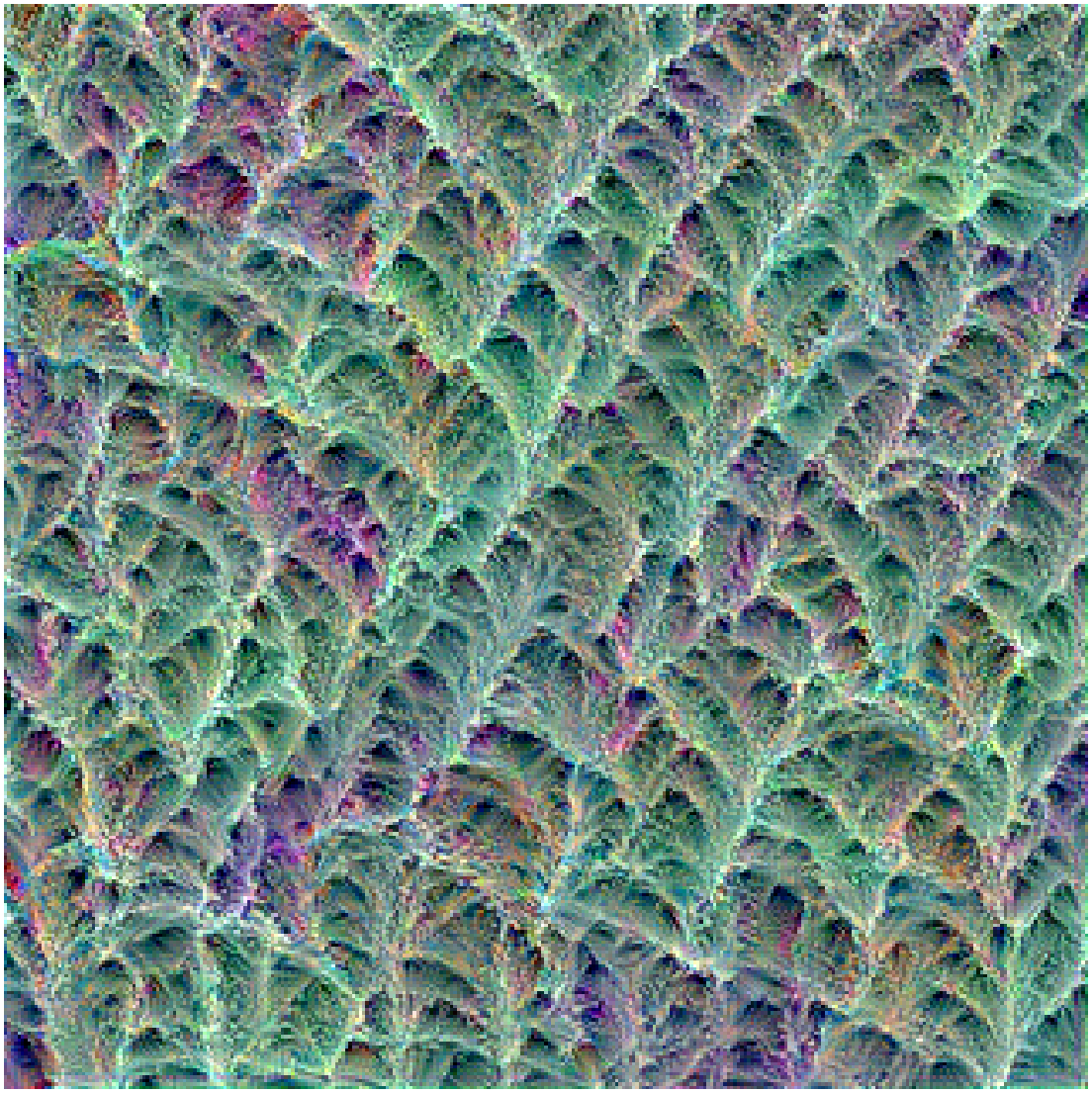


Exercise solutions

Machine learning with neural networks
An introduction for scientists and engineers

BERNHARD MEHLIG

Department of Physics
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden 2023

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

BERNHARD MEHLIG

Department of Physics
University of Gothenburg
Göteborg, Sweden 2023

1 Introduction

This document contains solutions for the exercises in *Machine learning with neural networks. An Introduction for scientists and engineers* (Cambridge University Press, 2021). Students, 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, Johan Gustafsson, Hampus Linander, Vitalii Iarko, Robert Mehlig, Jan Meibohm, Navid Mousavi, Marina Rafajlovic, Jan Schiffeler, Ludvig Storm, Linus Sundberg, Arvid Wenzel Wartenberg, and Erik Werner for their contributions. I would also like to thank John Segerstedt who took the initiative to collect 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* are typeset in red. A list of *errata* for this book is available from Cambridge University press at www.cambridge.org/9781108494939. Please let me know if you find further errors or inaccuracies in the book, or in the present document.

Gothenburg, January 13 (2023) Bernhard Mehlig

2 Solutions for 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

$$\sum_{j=1}^{N} w_{ij} x_j = \sum_{j=1}^{N} \tfrac{1}{N}(x_i x_j - \delta_{ij}) x_j = (1 - \tfrac{1}{N}) x_i \,, \tag{2.1}$$

using that $x_j = \pm 1$, and therefore $\sum_{j=1}^{N} x_j^2 = N$. Then apply the signum function. This gives $\mathrm{sgn}[(1-\frac{1}{N})x_i] = x_i$ for $N > 1$. It follows that under this condition, the network dynamics reproduces $\boldsymbol{x}$. 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 $\mathbb{W} = \frac{1}{N}\sum_{\mu}(\boldsymbol{x}^{(\mu)} \boldsymbol{x}^{(\mu)\mathsf{T}} - \mathbb{I})$, where $\mathbb{I}$ is the unit matrix with elements δ_{ij}. For this weight matrix, we find that $\mathbb{W}\boldsymbol{x}^{(\nu)} = (1-\frac{1}{N})\boldsymbol{x}^{(\nu)}$ for orthogonal patterns. This implies that the cross-talk term vanishes for $N > 1$.

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:

$$C_i^{(\nu)} = -x_i^{(\nu)} \frac{1}{N} \sum_{j=1}^{N} \sum_{\mu\neq\nu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)} \,. \tag{2.2}$$

Now average over the independent patterns, using that $\langle x_i^{(\nu)} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)} \rangle = 0$ when $i \neq j$ and $\mu \neq \nu$, because the average factorises in this case, and $\langle x_k^{(\mu)} \rangle = 0$. When $i = j$, there are $p-1$ terms that average to $\langle [x_j^{(\nu)}]^2 [x_j^{(\mu)}]^2 \rangle = 1$. We conclude that $\langle C_i^{(\nu)} \rangle = -(p-1)/N \approx -p/N$ 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, by contrast, the mean $\langle C \rangle$ cannot be neglected. In the limit $\alpha \to \infty$, the mean of the weight matrix, $\langle \mathbb{W} \rangle = \frac{p}{N}\mathbb{I}$, 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.

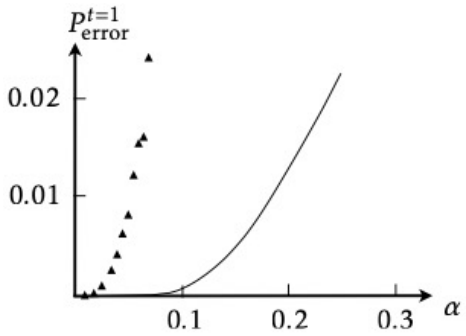


Figure 2.1: Numerical results for the one-step error probability (▲) for the update rule (**2.5**), for mixed states of the form (**2.52**) with $+++$. For comparison, the one-step error probability (**2.53**) for a stored pattern is also shown (solid line). Schematic, after simulations by Robert Mehlig. Exercise 2.5.

Answer (2.4) — Consider $\text{sgn}(\pm x_i^{(1)} \pm x_i^{(2)})$. For $x_i^{(\nu)} = \pm 1$, the argument of the signum function may evaluate to zero, but sgn(0) is not defined. Therefore such states cannot be recognised. The same is true for superpositions of any even number of stored patterns. As a consequence one considers only odd superpositions. See also Exercises 2.5 and 3.1.
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 (**2.52**) with $+++$. We see that the error probability is much larger for this state than the one-step error probability (**2.39**) for a stored pattern. But just like Equation (**2.39**), the error probability tends to zero as $\alpha \to 0$. To explain this observation, we show that the condition

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

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**). The goal is to determine whether the first term on the r.h.s. reproduces $x_i^{(\text{mix})}$. For large N, the sum over j on the r.h.s. of Equation (2.4) is an average over the quantity $\sigma_\mu = x_j^{(\mu)} x_j^{(\text{mix})}$.

