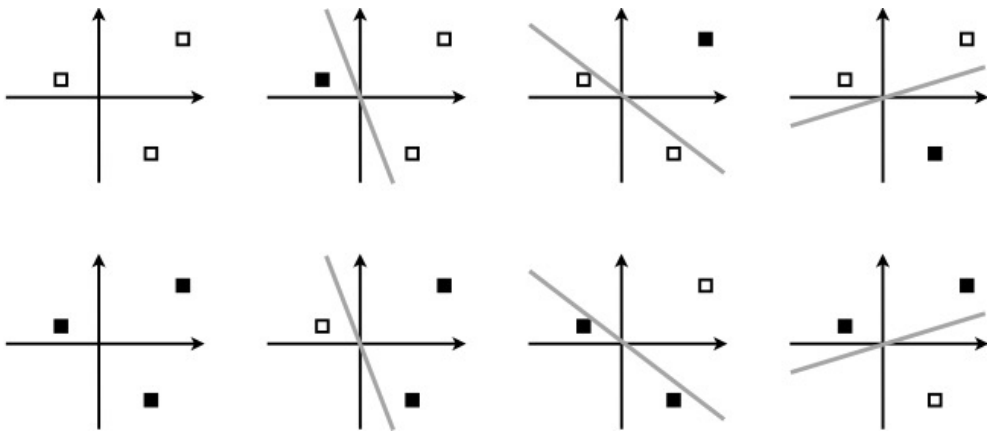


Figure 5.8: All problems obtained by randomly colouring three points in general position with the origin. Six of the problems are homogeneously linearly separable. Exercise 5.10.

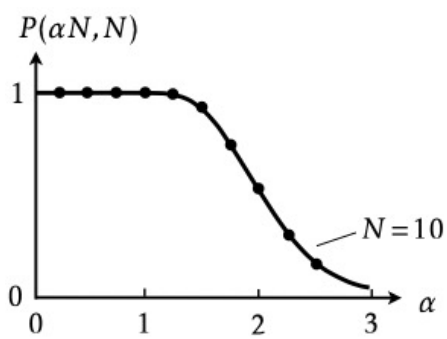


Figure 5.9: $P(\alpha N, N)$ versus α for $N = 10$ (schematic). Numerical evaluation of Equation (**5.29**), symbols. Approximate expression (**??**), solid line. Exercise 5.12.

tained it using the approximation

$$\binom{p-1}{k} \sim \frac{2^{p-1}}{\sqrt{(p-1)\pi/2}} \exp\left[-\frac{(k-(p-1)/2)^2}{(p-1)/2}\right] \tag{5.15}$$

which is valid for large p at fixed k. One inserts this into Equation (**5.29**), and approximates the sum over k as an integral (the lowest-order term in a Poisson summation). Integrating the Gaussian in (5.15) one obtains an error function. Taking the limit of $N \to \infty$ at fixed α gives Equation (**??**). This approximation works very well for N as low as 10 (Figure 5.9), and it demonstrates how $P(\alpha N, N)$ narrows to a step function when $N \to \infty$ at fixed α: when $\alpha > 2$, the error function tends to zero in the limit, when $\alpha < 2$ it tends to unity.

Table 5.1: First row: number $\mathcal{N}_n$ of linearly separable Boolean functions with n input components, from Ref. [74]. Second row: fraction $\mathcal{N}_n/2^{2^n}$. Third row: numerical estimate obtained using the procedure described in the solution to Exercise **??**. For $n = 4, 5$ only 10000 Boolean functions were sampled. Training for 20 epochs with learning rate $\eta = 0.05$, averaging over Gaussian distributed initial weights with mean zero and standard deviation $1/n$. The initial thresholds were set to zero. Exercise 5.13.

n	2	3	4	5
$\mathcal{N}_n$	14	104	1882	94572
$\mathcal{N}_n/2^{2^n}$	$\frac{7}{8}$	$\frac{13}{32}$	$\frac{941}{32768}$	$\frac{23643}{1073741824}$
	0.875	0.406	0.0287	$2.177\,10^{-5}$

Answer (5.13) — An n-dimensional cube has 2^n corners. So there are $\mathcal{N}_n = 2^{2^n}$ Boolean functions with n-dimensional inputs. This gives $\mathcal{N}_2 = 16$ and $\mathcal{N}_3 = 256$ as derived in Exercise **??**.
The McCulloch-Pitts neuron has weights $\boldsymbol{w}_j$ and a threshold θ. The weights are updated using the learning rule (5.18). For one output unit, this rule simplifies to:

$$\delta w_n^{(\mu)} = \eta(t^{(\mu)} - O^{(\mu)}) x_n^{(\mu)}. \tag{5.16}$$

As illustrated in Figure 5.7, the decision boundary for linearly separable Boolean functions has non-zero thresholds. So we need a learning rule for the thresholds as well. A guess is to simply replace $x_n^{(\mu)}$ by -1 in Equation (5.16):

$$\delta\theta^{(\mu)} = -\eta(t^{(\mu)} - O^{(\mu)}). \tag{5.17}$$

To see that this rule does the trick, consider its geometric interpretation, shown in Figure 5.10. Panel (**a**) shows a configuration where $\boldsymbol{x}^{(\mu)}$ is on the wrong side of the decision boundary. We can rectify the error by increasing the threshold a little bit, $\delta\theta^{(\mu)} = \eta > 0$, as shown in panel (**b**). Since $\square$ stands for $t^{(\mu)} = -1$, we can write this rule as $\delta\theta^{(\mu)} = -\eta t^{(\mu)}$. A second example is shown in panel (**c**). Note that $w_2 < 0$, so the threshold shown in this panel is negative. If we want to shift the decision boundary to the location shown in panel (**d**), we need to subtract a small positive number from θ, to increase its magnitude. So $\delta\theta = -\eta$. Since $\blacksquare$ stands for $t^{(\mu)} = 1$, this rule can again be written as $\delta\theta = -\eta t^{(\mu)}$. Both conclusions are consistent with Equation (5.17), because subtracting $O^{(\mu)}$ does not make a difference because by assumption we try to correct an error, for which $O^{(\mu)} = -t^{(\mu)}$.
Now train the McCulloch-Pitts neuron on Boolean functions with n input components for a large number of steps. A given function is linearly separable if the

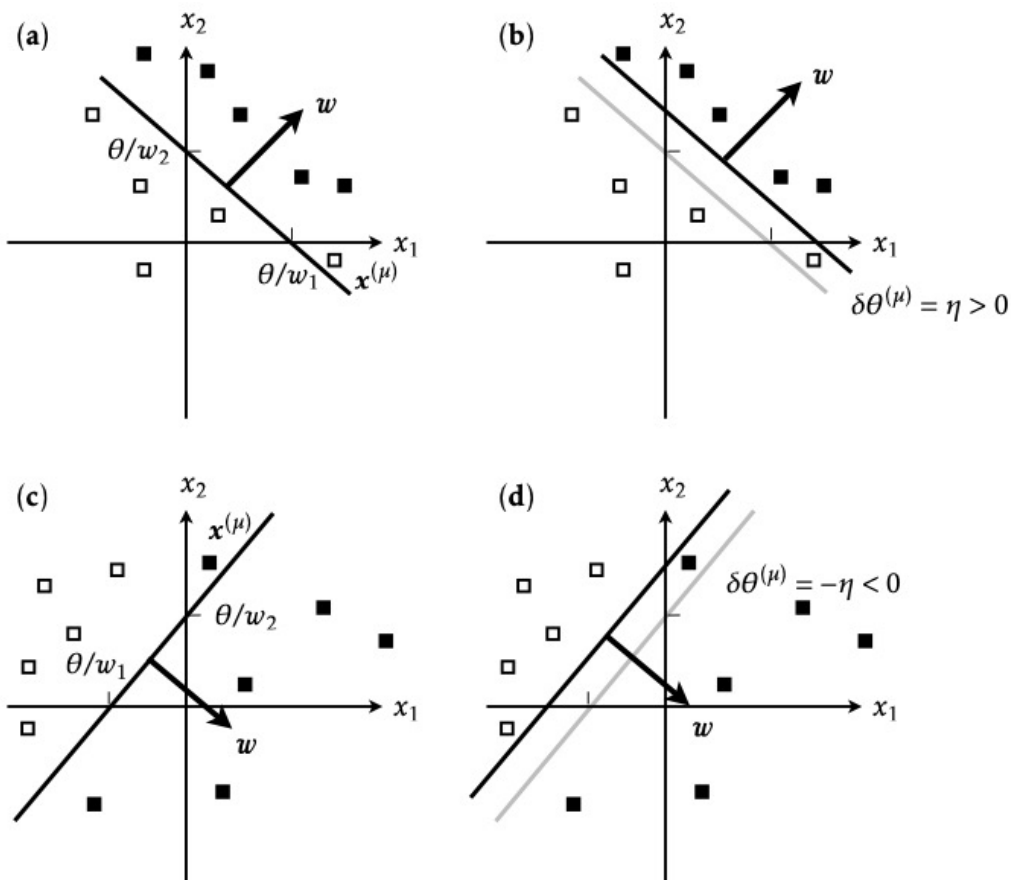


Figure 5.10: Geometrical interpretation of the learning rule (5.17) for the threshold of a binary threshold neuron. Exercise 5.13.

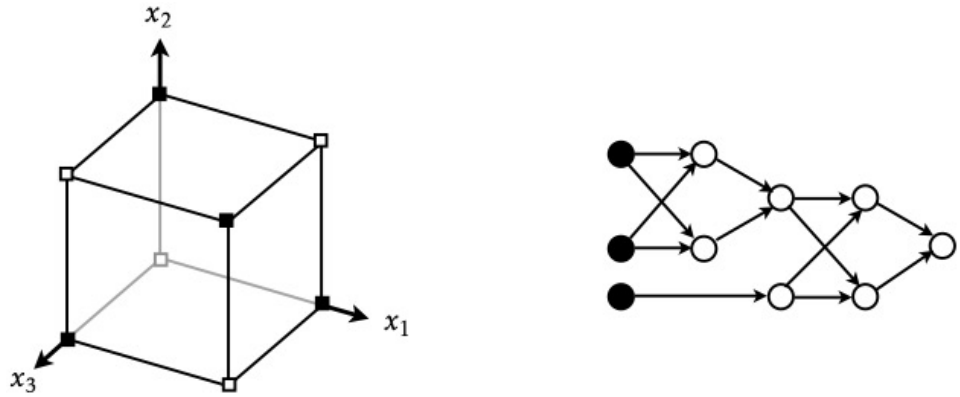


Figure 5.11: Input space and network layout for the Boolean function shown in Figure 5.25. Exercise 5.14.

algorithm finds a solution. If it doesn't, then it is likely (but not certain) that the function is not linearly separable. So the algorithm produces a lower bound to the number of linearly separable Boolean function. Table 5.1 summarises the results obtained by repeating this procedure many times. The table lists the actual number $\mathcal{N}_n$ of linearly separable Boolean functions, their fraction $\mathcal{N}_n/2^{2^n}$, and the numerically determined fraction, obtained by training for 20 epochs with learning rate $\eta = 0.05$ and averaging over Gaussian initial weights with mean zero and standard deviation n^{-1}. The initial thresholds were set to zero. The algorithm tends to find all linearly separable Boolean functions for $n = 2, 3$, and 4. For $n = 4, 5$, only 10000 functions were randomly sampled to determine the fraction. The numerical result for $n = 5$ is slightly lower than the exact one, because the algorithm misses some linearly separable functions. Finally, we infer from Table 5.1 that the fraction of linearly separable Boolean functions decreases very rapidly as the number n of input components increases, because the total number of Boolean functions increases much more rapidly than the number of linearly separable ones.

Answer (5.14) — The Boolean function shown in Figure 5.25 is the three-dimensional parity function. The input-space representation shown in Figure 5.11 demonstrates that this function is not linearly separable. A network layout with two XOR networks represents this function (Figure 5.11).
The three-dimensional exclusive parity function is similar to the three-dimensional parity function, except that the target for 111 is $t = 0$. Draw a network that represents this function. See Exercise **??**.

6 Exercises in Chapter 6

Answer (6.1) — This is demonstrated on page 105. Since matrix $\mathbb{C} = \langle \delta \boldsymbol{x} \delta \boldsymbol{x}^\mathsf{T} \rangle$ is symmetric, it has a complete orthonormal basis of eigenvectors $\boldsymbol{u}_\alpha$. This allows us to write $\mathbb{C} = \sum_\alpha \boldsymbol{u}_\alpha \boldsymbol{u}_\alpha^\mathsf{T}$ with real eigenvalues λ_α. To show that the eigenvalues are non-negative, we use $\lambda_\beta = \boldsymbol{u}_\beta^\mathsf{T} \mathbb{C} \boldsymbol{u}_\beta$ to show that $\lambda_\beta = \langle (\delta \boldsymbol{x}^\mathsf{T} \boldsymbol{u}_\beta)^2 \rangle \geq 0$. Here we used that the scalar product (2.14) is symmetric, $\delta \boldsymbol{x}^\mathsf{T} \boldsymbol{u}_\beta = \boldsymbol{u}_\beta^\mathsf{T} \delta \boldsymbol{x}$.

Answer (6.2) — Consider first Figure 6.10. The patterns are

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} -2 \\ -\frac{1}{2} \end{bmatrix}, \quad \boldsymbol{x}^{(2)} = \begin{bmatrix} -1 \\ -\frac{1}{4} \end{bmatrix}, \quad \boldsymbol{x}^{(3)} = \begin{bmatrix} 1 \\ \frac{1}{4} \end{bmatrix}, \quad \boldsymbol{x}^{(4)} = \begin{bmatrix} 2 \\ \frac{1}{2} \end{bmatrix}. \tag{6.1}$$

Since the mean $\langle \boldsymbol{x} \rangle = p^{-1} \sum_{\mu=1}^{p} \boldsymbol{x}^{(\mu)}$ is zero, the elements of the covariance matrix are given by $C_{ij} = \langle x_i x_j \rangle$. We find

$$\mathbb{C} = \frac{1}{4} \begin{bmatrix} 10 & \frac{5}{8} \\ \frac{5}{8} & \frac{5}{8} \end{bmatrix}. \tag{6.2}$$

The largest eigenvalue is $\lambda_1 = 85/32$, with eigenvector $\boldsymbol{u}_1 \propto [4,1]^\mathsf{T}$. The second eigenvalue vanbishes, $\lambda_2 = 0$, because there is no data variance orthogonal to the principal direction.
The pattern vectors in Figure 6.11 are

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} -6 \\ -5 \end{bmatrix}, \quad \boldsymbol{x}^{(2)} = \begin{bmatrix} -2 \\ -4 \end{bmatrix}, \quad \boldsymbol{x}^{(3)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \boldsymbol{x}^{(4)} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad \boldsymbol{x}^{(1)} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}, \tag{6.3}$$

Their mean vanishes, and the covariance matrix is given by

$$\mathbb{C} = \frac{1}{5} \begin{bmatrix} 70 & 65 \\ 65 & 70 \end{bmatrix}. \tag{6.4}$$

Its largest eigenvalue is $\lambda_1 = 27$, and the corresponding eigenvector is $\boldsymbol{u}_1 \propto [1,1]^\mathsf{T}$. This is the principal direction. The second eigenvalue is $\lambda_2 = 1$. It is not zero because the data in Figure 6.11 scatters a little bit around the principal direction.

Answer (6.3) — In the notation used here, Eq. (5) from Ref. [77] reads:

$$w_t = y_t - \eta_t \left. \frac{\partial H}{\partial w} \right|_{w_t}, \tag{6.5a}$$

$$a_{t+1} = \left(1 + \sqrt{4a_t^2 + 1}\right)/2 \quad \text{with} \quad a_0 = 1, \tag{6.5b}$$

$$y_{t+1} = w_t + (a_t - 1)(w_t + w_{t-1})/a_{t+1}, \tag{6.5c}$$

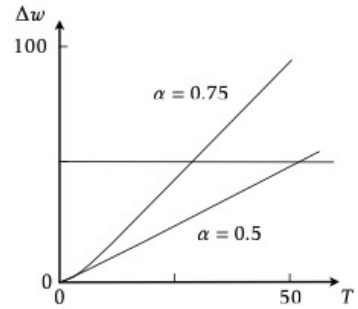


Figure 6.1: Weight change $\Delta w = w_{\mathrm{B}} - w_{\mathrm{A}}$ versus iteration number T. Exercise 6.4.

where η_t is a learning rate that may depend on the time index t. The steps necessary to derive Equation (**6.35**) are described by Sutskever [**78**]. With the definitions

$$\delta w_t = w_t - w_{t-1} \quad \text{and} \quad \alpha_t = \frac{a_t}{a_t - 1}, \tag{6.6}$$

Equation (6.5c) can be rewritten as

$$y_t = w_{t-1} + \alpha_{t-1} \delta w_{t-1}. \tag{6.7}$$

Inserting this into Equation (6.5a), one finds

$$\delta w_t = \alpha_{t-1} w_{t-1} + \eta_{t-1} \frac{\partial H}{\partial w}\Big|_{w_{t-1} + \alpha_{t-1} w_{t-1}}. \tag{6.8}$$

This is Equation (**6.31**). As one iterates Equation (6.5b), a_t increases. As a consequence, $\left(1+\sqrt{4a_t^2+1}\right)/2 \to a_t + \frac{1}{2}$. The recursion $a_{t+1} = a_t + \frac{1}{2}$ is solved by $a_t = c + t/2$ for large t. This means that $(a_t - 1)/a_{t+1} \to 1 - 3/(t + 2c + 1)$. So α_t approaches unity from below as stated in Section **6.5**. Nesterov developed the method to optimise convex and continuous functions H. He derived a particular form of the adaptive learning rate η_t for this case, that ensures rapid convergence. In neural-network applications, learning rate η_t and momentum parameter α_t are usually adjusted by trial and error [**78**].