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

$x_j^{(1)}$	$x_j^{(2)}$	$x_j^{(3)}$	$x_j^{(\mathrm{mix})}$	σ_1	σ_2	σ_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

Table 2.1 lists all possible combinations of bits of pattern j and the corresponding values of σ_μ. Using that $\mathrm{Prob}(x_i^{(\nu)} = \pm 1) = \frac{1}{2}$ [Equation (**2.29**)], we see that $\langle \sigma_\mu \rangle = \frac{1}{2}$. This implies

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

Taking the signum-function, we conclude that $\boldsymbol{x}^{(\mathrm{mix})}$ is reproduced. This demonstrates that the one-step error probability for the mixed state tends to zero as $\alpha \to 0$ (Figure 2.1), for deterministic updates. Exercise 3.1 concerns the effect of noise on the stability of mixed states.

Answer (2.6) — We have that $s_2' = \mathrm{sgn}(w_{21} s_1) = -\mathrm{sgn}(s_1) = -s_1$ because $w_{21} = -1$. It follows that $H' - H = -\frac{1}{2}(w_{21} + w_{12})(s_1 s_2' - s_1 s_2) = \frac{1}{2}(w_{21} + w_{12}) s_1 (s_1 + s_2)$. 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. This shows that H can increase under the asynchronous network dynamics if the weights are not symmetric.

Answer (2.7) — Assume that the second-order weights $w_{ij}^{(2)}$ are symmetric, and that the diagonal weights $w_{ii}^{(2)}$ are zero. Further, assume that the third-order weights $w_{ijk}^{(3)}$ are symmetric,

$$w_{ijk}^{(3)} = w_{jik}^{(3)} = w_{ikj}^{(3)} = w_{kji}^{(3)} , \tag{2.6a}$$

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

$$w_{iik}^{(3)} = w_{iji}^{(3)} = w_{ijj}^{(3)} = 0 . \tag{2.6b}$$

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

$$s'_m = \mathrm{sgn}(b_m) \quad \text{with} \quad b_m = \sum_j w^{(2)}_{mj} s_j + \frac{1}{2}\sum_{jk} w^{(3)}_{mjk} s_j s_k \,. \tag{2.7}$$

A calculation analogous to the one 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

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

For the thresholds, take

$$\mu_i = \tfrac{1}{2}\sum_j w_{ij}\,. \tag{2.8b}$$

These values of the thresholds ensure that the local fields in the s- and n-representation are the same, up to a factor of two [assuming $\theta_i = 0$ in Equation (**1.5**)].
The first task is to show that this rule works for one stored pattern. The calculation is analogous to Equations (**2.10**) and (**2.11**):

$$\sum_j w_{ij} n_j - \mu_i = \sum_j w_{ij}(n_j - \tfrac{1}{2}) = \frac{1}{2}\sum_j w_{ij}(2n_j - 1) \tag{2.9}$$

$$= \frac{1}{2N}\sum_{j\neq i}(2n_i - 1)(2n_j - 1)^2 = (2n_i - 1)\frac{1}{N}\sum_{j\neq i}(2n_j - 1)^2\,. \tag{2.10}$$

Since the right factor on the r.h.s. of Eq. (2.10) is positive, one finds:

$$n'_i = \theta_{\mathrm{H}}(2n_i - 1)\,. \tag{2.11}$$

Since $\theta_{\mathrm{H}}(2n_i - 1) = n_i$, we conclude that n_i remains unchanged. So the pattern is recognised.
The answer to the second part of the question is obtained by a calculation analogous to the argument leading from Equation (**2.45**) 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

$$H' - H = -\tfrac{1}{2}\sum_{j\neq m}(w_{mj} + w_{jm})(n'_m n_j - n_m n_j) + \mu_m(n'_m - n_m)\,. \tag{2.12}$$

Since the weights are symmetric, the first term can be evaluated further,

$$H' - H = -(n'_m - n_m)\Big(\sum_{j\neq m} w_{mj} n_j - \mu_m\Big)\,. \tag{2.13}$$

Now evaluate $\mathrm{sgn}(H'-H)$. If $n_m = 1$ then $n'_m = 0$. As a consequence, the factor $\left(\sum_{j\neq m} w_{mj} n_j - \mu_m\right)$ is negative, so that $H'-H<0$. If $n_m = 0$, then $n'_m = 1$. In this case the factor $\left(\sum_{j\neq m} w_{mj} n_j - \mu_m\right)$ is positive, so that again $H'-H<0$.

Answer (2.9) — Consider the effect of one update with 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 changes, $s'_i = -s_i$. The corresponding change in the energy function (**2.44**) reads

$$H'-H = 2\sum_{\substack{i\in\mathscr{S}\\ j\in\mathscr{S}^{\mathrm{c}}}} w_{ij} s_i s_j \,. \tag{2.14}$$

Here it is assumed that the weights are symmetric, and that the diagonal weights vanish. Equation (2.14) simplifies to Equation (**2.48**) when $\mathscr{S}^{\mathrm{c}}$ contains only one neuron, number m. Now consider the case $N>2$ and assume that all neurons except the first one change state,[1] that is $\mathscr{S}=\{1\}$ and $\mathscr{S}^{\mathrm{c}}=\{2,\ldots,N\}$. The corresponding change in H is

$$H'-H = 2\sum_{j=2}^{N} w_{1j} s_1 s_j = 2\sum_{j=1}^{N} w_{1j} s_1 s_j = 2 s_1 b_1 \,. \tag{2.15}$$

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 how H changes 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 $\frac{\mathrm{d}}{\mathrm{d}t}E \leq 0$. Differentiating E w.r.t time t yields

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -\sum_{ij} w_{ij}\frac{\mathrm{d}b_i}{\mathrm{d}t} g'(b_i) g(b_j) + \sum_i \theta_i \frac{\mathrm{d}b_i}{\mathrm{d}t} g'(b_i) + \sum_i \frac{\mathrm{d}b_i}{\mathrm{d}t} b_i g'(b_i) \tag{2.16}$$

$$= \sum_i \frac{\mathrm{d}b_i}{\mathrm{d}t} g'(b_i)\Big[-\sum_j w_{ij} g(b_j) + \theta_i + b_i\Big], \tag{2.17}$$

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

$$\tau\frac{\mathrm{d}b_i}{\mathrm{d}t} = \tau\sum_j w_{ij}\frac{\mathrm{d}n_i}{\mathrm{d}t} = -b_i - \theta_i + \sum_j w_{ij} g(b_j). \tag{2.18}$$

[1] 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 $N>2$.

Combining Equations (2.16) and (2.18) yields

$$\frac{\mathrm{d}E}{\mathrm{d}t} = -\frac{1}{\tau}\sum_i g'(b_i)\Big[b_i + \theta_i - \sum_j w_{ij} g(b_j)\Big]^2 . \tag{2.19}$$

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 9.2 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

$$\mathbb{W} = \frac{1}{4}\begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix} . \tag{2.20}$$

Now evaluate $\boldsymbol{s}(t=1)$. 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

$$Q_{\mu\nu} = \frac{1}{N}\sum_{j=1}^{N} x_j^{(\mu)} x_j^{(\nu)} , \tag{2.21}$$

as in Equation (**3.50**). 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

$$b_i^{(\nu)} = \sum_j w_{ij} x^{(\nu)} = x_i^{(1)} Q_{1\nu} + x_i^{(2)} Q_{2\nu} . \tag{2.22}$$

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	$-\frac{6}{32}$

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

$$\mathbb{W} = \frac{4}{3}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}. \tag{2.23}$$

Since the weight matrix is proportional to the unit matrix, all patterns are reproduced, not only the stored ones. Therefore the network cannot single out the XOR patterns. The network fails in this way 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

$$H = -2N(\tfrac{1}{2} - d)^2. \tag{2.24}$$

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. In other words, the distance does not account for the fact that the inverted pattern $-\boldsymbol{x}^{(1)}$ is closely related to $\boldsymbol{x}^{(1)}$.