Answer (6.4) — We start from Equation (**6.31**) which takes the form

$$\delta w^{(t)} = -\eta \frac{\partial H}{\partial w}\Big|_{w^{(t)}} + \alpha \delta w^{(t-1)}. \tag{6.9}$$

Assume that $\delta w^{(0)} = 0$. Between w_{A} and w_{B}, H decreases as a function of w with a constant slope. Denote this slope by $\partial H / \partial w = -s$ with $s > 0$. Iterating Equation (6.9), we obtain $\delta w^{(t)} = \eta s (1 - \alpha^t)/(1 - \alpha)$. As a consequence, the total weight change after T iterations reads

$$w_T - w_0 = \frac{\eta s}{1 - \alpha}\left(T - \frac{1 - \alpha^T}{1 - \alpha}\right). \tag{6.10}$$

How many steps are required to get from w_A to w_B for $\eta s = \frac{1}{2}$, say? Equation (6.10) gives

$$\Delta w = \tfrac{1}{2}\frac{1}{1-\alpha}\Big(T - \frac{1-\alpha^T}{1-\alpha}\Big) \tag{6.11}$$

for the total weight change Δw. Figure 6.1 shows Δw versus T for different values of α. We see that Δw is larger for larger α. Since $\alpha < 1$, Equation (6.11) simplifies for $T \gg 1$:

$$\Delta w \approx \tfrac{1}{2}\frac{1}{1-\alpha}\Big(T - \frac{1}{1-\alpha}\Big). \tag{6.12}$$

In this case the number of steps to go from w_A to w_B is approximately

$$T \approx 2(1-\alpha)(w_B - w_A) + \frac{1}{1-\alpha}. \tag{6.13}$$

Now assume that we iterated T_1 steps to reach w_B, or more precisely to reach a weight value just larger than w_B. Consider what happens next, between w_B and w_C (Figure **6.12**). Here $\partial H/\partial w = 0$, so that Equation (6.9) simplifies to

$$\delta w^{(t)} = \alpha \delta w^{(t-1)}. \tag{6.14}$$

We see: for $\alpha = 0$ the algorithm arrests, but not for $\alpha < 0 \leq 1$. The additional number T_2 of steps to reach w_C is given by

$$w_C - w_{T_1} = \delta w^{(T_1)} \sum_{t=0}^{T_2-1} \alpha^t = \eta s \frac{1-\alpha^{T_1}}{1-\alpha}\frac{1-\alpha^{T_2}}{1-\alpha}. \tag{6.15}$$

Answer (6.5) — Consider first the learning rule for the output weights, W_{mn}. Using Equation (**7.45**), we find that the derivative of H w.r.t. W_{mn} evaluates to

$$\frac{\partial H}{\partial W_{mn}} = \sum_{i\mu} \frac{t_i^{(\mu)} - O_i^{(\mu)}}{O_i^{(\mu)}(1-O_i^{(\mu)})} \frac{\partial \sigma(B_i^{(\mu)})}{\partial W_{mn}}, \tag{6.16}$$

with $B_i^{(\mu)} = \sum_j W_{ij} V_j^{(\mu)} - \Theta_i$. We compute the derivative of σ using Equation (**6.20**). This gives

$$\frac{\partial \sigma(B_i^{(\mu)})}{W_{mn}} = \sigma(B_i^{(\mu)})[1-\sigma(B_i^{(\mu)})]\delta_{im} V_n^{(\mu)}. \tag{6.17}$$

This gives

$$\delta W_{mn} = -\eta \frac{\partial H}{\partial W_{mn}} = \eta \sum_{\mu} (t_m^{(\mu)} - O_m^{(\mu)}) V_n^{(\mu)}. \tag{6.18}$$

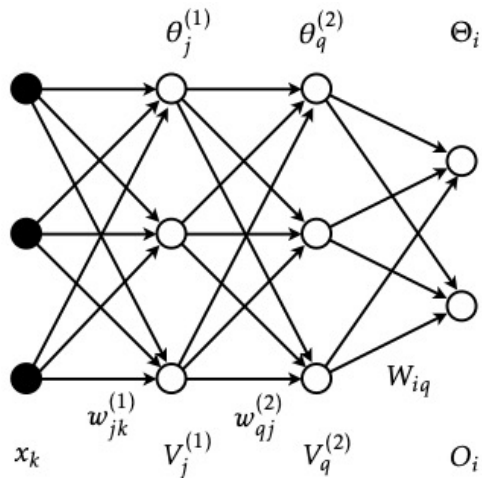


Figure 6.2: Network layout for Exercise 6.6. See also Figure **6.13**.

Now consider the learning rule for w_{mn}. Following the steps outlined in Section **6.1**, one finds

$$\delta w_{mn} = \eta \sum_{i\mu} \Delta_i^{(\mu)} W_{im} \sigma'(b_m^{(\mu)}) x_n^{(\mu)}, \tag{6.19}$$

with $\Delta_i^{(\mu)} = t_i^{(\mu)} - O_i^{(\mu)}$. This expression differs from Equation(**6.6b**) by a factor of $\sigma'(B_i^{(\mu)})$.

Answer (6.6) — The network from Figure **6.13** is reproduced in Figure 6.2. How to derive the update formulae (or learning rules) for weights and thresholds is described in Section **6.1**. The learning rules for the output weights and thresholds are simplest. For the weights, we have

$$\delta W_{mn} = -\eta \frac{\partial H}{\partial W_{mn}} = \eta \sum_{\mu} (t_m^{(\mu)} - O_m^{(\mu)}) g'(B_m^{(\mu)}) V_n^{(2,\mu)}, \tag{6.20}$$

corresponding to Equation (**6.6a**). The sequential learning rule is obtained by removing the sum over pattern indices μ. The learning rule for Θ_m is obtained from Equation (6.20) by setting $V_n^{(2,\mu)} = -1$ [Equation (**6.11a**)].
To find the learning rule for $w_{mn}^{(2)}$, we need to calculate

$$\frac{\partial O_i}{\partial w_{mn}^{(2)}} = \sum_q \frac{\partial O_i}{\partial V_q^{(2)}} \frac{\partial V_q^{(2)}}{\partial w_{mn}^{(2)}}. \tag{6.21}$$

Here we left out the pattern index μ. Using $\partial O_i / \partial V_q^{(2)} = g'(B_i) W_{iq}$ and $\partial V_q^{(2)} / \partial w_{mn}^{(2)} =$

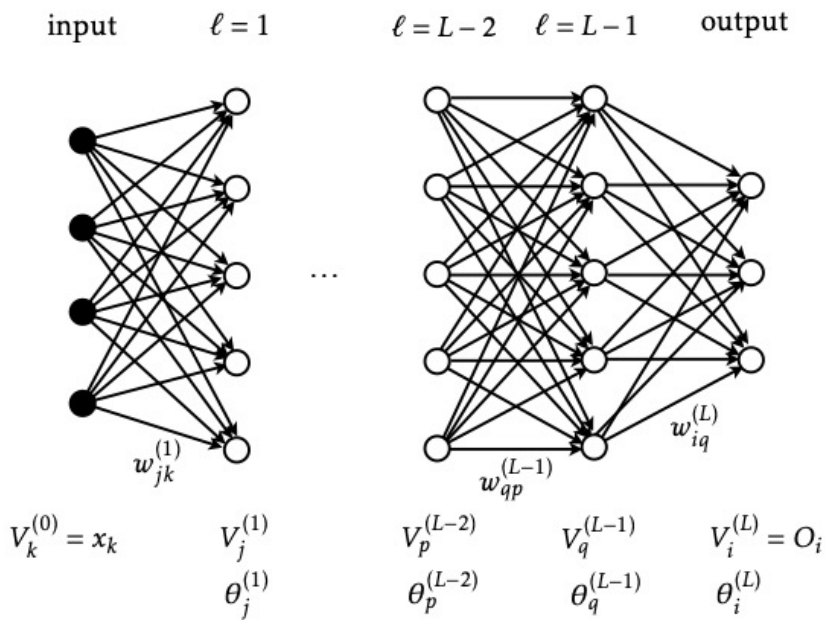


Figure 6.3: Network layout for Exercise 6.7.

$g'(b_q^{(2)})\delta_{qm}V_n^{(1)}$, we find

$$\delta w_{mn}^{(2)} = \eta \sum_i (t_i - O_i) g'(B_i) W_{im} g'(b_m^{(2)}) V_m^{(1)}. \tag{6.22}$$

This is equivalent to Equation (6.8). The learning rule for $\theta_m^{(2)}$ is obtained upon replacing $V_m^{(1)}$ by -1.
The learning rule for $w_{mn}^{(1)}$ requires one more application of the chain rule

$$\frac{\partial O_i}{\partial w_{mn}^{(1)}} = \sum_q \frac{\partial O_i}{\partial V_q^{(2)}} \sum_j \frac{\partial V_q^{(2)}}{\partial V_j^{(1)}} \frac{\partial V_j^{(1)}}{\partial w_{mn}^{(1)}}. \tag{6.23}$$

Using $\partial V_q^{(2)}/\partial V_j^{(1)} = g'(b_q^{(2)}) w_{qj}^{(2)}$ and $\partial V_j^{(1)}/\partial w_{mn}^{(1)} = g'(b_j^{(1)})\delta_{jm} x_n$, we find

$$\delta w_{mn}^{(1)} = \eta \sum_i (t_i - O_i) g'(B_i) \sum_q W_{iq} g'(b_q^{(2)}) w_{qm}^{(2)} g'(b_m^{(1)}) x_n. \tag{6.24}$$

Compare with Equation (6.28). The learning rule for $\theta_m^{(1)}$ is obtained by setting $x_n = -1$.

Answer (6.7) — The network is drawn in Figure 6.3. First, to compute the recursion

for the derivatives of $V_i^{(\ell)}$ with respect to $w_{mn}^{\ell'}$ for $\ell' < \ell$, one uses the chain rule

$$\frac{\partial V_i^{(\ell)}}{\partial w_{mn}^{(\ell')}} = \frac{\partial}{\partial w_{mn}^{(\ell')}} g\Big(\sum_j w_{ij}^{(\ell)} V_j^{(\ell-1)} - \theta_i^{(\ell)}\Big) = g'(b_i^{(\ell)}) \sum_j w_{ij}^{(\ell)} \frac{\partial V_j^{(\ell-1)}}{\partial w_{mn}^{(\ell')}} . \tag{6.25}$$

Second, for $\ell' = \ell$ the result is different. Note that $V_j^{(\ell-1)}$ does not depend on $w_{mn}^{(\ell)}$ because of the feed-forward layout of the network (Figure 6.3). Therefore

$$\frac{\partial V_i^{(\ell)}}{\partial w_{mn}^{(\ell)}} = g'(b_i^{(\ell)}) \sum_j \frac{\partial w_{ij}^{(\ell)}}{\partial w_{mn}^{(\ell)}} V_j^{(\ell-1)} = g'(b_i^{(\ell)}) \delta_{im} V_n^{(\ell-1)} . \tag{6.26}$$

This is analogous to Equation (**6.7d**). Third, put these results together to derive the learning rule for layer $L-2$. We feed pattern $\boldsymbol{x}^{(\mu)}$ and minimise $H = \frac{1}{2}\sum_i (t_i^{(\mu)} - O_i^{(\mu)})^2$, dropping the index μ in the following.

$$\delta w_{mn}^{(L-2)} = \eta \sum_i (t_i - V_i^{(L)}) \frac{\partial V_i^{(L)}}{\partial w_{mn}^{(L-2)}} \tag{6.27}$$

Iterating twice with the recursion (6.25) and then using (6.26) gives

$$\delta w_{mn}^{(L-2)} = \eta \sum_i (t_i - V_i^{(L)}) g'(b_i^{(L)}) \sum_j w_{ij}^{(L)} g'(b_j^{(L-1)}) w_{jm}^{(L-1)} g'(b_m^{(L-2)}) V_n^{(L-3)} . \tag{6.28}$$

Answer (6.8) — To derive Equation (**6.16**), we start from

$$\delta w_{mn}^{(\ell)} = -\eta \frac{\partial H}{\partial w_{mn}^{(\ell)}} \quad \text{with} \quad H = \tfrac{1}{2} \sum_i \big(t_i - V_i^{(L)}\big)^2 . \tag{6.29}$$

Here we left out the sum over pattern indices μ in H, in order to get the stochastic gradient-descent algorithm (Sections **6.1** and **6.2**). Evaluating the derivative yields

$$\delta w_{mn}^{(\ell)} = \eta \sum_i \big(t_i - V_i^{(L)}\big) \frac{\partial V_i^{(L)}}{\partial w_{mn}^{(\ell)}} = \eta \sum_i (t_i - V_i^{(L)}) \sum_q \frac{\partial V_i^{(L)}}{\partial V_q^{(\ell)}} \frac{\partial V_q^{(\ell)}}{\partial w_{mn}^{(\ell)}} , \tag{6.30}$$

where we applied the chain rule twice. Equation (**6.14**) allows us to compute the right-most derivative

$$\frac{\partial V_q^{(\ell)}}{\partial w_{mn}^{(\ell)}} = g'(b_q^{(\ell)}) \delta_{mq} V_n^{(\ell-1)} . \tag{6.31}$$

In summary,

$$\delta w_{mn}^{(\ell)} = \eta \sum_i \big(t_i - V_i^{(L)}\big) g'(b_m^{(\ell)}) \frac{\partial V_i^{(L)}}{\partial V_m^{(\ell)}} V_n^{(\ell-1)}. \tag{6.32}$$

Comparing with Equation (6.15), $\delta w_{mn}^{(\ell)} = \eta \delta_m^{(\ell)} V_n^{(\ell-1)}$, we find

$$\delta_m^{(\ell)} = \sum_i \big(t_i - V_i^{(L)}\big) \frac{\partial V_i^{(L)}}{\partial V_m^{(\ell)}} g'(b_m^{(\ell)}). \tag{6.33}$$

This is equivalent to Equation (6.16),

$$\delta_j^{(\ell-1)} = \sum_i \big(t_i - V_i^{(L)}\big) \frac{\partial V_i^{(L)}}{\partial V_j^{(\ell-1)}} g'(b_j^{(\ell-1)}), \tag{6.34}$$

which answers the first part of the question. To derive the recursion (6.17), we use the chain rule once more,

$$\frac{\partial V_i^{(L)}}{\partial V_j^{(\ell-1)}} = \sum_q \frac{\partial V_i^{(L)}}{\partial V_q^{(\ell)}} \frac{\partial V_q^{(\ell)}}{\partial V_j^{(\ell-1)}} = \sum_q \frac{\partial V_i^{(L)}}{\partial V_q^{(\ell)}} g'(b_q^{(\ell)}) w_{qj}^{(\ell)}. \tag{6.35}$$

Substituting this expression into Equation (6.34) gives

$$\begin{aligned} \delta_j^{(\ell-1)} &= \sum_i \big(t_i - V_i^{(L)}\big) \sum_q \frac{\partial V_i^{(L)}}{\partial V_q^{(\ell)}} g'(b_q^{(\ell)}) w_{qj}^{(\ell)} g'(b_j^{(\ell-1)}), \\ &= \sum_q \left(\sum_i \big(t_i - V_i^{(L)}\big) \frac{\partial V_i^{(L)}}{\partial V_q^{(\ell)}} g'(b_q^{(\ell)}) \right) w_{qj}^{(\ell)} g'(b_j^{(\ell-1)}). \end{aligned} \tag{6.36}$$

The last step is to compare with Equation (6.33). This yields Equation (6.17):

$$\delta_j^{(\ell-1)} = \sum_q \delta_q^{(\ell)} w_{qj}^{(\ell)} g'(b_j^{(\ell-1)}). \tag{6.37}$$

Answer (6.9) — Assume that there are two patterns, $\boldsymbol{x}^{(1)} = [10, 10]^\mathsf{T}$ with target $t^{(1)} = -1$, and $\boldsymbol{x}^{(2)} = [0, -1]^\mathsf{T}$ with $t^{(2)} = 1$. Figure 6.4 shows the energy function $H(w_1, w_1, \theta = 0)$ for a single neuron with sigmoid activation function. The steep gradient near the anti-diagonal is a consequence of the large mean $\langle \boldsymbol{x} \rangle$. To see this, evaluate the gradient of H with respect to w_1, set $w_2 = 0$ and $\boldsymbol{x}^{(1)} = [x, x]^\mathsf{T}$,

$$\frac{\partial H}{\partial w_1} = \frac{(2 + \mathrm{e}^{-x w_1}) x \mathrm{e}^{-x w_1}}{(1 + \mathrm{e}^{-x w_1})^3}. \tag{6.38}$$

We see that the gradient is proportional to x. So the large gradient is due to the large mean value of the patterns.

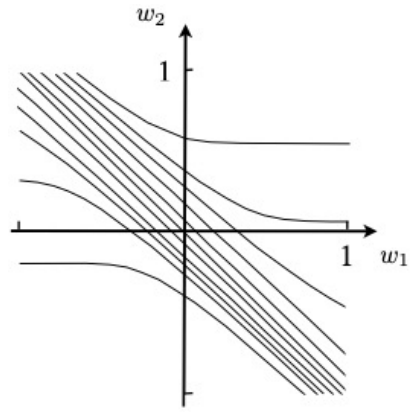


Figure 6.4: Contour plot of the energy function $H(w_1, w_2, \theta = 0)$ for Exercise 6.9.

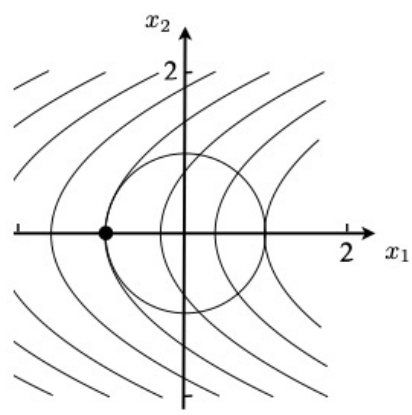


Figure 6.5: Contours of the function $f(x_1, x_2)$ and the constraint $x_1^2 + x_2^2 = 1$. Exercise 6.11.

Answer (6.10) — The function $f(x_1, x_2) = \frac{1}{2}[(x_1 - 1)^2 + x_2^2]$ assumes a minimum at $\boldsymbol{x}^* = [1, 0]^\mathsf{T}$ subject to the constraint $g(x_1, x_2) = x_2 = 0$. The Lagrangian $\mathscr{L} = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$ has no singular point at the minimum $\boldsymbol{x}^*$. To see this, compute the gradients of f and g with respect to $\boldsymbol{x}$:

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} x_1 - 1 \\ x_2 \end{bmatrix}, \quad \text{and} \quad \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{6.39}$$

Since the vectors are orthogonal at $\boldsymbol{x}^*$, the gradient of $\mathscr{L}$ with respect to $\boldsymbol{x}$ cannot vanish at this point.