3 Solutions for exercises in Chapter 3

Answer (3.1) — We start from the mean-field equation (3.21) for the mixed order parameter (3.19). Assume that only the first n components ($\mu = 1,\ldots,n$) are non-zero, and that they are all equal to $m > 0$. Defining $z_i^{(n)} = \sum_{\mu=1}^{n} x_i^{(\mu)}$, Equation (3.21) can be written as

$$m = \tfrac{1}{n}\langle z_i^{(n)} \tanh(\beta\, m\, z_i^{(n)})\rangle\,. \tag{3.1}$$

Here $\langle\cdots\rangle$ is an average over random bits $x_i^{(\mu)} = \pm 1$ with equal probability, for $\mu = 1,\ldots,n$. For $n = 1$, this mean-field equation is equivalent to Equation (3.14).
Figure 3.1(**a**) shows the solution of (3.1) for $n = 1$ and $n = 3$ as a function of the noise level β^{-1}. The curve for $n = 1$ is the same as that in Figure 3.3. The order parameter for $n = 3$ is non-zero below a critical noise level given by $\beta_c = 1$, and it is smaller than the $n=1$-order parameter in this range. The $n=3$-order parameter m approaches $\frac{1}{2}$ as the noise level tends to zero. This follows from the solution to Exercise 2.5: Equation (3.1) implies that m approaches $\frac{1}{3}\sum_{\mu=1}^{3}\langle\sigma_\mu\rangle = \frac{1}{2}$. In this limit, Equation (3.20) shows that $\langle s_i\rangle$ tends to the mixed state (2.52), with all plus signs,

$$\langle s_i\rangle \to \mathrm{sgn}(x_i^{(1)} + \ldots + x_i^{(n)})\,. \tag{3.2}$$

Now consider the network dynamics. For a given noise level, we iterate the McCulloch-Pitts dynamics starting at $s_i(t=0) = \mathrm{sgn}(x_i^{(1)} + x_i^{(2)} + x_i^{(3)})$, and evaluate the components of the order parameter (3.19). The results are shown in Figure 3.1(**b**)-(**d**), for different values of the noise level β^{-1}. The order-parameter components are plotted versus the iteration number, as in Figure 3.2. Also shown is the solution of Equation (3.1).
For small noise levels, the simulations appear to converge to steady-state values that agree well with the mean-field theory. In particular, the non-zero components take the same value, determined by Equation (3.1) [panel (**b**)]. Simulations at smaller values of N show, however, that the mixed states are metastable, even at small noise levels. As the iteration number tends to ∞, the network dynamics explores a single pattern. Also this state is metastable, as noted on pages 39 and 40.
For larger noise levels [Figure 3.1(**c**) and (**d**)], the non-zero order-parameter components immediately drift away from the mean-field prediction. This happens because the $n=3$-mean-field solution bifurcates at $\beta^{-1} = 4.6$, it becomes unstable [1,33]. As outlined in these references, this conclusion follows from the properties of the *free energy* of the Hopfield model,

$$F(\beta) = -\frac{1}{\beta}\langle \log Z\rangle \quad \text{with} \quad Z = \sum_{\boldsymbol{S}} \mathrm{e}^{-\beta H(\boldsymbol{S})}\,. \tag{3.3}$$

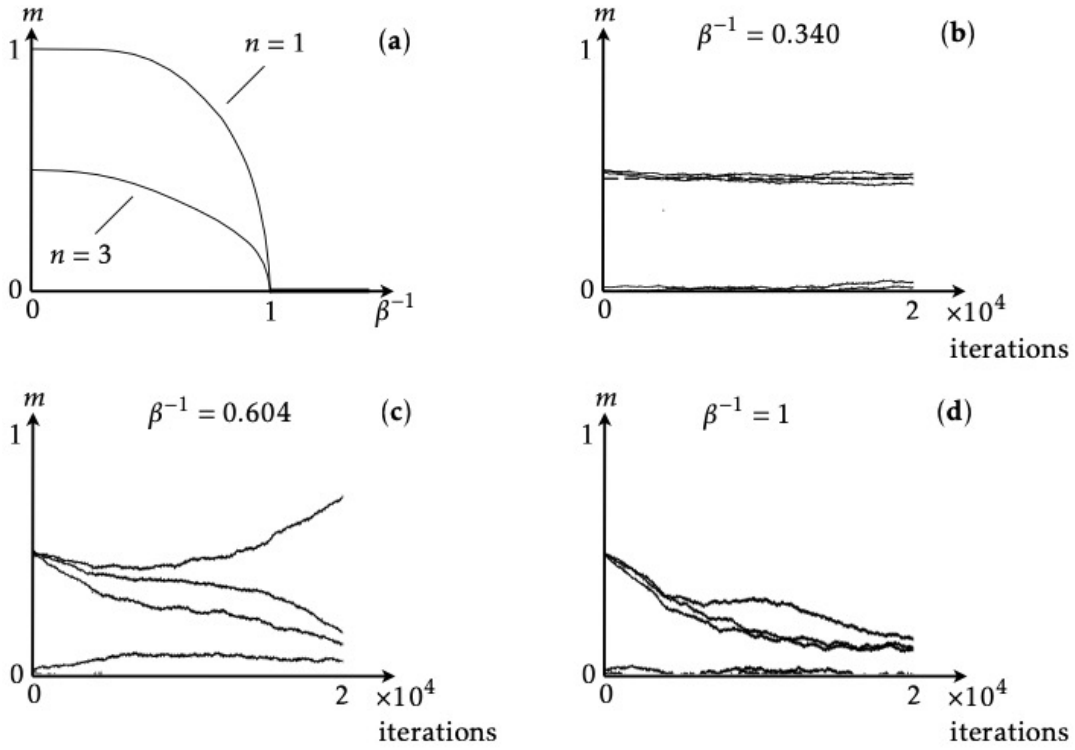


Figure 3.1: (**a**) Numerical solution of the mean-field equation (3.1) as a function of the noise level β^{-1} for $n = 1$ and 3. (**b**) Numerical result for order parameters (**3.19**), for $n = 3$, $p = 5$, $N = 4000$, and $\beta^{-1} = 0.34$. The results are obtained by iterating the McCulloch-Pitts dynamics (**1.9**) with initial condition $s_i(t = 0) = \mathrm{sgn}(x_i^{(1)} + x_i^{(2)} + x_i^{(3)})$. Also shown is the solution of (3.1), dashed line. (**c**) Same, but for $\beta = 0.604$. Mean-field theory not shown. (**d**) Same, but for $\beta = 1$. Schematic, after simulations by Linus Sundberg. Exercise 3.1.

The average is over different realisations of random patterns. It can be shown that the the steady-state distribution corresponds to a local minimum of the free energy.[1] In mean-field theory, one approximates the energy function $H(\boldsymbol{s}) = -\frac{1}{2}\sum_{i\neq j} w_{ij} s_i s_j$ using

$$s_i s_j \approx s_i \langle s_j \rangle + \langle s_i \rangle s_j - \langle s_i \rangle \langle s_j \rangle \,. \tag{3.4}$$

In other words, one neglects terms quadratic in the fluctuations $s_i - \langle s_i \rangle$. Here $\langle s_i \rangle$ is the average of s_i in the steady state of the McCulloch-Pitts dynamics, given a certain realisation of random patterns. Using the definition of the order parameter, Equation (**3.9**), one finds

$$H(\boldsymbol{s}) \approx -\tfrac{1}{2}\sum_{\mu} m_\mu^2 + \tfrac{1}{2}\sum_{i\neq j} w_{ij} s_i \langle s_j \rangle + \tfrac{1}{2}\sum_{i\neq j} w_{ij} \langle s_i \rangle s_j \,. \tag{3.5}$$

Using Hebb's rule gives

$$H(\boldsymbol{s}) = -\tfrac{1}{2}\sum_{\mu} m_\mu^2 + \frac{1}{N}\sum_{\mu}\sum_{i} s_i x_i^{(\mu)} m_\mu \,. \tag{3.6}$$

Finally, one inserts this expression for H into Equation (3.3), performs the sum over s_i, and averages over the stored patterns. This yields

$$F(\beta) = -\tfrac{1}{2}\sum_{\mu} \langle m_\mu^2 \rangle - \frac{1}{\beta N}\sum_{i} \Big\langle \log\Big[2\cosh\big(\beta \sum_{\mu} m_\mu x_i^{(\mu)}\big)\Big]\Big\rangle . \tag{3.7}$$

Here $\langle \cdots \rangle$ denotes the average over random patterns, as in Equation (3.3).
A necessary condition for a local minimum of F is $\partial F / \partial m_\mu = 0$. For a single state, we obtain Equation (**3.14**) from this condition, if we neglect the cross-talk term, $\sum_\mu m_\mu x_j^{(\mu)} \approx m x_j^{(1)}$. For a mixed state (**3.19**) with three non-zero components $m > 0$, we find Equation (3.1) if we approximate $\sum_\mu m_\mu x_j^{(\mu)} \approx \sum_{\mu=1}^{n} m_\mu x_j^{(\mu)} = m z_j^{(n)}$. Solving these two equations allows us to determine the corresponding free energies.
It turns out that the free energy of the single state is lower than that for the mixed state. The networks dynamics may nevertheless stay near the mixed state for a long time, if that state is a local minimum of the free energy (a metastable state). This happens if the Hessian matrix $\mathbb{H}$ with elements $H_{\mu\nu} = \partial^2 F / \partial m_\mu \partial m_\nu$ is positive definite. The calculations summarised in Ref. [**33**] show that for $n = 3$, this is the case when the noise level is below 0.46. Above this critical noise level, the mixed state becomes unstable (the matrix of second free-energy derivatives is no longer positive definite). This is consistent with the behaviour seen in Figure 3.1.
For larger values of p (or $\alpha = p/N$), one cannot neglect the cross-talk term anymore. As a consequence it is more difficult to calculate the free energy. The corresponding

[1] H. Horner, Statistische Physik, Institut für theoretische Physik, Universität Heidelberg (2004).

mean-field calculation is described in Refs. [1,34]. For single states, the condition $\partial F/\partial m_\mu = 0$ yields the self-consistent theory summarised in Section 3.4.