Answer (6.11) — The Lagrangian for the problem reads $\mathscr{L} = x_1 - x_2^2/2 + \lambda(x_1^2 + x_2^2 - 1)$.

Its derivatives are

$$\frac{\partial \mathscr{L}}{\partial x_1} = (2\lambda x_1 + 1), \quad \frac{\partial \mathscr{L}}{\partial x_2} = (2\lambda - 1)x_2, \quad \frac{\partial \mathscr{L}}{\partial \lambda} = x_1^2 + x_2^2 - 1. \tag{6.40}$$

Setting the first two gradients to zero yields $x_1^* = -1/(2\lambda^*)$, and $x_2^* = 0$ or $\lambda^* = \frac{1}{2}$. Substituting the first solution into $\frac{\partial \mathscr{L}}{\partial \lambda}$ gives $\lambda^* = \pm\frac{1}{2}$, and so

$$x_1^* = \mp 1, \quad x_2^* = 0, \quad \lambda^* = \pm\frac{1}{2}. \tag{6.41}$$

For the second solution, one obtains one of the above solutions once more, $x_1^* = 1$, $x_2^* = 0$, and $\lambda^* = \frac{1}{2}$. Substituting these solutions into $f(\boldsymbol{x})$ we find that its minimum subject the constraint is assumed at $\boldsymbol{x}^* = [-1, 0]^\mathsf{T}$ (Figure 6.5). Note that the Lagrangian has a saddle at this solution.

7 Exercises in Chapter 7

Answer (7.1) — To start with, note that all singular points of $\mathscr{L}$ are saddle points. This can be shown as follows. From Equation (7.58) we find for the elements of the second variation of $\mathscr{L}$:

$$\frac{\partial^2 \mathscr{L}}{\delta \boldsymbol{w}^2} = \mathbb{M}, \quad \frac{\partial^2 \mathscr{L}}{\delta \lambda \delta \boldsymbol{w}} = \hat{\mathbf{e}}_q, \quad \text{and} \quad \frac{\partial^2 \mathscr{L}}{\delta \lambda^2} = 0. \tag{7.1}$$

The corresponding Hessian matrix is

$$\begin{bmatrix} \mathbb{M} & \hat{\mathbf{e}}_q \\ \hat{\mathbf{e}}_q^{\mathsf{T}} & 0 \end{bmatrix}. \tag{7.2}$$

We deduce that it must have eigenvalues of different signs because sandwiching this matrix between $[0,\ldots,0,1]$ and $[0,\ldots,0,1]^{\mathsf{T}}$ yields zero.
The Lagrange equations (7.58) are a necessary, not a sufficient condition for a minimum of $\frac{1}{2}\delta \boldsymbol{w} \cdot \mathbb{M}\delta \boldsymbol{w}$ subject to the constraint, but Figure 7.14 illustrates that $[\delta \boldsymbol{w}^*, \lambda^*]$ is a miminum for any given value of q.

Answer (7.2) — The solution is shown in Figure 7.1 (see also Figure 7.6). The four weight vectors $\boldsymbol{w}_j$ for $j = 0,1,2,3$ are obtained from Equation (7.6):

$$\boldsymbol{w}_0 = \begin{bmatrix} -\delta \\ -\delta \end{bmatrix}, \quad \boldsymbol{w}_1 = \begin{bmatrix} -\delta \\ \delta \end{bmatrix}, \quad \boldsymbol{w}_2 = \begin{bmatrix} \delta \\ -\delta \end{bmatrix}, \quad \boldsymbol{w}_3 = \begin{bmatrix} \delta \\ \delta \end{bmatrix}, \tag{7.3}$$

The thresholds are all the same. The intersections of the decision boundaries with the x_2-axis are determined by Equation (5.13). The 4-digit codes describe the output of the hidden neurons, one verifies that the output layer $O = \mathrm{sgn}(-V_0 + V_1 + V_2 - V_3)$ does the trick.

Answer (7.3) — Figure 7.2 shows the r.m.s. error $\delta^{(\ell)}_{\mathrm{rms}} = \frac{1}{N_\ell}\sum_{j=1}^{N_\ell}[\delta^{(\ell)}_j]^2$ for different layers ℓ as a function of the number of training epochs. During the first 500 epochs, the gradients are small, their magnitude decreases exponentially as one moves away from the output layer. Around 1000 training epochs, the network starts to learn. The errors increase at first, resulting in desired weight changes reducing the energy function. At later times, the errors decrease as the network learns to classify the training set.

Answer (7.4) — We start with Equation (7.30),

$$\delta^{(L-1)} = \delta^{(L)} w^{(L,L-1)} g'(b^{(L-1)}). \tag{7.4}$$

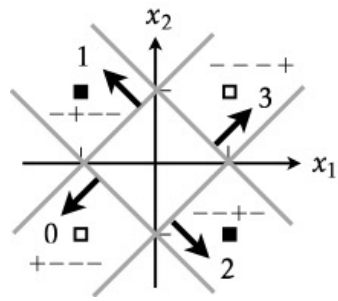


Figure 7.1: Shows solution of XOR problem. Exercise **??**.

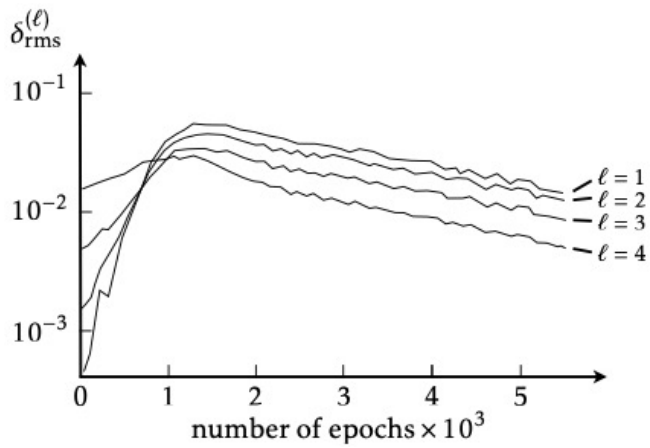


Figure 7.2: Vanishing-gradient problem. Shown is how the r.m.s error $\delta^{(\ell)}_{\text{r.m.s.}}$ in layer ℓ as a function of the number of training epochs. Exercise 7.3.

The error $\delta^{(L-2)}$ is obtained using the recursion (**7.33**):

$$\delta^{(\ell-1)} = \delta^{(\ell)} w^{(\ell,\ell-1)} g'(b^{(\ell-1)}) + \delta^{(\ell+1)} w^{(\ell+1,\ell-1)} g'(b^{(\ell-1)}), \tag{7.5}$$

valid for $\ell - 1 \leq L - 2$. This recursion reflects that every neuron $\ell - 1 < L - 2$ can be reached backwards directly from ℓ, and also from $\ell + 1$ via a skipping connection. So we have for $\delta^{(L-2)}$:

$$\delta^{(L-2)} = \delta^{(L-1)} w^{(L-1,L-2)} g'(b^{(L-2)}) + \delta^{(L)} w^{(L,L-2)} g'(b^{(L-2)}). \tag{7.6}$$

Iterating once more yields three terms for $\delta^{(L-3)}$:

$$\begin{aligned}\delta^{(L-3)} = {} & \delta^{(L)} w^{(L,L-1)} g'(b^{(L-1)}) w^{(L-1,L-2)} g'(b^{(L-2)}) w^{(L-2,L-3)} g'(b^{(L-3)}) \\ & + \delta^{(L)} w^{(L,L-2)} g'(b^{(L-2)}) w^{(L-2,L-3)} g'(b^{(L-3)}) \\ & + \delta^{(L)} w^{(L,L-1)} g'(b^{(L-1)}) w^{(L-1,L-3)} g'(b^{(L-3)}).\end{aligned} \tag{7.7}$$

Each term in this expression corresponds to one of all possible paths from L to $L-3$,

$$\begin{aligned} &L \to \ell_1 = L-1 \to \ell_2 = L-2 \to L-3\,, \\ &L \to \ell_1 = L-2 \to L-3\,, \\ &L \to \ell_1 = L-1 \to L-3\,. \end{aligned} \tag{7.8}$$

The first path visits $n = 2$ intermediate neurons, it has no skipping connections. The other two paths have one skipping connection, each of them visits only $n = 1$ intermediate neuron. There are no paths that involve two or more skipping connections. We conclude that the error $\delta^{(L-3)}$ can be written as a sum over all paths, as stated in Equation (**7.34**). The general form of Equation (**7.34**) is obtained by iterating backwards using Equation (7.5).

Answer (7.5) — We start with Equation (**7.38**):

$$H = -\sum_{i\mu} t_i^{(\mu)} \log O_i^{(\mu)}\,. \tag{7.9}$$

First show that $H = 0$ when $O_i^{(\mu)} = t_i^{(\mu)}$. This follows because $t_i^{(\mu)} = 0, 1$ and $z \log z = 0$ for $z = 0$ and $z = 1$. Second, since $0 \le O_i^{(\mu)} \le 1$, we conclude that $\log O_i^{(\mu)} \le 0$. This means that $H \ge 0$. Therefore H assumes a global minimum at $O_i^{(\mu)} = t_i^{(\mu)}$.

Answer (7.6) — Instead of (**7.44**), we use the energy function

$$H = -\sum_{i\mu} \Big[\tfrac{1+t_i^{(\mu)}}{2} \log\Big(\tfrac{1+O_i^{(\mu)}}{2}\Big) + \tfrac{1-t_i^{(\mu)}}{2} \log\Big(\tfrac{1-O_i^{(\mu)}}{2}\Big) \Big], \tag{7.10}$$

where $O_i^{(\mu)} = \tanh(b_i^{(\mu)})$ and $t_i^{(\mu)} = \pm 1$. See also Eq. (5.52) in Ref. [**1**]. First, show that $H = 0$ when $O_i^{(\mu)} = t_i^{(\mu)}$. This follows because $z \log z = 0$ for $z = 0$ and $z = 1$. Second, show that H cannot be negative. Using the inequality $-\log z \ge -(z-1)$, we have

$$H \ge -\sum_{i\mu} \Big(\tfrac{1+t_i^{(\mu)}}{2} \tfrac{O_i^{(\mu)}-1}{2} + \tfrac{1-t_i^{(\mu)}}{2} \tfrac{-1-O_i^{(\mu)}}{2} \Big) = -\sum_{i\mu} \tfrac{t_i^{(\mu)} O_i^{(\mu)} - 1}{2} = \sum_{i\mu} \tfrac{1 - t_i^{(\mu)} O_i^{(\mu)}}{2}\,. \tag{7.11}$$

Since $-1 \le O_i^{(\mu)} \le 1$ and $t_i^{(\mu)} = \pm 1$. This implies that $1 - t_i^{(\mu)} O_i^{(\mu)} \ge 0$, and therefore $H \ge 0$.

Answer (7.7) — To simplify the notation, leave out the superscripts L. The derivative of H w.r.t. w_{mn} evaluates to

$$\frac{\partial H}{\partial w_{mn}} = -\sum_{i\mu} \frac{t_i^{(\mu)}}{O_i^{(\mu)}} \frac{\partial O_i^{(\mu)}}{\partial w_{mn}}\,. \tag{7.12}$$

The derivative of the output is evaluated using Equation (7.36) with $\alpha = 1$:

$$\frac{\partial O_i^{(\mu)}}{\partial b_l} = \delta_{il} O_i^{(\mu)} - \frac{\mathrm{e}^{b_i}\mathrm{e}^{b_l}}{\left(\sum_j \mathrm{e}^{b_j}\right)^2} = O_i^{(\mu)}(\delta_{il} - O_l^{(\mu)}). \tag{7.13}$$

Now

$$\frac{\partial O_i^{(\mu)}}{\partial w_{mn}} = \sum_l \frac{\partial O_i^{(\mu)}}{\partial b_l} \frac{\partial b_l}{\partial w_{mn}}. \tag{7.14}$$

Using $b_l = \sum_k w_{lk} V_k - \theta_l$, we have $\partial b_l / \partial w_{mn} = \delta_{lm} V_n$, and therefore:

$$\frac{\partial O_i^{(\mu)}}{\partial w_{mn}} = O_i^{(\mu)}(\delta_{im} - O_m^{(\mu)}) V_n^{(\mu)}. \tag{7.15}$$

Inserting this into Equation (7.12) yields

$$\delta w_{mn} = -\eta \frac{\partial H}{\partial w_{mn}} = \eta \sum_{i\mu} t_i^{(\mu)} (\delta_{im} - O_m^{(\mu)}) V_n^{(\mu)}. \tag{7.16}$$

This becomes Equation (7.42) because the targets sum to one, $\sum_i t_i^{(\mu)} = 1$.

Answer (7.8) — The input-space representation of the three-dimensional generalised XOR function is shown in Figure 7.3. It is not linearly separable. A network that solves this classification problem is shown on the right of Figure 7.3. Contrast this with Figure 5.11. Here there is an additional hidden neuron connected to the three inputs. It outputs 1 if all inputs are 1. This output is fed into a third XOR unit, to make sure that the function evaluates to zero if all three inputs equal unity. The generalised XOR function with four inputs is represented in an analogous way. An alternative is to use winning neurons. The construction outlined in Section 7.1 requires 2^N hidden neurons, so eight hidden neurons for $N = 3$, whereas the network layout shown in Figure 7.3 has ten hidden neurons. For $N = 4$, this approach requires 12 hidden neurons, while the construction with winning neurons requires 16 hidden neurons.

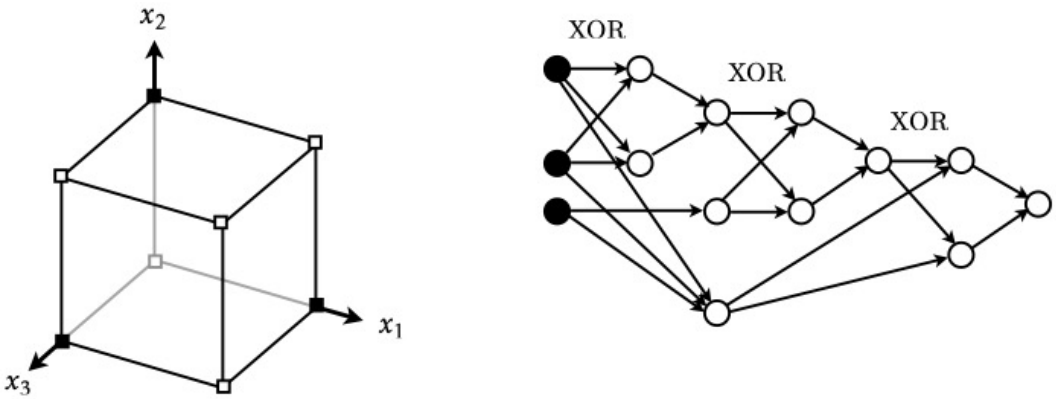


Figure 7.3: Input space of the three-dimensional exclusive XOR problem. Exercise 7.8.

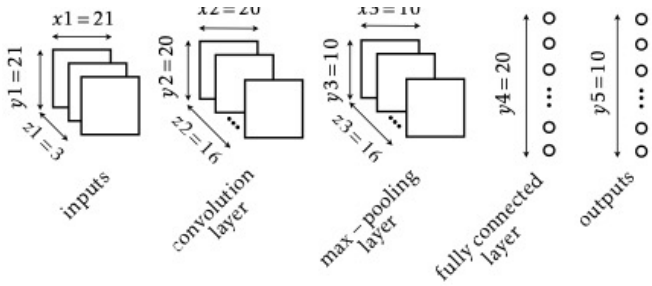


Figure 8.1: Parameter values for the network described in Exercise 8.1.

8 Exercises in Chapter 8

Answer (8.1) — Consider first the parameter values given in Figure 8.1. The input dimensions x1= 21, y1= 21, and z1= 3 are given in the question. The convolution layers have dimension 20×20 for a 2×2 local receptive field with stride $(1,1)$. So x2=y2= 20, and z2= 16. The pooling layer has a 2×2 local receptive field with stride $(2,2)$, so its dimensions are x3=y3= 10, and z3= 16. The dimensions of the fully connected layer (y4= 20) and of the output layer (y5= 10) are given in the question.
Now consider the number of trainable parameters. To count the number of trainable parameters for the convolution layer, we start from Equation (**8.3**). There, $P = 2$ and $Q = 2$ are the dimensions of the local receptive field, and $R = 3$ is the number of colour channels. Since there are 16 kernels, the number of weights w_{pqrk} is $2 \times 2 \times 3 \times 16 = 192$. In addition there are 16 thresholds θ_k. The number of weights in the fully connected layer is $10 \times 10 \times 16 \times 20 = 32000$, plus 20 thresholds. The output layer has $20 \times 10 = 200$ weights and 10 thresholds. So the total number of trainable

parameters is $208+32020+210=32438$.

Answer (8.2) — This is summarised in Sections **8.1** and **8.2**.

Answer (8.3) — Applying the kernel shown in Figure **8.14**(b) to the digits in Figure **8.14**(a) yields the following states of the hidden neurons:

$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 3 & 3 \\ 3 & 2 & 2 \\ 3 & 2 & 2 \\ 3 & 2 & 1 \end{bmatrix}, \quad \begin{bmatrix} 3 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 2 & 3 \\ 3 & 3 & 3 \\ 3 & 2 & 2 \end{bmatrix} \tag{8.2}$$

(left for input 2, right for input 8). The pooling layer maps these states into

$$\begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}, \quad \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix}. \tag{8.3}$$

Now we need 2×3 weight matrix $\mathbb{W}$ and a threshold vector $\boldsymbol{\theta}=[\theta_1,\theta_2]^\top$ to distinguish these two states. One possibility is

$$\mathbb{W}=\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \boldsymbol{\theta}=\begin{bmatrix} 10 \\ -10 \end{bmatrix}. \tag{8.4}$$

For input 2, this gives

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}\begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}-\begin{bmatrix} 10 \\ -10 \end{bmatrix}=\begin{bmatrix} -1 \\ 1 \end{bmatrix}, \tag{8.5}$$

and

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}\begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix}-\begin{bmatrix} 10 \\ -10 \end{bmatrix}=\begin{bmatrix} 2 \\ -2 \end{bmatrix} \tag{8.6}$$

for input 8.