Answer (3.2) — The calculation outlined below is described in Ref. [1]. One starts from Equation (3.32),

$$\beta(1-q) = \int_{-\infty}^{\infty} \frac{\mathrm{d}z}{\sqrt{2\pi}\sigma_z} e^{-z^2/2\sigma_z^2} \left[\beta - \beta \tanh^2(\beta m_1 + \beta z)\right]. \tag{3.8}$$

Equation (6.20) shows that the argument in the square brackets is proportional to the Dirac δ-function in the limit of large β,

$$\beta\left[1-\tanh^2(\beta m_1 + \beta z)\right] = \tfrac{\mathrm{d}}{\mathrm{d}z}\tanh(\beta m_1 + \beta z) \approx \tfrac{1}{2}\delta(m_1 + z). \tag{3.9}$$

Inserting this approximation into Equation (3.8) gives Equation (3.39b):

$$\beta(1-q) \approx \sqrt{\frac{2}{\pi\sigma_z^2}} e^{-m_1^2/2\sigma_z^2}. \tag{3.10}$$

We see, in particular, that $\beta(1-q)$ assumes a finite limit as $\beta \to \infty$. As a consequence, q must tend to unity. In this deterministic limit, it therefore follows from Equation (3.38) that

$$\sigma_z^2 = \frac{\alpha}{[1-\beta(1-q)]^2}. \tag{3.11}$$

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

$$m_1 = \mathrm{erf}\left(m_1/\sqrt{2\sigma_z^2}\right). \tag{3.12}$$

This is Equation (3.39c).

Answer (3.3) — To derive Equation (3.41), we insert the definition $y = m_1/\sqrt{2\sigma_z^2}$ (page 46), into (3.39c). This gives

$$\sqrt{2}\sigma_z\, y = \mathrm{erf}(y). \tag{3.13}$$

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

$$\sigma_z - \sqrt{2/\pi}\mathrm{e}^{-y^2} = \sqrt{\alpha}. \tag{3.14}$$

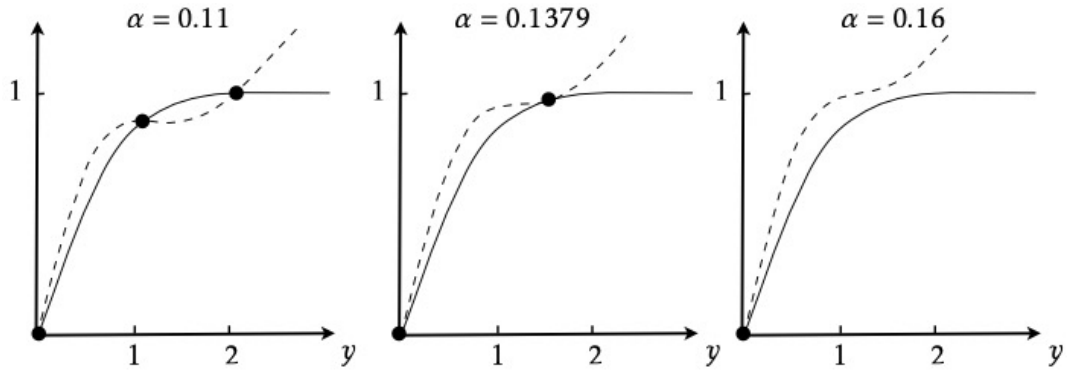


Figure 3.2: Numerical solution of Equation (3.15) for three values of α, $\alpha < \alpha_c$ (left), $\alpha = \alpha_c$ (center), and $\alpha > \alpha_c$ (right). Shown are the l.h.s. of Equation (3.15), dashed lines, and the r.h.s. of Equation (3.15), solid lines. See also Figure 2.16 in Ref. [1]. Exercise 3.3.

Substituting (3.14) into (3.13) gives

$$\sqrt{2}y\left[\sqrt{\alpha}+\sqrt{2/\pi}\mathrm{e}^{-y^2}\right]=\mathrm{erf}(y). \tag{3.15}$$

This expression is equivalent to Equation (**3.41**). The solutions of Equation (3.15) are illustrated in Figure 3.2. 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 (3.15) 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:

$$m_1 = \int \mathrm{d}z P(z)\tanh(\beta m_1 + \beta z), \tag{3.16a}$$

$$q = \int \mathrm{d}z P(z)\tanh^2(\beta m_1 + \beta z), \tag{3.16b}$$

$$P(z) = (2\pi\sigma_z^2)^{-1/2}\exp[-z^2/(2\sigma_z^2)] \quad \text{with} \quad \sigma_z^2 = \alpha q/[1-\beta(1-q)]^2. \tag{3.16c}$$

These self-consistent equations 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 3.3 shows numerical solutions of Equations (3.16). The Figure demonstrates how the order parameter decreases as the noise level increases,

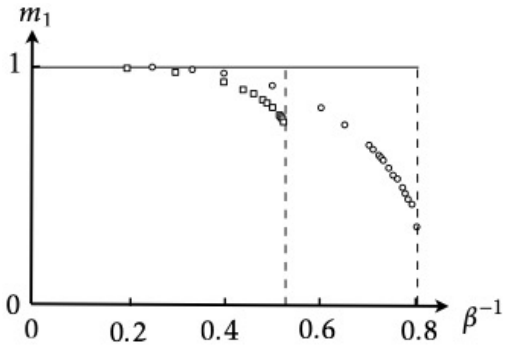


Figure 3.3: Solutions of Equation (3.15) for $\alpha = 0.05$ (□), and $\alpha = 0.01$ (∘). Shown is m_1 as a function of the noise level β^{-1}. The vertical dashed lines indicate the critical noise level above which the network ceases to function. See Fig. 3 in Ref. [34]. Exercise 3.3.

for $\alpha = 0.01$ and $\alpha = 0.05$. The vertical dashed lines in Figure 3.3 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) — The task is to demonstrate that Equation (3.51),

$$w_{ij} = \frac{1}{N} \sum_{\mu\nu} x_i^{(\mu)} [\mathbb{Q}^{-1}]_{\mu\nu} x_j^{(\nu)} \quad \text{with} \quad Q_{\mu\nu} = \frac{1}{N} \sum_i x_i^{(\mu)} x_i^{(\nu)}, \tag{3.17}$$

ensures that all stored patterns are reproduced. To check this, we evaluate $\sum_j w_{ij} x_j^{(\delta)}$ for arbitrary $\delta = 1, \ldots, p$:

$$\frac{1}{N} \sum_j \sum_{\mu,\nu} x_i^{(\mu)} [\mathbb{Q}^{-1}]_{\mu\nu} x_j^{(\nu)} x_j^{(\delta)} = \sum_{\mu,\nu} x_i^{(\mu)} [\mathbb{Q}^{-1}]_{\mu\nu} Q_{\nu\delta} = \sum_{\mu} x_i^{(\mu)} \delta_{\mu\delta} = x_i^{(\delta)}. \tag{3.18}$$

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

$$\langle b_i \rangle = J \sum_{j=\mathrm{nn}(i)} \langle s_j \rangle + h = J z \langle s_i \rangle + h \,. \tag{3.19}$$

Here z is the coordination number, the number of nearest neighbours in d dimensions (Table 3.1). With the definition of the order parameter, the *magnetisation*

Table 3.1: Critical temperature of the d-dimensional Ising model on a hypercubic lattice, for $h = 0$. Exercise 3.5.

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

$m = \langle \frac{1}{N} \sum_i s_i \rangle$, it follows that

$$m = \tanh(\beta J z m + \beta h). \tag{3.20}$$

Here we consider $h = 0$. In this case, Equation (3.20) has only the solution $m = 0$ if $\beta J z < 1$. For $\beta J z > 1$, by contrast, there are three solutions, $m = 0$, as well as $m = \pm|m| \neq 0$. Using $\beta_c = (k_B T_c)^{-1}$, one defines the critical temperature T_c as

$$k_B T_c = J z\,. \tag{3.21}$$

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

$$m \sim \pm(3|\delta|)^{1/2} \quad (\text{with} \quad \delta = \tfrac{T - T_c}{T_c} \leq 0). \tag{3.22}$$

as $T \to T_c$ from below. We say that the magnetisation tends to zero with critical exponent $\frac{1}{2}$ (Figure 3.4). It turns out that this is the correct behaviour for $d \geq 4$. In $d = 1, 2, 3$ dimensions, mean-field theory gets the critical exponent wrong because the mean-field approximation neglects fluctuations.

The mean-field result (3.21) for the critical temperature becomes more accurate as the dimension increases. This is shown in Table 3.1 which compares the result of the mean-field approximation for T_c with the actual value of the critical temperature. In the limit of $d \to \infty$, the mean-field prediction for T_c becomes exact (not shown). 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**.

[2] Exact solution of one-dimensional Ising model: $k_B T_c = 0$.

[3] Exact solution of two-dimensional Ising model: $k_B T_c = 2J/\log(1+\sqrt{2})$. See Equation (3.2) in Newell & Montroll, Rev. Mod. Phys. **25** (1953) 353, and the discussion in the paragraphs below that equation.

[4] Ferrenberg & Landau, Phys. Rev. B **44** (1991) 5081.

[5] Gaunt, Sykes & McKenzie, J. Phys. A **12** (1979) 871.

[6] Luijten, Binder & Blöte, Eur. Phys. J. B **9** (1999) 289.

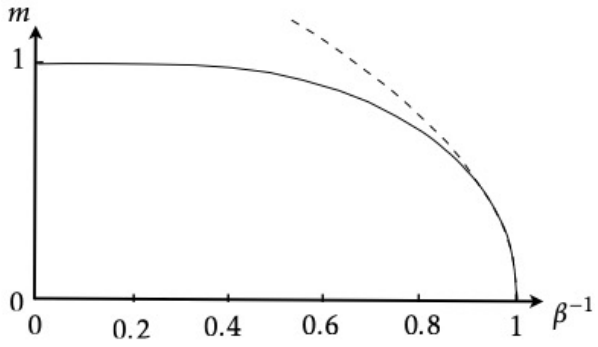


Figure 3.4: Solution $m > 0$ of the mean-field equation (3.20), for $Jz = 1$ and $h = 0$ as a function of $k_B T = \beta^{-1}$ (solid line). Also shown is the approximate mean-field solution near the critical temperature, Equation (3.22), dashed line. Exercise 3.5.

Answer (3.6) — To demonstrate that the one-step error probabilities (**2.39**) and (**3.40**) have the same limit for small values of α, 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, we conclude that

$$y = 1/\sqrt{2\alpha} \tag{3.23}$$

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

4 Solutions for 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, on the other hand,

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

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 2.8, $n_m' = \theta_{\mathrm{H}}(b_m)$ where $\theta_{\mathrm{H}}(b)$ 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

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

with $p(b)=[1+\exp(-\beta b)]^{-1}$. Note that the argument of the exponential function 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

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

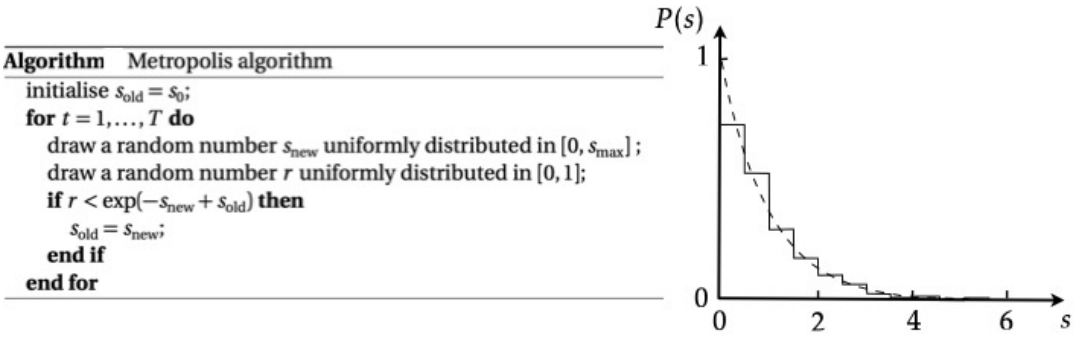


Figure 4.1: Left: algorithm to generate samples from the exponential distribution $\exp(-s)$. Right: histogram obtained from 1000 iterations of the algorithm (solid line). The dashed line shows $\exp(-s)$. Exercise 4.3.

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 corresponding Markov chain. The dashed line is $\exp(-s)$.

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

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

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 $P_1^* = P_2^* = P_3^* = \frac{1}{3}$.

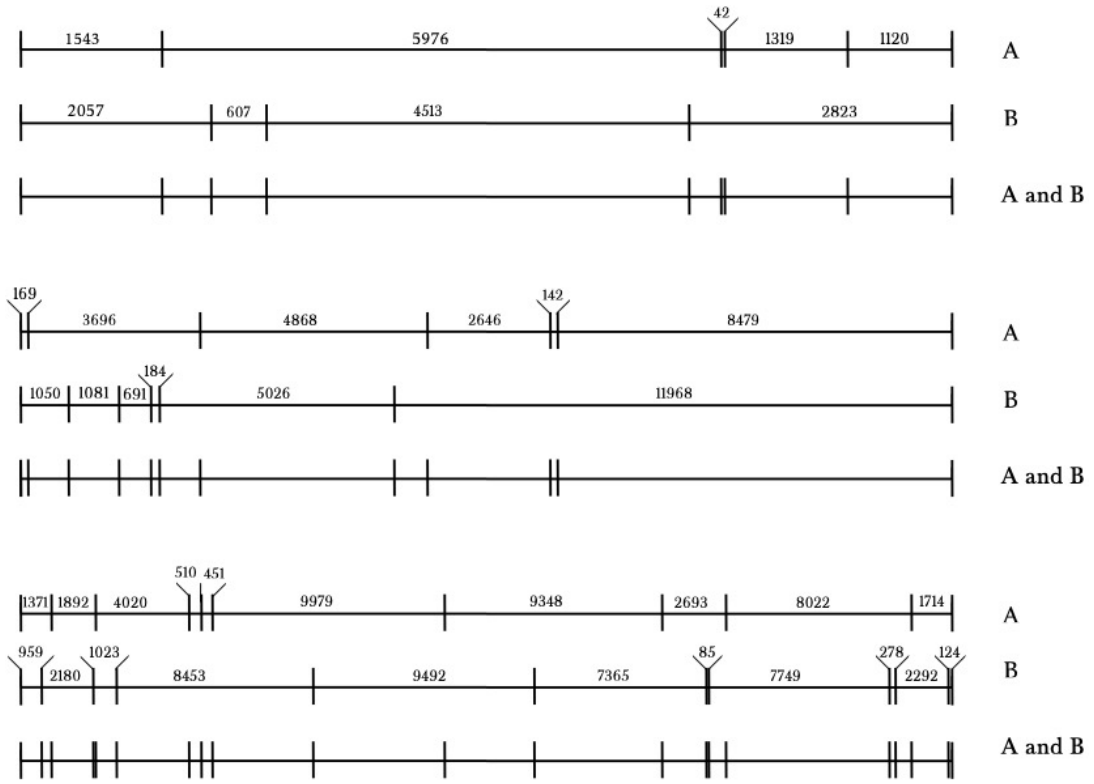


Figure 4.2: Solutions of the three double-digest problems from Table 4.1. Solutions obtained by Vitalii Iarko and Erik Werner. Exercise 4.5.

Now check whether the detailed-balance condition holds. 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 satisfied. The reason is that the Markov chain represents a cycle. Its steady state is an example of a non-equilibrium steady state, a steady state with a non-zero probability current.[1]

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 unique. 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 is 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

[1] See for example M. Wilkinson & B. Mehlig, Phys. Rev. E **68** (2003) 040101(R).

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 in Adv. Appl. Math. **8** (1987) 194.