Answer (8.4) — The network layout is shown in Figure 8.2(**a**). The network contains a total of 260458 trainable parameters. They are trained by stochastic gradient descent using an adaptive learning rate with momentum[1] as described in Chapter **6.5**. The initial learning rate was $\eta=0.01$, with η defined as in Equation (**6.18**). A mini-batch size of $m_{\mathrm{B}}=128$ was used.

[1]For this solution the Adam optimiser was used. Kingma, D.P. & Ba, J., Adam: a method for stochastic optimization, arXiv:1412.6980.

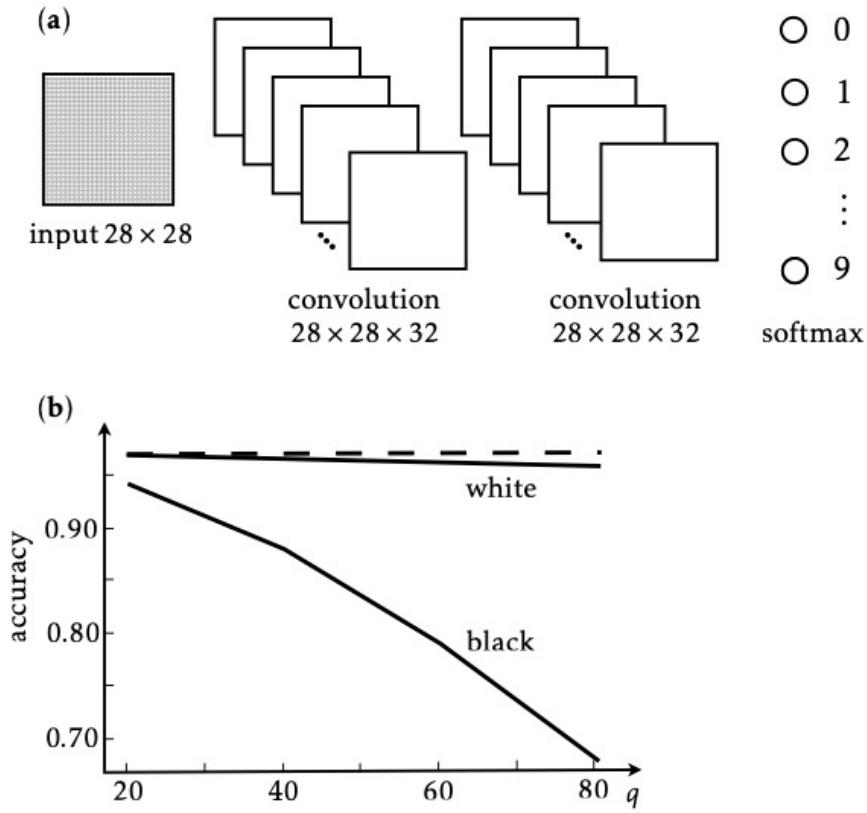


Figure 8.2: (**a**) Layout of the convolutional network for Exercise 8.4. (**b**) Accuracy as a function of the noise strength q. Also shown is the accuracy without noise (dashed line), as a benchmark. Exercise 8.4.

Figure 8.2(**b**) shows that noise reduces network performance, much more so for 'black' noise than 'white'. Part of this asymmetry comes from the fact that the white noise affects less pixels on average, since the background for the MNIST digits is white (Figure **8.5**). Note however that if one compares accuracy at similar number of changes pixels there is still a significant difference, possibly since the black noise introduces black pixels in regions close to the border, which confuses the network.

Answer (8.5) — Figure 8.3 shows the validation accuracy for two neural networks, trained on the CIFAR-10 data set. The multi-layer perceptron has two hidden layers with 105 neurons each, and an output layer with 10 softmax units (in total this network has 334855 parameters). The convolutional network has two convolutional layers with 32 3×3 kernels, a fully connected layer, and a softmax-output layer with 10 units (in total 337834 trainable parameters). Both networks were trained on

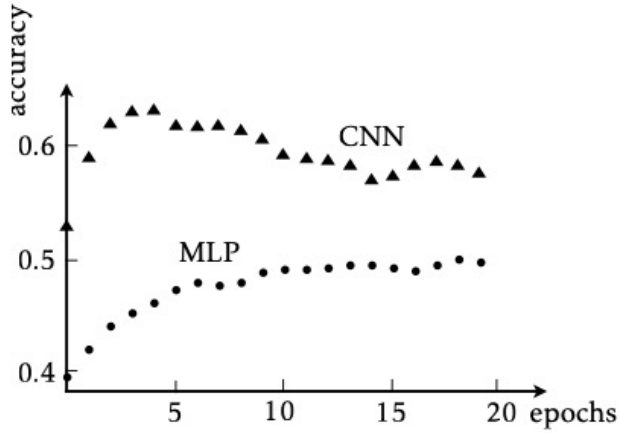


Figure 8.3: Validation accuracy for the CIFAR-10 data set [132] for a multi-layer perceptron (MLP, •) with two hidden layers with 105 neurons each, and a softmax-output layer with 10 units, and for a convolutional network (CNN, ▲) with two convolutional layers with 3×3 kernels and a fully connected softmax-output layer with 10 units. All hidden neurons have ReLU activation functions. Exercise 8.5.

CIFAR-10 using adaptive learning rate with momentum[2] without any regularisation, see Chapter 6.5, using learning rate $\eta = 0.001$, with η defined as in Equation (6.18), and mini-batch size $m_{\mathrm{B}} = 128$. [3]
Without regularisation the convolutional network overfits after around five epochs of training. Still, the onvolutional network has a validation accuracy that is approximately 10% higher than the multi-layer perceptron.

Answer (8.6) — One possibility is to use the kernels

$$\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}. \tag{8.7}$$

with ReLU neurons. Assume zero thresholds, unit stride, and zero padding. Feature-map outputs for the first two kernels are shown in Figure 8.4. For stripe patterns, applying 2×2-pooling gives $\boldsymbol{V}_1 = [4,4,0,4,0,4]^{\mathsf{T}}$ for the first kernel, and $\boldsymbol{V}_2 = [0,4,4,0,4,4]^{\mathsf{T}}$ for the second one. For bar patterns one finds $\boldsymbol{V}_1 = \boldsymbol{V}_2 = [0,0,0,0,0,0]^{\mathsf{T}}$. For the other two kernels, $\boldsymbol{V}_1$ and $\boldsymbol{V}_3$ are exchanged, and so are $\boldsymbol{V}_2$ and $\boldsymbol{V}_4$. The following output layer gives $+1$ for stripes and -1 for bars, $O = \mathrm{sgn}(V_1 + V_2 - V_3 - V_4)$.

[2]For this solution the Adam optimiser (without weight decay) was used. Kingma, D.P. & Ba, J., Adam: a method for stochastic optimization, arXiv:1412.6980.

[3]The particular values of η and m_{B} was picked after testing a number of combinations and looking for reasonable training times and accuracy.

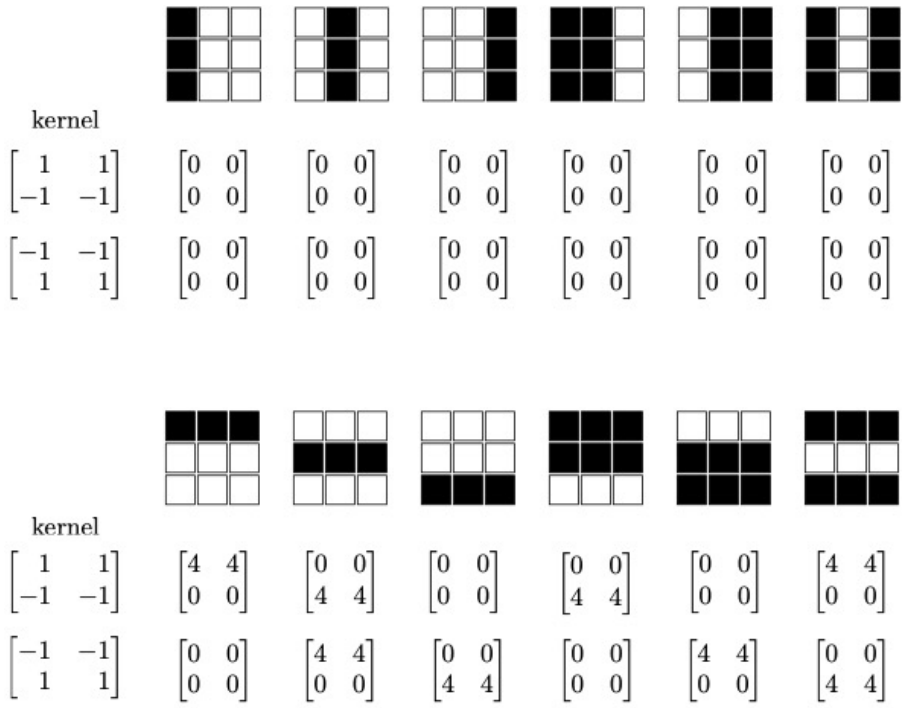


Figure 8.4: Result of applying the first two kernels in Equation (8.7) to bar and to stripe patterns. Exercise 8.6.

Answer (8.7) — Starting from a trained convolutional model (same as network architecture and training parameters as in Exercise 8.5), the feature map [4] at each layer ℓ in the model is extracted as $V^{(\ell)}_{ijk}(\boldsymbol{x})$ for an input image x. Starting with a uniform random $\boldsymbol{x}$ of size $32 \times 32 \times 3$ (the CIFAR RGB-image tensor size). Stochastic gradient ascent is used to maximize the average activation $V^{(\ell)}_k(x) = \sum_{ij} V^{(\ell)}_{ijk}$ with respect to the input x viewed as a trainable tensor. Since this optimization procedure does not know about the limits of pixel values (e.g. [01] or [0,255], depending on image format choosen) in the input image domain it is also necessary to clip the output to this range after each step. Using an initial learning rate of $\eta = 0.01$ and performing 500 steps of gradient ascent using the Adam optimiser[5], the process is repeated for different choices of layer ℓ and feature map k.

Figure 8.5 shows the optimized input $\boldsymbol{x}$ for $\ell = \{1,2,3\}$ and $k = \{5,5,8\}$ (i.e. first panel shows an optimized input for $k = 5$ of layer 1) corresponding to the different layers in the model (first convolution, second convolution, softmax output). We see that the filter corresponding to $k = 5$ in the first convolutional layer (left panel in Figure 8.5)

[4] By slight abuse of notation we also refer to the output of the softmax layer as a feature map.

[5] Kingma, D.P. & Ba, J., Adam: a method for stochastic optimization, arXiv:1412.6980.

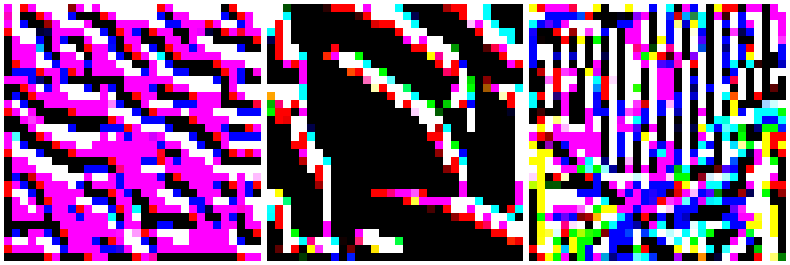


Figure 8.5: Input images maximising the average activation of a randomly choosen feature map in the first convolution layer (left), in the second convolution layer (middel), and in the softmax output layer (right). Exercise 8.7.

looks for straight patterns in the top-left to bottom-right direction. The middle panel exhibits more complex shapes: the deeper position in the model can utilize the previous layers features to build more expressive representations and we observe thicker patterns with varying orientations. The right panel in Figure 8.5 shows an image that maximizes the output for the 9th softmax output layer, corresponding to the ship class *ship* of CIFAR-10. Even though this image does not look like a ship at all, it is still be classified as such by this network.

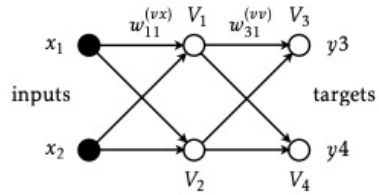


Figure 9.1: Multi-layer feedforward network obtained by removing the feedback connection in Figure 9.1. Exercise 9.1.

9 Exercises in Chapter 9

Answer (9.1) — We use gradient descent on the energy function (9.4) to derive the learning rule for $w_{mn}^{(vx)}$,

$$\delta w_{mn}^{(vx)} = -\eta \frac{\partial H}{\partial w_{mn}^{(vx)}} = \eta \sum_k E_k^* \frac{\partial V_k^*}{\partial w_{mn}^{(vx)}} . \tag{9.1}$$

To evaluate the derivative of V_k^*, we use Equation (9.6):

$$\frac{\partial V_k^*}{\partial w_{mn}^{(vx)}} = g'(b_k^*)\Big(\delta_{km} x_n + \sum_j w_{kj}^{(vv)} \frac{\partial V_j^*}{\partial w_{mn}^{(vx)}}\Big). \tag{9.2}$$

The only difference to Equation (9.8) is that we have x_n instead of V_n^* in the first term on the right. So the learning rule for $w_{mn}^{(vx)}$ is obtained from Equation (9.14) upon replacing V_n^* by x_n. This gives Equation (9.15).
For the second part consider the network shown in Figure 9.1. It is very similar to Figure 9.1 except that there is no feedback connection. The goal is to show that the recurrent backprogation rule (9.13)

$$\Delta_m^* = g'(b_m^*) \sum_k E_k^* [\mathbb{L}^{-1}]_{km} , \tag{9.3}$$

simplifies to Equations (6.6) and (6.8). The matrix $\mathbb{L}$ in Equation (9.3) has entries $L_{ij} = \delta_{ij} - g'(b_i^*) w_{ij}^{(vv)}$ [Equation (9.9)] and the errors are $E_k^* = y_k - V_k^*$ for $k = 3,4$ (and zero otherwise). For the network shown in Figure 9.1, the matrix $\mathbb{L}$ reads

$$\mathbb{L} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ -g'(b_3^*) w_{31} & -g'(b_3^*) w_{32} & 1 & \\ -g'(b_4^*) w_{41} & -g'(b_4^*) w_{42} & & 1 \end{bmatrix} , \tag{9.4}$$

with inverse

$$\mathbb{L}^{-1} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ g'(b_3^*)w_{31} & g'(b_3^*)w_{32} & 1 & \\ g'(b_4^*)w_{41} & g'(b_4^*)w_{42} & & 1 \end{bmatrix}. \tag{9.5}$$

Inserting $\mathbb{L}^{-1}$ into Equation (9.3), we find

$$[\boldsymbol{\Delta}^*]^\mathsf{T} = [0, 0, E_3^*, E_4^*] \begin{bmatrix} g'(b_1^*) & & & \\ & g'(b_2^*) & & \\ g'(b_3^*)w_{31}g'(b_1^*) & g'(b_3^*)w_{32}g'(b_2^*) & g'(b_3^*) & \\ g'(b_4^*)w_{41}g'(b_1^*) & g'(b_4^*)w_{42}g'(b_2^*) & & g'(b_4^*) \end{bmatrix}. \tag{9.6}$$

Using this result together with Equation (**9.14**) yields

$$\delta w_{3n}^{(vv)} = \eta(y_3 - V_3^*)g'(b_3^*)V_n^* \quad \text{and} \quad \delta w_{4n}^{(vv)} = \eta(y_4 - V_4^*)g'(b_3^*)V_n^* \tag{9.7}$$

for the output weights. In this formula, the index n takes the values 1 and 2 (corresponding to the two hidden neurons). Equation (9.7) is the update rule (**6.6**) for the output weights, derived in Section **6**. From Equation (**9.15**) we obtain

$$\delta w_{1n}^{(vx)} = \eta \sum_{i=3,4} (y_i - V_i^*)g'(b_i^*)w_{i1}^{(vv)}g'(b_1^*)x_n\,, \tag{9.8a}$$

$$\delta w_{2n}^{(vx)} = \eta \sum_{i=3,4} (y_i - V_i^*)g'(b_i^*)w_{i2}^{(vv)}g'(b_2^*)x_n \tag{9.8b}$$

for the hidden weights. This is the update rule (**6.9**) for the hidden weights. Since all eigenvalues of $\mathbb{L}$ equal unity, the discussion on p. **161** shows that the steady state $\boldsymbol{V}^*$ is linearly stable. We conclude that the recurrent-backpropagation algorithm reduces to backpropagation (Algorithm **4** in Chapter **6**) for a multilayer feed-forward network.

Answer (9.2) — The steady state $\boldsymbol{V}^*$ of Equation (**9.3**) is defined by $\mathrm{d}\boldsymbol{V}^*/\mathrm{d}t = 0$. This means that the the r.h.s. of Equation (**9.3**) must evaluate to zero at the steady state. This gives

$$V_i^* = g\Big(\sum_j w_{ij}^{(vv)}V_j^* + \sum_k w_{ik}^{(vx)}x_k - \theta_i^{(v)}\Big). \tag{9.9}$$

This is Equation (**9.6**). The stability of $\boldsymbol{V}^*$ is determined by linearising the dynamics around $\boldsymbol{V}^*$, $\boldsymbol{V}(t) = \boldsymbol{V}^* + \delta\boldsymbol{V}(t)$. Expanding the r.h.s. of (**9.3**) in small $|\delta\boldsymbol{V}|$ yields $\tau\frac{\mathrm{d}}{\mathrm{d}t}\delta\boldsymbol{V} = -\mathbb{L}\delta\boldsymbol{V}$. The matrix $\mathbb{L}$ is given by Equation (**9.9**). We conclude that the fixed point $\boldsymbol{V}^*$ is stable if all eigenvaluels of $\mathbb{L}$ are positive. If this is not the case, the steady state is a saddle point and training fails. See also Exercise **??**.

Answer (9.3) — We start with Equation (9.28),

$$\delta w^{(ov)} = -\eta \frac{\partial H}{\partial w^{(ov)}} = -\eta \frac{\partial}{\partial w^{(ov)}} \frac{1}{2} \sum_t (y_t - O_t)^2 = \eta \sum_t E_t g'(B_t) V_t \,, \tag{9.10}$$

using Equation (9.19b). The quantity $E_t g'(B_t) \equiv \Delta_t$ is an output error. Now consider (9.27),

$$\delta w^{(vx)} = -\eta \frac{\partial H}{\partial w^{(vx)}} = -\eta \frac{\partial}{\partial w^{(vx)}} \frac{1}{2} \sum_t (y_t - O_t)^2 = \eta \sum_t E_t \frac{\partial O_t}{w^{(vx)}} \,. \tag{9.11}$$

Evaluating the derivative of O_t, one finds:

$$\frac{\partial O_t}{w^{(vx)}} = w^{(ov)} g'(B_t) \frac{\partial V_t}{w^{(vx)}} x_t \,. \tag{9.12}$$

Evaluating the derivative of V_t using Equation (19.a), one finds the recursion (9.26)

$$\frac{\partial V_t}{w^{(vx)}} = g'(b_t)\Big(x_t + w^{(vv)} \frac{\partial V_{t-1}}{w^{(vx)}} \Big). \tag{9.13}$$

This is the same as (9.23), but with x_t instead of V_{t-1}. Therefore the weight increment $\delta w^{(vx)}$ has the same form as Equation (9.25), but V_{t-1} is replaced by x_t. This yields Equation (9.27).

Answer (9.4) — The network dynamics is given by Equations (9.19):

$$V(t) = g(w^{(vv)} V(t-1) + w^{(vx)} x(t) - \theta^{(v)}), \tag{9.14a}$$

$$O(t) = g(w^{(ov)} V(t) - \theta^{(o)}). \tag{9.14b}$$

The learning rule for $w^{(ov)}$ is derived from

$$\delta w^{(ov)} = -\eta \frac{\partial H}{\partial w^{(ov)}} \,, \tag{9.15}$$

with energy function (9.20). Using the chain rule, we find:

$$\delta w^{(ov)} = \eta \sum_{t=1}^{T} E_t \frac{\partial O_t}{\partial w^{(ov)}} \,. \tag{9.16}$$

Using Equation (9.14b), we obtain the learning rule for $w^{(ov)}$:

$$\delta w^{(ov)} = \eta \sum_{t=1}^{T} E_t g'(B_t) V_t = \eta \sum_{t=1}^{T} \Delta_t V_t \,. \tag{9.17}$$

Here $B_t = w^{(ov)} V(t) - \theta^{(o)}$ and $\Delta_t = E_t g'(B_t)$ [Equation (9.22)].

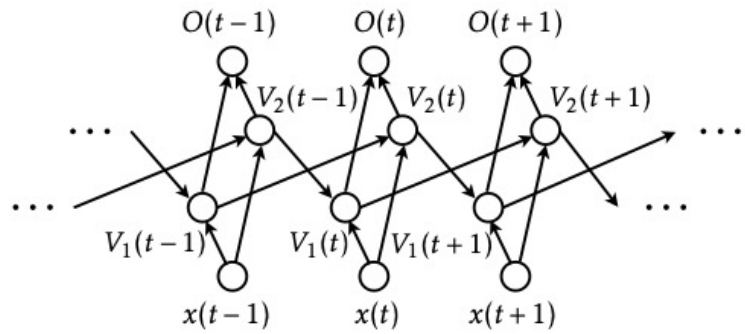


Figure 9.2: Network from Figure 9.8 unfolded in time. Exercise 9.5.

Answer (9.5) — We denote the input weights as $w_i^{(vx)}$, the hidden weights as $w_{ij}^{(vv)}$, and the output weights as $w_j^{(ov)}$. We put the thresholds to zero to simplify the notation. The update rule for the output is given by Equation (9.19b):

$$O^{(t)} = g\left(\sum_{j=1}^{2} w_j^{(ov)} V_j^{(t)}\right) \tag{9.18}$$

where $V_j(t)$ is the output of the j:th hidden neuron at time t. The update rule for the hidden neurons is given by Equation (9.19a). For hidden neuron $i = 1$, for example, one has:

$$V_1^{(t)} = g\left(w_1^{(vx)} x^{(t)} + w_{12}^{(vv)} V_2^{(t-1)}\right), \tag{9.19}$$

where $x(t)$ is the input at time t. Note that the hidden neurons have no self connections. The unfolded network is shown in Figure 9. We now derive the learning rules for the weights using the energy function

$$H = \frac{1}{2}\sum_{t=1}^{T}[E^{(t)}]^2, \qquad E^{(t)} = y^{(t)} - O^{(t)}, \tag{9.20}$$

where $y^{(t)}$ is the target value at time t. We start with the output weights:

$$\delta w_m^{(ov)} = -\eta \frac{\partial H}{\partial w_m^{(ov)}} = \eta \sum_{t=1}^{T} E^{(t)} \frac{\partial O_t}{\partial w_m^{(ov)}} = \eta \sum_{t=1}^{T} E^{(t)} g'(B^{(t)}) \sum_{j=1}^{2} \delta_{jm} V_j^{(t)}. \tag{9.21}$$

Here, $B^{(t)}$ is the local field of the output neuron at time t. Evaluating the sum over j, we find:

$$\delta w_m^{(ov)} = \eta \sum_{t=1}^{T} E^{(t)} g'(B^{(t)}) V_m^{(t)} = \eta \sum_{t=1}^{T} \Delta^{(t)} V_m^{(t)}, \tag{9.22}$$

where $\Delta^{(t)} = E^{(t)} g'(B^{(t)})$. Now consider the learning rule for the hidden weights $w_{ij}^{(vv)}$. We begin by deriving the learning rule for $\delta w_{12}^{(vv)}$:

$$\begin{aligned}\delta w_{12}^{(vv)} &= -\eta \frac{\partial H}{\partial w_{12}^{(vv)}} = \eta \sum_{t=1}^{T} E^{(t)} \frac{\partial O^{(t)}}{\partial w_{12}^{(vv)}} \\ &= \eta \sum_{t=1}^{T} \Delta^{(t)} \left(w_1^{(ov)} \frac{\partial V_1^{(t)}}{\partial w_{12}^{(vv)}} + w_2^{(ov)} \frac{\partial V_2^{(t)}}{\partial w_{12}^{(vv)}} \right). \end{aligned} \tag{9.23}$$

The next step is to derive a recursion formula for $\partial V_1^{(t)} / \partial w_{12}^{(vv)}$ and $\partial V_2^{(t)} / \partial w_{12}^{(vv)}$:

$$\frac{\partial V_1(t)}{\partial w_{12}^{(vv)}} = g'(b_1^{(t)}) \left[V_2^{(t-1)} + w_{12}^{(vv)} \frac{\partial V_2^{(t-1)}}{\partial w_{12}^{(vv)}} \right], \tag{9.24a}$$

$$\frac{\partial V_2^{(t)}}{\partial w_{12}^{(vv)}} = g'(b_2^{(t)}) w_{21}^{(vv)} \frac{\partial V_1^{(t-1)}}{\partial w_{12}^{(vv)}}. \tag{9.24b}$$

Inserting Equation (9.24) into (9.23), we find recursions for the gradients:

$$\frac{\partial V_1^{(t)}}{\partial w_{12}^{(vv)}} = g'(b_1^{(t)}) \left[V_2^{(t-1)} + w_{12}^{(vv)} g'[b_2^{(t-1)}] w_{21}^{(vv)} \frac{\partial V_1^{(t-2)}}{\partial w_{12}^{(vv)}} \right], \tag{9.25a}$$

$$\frac{\partial V_2^{(t)}}{\partial w_{12}^{(vv)}} = g'(b_2^{(t)}) w_{21}^{(vv)} g'(b_1^{(t-1)}) \left[V_2^{(t-2)} + w_{12}^{(vv)} \frac{\partial V_2^{(t-2)}}{\partial w_{12}^{(vv)}} \right]. \tag{9.25b}$$

As expected, this is a two-step recursion. We iterate the recursion starting from $t = 1$ to see whether any pattern emerges:

$$\frac{\partial V_1^{(1)}}{\partial w_{12}^{(vv)}} = g'(b_1^{(1)}) V_2^{(0)}, \quad \frac{\partial V_2^{(1)}}{\partial w_{12}^{(vv)}} = 0 \tag{9.26a}$$

Here we used Equations (9.24a) and (9.24b), as well as $\partial V_i^{(0)} / \partial w_{12}^{(vv)} \equiv 0$. Iterating,

we find:

$$\frac{\partial V_1^{(2)}}{\partial w_{12}^{(\nu\nu)}} = g'(b_1^{(2)})V_2^{(1)}, \tag{9.26b}$$

$$\frac{\partial V_2^{(2)}}{\partial w_{12}^{(\nu\nu)}} = g'(b_2^{(2)})w_{21}^{(\nu\nu)}g'(b_1^{(1)})V_2^{(0)}, \tag{9.26c}$$

$$\frac{\partial V_1^{(3)}}{\partial w_{12}^{(\nu\nu)}} = g'(b_1^{(3)})V_2^{(2)} + g'(b_1^{(3)})w_{12}^{(\nu\nu)}g'(b_2^{(2)})w_{21}^{(\nu\nu)}g'(b_1^{(1)})V_2^{(0)}, \tag{9.26d}$$

$$\frac{\partial V_2^{(3)}}{\partial w_{12}^{(\nu\nu)}} = g'(b_2^{(3)})w_{21}^{(\nu\nu)}g'(b_1^{(2)})V_2^{(1)}, \tag{9.26e}$$

$$\frac{\partial V_1^{(4)}}{\partial w_{12}^{(\nu\nu)}} = g'(b_1^{(4)})V_2^{(3)} + g'(b_1^{(4)})w_{12}^{(\nu\nu)}g'(b_2^{(3)})w_{21}^{(\nu\nu)}g'(b_1^{(2)})V_2^{(1)}, \tag{9.26f}$$

$$\begin{aligned}\frac{\partial V_2^{(4)}}{\partial w_{12}^{(\nu\nu)}} &= g'(b_2^{(4)})w_{21}^{(\nu\nu)}g'(b_1^{(3)})V_2^{(2)} \\ &\quad + g'(b_2^{(4)})w_{21}^{(\nu\nu)}g'(b_1^{(3)})w_{12}^{(\nu\nu)}g'(b_2^{(2)})w_{21}^{(\nu\nu)}g'(b_1^{(1)})V_2^{(0)},\end{aligned} \tag{9.26g}$$

and so forth. In the sum over t in Equation (9.23), we regroup the terms as described

on page 164:

$$
\begin{aligned}
\delta w_{12}^{(vv)} = \eta\Big[&\Delta^{(1)}\left(w_1^{(ov)}\frac{\partial V_1^{(1)}}{\partial w_{12}^{(vv)}} + w_2^{(ov)}\frac{\partial V_2^{(1)}}{\partial w_{12}^{(vv)}}\right)\\
&+\Delta^{(2)}\left(w_1^{(ov)}\frac{\partial V_1^{(2)}}{\partial w_{12}^{(vv)}} + w_2^{(ov)}\frac{\partial V_2^{(2)}}{\partial w_{12}^{(vv)}}\right)+\dots\Big]\\
= \eta w_1^{(ov)}\Big[&\Delta^{(T)}g'(b^{(T)})V_2^{(T-1)} + \Delta^{(T-1)}g'(b_1^{(T-1)})V_2^{(T-2)}\\
&+\big(\Delta^{(T-2)}g'(b_1^{(T-2)})\\
&\quad+\Delta^{(T)}g'(b_1^{(T)})w_{12}^{(vv)}g'(b_2^{(T-1)})w_{21}^{(vv)}g'(b_1^{(T-2)})\big)V_2^{(T-3)}\\
&+\big(\Delta^{(T-3)}g'(b_1^{(T-3)})\\
&\quad+\Delta^{(T-1)}g'(b_1^{(T-1)})w_{12}^{(vv)}g'(b_2^{(T-2)})w_{21}^{(vv)}g'(b_1^{(T-3)})\big)V_2^{(T-4)}\\
&+\dots\Big]\\
+\eta w_2^{(ov)}\Big[&\Delta^{(T)}g'(b_2^{(T)}w_{21}^{(vv)}g'(b_1^{(T-1)})V_2^{(T-2)}\\
&+\Delta^{(T-1)}g'(b_2^{(T-1)})w_{21}^{(vv)}g'(b_1^{(T-2)})V_2^{(T-3)}\\
&+\big(\Delta^{(T-2)})g'(b_2^{(T-2)})w_{21}^{(vv)}g'(b_1^{(T-3)})\\
&\quad+\Delta^{(T)}g'(b_2^{(T)})w_{21}^{(vv)}g'(b_1^{(T-1)})w_{12}^{(vv)}g'(b_2^{(T-2)})w_{21}^{(vv)}g'(b_1^{(T-3)})\big)V_2^{(T-4)}\\
&+\dots\Big].
\end{aligned}
$$

This expression implies the following recursions for the error $\delta_1^{(t)} = a_1^{(t)} + c_1^{(t)}$

$$
a_1^{(t)} = \begin{cases}
\Delta^{(T)}w_1^{(ov)}g'(b_1^{(T)}) & \text{if } t = T\,,\\
\Delta^{(T-1)}w_1^{(ov)}g'(b_1^{(T-1)}) & \text{if } t = T-1\,,\\
\Delta^{(t)}w_1^{(ov)}g'(b_1^{(t)}) & \\
\qquad +a_1^{(t+2)}w_{12}^{(vv)}g'(b_2^{(t+1)}]w_{21}^{(vv)}g'(b_1^{(t)}) & \text{if } 0 < t < T-1\,,
\end{cases}
$$

$$
c_1^{(t)} = \begin{cases}
0, & \text{if } t = T\\
\Delta^{(T)}w_2^{(ov)}g'(b_2^{(T)})w_{21}^{(vv)}g'(b_1^{(T-1)}) & \text{if } t = T-1\,,\\
\Delta^{(T-1)}w_2^{(ov)}g'(b_2^{(T-1)})w_{21}^{(vv)}g'(b_1^{(T-2)}) & \text{if } t = T-2\,,\\
\Delta^{(t)}w_2^{(ov)}g'(b_2^{(t)})w_{21}^{(vv)}g'(b_1^{(t)}) & \\
\qquad +c_1^{(t+2)}w_{12}^{(vv)}g'(b_2^{(t+1)})w_{21}^{(vv)}g'(b_1^{(t)}) & \text{if } 0 < t < T-2\,.
\end{cases}
$$

Using these definitions expressions for the errors, we finally obtain:

$$
\delta w_{12}^{(vv)} = \eta\sum_{t=1}^{T}\delta_1^{(t)}V_2^{(t-1)}. \tag{9.27}
$$

To obtain the update rule for w_{21}, one simply exchanges the indices '1' and '2' in the above formulae. Through a similar calculation, the update rules for the input weights become

$$\delta w_1^{(vx)} = \eta \sum_{t=1}^{T} \delta_1^{(t)} x^{(t)} \quad \text{and} \quad \delta w_2^{(vx)} = \eta \sum_{t=1}^{T} \delta_2^{(t)} x^{(t)}. \tag{9.28}$$

Answer (9.6) — The learning rules for the thresholds summarised in Algorithm 7 follow from the learning rules for the weights. Looking at the network dynamics (9.19) we see that derivatives w.r.t. the thresholds $\theta_i^{(v)}$ and $\theta_i^{(o)}$ are obtained from derivatives w.r.t. to the corresponding weights $w_{mn}^{(vv)}$ and $w_{mn}^{(ov)}$ by setting $V_n(t-1)$ [or $V_n(t)$] to -1. This implies

$$\delta \theta_m^{(v)} = -\eta \sum_{t=1}^{T} \delta_m(t) \quad \text{and} \quad \delta \theta_m^{(o)} = -\eta \sum_{t=1}^{T} \Delta_m(t). \tag{9.29}$$

Answer (9.7) — The proof is given by Hertz, Krogh & Palmer [1]. The steps are as follows. The steady state $\mathbf{\Delta}^*$ of (9.16) is obtained by setting the r.h.s. of Equation (9.16) to zero. This gives

$$Y_j^* - \sum_k Y_k^* g'(b_k^*)] w_{kj}^{(vv)} = E_j^*, \tag{9.30}$$

with $Y_j^* = \Delta_j^* / g'(b_j^*)$. We want to show that this is consistent with Equation (9.13), which is equivalent to

$$\sum_k Y_k^* \mathbb{L}_{kj} = E_j^*. \tag{9.31}$$

According to Equation (9.9), $\mathbb{L} = \delta_{kj} - g'(b_k^*) w_{kj}^{(vv)}$. This shows that (9.30) and (9.31) are equivalent.

Answer (9.8) — Consider first $\mathbb{A}_1$. It is a symmetric matrix, so it has real eigenvalues, $\nu_{1,2} = \frac{1}{2}(3 \pm \sqrt{17})$. The singular values Λ_α of $\mathbb{A}_1$ are the square roots of the eigenvalues σ_α of $\mathbb{A}_1^\mathsf{T} \mathbb{A}_1$. Since $\mathbb{A}_1$ is symmetric, we have $\sigma_\alpha = \nu_\alpha^2$, and thus $\Lambda_\alpha = \nu_\alpha$.
The matrix $\mathbb{A}_2$ has complex eigenvalues, $\nu_\alpha = 1 \pm \mathrm{i}$. Its singular values equal $\Lambda_\alpha = \sqrt{2} = |\nu_\alpha|$. This is generally true for normal matrices for which $\mathbb{A}^\mathsf{T}\mathbb{A} = \mathbb{A}\mathbb{A}^\mathsf{T}$. The matrix $\mathbb{A}_2$ is normal:

$$\mathbb{A}_2^\mathsf{T} \mathbb{A}_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \mathbb{A}_2 \mathbb{A}_2^\mathsf{T}. \tag{9.33}$$

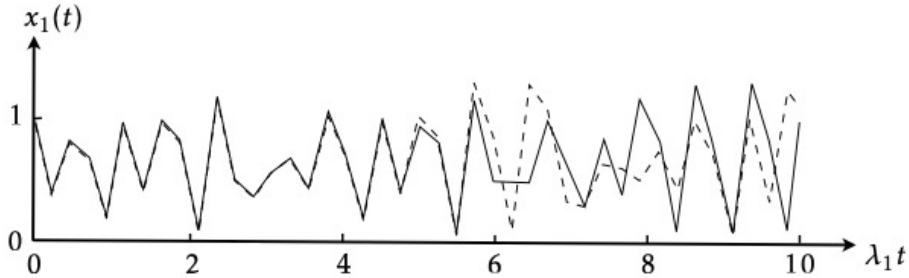


Figure 9.3: Time series generated using the Ikeda map (**??**), solid line. Prediction of the time series using a reservoir computer, dashed line. Exercise 9.9.