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

$$D_{\mathrm{KL}} = -\sum_{\boldsymbol{x}} P_{\mathrm{data}}(\boldsymbol{x}) \log[P_{\mathrm{B}}(\boldsymbol{s} = \boldsymbol{x}) / P_{\mathrm{data}}(\boldsymbol{x})], \tag{4.8}$$

where the sum is over the *distinct* patterns $\boldsymbol{x}$ that occur with different frequencies in the data set. Now we use the inequality $-\log z \geq -(z-1)$ to obtain a bound for the Kullback-Leibler divergence:

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

Finally, we use that the distributions are normalised,

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

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

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

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

Answer (4.7) — A restricted Boltzmann machine with M hidden neurons was trained on the XOR problem with the CD-k algorithm (Algorithm **3**). The weights were initially Gaussian random with mean zero and unit variance, the initial thresholds

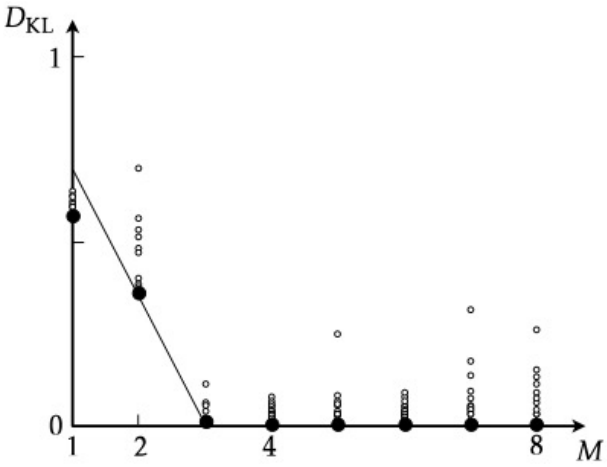


Figure 4.3: Kullback-Leibler divergence D_{KL} achieved by training a restricted Boltzmann machine on the XOR problem, versus the number M of hidden neurons. Upper bound (**4.40**), solid line, and numerical results using the CD-k algorithm (Algorithm **3**). Training parameters: $\nu_{\max} = 10^4$, $p_0 = 20$, $k = 10$, and $\eta = 0.005$. Sampling parameters: the Markov chain was iterated for 10^7 iterations. Shown are the results of 20 independent runs ($\circ$). For each value of M, the best result is shown as $\bullet$. Schematic, adapted from numerical results obtained by Ehsan Ghane. Exercise 4.7.

were set to zero. The remaining training parameters are given in the caption. After training, the model (Boltzmann) distribution was sampled using the Markov chain (**4.36**). Then D_{KL} was computed using Equation (**4.18**). For each value of M, this process was repeated 20 times, for different initialisations of weights and thresholds. The results are shown in Figure 4.3. We see that there is some spread of the D_{KL}-values found, likely because the CD-k algorithm fails to explore the global maximum. For $M > 1$, the best training 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.

Instead of using the Markov chain (**4.36**) to sample the model probabilities after training, one can evaluate them exactly since the number of neurons is quite small. The model probabilities $P_{\mathrm{B}}(\boldsymbol{v} = \boldsymbol{x})$ are obtained from $P_{\mathrm{B}}(\boldsymbol{v}, \boldsymbol{h})$ by summing over the states of the hidden neurons:

$$P_{\mathrm{B}}(\boldsymbol{v} = \boldsymbol{x}) \quad = \sum_{h_1 = \pm 1, \ldots, h_M = \pm 1} Z^{-1} \exp\bigl[-H(\boldsymbol{v} = \boldsymbol{x}, \boldsymbol{h})\bigr]. \tag{4.11}$$

Here H is the energy function (**4.29**), and Z is the partition function (a normalisation factor).

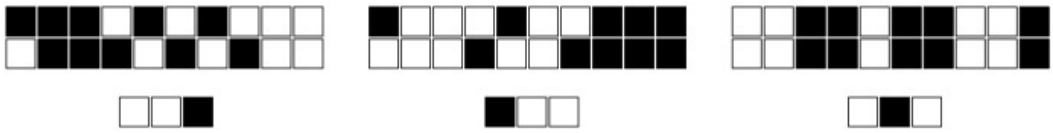


Figure 4.4: Illustration of shifter ensemble [15,47]. Exercise 4.8.

Answer (4.8) — The shifter ensemble is illustrated in Figure 4.4. The patterns are constructed as follows [15,47]. 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 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 in terms of a Boltzmann machine that relies only on two-point correlations, 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) [15,47].

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

$$H = -\sum_{i=1}^{M}\sum_{j=1}^{N} w_{ij} h_i v_j + \sum_{j=1}^{N} \theta_j^{(\text{v})} v_j + \sum_{i=1}^{M} \theta_i^{(\text{h})} h_i \,. \tag{4.12}$$

The deterministic update rule follows from Equation (4.30),

$$h'_m = \text{sgn}(b_m^{(\text{h})}), \quad \text{and} \quad v'_n = \text{sgn}(b_n^{(\text{v})}), \tag{4.13}$$

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

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

This allows us to express the change in H as

$$H' - H = -\sum_{i=1}^{M} (h'_i - h_i) \Big(\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\text{h})} \Big). \tag{4.15}$$

Suppose that $h_i = 1$ and $h_i' = -1$, so that $h_i' - h_i < 0$. It follows from Equation (4.13) that the sign of $\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\mathrm{h})}$ 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 $\sum_{j=1}^{N} w_{ij} v_j - \theta_i^{(\mathrm{h})} > 0$. 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 2.9). 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

$$P_{\mathrm{B}}(\boldsymbol{s} = \boldsymbol{x}) = Z^{-1} \exp\Big(\tfrac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j - \sum_i \theta_i x_i\Big). \tag{4.16}$$

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

$$\frac{\partial \log \mathscr{L}}{\partial \theta_m} = \frac{\partial}{\partial \theta_m} \sum_{\mu} \Big(-\log Z + \tfrac{1}{2} \sum_{i \neq j} w_{ij} x_i^{(\mu)} x_j^{(\mu)} - \sum_i \theta_i x_i^{(\mu)} \Big), \tag{4.17}$$

with

$$\log Z = \sum_{s_i = \pm 1, \ldots, s_N = \pm 1} \exp\Big(\tfrac{1}{2} \sum_{i \neq j} w_{ij} s_i s_j - \sum_i \theta_i s_i\Big). \tag{4.18}$$

The derivative of $\log Z$ is

$$\frac{\partial \log Z}{\partial \theta_m} = -\sum_{s_i = \pm 1, \ldots, s_N = \pm 1} s_m P_{\mathrm{B}}(\boldsymbol{s}) = -\langle s_m \rangle_{\mathrm{model}}. \tag{4.19}$$

Evaluating the derivative of the second term in Equation (4.17) in a similar way, one obtains

$$\frac{\partial \log \mathscr{L}}{\partial \theta_m} = -p\big(\langle x_m \rangle_{\mathrm{data}} - \langle s_m \rangle_{\mathrm{model}}\big). \tag{4.20}$$

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

$$\frac{\partial \log \mathscr{L}}{\partial \theta_n^{(\mathrm{v})}} = -\big(x_n^{(\mu)} - \langle v_n \rangle_{\mathrm{model}}\big). \tag{4.21}$$

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, $\langle v_n \rangle_{\mathrm{model}} \approx v_{n,t=k}$ (CD-k algorithm). This gives

$$\delta\theta_n^{(\mathrm{v})} = \eta \frac{\partial \log \mathscr{L}}{\partial \theta_n^{(\mathrm{v})}} \approx -\eta(v_{n,t=0} - v_{n,t=k}). \tag{4.22}$$

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

$$\frac{\partial \log \mathscr{L}}{\partial \theta_m^{(\mathrm{h})}} = -\big(\langle h_m \rangle_{\mathrm{data}} - \langle h_m \rangle_{\mathrm{model}}\big), \tag{4.23}$$

where $\langle h_m \rangle_{\mathrm{data}}$ is the average of h_m over the Boltzmann distribution conditional on the data point $\boldsymbol{v} = \boldsymbol{x}^{(\mu)}$ (note that the average $\langle \cdots \rangle_{\mathrm{data}}$ in Equation (4.20) is defined differently). Using (4.33) and (4.34), one has $\langle h_m \rangle_{\mathrm{data}} = \tanh\big(\sum_{j=1}^N w_{mj} x_j^{(\mu)} - \theta_m^{(\mathrm{h})}\big)$. The average $\langle h_m \rangle_{\mathrm{model}}$ 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 $\tanh\big(\sum_{j=1}^N w_{mj} v_j - \theta_m^{(\mathrm{h})}\big)$. Then one averages $\tanh\big(\sum_{j=1}^N w_{mj} v_j - \theta_m^{(\mathrm{h})}\big)$ over the Boltzmann distribution using a Markov chain with $\boldsymbol{v}_{t=0} = \boldsymbol{x}^{(\mu)}$. Using the CD-$k$ approximation gives

$$\begin{aligned}\delta\theta_m^{(\mathrm{h})} &= \eta \frac{\partial \log \mathscr{L}}{\partial \theta_m^{(\mathrm{h})}} \\ &\approx -\eta\Big[\tanh\Big(\sum_{j=1}^N w_{mj} v_{j,t=0} - \theta_m^{(h)}\Big) - \tanh\Big(\sum_{j=1}^N w_{mj} v_{j,t=k} \theta_m^{(h)}\Big)\Big].\end{aligned} \tag{4.24}$$

This is Equation (4.39b).