The matrix A_3 is not normal. Its eigenvalues are $\nu_1 = 2$ and $\nu_2 = 1$, and its singular values are $\Lambda_\alpha = \frac{1}{2}(\sqrt{13} \pm \sqrt{5})$. This illustrates that singular values are different from the eigenvalues, in general.
Finally consider the singular values of $\mathbb{B}(t) = \mathbb{A}^t$ for large integer t. We express $\mathbb{B}(t)$ in terms of its left and right eigenvectors, $\mathbb{B}(t) = \sum_\alpha \nu_\alpha^t \boldsymbol{R}_\alpha \overline{\boldsymbol{L}}_\alpha^\mathsf{T}$. Here $\overline{\boldsymbol{L}}^\mathsf{T}$ denotes the Hermitian transpose of $\boldsymbol{L}$. We find $\mathbb{B}(t)^\mathsf{T}\mathbb{B}(t) = \sum_{\alpha,\beta} \nu_\alpha^t \overline{\nu}_\beta^t \boldsymbol{L}_\alpha (\overline{\boldsymbol{R}}_\alpha^\mathsf{T} \boldsymbol{R}_\alpha) \overline{\boldsymbol{L}}_\alpha$. At large times, only one term in the double sum survives, corresponding to the eigenvalue ν_1 with largest modulus $|\nu_1|$. This means that the maximal singular value of $\mathbb{B}(t)$ converges to $|\nu_1|^t = \exp(t \log|\nu_1|)$.

Answer (9.9) — The Ikeda map is chaotic because it has a positive maximal Lyapunov exponent. Therefore it is difficult to predict the Ikeda time series for much longer than $\lambda_1 t$. For the parameters specified in the question, $\lambda_1 \approx 0.24$. Figure 9.3 shows a time series $x_1(t)$ generated using the Ikeda map (solid line), and the prediction obtained using a reservoir computer (dashed line). We see that the reservoir computer manages to predict the time series up to $\lambda_1 t \approx 5$.
The reservoir computer was set up as follows. The reservoir contained $\mathcal{N} = 500$ neurons. Its weights were sampled from a normal distribution with mean zero and variance $\mathcal{N}\sigma_w^2 = 1$. There was one input, $x(t)$. The input weights $w_i^{(\text{in})}$ were sampled from a normal distribution with mean zero and variance $\sigma_{\text{in}}^2 = 1$. The reservoir states were initialised to zero $r_i(0) = 0$, and the reservoir dynamics (**9.34a**) was iterated using the second component $x_2(t)$ of the Ikeda time series as the input $x(t)$. The first 100 iterations were discarded. From the next T iterations, the energy function is computed. Using Equation (**9.35b**), it can be written as

$$H = \frac{1}{2}\sum_{t=0}^{T-1}[y(t) - O(t)]^2 = \frac{1}{2}\left\|\boldsymbol{y} - \mathbb{R}^\mathsf{T}\boldsymbol{w}^{(\text{out})}\right\|^2 \tag{9.35}$$

For time-series prediction, the target $y(t)$ equals the input $x(t)$. Furthermore, $\mathbb{R} = [\boldsymbol{r}(0), \ldots, \boldsymbol{r}(T-1)]$, and $\boldsymbol{y} = [x(0), \ldots, x(T-1)]^\mathsf{T}$. Minimising H is a regression

problem to determine the parameters $\boldsymbol{w}^{(\text{out})}$. It was solved with ridge regression[1] with ridge parameter 0.001.
After determining the outputs weights as described above, the reservoir computer was used to predict the time series. To this end, the outputs were fed into the inputs. This corresponds to replacing $x(t)$ in Equation (**9.35a**) by $O(t)$ (there is no index k because there is only one input component and one output neuron). Iterating the reservoir dynamics once, one gets $\boldsymbol{r}(T)$ from $\boldsymbol{r}(T-1)$, and the predicted value from $O(T) = \boldsymbol{w}^{(\text{out})} \cdot \boldsymbol{r}(T)$, using Equation (**9.35b**). Figure 9.3 was obtained by iterating Equation (**9.35a**), always with $x(t) = O(t)$ and fixed output weights $\boldsymbol{w}^{(\text{out})}$.

[1] See Section 1.5 of [W. N. van Wieringen, *Lecture notes on ridge regression*, arxiv:1509.09169].

10 Exercises in Chapter 10

Answer (10.1) — Write $\boldsymbol{q} = \alpha \boldsymbol{w}$ and assume $\boldsymbol{w}$ is a unit vector, so that α is the length of $\boldsymbol{q}$. Equation (10.4) implies

$$\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q} = \dot{\alpha}\boldsymbol{w} + \alpha\tfrac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{w} = \alpha\mathbb{A}\boldsymbol{w}\,. \tag{10.1}$$

The norm of $\boldsymbol{q}$ changes as $\frac{\mathrm{d}}{\mathrm{d}t}|\boldsymbol{q}|^2 = 2\alpha\dot{\alpha} = 2\boldsymbol{q}\cdot\mathbb{A}\boldsymbol{q} = 2\alpha^2\boldsymbol{w}\cdot\mathbb{A}\boldsymbol{w}$, where we used $|\boldsymbol{w}| = 1$ for the first equality. We conclude that $\dot{\alpha} = \alpha(\boldsymbol{w}\cdot\mathbb{A}\boldsymbol{w})$. Substituting this into Equation (10.1) gives Equation (10.5).

Answer (10.2) — The data in Figure 10.4 has non-zero mean. Therefore the matrix $\mathbb{C}'$ defined in Equation (10.7) is different from the data-covariance matrix $\mathbb{C}$ with elements

$$C_{ij} = \frac{1}{3}\sum_{\mu=1}^{3}(x_i - \langle x_i\rangle)(x_j - \langle x_j\rangle) = \frac{1}{3}\sum_{\mu=1}^{3}(x_i - \tfrac{2}{3})(x_j - \tfrac{2}{3})\,. \tag{10.2}$$

From Figure 10.4 we read off that

$$\mathbb{C} = \frac{1}{3}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \tag{10.3}$$

This matrix has eigenvalues and eigenvectors

$$\lambda_1 = 1\,, \quad \boldsymbol{u}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \text{and} \quad \lambda_2 = \tfrac{1}{3}\,, \quad \boldsymbol{u}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{10.4}$$

We conclude that the principal direction is $\boldsymbol{u}_2$.

Answer (10.3) — This is explained in Section 10.1.

Answer (10.4) — We start at $\boldsymbol{w} = \boldsymbol{w}_0$ and iterate Equation (10.3). The result is a sequence of updates $\boldsymbol{w}_k = \boldsymbol{w}_{k-1} + \delta\boldsymbol{w}_k$ with $\delta\boldsymbol{w}_k = \eta\big(\mathbb{X}_k\boldsymbol{w}_{k-1} - (\boldsymbol{w}_{k-1}\cdot\mathbb{X}_1\boldsymbol{w}_{k-1})\boldsymbol{w}_{k-1}\big)$, with $\mathbb{X}_k = \boldsymbol{x}^{(k)}\boldsymbol{x}^{(k)\mathsf{T}}$. Expanding to linear order in η one finds

$$\delta\boldsymbol{w}_k = \eta\big(\mathbb{X}_k\boldsymbol{w}_0 - (\boldsymbol{w}_0\cdot\mathbb{X}_k\boldsymbol{w}_0)\boldsymbol{w}_0\big). \tag{10.5}$$

Averaging gives

$$\langle\delta\boldsymbol{w}\rangle = \eta\big(\mathbb{C}'\boldsymbol{w}_0 - (\boldsymbol{w}_0\cdot\mathbb{C}'\boldsymbol{w}_0)\boldsymbol{w}_0\big). \tag{10.6}$$

Now we can set $\boldsymbol{w}_0 = \boldsymbol{u}_\alpha + \boldsymbol{\varepsilon}_0$ and analyse how $\langle\delta\boldsymbol{w}\rangle$ depends on $\boldsymbol{\varepsilon}_0$. The conclusions are the same as obtained in Section 10.1.

Answer (10.5) — The steady-state condition (**10.6**) inmplies

$$0 = \langle \delta\, w_{i_0 j} \rangle = \left\langle \left(\frac{x_j}{\sum_k x_k} - w^*_{i_0 j} \right) \right\rangle . \tag{10.7}$$

We conclude

$$\boldsymbol{w}^*_{i_0} = \left\langle \frac{x_j}{\sum_k x_k} \right\rangle . \tag{10.8}$$

Since $x_k = 0$ or 1, it follows that the the elements of $\boldsymbol{w}^*_{i_0}$ cannot be negative. Moreover,

$$\sum_j w^*_{i_0 j} = \sum_j \left\langle \frac{x_j}{\sum_k x_k} \right\rangle = 1 . \tag{10.9}$$

Answer (10.6) — To do.

Answer (10.7) — A self-organising map with distance function (**10.18**) with dimension 50×1 was trained on the data described in the exercise for 50 epochs with an initial learning rate of $\eta_0 = 0.1$ with a decay rate of $d_\eta = 0.01$ and an initial with $\sigma_0 = 10$ with a decay rate of $d_\sigma = 0.05$ [see Equation (10.11)]. The result is shown in Fig. 10.1, where the solid line is the principal component and the dashed line is the principal manifold. To quantify the variance unexplained by the model, consider the sum of squared residuals

$$SSR = \sum_{i=1}^{N} \left(y_i - \hat{y}_i \right)^2 , \tag{10.10}$$

where $N = 50$ is the number of data points, y_i is the target value, and $\hat{y}_i$ is the model output. SSR equals 0.52 for the principal-component model, and 0.08 for the principal-manifold model. The unexplained variance is, as expected, lower for the principal manifold.

Answer (10.8) — Figure 10.2 shows the results of a self-organising map that maps the iris data set to a 40×40 output array. Symbols denote the locations of the winning neurons in the output array, colour-coded according to the classification of the input patterns, as described in Section **10.3**. The map was obtained by iterating the learning rule (**10.17**) with neighbourhood function (**10.18**). The learning rate η and the width σ of the neighbourhood function were reduced during the learning,

$$\eta = \eta_0 \exp(-\text{epoch}\tau_\eta), \quad \sigma = \sigma_0 \exp(-\text{epoch}\tau_\sigma), \tag{10.11}$$

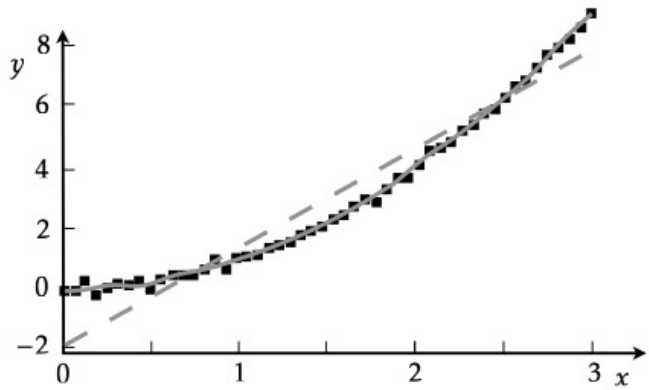


Figure 10.1: Principal component and principal manifold fitted to data described in Exercise 10.7. Shown is the principal component (gray dashed line) as well as the principal manifold (solid gray line).

with $\eta_0 = 0.1, \tau_\eta = 10^{-2}, \sigma_0 = 10$, and $\tau_\sigma = 0.05$. Here 'epoch' counts iterations divided by the number of patterns in the data set. The input data was preprocessed by dividing the components of the inputs by the largest value of that component found in the data set. The weights of the output neurons were initialised randomly, from a uniform distribution in $[0,1]$. The learning rule was iterated for ten epochs. We see that the self-organising map clusters the data into three distinct clusters in the output array. This data can be classified using a perceptron with one hidden layer with two sigmoid neurons, and an output layer with three sigmoid neurons and targets $t_i^{(\mu)} = \delta_{i\mu}$. The gray lines Figure 10.2 show the decision boundaries of the hidden neurons.

Answer (10.9) — The source for this answer is Ref. [161]. The starting point is Equation (10.23):

$$0 = \int \mathrm{d}\boldsymbol{r}_0 |\det \mathbb{J}| Q(\boldsymbol{r}_0) h(\boldsymbol{r} - \boldsymbol{r}_0) \big[\boldsymbol{w}^*(\boldsymbol{r}_0) - \boldsymbol{w}^*(\boldsymbol{r}) \big]. \tag{10.12}$$

Now expand the integrand in $\delta\boldsymbol{r} = \boldsymbol{r} - \boldsymbol{r}_0$ around $\boldsymbol{r}$. Following the same steps as outlined in Section 10.3, one finds

$$\int \mathrm{d}\delta\boldsymbol{r}\, \delta r_i \delta r_j h(\delta\boldsymbol{r}) \Big[\partial_i \boldsymbol{w}^* \partial_j (JD) + \tfrac{1}{2} JD \partial_i \partial_j \boldsymbol{w}^* \Big], \tag{10.13}$$

with $J = |\det \mathbb{J}(\boldsymbol{r})|$. Assuming that the neighbourhood function is isotropic

$$\int \mathrm{d}\delta\boldsymbol{r}\, \delta r_i \delta r_j h(\delta\boldsymbol{r}) = \sigma^2 \delta_{ij}, \tag{10.14}$$

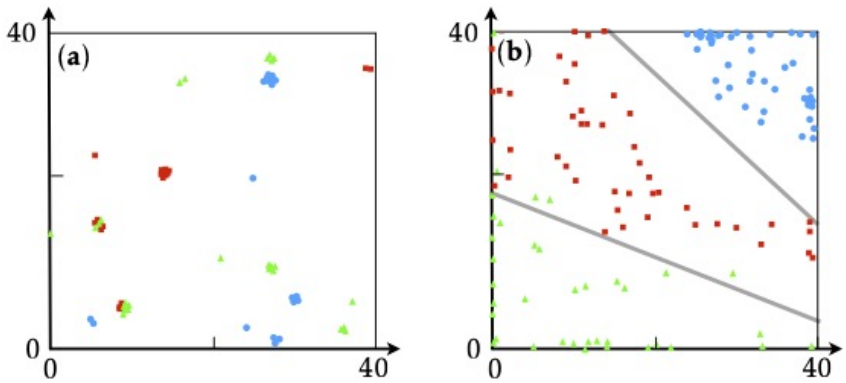


Figure 10.2: Classification of Iris data set (Figure 5.1) using a self-organising map with a 40×40 output array, and a perceptron with one hidden layer with two neurons and and output layer with three neurons. (**a**) Initial output the self-organising map, *iris setosa* (•), , *iris versicolor* (■), *iris virginica* (▲). (**b**) Final output of the self-organising map, and decision boundaries of the hidden neurons of the perceptron. Exercise 10.8.

Equation (10.13) is equivalent to

$$\partial_i \boldsymbol{w}^* \left(\frac{\partial_i Q}{Q} + \frac{\partial_i J}{J} \right) = -\tfrac{1}{2} \partial_i^2 \boldsymbol{w}^* . \tag{10.15}$$

First, if the distribution P_{data} factorises in a square domain, we assume that $w_1^*(r_1)$ and $w_2 *(r_2)$. With this ansatz, Equation (10.15) splits into two conditions

$$\partial_1 w_1^* \left(\frac{\partial_1 Q_1}{Q_1} + \frac{\partial_1 J_1}{J_1} \right) = -\tfrac{1}{2} \partial_1^2 w_1^* , \quad \partial_2 w_2^* \left(\frac{\partial_2 Q_2}{Q_2} + \frac{\partial_2 J_2}{J_2} \right) = -\tfrac{1}{2} \partial_2^2 w_2^* . \tag{10.16}$$

Using that $J_i = \partial_i w_i^*$, both Equations are equivalent to the one-dimensional Equation (10.29). We conclude that $\varrho = |\det \mathbb{J}|^{-1} \propto P_{\text{data}}^{3/2}$, just as in the one-dimensional case discussed in Section 10.3. So in this case, too, the weight density ϱ imitates the data distribution P_{data}, as anticipated by Kohonen [18]. But the weight density does not equal the data distribution [161]. One may speculate that the difference is a consequence of the difficulty of approximating the data distribution near its boundaries, but there is no general proof.

There is a special case where the two distributions are equal, namely when $\boldsymbol{w}^*$ is an analytic function of $\boldsymbol{r} = r_1 + i\, r_2$ [161]. In this case the r.h.s. of Equation (10.15) vanishes. As a consequence, it follows from Equation (10.15) that the distributions are equal, subject to normalisation: $\varrho = |\det \mathbb{J}|^{-1} \propto P_{\text{data}}$.

Answer (10.10) — This Exercise is solved in Ref. [2]. Figure 10.3 shows the XOR problem in the x_1-x_2 plane (left), and in the u_1-u_2 plane. This mapping is obtained by evaluating the radial basis functions given in the problem formulation

$$\boldsymbol{u}^{(1)} \approx \begin{bmatrix} 1 \\ 0.14 \end{bmatrix}, \quad \boldsymbol{u}^{(2)} \approx \begin{bmatrix} 0.37 \\ 0.37 \end{bmatrix}, \quad \boldsymbol{u}^{(3)} \approx \begin{bmatrix} 0.37 \\ 0.37 \end{bmatrix}, \quad \boldsymbol{u}^{(4)} \approx \begin{bmatrix} 0.14 \\ 1 \end{bmatrix}. \tag{10.17}$$

Note that $\boldsymbol{x}^{(2)}$ and $\boldsymbol{x}^{(3)}$ are mapped to the same point in the u_1-u_2-plane, so the mapping is not one-to-one. The problem is linearly separable in this plane.

Answer (10.11) — Figure 10.4 shows that the problem given in Table 10.1 is not linearly separable. Mapping input space as described in the problem, results in the problem shown on the right of Figure 10.4. In the new coordinates, the problem is linearly separable with $\boldsymbol{W} = [-1, -1]^\mathsf{T}$ and $\Theta = -1$.

Answer (10.12) — To do.

Answer (10.13) — The result of training the autoencoder from Figure 10.18 on the MNIST data set is shown in Figure 10.5. All neurons used sigmoid activations [Equation (6.19a)]. The model was trained for 100 epochs with learning rate $\eta = 0.1$, mini-batch size 128, using the Adam optimiser.[1]
Figure 10.5(**a**) demonstrates that the autoencoder creates a non-linear two-dimensional representation of the MNIST digits, much like the self-organising map (Figure 10.11). Like the self-organising map, the autoencoder tends to confuse the digits 4 and 9, and to some extent also 3 and 8. Figure 10.5(**b**) gives examples for input-output pairs. Panel (**c**) shows how to generate artificial digits: one samples a point from a cluster latent space, for example from the upper left-hand corner. Applying the decoder generates an artificial version of the digit 2, in this case.

[1] Kingma, D.P. & Ba, J., Adam: a method for stochastic optimization, arXiv:1412.6980.

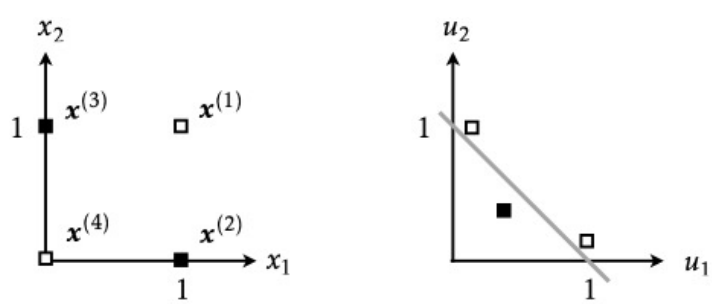


Figure 10.3: Left: input plane for Exercise **??**. Right: mapped problem in the u_1-u_2-plane and corresponding decision boundary for Exercise 10.10. After Fig. 5.2 in Ref. [2].

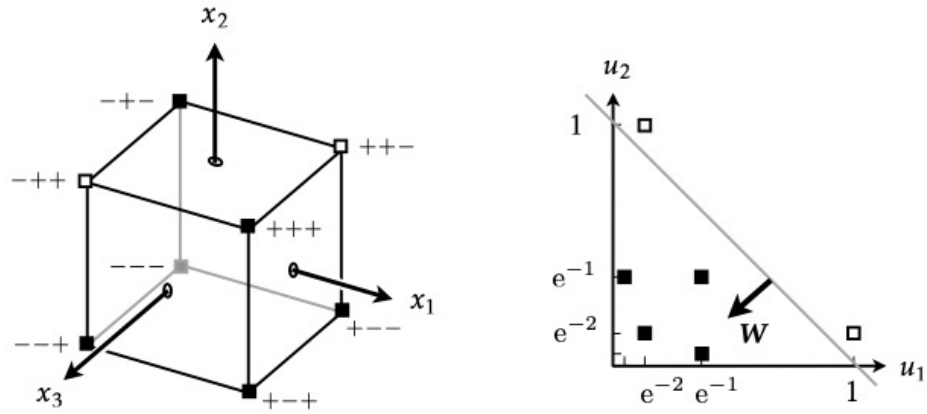


Figure 10.4: Left: input plane for Exercise **??**. Right: Right: mapped problem in the u_1-u_2-plane, weight vector $\boldsymbol{W}$, and corresponding decision boundary for Exercise 10.11.

Answer (10.14) — This solution follows Ref. [164]. Let $\mathbb{X}$ be the $N \times p$ matrix that has pattern vectors as its columns, $\mathbb{X} = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(p)}]$. For linear units with zero thresholds, the energy function of the autoencoder shown in Figure 10.19 is

$$H = \tfrac{1}{2}\left\|\mathbb{X} - \mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}}\mathbb{X}\right\|^2 \tag{10.20}$$

Here $\mathbb{W}_{\mathrm{e}}$ is the $2 \times N$ weight matrix of the encoder that maps patterns to the latent variables, $\boldsymbol{z}^{(\mu)} = \mathbb{W}_{\mathrm{e}}\boldsymbol{x}^{(\mu)}$, and $\mathbb{W}_{\mathrm{d}}$ is the $N \times 2$ matrix of the decoder. The matrix $\mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}}\mathbb{X}$ has rank two. So minimising H corresponds to finding the best rank-2 approximation $\mathbb{X}^{(2)} = \mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}}\mathbb{X}$ to $\mathbb{X}$. To determine $\mathbb{X}^{(2)}$, consider the singular-value decomposition of $\mathbb{X}$,

$$\mathbb{X} = \mathbb{U}\mathbb{S}\mathbb{V}^{\mathsf{T}}, \tag{10.21}$$

where $\mathbb{U}$ is an $N \times N$ orthogonal matrix, $\mathbb{V}$ is a $p \times p$ orthogonal matrix, and $\mathbb{S}$ is an $N \times p$ diagonal matrix. Its diagonal elements are the singular values $\Lambda_1 \geq \Lambda_2 \geq \cdots$ of $\mathbb{X}$. Then

$$\mathbb{X}^{(2)} = \mathbb{U}\mathbb{S}^{(2)}\mathbb{V}^{\mathsf{T}}, \tag{10.22}$$

where $\mathbb{S}^{(2)}$ is obtained from $\mathbb{S}$ by setting all singular values to zero except the largest two. Taking together Equations (10.21) and (10.22), $\mathbb{X}^{(2)} = \mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}}\mathbb{X}$ must satisfy $\mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}}\mathbb{U}\mathbb{X}\mathbb{V}^{\mathsf{T}} = \mathbb{U}\mathbb{S}^{(2)}\mathbb{V}^{\mathsf{T}}$. In other words, $\mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}} = \mathbb{U}\mathbb{S}^{(2)}\mathbb{V}^{\mathsf{T}}\mathbb{V}\mathbb{S}'\mathbb{U}^{\mathsf{T}}$, where $\mathbb{S}'$ is obtained from $\mathbb{S}$ by taking the reciprocal of all non-zero diagonal elements. This implies

$$\mathbb{W}_{\mathrm{d}}\mathbb{W}_{\mathrm{e}} = \mathbb{U}\mathbb{1}^{(2)}\mathbb{U}^{\mathsf{T}}, \tag{10.23}$$

where $\mathbb{1}^{(2)}$ is a diagonal matrix with entires equal to unity in the first two diagonal elements, and zero for all other matrix elements. We choose $\mathbb{W}_{\mathrm{d}} = \mathbb{U}\mathbb{1}^{(2)}$ and $\mathbb{W}_{\mathrm{e}} =$

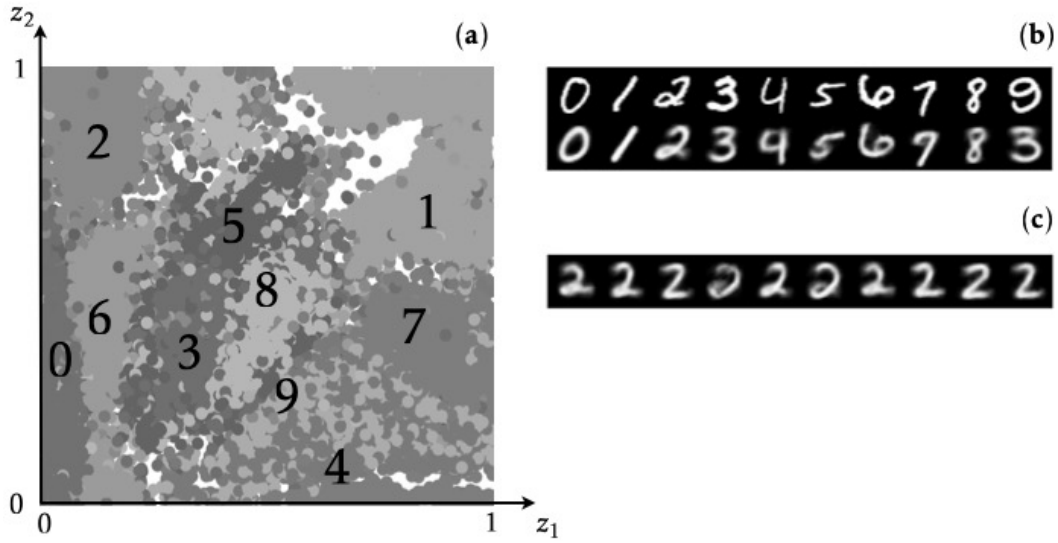


Figure 10.5: (**a**) Representation of the MNIST digits in the latent plane. (**b**) Examples of input-output pairs. The inputs are in the top row, the corresponding outputs in the bottom row. (**c**) Artificial digits generated by sampling points from a cluster in latent space (here: top left corner), and then applying the decoder. Exercise 10.13.

$\mathbb{1}^{(2)}\mathbb{U}^{\mathsf{T}}$. Then $\mathbb{Z} = \mathbb{W}_{\mathrm{e}}\mathbb{X}$ contains the coefficients of the data vectors along the two principal components of the input data (Section **6.3**). This can be seen as follows. The two principal components are the two eigenvectors of the correlation matrix $\mathbb{C} = \mathbb{X}\mathbb{X}^{\mathsf{T}} = \mathbb{U}\mathbb{S}^2\mathbb{U}^{\mathsf{T}}$ [Equation (**6.24**)] with the largest eigenvalues, the two first columns of $\mathbb{U}$.

The derivation summarised above assumes zero thresholds. The corresponding calculations for non-zero thresholds are described in Ref. [**164**].

11 Exercises in Chapter 11

Answer (11.1) — The gradient of H' w.r.t. w_{mn} evaluates to

$$\frac{\partial H'}{\partial w_{mn}} = \sum_{i\mu} (t_i^{(\mu)} - \langle y_i^{(\mu)} \rangle) \frac{\partial \langle y_i^{(\mu)} \rangle}{\partial w_{mn}} . \tag{11.1}$$

From Equation (3.7) we infer that $\langle y_i^{(\mu)} \rangle = \tanh(\beta\, b_i^{(\mu)})$. Using Equation (6.20) we find

$$\frac{\partial \langle y_i^{(\mu)} \rangle}{\partial w_{mn}} = \beta (1 - \langle y_i^{(\mu)} \rangle^2) \frac{\partial b_i^{(\mu)}}{\partial w_{mn}} . \tag{11.2}$$

The derivative of the local field is evaluated using $b_i^{(\mu)} = \sum_j w_{ij} x_j^{(\mu)}$. In summary,

$$\delta w'_{mn} \equiv -\eta \frac{\partial H'}{\partial w_{mn}} = \eta \sum_{\mu} (t_m^{(\mu)} - \langle y_m^{(\mu)} \rangle) \beta (1 - \langle y_m^{(\mu)} \rangle^2) x_n^{(\mu)} . \tag{11.3}$$

Comparing with the form of the learning rule stated in the Exercise, we find $\delta_m^{(\mu)} = (t_m^{(\mu)} - \langle y_m^{(\mu)} \rangle) \beta (1 - \langle y_m^{(\mu)} \rangle^2)$.
Now consider the average energy: $\langle H \rangle = \sum_{i\mu} (1 - t_i^{\mu} \langle y_i^{(\mu)} \rangle)$. To average, we used that $t_i^{(\mu)} = \pm 1$ and $y_i^{(\mu)} = \pm 1$. The gradient evaluates to

$$\frac{\partial \langle H \rangle}{\partial w_{mn}} = -\sum_{\mu} t_m^{(\mu)} \beta (1 - \langle y_m^{(\mu)} \rangle^2) x_n^{(\mu)} \tag{11.4}$$

The change in $\langle H \rangle$ under the learning rule (11.3) is given by

$$\sum_{mn} \frac{\partial \langle H \rangle}{\partial w_{mn}} \delta w'_{mn} . \tag{11.5}$$

Inserting Equations (11.3) and (11.4), we see that the resulting expression can be positive. The reason is that the result contains a double sum over pattern indices, as a consequence, different patterns can interfere to give a positive result.

Answer (11.2) — We start from Equation (11.5) for the average immediate reward given an input $\boldsymbol{x}$:

$$\frac{\partial \langle r \rangle}{\partial w_{mn}} = \sum_{y_1 = \pm 1, \ldots, y_M = \pm 1} \langle r(\boldsymbol{x}, \boldsymbol{y}) P(\boldsymbol{y} | \boldsymbol{x}) \rangle_{\text{reward}} = \sum_{y_1 = \pm 1, \ldots, y_M = \pm 1} \langle r(\boldsymbol{x}, \boldsymbol{y}) \frac{\partial}{\partial w_{mn}} P(\boldsymbol{y} | \boldsymbol{x}) \rangle_{\text{reward}} . \tag{11.6}$$

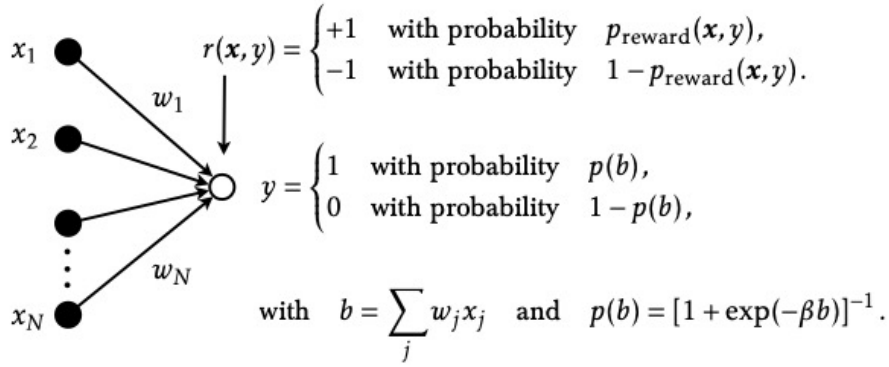


Figure 11.1: Stochastic binary neuron with 0/1 outputs. The learning rule is the same as in Exercise **??**. Exercise 11.3.

According to Equation (**11.6**),

$$P(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{M} \begin{cases} p(b_i) & \text{for} \quad y_i = 1\,, \\ 1-p(b_i) & \text{for} \quad y_i = -1\,, \end{cases} \tag{11.7}$$

with $p(b) = [1+\exp(-2\beta\, b)]^{-1}$ and local field $b_i = \sum_j w_{ij} x_j$. Now evaluate the derivative w.r.t. w_{mn} using the product rule:

$$\frac{\partial}{\partial\, w_{mn}} P(\boldsymbol{y}|\boldsymbol{x}) = \sum_i \left(\prod_{k\neq i} \begin{cases} p(b_k) & \text{for} \quad y_k = 1 \\ 1-p(b_k) & \text{for} \quad y_k = -1 \end{cases} \right) \begin{cases} p'(b_i)\delta_{im} x_n & \text{for} \quad y_i = 1\,, \\ -p'(b_i)\delta_{im} x_n & \text{for} \quad y_i = -1\,, \end{cases} \tag{11.8}$$

$$= \left(\prod_{k\neq m} \begin{cases} p(b_k) & \text{for} \quad y_k = 1 \\ 1-p(b_k) & \text{for} \quad y_k = -1 \end{cases} \right) \begin{cases} p'(b_m) x_n & \text{for} \quad y_m = 1\,, \\ -p'(b_m) x_n & \text{for} \quad y_m = -1\,, \end{cases}$$

$$= P(\boldsymbol{y}|\boldsymbol{x}) x_n \begin{cases} p'(b_m)/p(b_m) & \text{for} \quad y_m = 1\,, \\ -p'(b_m)/[1-p(b_m)] & \text{for} \quad y_m = -1\,. \end{cases} \tag{11.9}$$

Now use $p'/p = \beta(1-\tanh\beta\, b)$ and $-p'/(1-p) = -\beta(1+\tanh\beta\, b)$ to obtain

$$\frac{\partial}{\partial\, w_{mn}} P(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{y}|\boldsymbol{x})\beta[y_m - \tanh(\beta\, b_m)]x_n\,. \tag{11.10}$$

Multiplication with $r(\boldsymbol{x},\boldsymbol{y})$ and averaging over the stochastic outputs $\boldsymbol{y}$ and over the reward distribution gives Equation (**11.7**).

Answer (11.3) — Consider the binary stochastic neuron shown in Figure 11.1. The learning rule for the weights w_n is derived by maximising the average immediate

reward using gradient ascent. We start from Equation (11.5),

$$\frac{\partial \langle r \rangle}{\partial w_n} = \sum_{y=\pm 1} \langle r(\boldsymbol{x}, y) P(y|\boldsymbol{x}) \rangle_{\text{reward}} = \sum_{y=\pm 1} \langle r(\boldsymbol{x}, y) \frac{\partial}{\partial w_n} P(y|\boldsymbol{x}) \rangle_{\text{reward}}, \tag{11.11}$$

$$\langle r(\boldsymbol{x}, 1) \frac{\partial}{\partial w_n} p(b) + r(\boldsymbol{x}, 0) \frac{\partial}{\partial w_n} [1 - p(b)].$$

Using $\partial p(b)/\partial w_n = p'(b) x_n$ as well as Equation (6.20) gives

$$\frac{\partial \langle r \rangle}{\partial w_n} = \sum_{y=\pm 1} \langle r(\boldsymbol{x}, y) P(y|\boldsymbol{x}) \beta [y - p(b)] \rangle_{\text{reward}} x_n . \tag{11.12}$$

So the following learning rule increases the expected immediate reward,

$$\delta w_n = \alpha r [y - p(b)] x_n , \tag{11.13}$$

at least for small enough learning rate α. This is the analogue of Equation (11.9) for 0/1 neurons.

Answer (11.4) — The association task given in Table 11.1 cannot be solved using the associative reward penalty algorithm [185]. But if one embeds the four patterns into a four-dimensional input space – so that the patterns become linearly independent – then the algorithm finds the optimal strategy (Figure 11.2).

Answer (11.5) — Figure 11.3 shows the reward obtained from Q-learning as a function of the number of training episodes for three different values of the parameter ε defining the ε-greedy policy (page 208). When $\varepsilon = 0$, the action with the highest Q-value for a given state is picked. As a consequence, the algorithm arrests in a local reward maximum, training fails. When $\varepsilon > 0$, the algorithm explores presently sub-optimal state-action pairs that may lead to higher rewards in the long run (exploit-versus-explore dilemma, page 206). Figure 11.3 indicates that convergence is faster for $\varepsilon = 0.1$ compared with $\varepsilon = 0.01$. As you increase ε, when does the algorithm start to fail?