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

$$\delta w_{mn}^{(\mu)} = \eta\big(\langle h_m x_n^{(\mu)} \rangle_{\mathrm{data}} - \langle h_m v_n \rangle_{\mathrm{model}}\big). \tag{4.25}$$

The term $\langle h_m x_n^{(\mu)} \rangle_{\mathrm{data}}$ is defined as the average over all states of the hidden neurons when the pattern $\boldsymbol{x}^{(\mu)}$ is clamped to the visible neurons:

$$\langle h_m x_n^{(\mu)} \rangle_{\mathrm{data}} = \sum_{h_1=0,1,\ldots,h_M=0,1} h_m x_n^{(\mu)} \Big[\prod_{i=1}^M P(h_i | \boldsymbol{v} = \boldsymbol{x}^{(\mu)})\Big]. \tag{4.26}$$

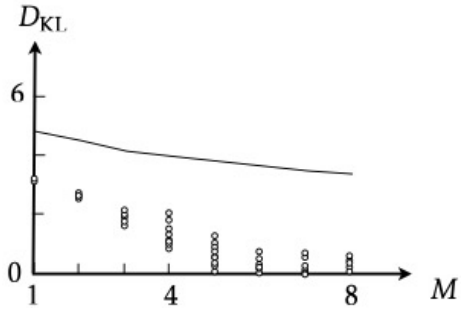


Figure 4.5: Kullback-Leibler divergence D_{KL} obtained by training a restricted Boltzmann machine on the bars-and-stripes ensemble, versus the number M of hidden neurons. Upper bound (**4.40**), solid line, and numerical results using the CD-k algorithm (Algorithm **3**), $\circ$. Training parameters: $\nu_{\max} = 10^4$, $p_0 = 20$, $k = 10$, and $\eta = 0.005$. Shown are the results of 20 independent runs for each value of M, in each case using the exact Boltzmann probabilities, Equation (4.11). Schematic, adapted from numerical results obtained by Ehsan Ghane. Exercise 4.12.

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

$$\langle h_m x_n^{(\mu)} \rangle_{\mathrm{data}} = \sum_{h_m=0,1} h_m x_n^{(\mu)} P(h_m|\boldsymbol{v}=\boldsymbol{x}^{(\mu)}). \tag{4.27}$$

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

$$h'_m = \begin{cases} 1 & \text{with probability} \quad p(b_m^{(\mathrm{h})}), \\ 0 & \text{with probability} \quad 1-p(b_m^{(\mathrm{h})}), \end{cases} \tag{4.28}$$

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 also Exercise 4.2.
We use (4.28) to evaluate the average in Equation (4.27):

$$\langle h_m x_n^{(\mu)} \rangle_{\mathrm{data}} = p(b_m^{(\mathrm{h})}) x_n^{(\mu)}. \tag{4.29}$$

The second average in (4.25) is evaluated in an analogous fashion

$$\langle h_m v_n \rangle_{\mathrm{model}} = \langle p(b_m^{(\mathrm{h})}) v_n \rangle_{\mathrm{model}}. \tag{4.30}$$

Contrast Equations (4.29) and (4.30) 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 function $p(b)$.

Answer (4.12) — Figure 4.5 shows the Kullback-Leibler entropy obtained by training a restricted Boltzmann machine with M hidden neurons on the bars-and-stripes data set, as a function of M. Also shown is the upper bound (**4.40**). 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 $N = 9$ bits.

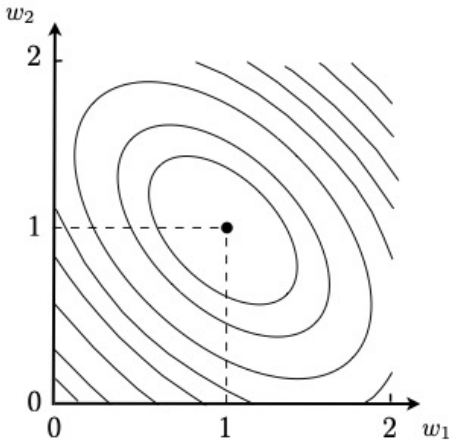


Figure 5.1: Energy function $H(w_1, w_2, \theta = \frac{3}{2})$ for the Boolean AND problem. Exercise 5.1.

5 Solutions for 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

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

Here $\langle\cdots\rangle = p^{-1}\sum_{\mu=1}^{p}\cdots$ is the average over patterns. For the AND problem (Figure 5.7),

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

Now set the derivatives to zero to determine $\boldsymbol{w}$ and θ. This gives $\boldsymbol{w} = [1,1]^{\mathsf{T}}$ and $\theta = \frac{3}{2}$. With these weights and threshold, 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 $O^{(\mu)} \neq t^{(\mu)}$. In other words: H is non-zero.
However, one can check that 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 (5.21). Since we have a non-zero

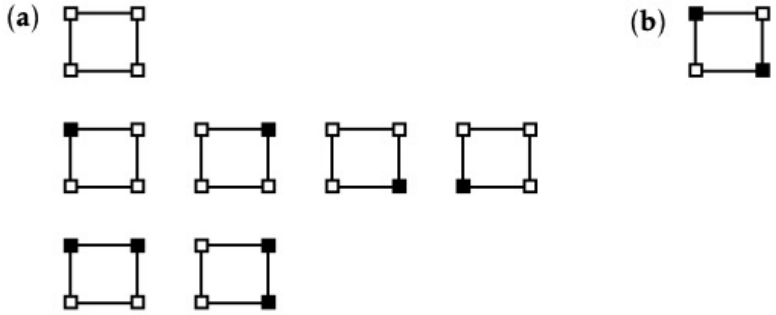


Figure 5.2: Half of all Boolean functions with two-dimensional inputs. The other half is obtained by conjugation. (**a**) These functions are linearly separable. (**b**) The XOR function is not linearly separable. Exercise 5.2.

threshold, and only one output, Equation (**5.21**) becomes

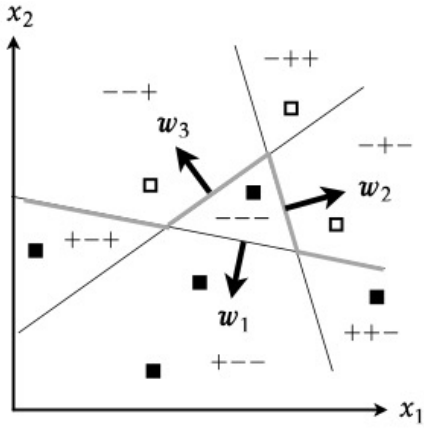
$$w_k = \frac{1}{4}\sum_{\mu\nu}(t^{(\mu)}+\theta)[\mathbb{Q}^{-1}]_{\mu\nu}x_k^{(\mu)}. \tag{5.3}$$

The matrix $\mathbb{Q}$ [Equation (**5.22**)] takes the form:

$$\mathbb{Q} = \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}. \tag{5.4}$$

Since this matrix is not invertible, we compute its pseudo-inverse using the singular-value decomposition $\mathbb{Q} = \mathbb{U}\mathbb{S}\mathbb{V}^\mathsf{T}$. Here $\mathbb{S}$ is the diagonal matrix containing the singular values of $\mathbb{Q}$. The pseudo-inverse is given by $\mathbb{V}\mathbb{S}'\mathbb{U}^\mathsf{T}$, where the diagonal matrix $\mathbb{S}'$ is obtained by taking the reciprocals of all non-zero singular values. See also Exercise 10.14. Replacing $\mathbb{Q}^{-1}$ by $\mathbb{V}\mathbb{S}'\mathbb{U}^\mathsf{T}$ in Equation (5.3) yields $\boldsymbol{w} = [1,1]^\mathsf{T}$ for $\theta = \frac{3}{2}$, the same as above.

Answer (5.2) — For two-dimensional inputs, there are $2^{(2^2)} = 16$ Boolean functions in total. Eight of them are shown in Figure 5.2. The other eight are obtained by conjugation ($\square \longleftrightarrow \blacksquare$). 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



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 5.4. (**b**) Encoding of the different regions and corresponding target values for Exercise 5.4.

edge are linearly separable. The cube has six faces. There are four functions with three ■ per face. They are linearly separable. 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 eight functions with four ■ connecting to a corner. Conjugation gives more 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. Such output problems are generally linearly separable, 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, so that the problem is linearly separable.