Answer (11.6) — Figure 11.4(**a**) shows the decision tree for your opponent. We want to use Q-learning to find the optimal strategy to compete against this player. The question is how this strategy depends on p and q. Figure 11.4(**b**) shows your average reward when you compete against this opponent. It is calculated as

$$\langle R \rangle = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} r_k \tag{11.14}$$

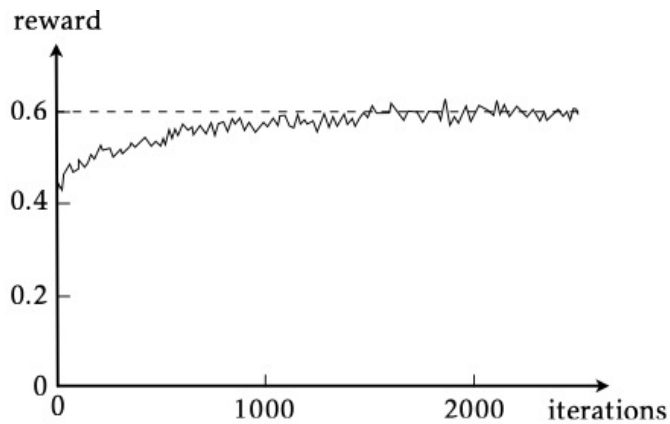


Figure 11.2: Associative reward-penalty algorithm for the XOR problem. The dashed line shows the maximal expected reward $r_{\max} = 0.6$, obtained from Table . The solid line shows the average reward as a function of the number of iterations of Equation (**11.10**) (how averaged?), obtained after embedding the four patterns from Table **11.1**, in four-dimensional input space. Parameters: $\delta = 0.01$, $\alpha = 0.01$. Exercise 11.4.

Define r_k. The average in Equation (11.14) is over many ($k_{\max}$) episodes. Each episode consists of only one iteration, a single round of the game. In other words, $T = 1$. Therefore the Q-learning rule (**11.24**) simplifies to (**11.26**) Moreover, $\boldsymbol{s}$ and $\boldsymbol{a}$ in Equation (11.14) are states and actions. The actions are to play rock, paper, or scissors. The states are listed in Table 11.1. Each state contains the move of your opponent in the previous round, and whether he won, drew, or lost that round
We infer from Figure 11.4(**b**) that $\langle R \rangle = 1$ when $p = q = 0$, where the opponent plays always rock. In this case, one learns to win all games by playing paper all the time. When $p = 0$ and $q = \frac{1}{3}$, the opponent chooses rock, paper, scissors with equal probabilities. There is nothing to learn, we expect to receive $+1, 0, -1$ with probability $\frac{1}{2}$. So the reward averages to zero.
For $p = 0$, the opponent's choice does not depend on history. If we average over many games, your average reward evaluates to

$$\begin{cases} 0 & \text{for } a = \text{rock}\,, \\ 1-3q & \text{for } a = \text{paper}\ , \\ 3q-1 & \text{for } a = \text{scissors}\ . \end{cases} \tag{11.15}$$

For $0 \le q < \frac{1}{3}$, it is advantageous for you to play always paper. In this case Eq. (11.14) evaluates to $\langle R \rangle = 1 - 3q$.
When $p > 0$, the opponent's choice depends on the outcome of the previous game. This is accounted for by defining the states to include your opponent's last move

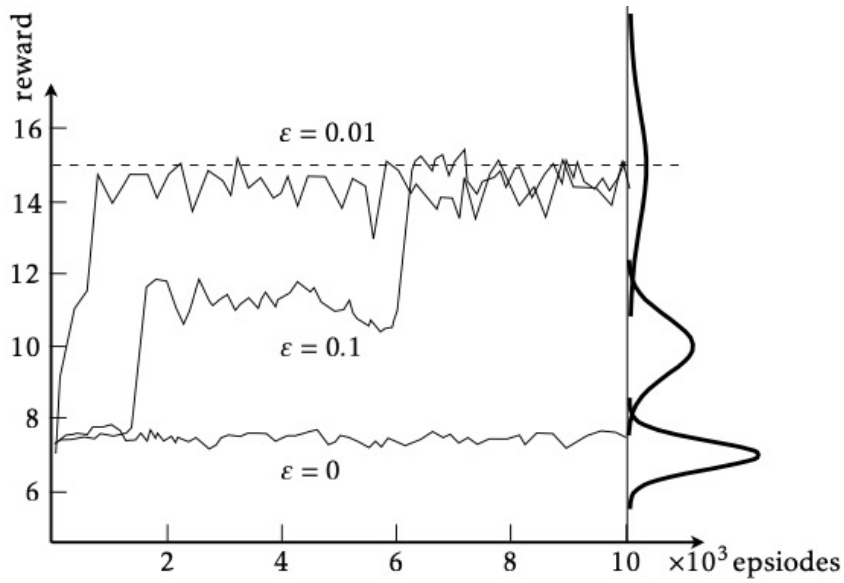


Figure 11.3: Three-armed bandit problem. The dashed line shows the maximal expected reward, $r_{\max} = 15$. The thick solid lines show the reward distributions from Figure **11.8**. The thin solid lines show the expected reward versus the number of episodes for three different values of ε defining the ε-greedy policy (page **208**). Schematic, using simulation results of Navid Mousavi. Exercise 11.5.

and whether he won, drew, or lost. Table 11.1 shows your average reward for playing rock, paper, and scissors, following the decision tree shown in Figure11.4(**a**). Each row corresponds to different outcome of the previous round, whether the opponent played rock, paper, or scissors, and whether he won, drew, or lost. For fixed $p > 0$, Table 11.1 shows that there are two critical values of q where the optimal action changes for a given state:

$$q_1^{(c)} = \frac{3p-1}{3(p-1)} \quad \text{and} \quad q_2^{(c)} = \frac{3p-2}{6(p-1)}. \tag{11.16}$$

The average reward (11.14) changes at these lines, as seen in Figure 11.4(**b**).

Answer (11.7) — Table 11.2 shows instructions for an 'expert' tic-tac-toe player [**191**]. The left column contains an ordered list of actions. At every step, one should take the action that is highest up in the column. The second column shows an example for a configuration of the playing board that calls for the action to the left. The third column shows the result of Q learning, giving the Q-matrices for each configuration. We see that the Q-matrices are consistent with the expert model: for each configuration, the Q element corresponding to the action proposed by Crowley & Siegler [**191**] is largest.

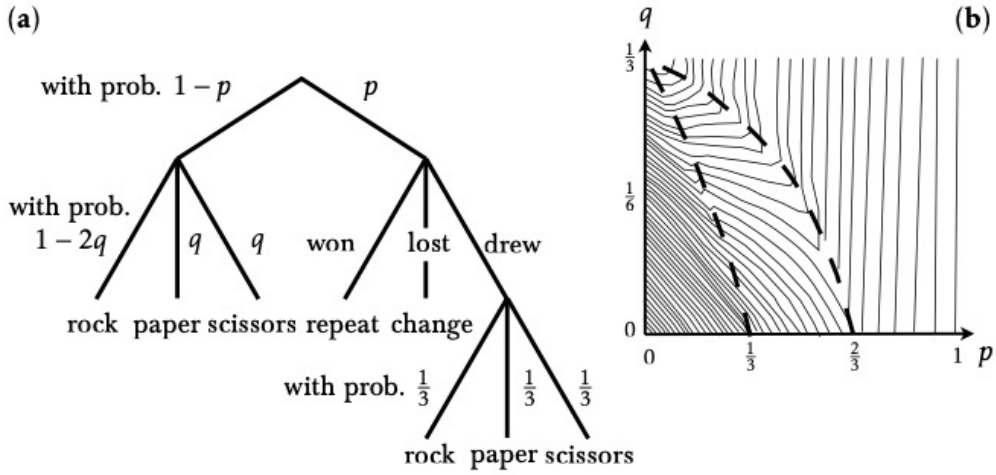


Figure 11.4: Rock-paper-scissors. (**a**) Decision tree of the opponent, as described in the problem formulation. (**b**) Shows contour lines of the average reward (11.14) as a function of p and q. Also shown are the critical values of the parameter q given by Eq. (11.16), dashed lines. The numerical results were obtained using Q-learning, letting a player compete with an opponent acting as described in the decision tree. Exercise 11.6.

Comparing the result of the Q-learning algorithm with the rules proposed by Crowley and Siegler [**191**], we see that the results are consistent, with two exceptions. Rewrite the rest, and modify Table 11.2 as needed. First, the Q-learning algorithm does not prioritise playing an 'opposite corner'. The move 'empty side' has the same Q-value. Second, no configurations we encountered where the algorithm prioritises to place a piece on an 'empty side'. Check modified scheme on p. 62 of A. Newell and H. A. Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs. J. J. (1972). One or two sentences about the numerical Q-values. Converged, but not equal to exact reward. Example: negative value of "play empty corner". What should the value be? Discuss changing ε. Comment on $\varepsilon = 1$. Why no difference to complete enumeration?

Answer (11.8) — Results for three different reward functions are plotted in Figure 11.5. Shown is how the probability for a game to end in a draw changes during training, as a function of the number of rounds of the game played. The reward function R_1 (see Figure caption) is the same as in Figure **11.7**. The fraction of games drawn increases in a way similar to Figure **11.7**. The results for the reward function R_2 are very similar, for expert players all games end in a draw. The reward function R_3 yields a different result, player 1 always wins. This is expected, because the reward is the same for 'draw' or 'lose'. Therefore the second player cannot learn to prevent

Table 11.1: Your average reward for playing rock, paper, or scissors. The average reward depends on the outcome of the previous game, whether your opponent played rock, paper, or scissors, and whether he won, lost, or drew. Exercise **??**.

		$\langle R\rangle_{\text{rock}}$	$\langle R\rangle_{\text{paper}}$	$\langle R\rangle_{\text{scissors}}$
rock	won	0	$1+(p-1)3q$	$(1-p)3q-1$
paper	won	$-p$	$(1-p)(1-3q)$	$(1-p)(3q-1)+p$
scissors	won	p	$(1-p)(1-3q)-p$	$(1-p)(3q-1)$
rock	lost	0	$(1-p)(1-3q)-\frac{p}{2}$	$(1-p)(3q-1)+\frac{p}{2}$
paper	lost	$\frac{p}{2}$	$(1-p)(1-3q)$	$(1-p)(3q-1)-\frac{p}{2}$
scissors	lost	$-\frac{p}{2}$	$(1-p)(1-3q)+\frac{p}{2}$	$(1-p)(3q-1)$
rock	drew	0	$(1-p)(1-3q)$	$(1-p)(3q-1)$
paper	drew	0	$(1-p)(1-3q)$	$(1-p)(3q-1)$
scissors	drew	0	$(1-p)(1-3q)$	$(1-p)(3q-1)$

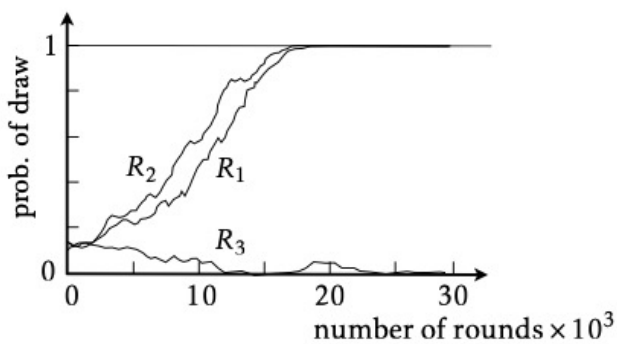


Figure 11.5: Learning to play tic-tac-toe using Q-learning, for three different reward functions, R_1: $r=1$ (win), $r=0$ (draw), $r=-1$ (lose); R_2: $r=2$ (win), $r=0$ (draw), $r=-1$ (lose); R_3: $r=1$ (win), $r=-1$ (draw, lose). Exercise 11.8.

the first one from winning. The first player learns to always win (it has the advantage of placing the first piece).

Answer (11.9) — Section **11.3** describes how two players can learn to play tic-tac-toe by means of the Q-learning algorithm. In the same way, two players can learn to play *connect four* (Figure **11.9**). An important difference though is that there are many more states to consider in connect four on a 6×6 playing field which has 3^{36} board configurations, as opposed to 3^9 for tic-tac-toe. But note that not all board configurations are allowed states. For tic-tac-toe, for example, the algorithm

Table 11.2: Table 1 from Ref. [191], summarising how to how to play tic-tac-toe as successfully as possible. At each round, one should make the move that comes first from the top. Also shown are board configurations for each of the moves, as well as Q-tables obtained by Q-learning.

action	board	Q-table
Win (x)	x x - - o - o - -	— — 1 0.62 — 0.8 — 0.8 0.8
Block (o)	x - - - o - x - -	— -0.054 -0.002 0 — -0.084 — -0.066 -0.092
Fork (x)	x - - o x - - - o	— 1 1 — — 0.970 0.957 0.986 —
Block fork (o) (get two in a row)	- x - x - - o - -	0.948 — 0.996 — 0.98 0.972 — 0.979 1
Block fork (o) (can't get two in a row)	o x - x - - - - -	— — -0.025 — 0 0 -0.029 0 -0.029
Play center (o)	x - - - - - - - -	— -0.002 -0.005 -0.027 0 -0.016 -0.012 -0.013 -0.034
Play opposite corner (o)	x - - - o - - x -	— -0.044 -0.014 0 — 0 0 — 0
Play empty corner (o)	- - - - x - - - -	0 -0.003 0 -0.004 — -0.005 0 -0.002 0
Play empty side (o)		

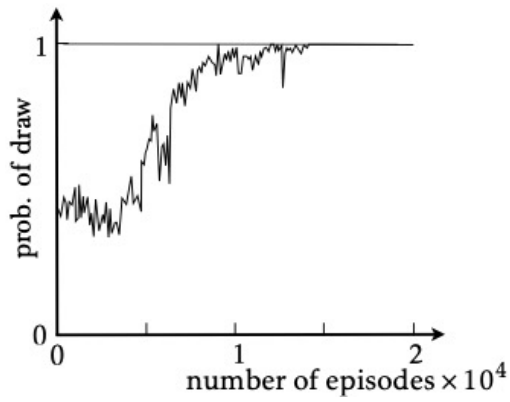


Figure 11.6: Q-learning for connect four on a 6×6 playing field. Exercise 11.9.

encounters between 2500 and 3000 states for each player. For example, the board

x	x	x
o	o	o
-	-	x

(11.17)

is not encountered. As a consequence, the Q-learning algorithm takes much longer to exploer the connect-four state space, compared with tic-tac-toe. There are too many states to explore when learning to play connect four. Figure 10.5 shows how two players learn to play connect four by competing each other, using Q-learning. We see that the game always ends in a draw for expert players, confirming the result of Ref. [196]. There it is also explained that the situation is different when the playing field has 7 columns. In this case the first player can learn to always win if she starts in the center column. If the first player begins in any other column instead, then the second player can learn to force a draw.
The Q-learning algorithm as described in Section 11.3 becomes quite inefficient for the 6×7 playing field, because there are even more states to consider. In this case it may be more efficient to approximate the Q-function by a neural network (Section 11.4).

Answer (11.10) — The expected stock of chocolate at time t, $\langle s_t \rangle$, obeys the recursion:

$$\langle s_{t+1} \rangle = \begin{cases} (\langle s_t \rangle + 1)(1-p) & \text{save} \\ 0 & \text{eat} \end{cases} \tag{11.18}$$

The expected daily reward, $\langle r_t \rangle$, is given by $\langle r_{t+1} \rangle = \langle s_t \rangle + 1$ if you save the chocolate, but $\langle r_{t+1} \rangle = 2(\langle s_t \rangle + 1)$ when you eat it. Now consider the expected future reward

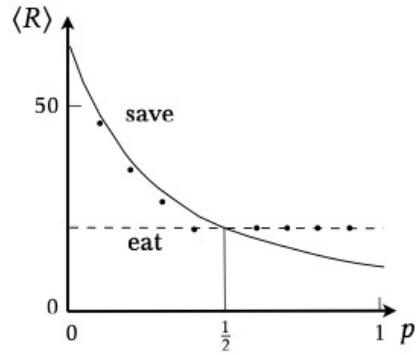


Figure 11.7: Eat or save the chocolate? Expected reward obtained by by Q learning for $T = 10$ days (•). Also shown is the expected reward for two strategies: to eat the chocolate immediately (Equation (11.19), 'eat'), and to wait and eat on the last day (Equation (11.22), 'save'). The optimal strategy changes for $p = \frac{1}{2}$ (vertical line). Exercise 11.10.

over a period of T days, $\langle R_T \rangle = \sum_{t=1}^{T} \langle r_t \rangle$. Let us compare two strategies. The first one is to eat the chocolate every day. In this case

$$\langle R_T \rangle_{\text{eat}} = 2T\,. \tag{11.19}$$

The second strategy is to save the chocolate for $T-1$ days, and to eat what's left on day T. The reward for this stragegy is

$$\langle R_T \rangle_{\text{save}} = \left[\sum_{t=1}^{T-1} \langle s_{t-1} \rangle + 1 \right] + 2\big(\langle s_{T-1} \rangle + 1\big). \tag{11.20}$$

To evaluate this expression further, one needs the solution of the recursion (11.18):

$$\langle s_t \rangle = \frac{1}{p}(1-p)\big[1-(1-p)^t\big]. \tag{11.21}$$

This yields:

$$\langle R_T \rangle_{\text{save}} = \frac{T}{p} + \frac{2p-1}{p^2}\big[1-(1-p)^T\big]. \tag{11.22}$$

Figure 11.7 shows $\langle R_T \rangle_{\text{save}}$ and $\langle R_T \rangle_{\text{eat}}$ as a function of p for $T = 10$. We conclude that it is better to save if $p < \frac{1}{2}$. For $p > \frac{1}{2}$ it is better to eat. It turns out that these are the optimal strategies. Also shown are results of Q-learning. For $p < \frac{1}{2}$, the results are slightly below $\langle R_T \rangle_{\text{save}}$. The problem is that the learning algorithm converges slowly when p approaches $\frac{1}{2}$ from below. As p increases, the fluctuations of the total reward increase