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

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

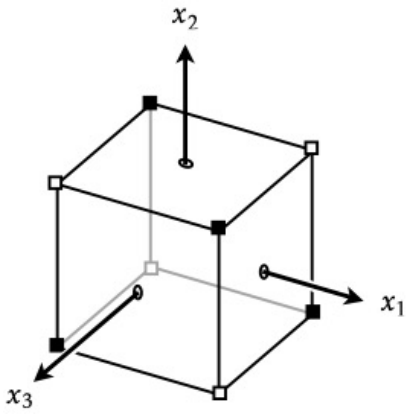
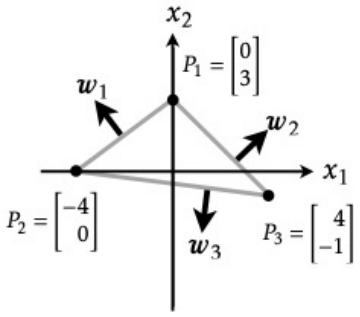


Figure 5.4: Three-dimensional parity function. Exercise 5.5.



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.

It follows from Equation (**5.35**) that $V_j^{(\mu)} = 1$ if $j = \mu$, but 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 $t^{(\mu)} = -1$. 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**)]: $\theta_1 = 12$, $\theta_2 = 12$, and $\theta_3 = 4$. A possible choice for the

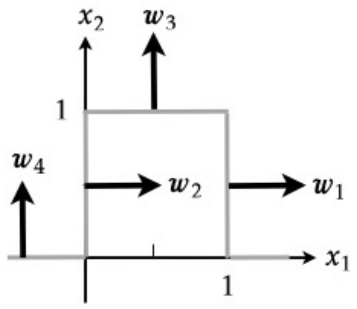


Figure 5.6: Decision boundaries and corresponding weights for Exercise 5.7.

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) — The decision boundaries of the four hidden neurons are shown in Figure **5.22**(left), the lines $[1, x_2]$, $[0, x_2]$, $[x_1, 1]$, and $[x_1, 0]$. Consider the decision boundary $[1, x_2]$ of neuron $j = 1$. Points $\boldsymbol{x}$ on the decision boundary satisfy Equation (**5.13**),

$$\boldsymbol{w}_1 \cdot \boldsymbol{x} - \theta_1 = \begin{bmatrix} w_{11} & w_{22} \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \end{bmatrix} - \theta_1 = 0\,. \tag{5.7}$$

This yields $w_{11} = \theta_1$ and $w_{22} = 0$. Choosing $w_{11} = 1$ gives $\theta_1 = 1$.
The other weight vectors and thresholds are found in a similar fashion. In summary:

$$\boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \theta_1 = 1\,, \quad \boldsymbol{w}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \theta_2 = 0\,, \quad \boldsymbol{w}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \theta_3 = 1\,, \quad \boldsymbol{w}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \theta_4 = 0\,. \tag{5.8}$$

The overall decision boundary is determined by Equation (**5.37**). It is shown in Figure 5.6.
For the second part of the question, 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 renders 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

$$\boldsymbol{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \theta_1 = 1\,, \quad \boldsymbol{w}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \theta_2 = \tfrac{1}{2}\,, \quad \boldsymbol{w}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \theta_3 = \tfrac{4}{5}\,. \tag{5.9}$$

The resulting output problem is shown on the right in Figure 5.7. The problem can be

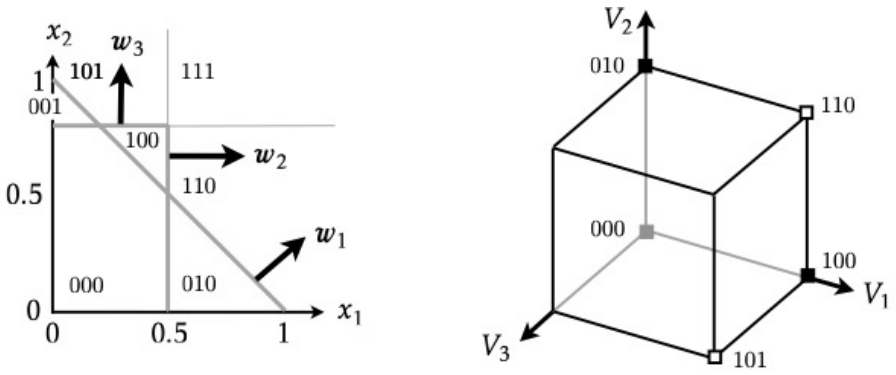


Figure 5.7: Left: weights and decision boundaries in the input plane for Exercise 5.8. Right: output problem for Exercise 5.8.

solved by a decision boundary that contains the points $\boldsymbol{V}_1 = [\frac{1}{2}, 1, 0]^\mathsf{T}$, $\boldsymbol{V}_2 = [1, \frac{1}{2}, , 0]^\mathsf{T}$, and $\boldsymbol{V}_3 = [0, 0, 1]^\mathsf{T}$. Equation (**5.13**) gives three conditions for these three points:

$$W_1 + \tfrac{1}{2} W_2 = \Theta\,, \quad W_2 + \tfrac{1}{2} W_1 = \Theta\,, \quad \text{and} \quad W_3 = \Theta\,. \tag{5.10}$$

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 $O = \theta_\mathrm{H}(-\frac{2}{3} V_1 - \frac{2}{3} V_2 - V_3 + 1)$.

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 is defined to be zero otherwise. As a consequence, p_n is normalised: $\sum_{n=0}^{\infty} p_n = 1$. To evaluate the mean $\sum_{n=0}^{\infty} n p_n$, use that $n\binom{n-1}{N-1} = N\binom{n}{N}$. This gives:

$$\sum_{n=0}^{\infty} n p_n = 2N \sum_{n=0}^{\infty} (\tfrac{1}{2})^{n+1} \binom{n}{N} = 2N\,, \tag{5.11}$$

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 w.r.t. 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 $O_i^{(\mu)} = t_i^{(\mu)}$ for $i = 1, \ldots, N$ and $\mu = 1, \ldots, p$ if the inverse of the overlap matrix (**5.22**) exists. This

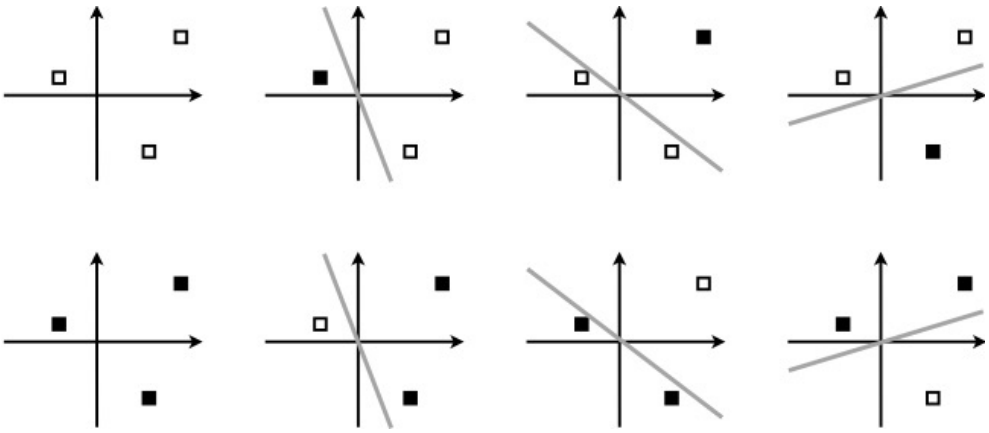


Figure 5.8: All problems obtained by randomly colouring three given points in general position with the origin. Six of the problems are homogeneously linearly separable. Exercise 5.10.

is the case if the columns of the overlap matrix are linearly independent, and 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, as explained in Ref. [1]: 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. One concludes that a single neuron with a continuous, non-linear, monotonically increasing activation function cannot solve the classification problem if there are more than N patterns. Contrast this with the solution of the AND problem in Figure 5.7 with a single neuron with activation function $g(b) = \mathrm{sgn}(b)$.

Answer (5.12) — Equation (5.38) is quoted by Hertz, Krogh & Palmer [1]. One obtains it using the asymptotic approximation

$$\binom{p-1}{k} \sim \frac{2^{p-1}}{\sqrt{(p-1)\pi/2}} \exp\left[-\frac{(k-(p-1)/2)^2}{(p-1)/2}\right], \tag{5.12}$$

which is valid for large p at fixed k. One inserts Equation (5.12) into (5.29), and approximates the sum over k as an integral (the lowest-order term in a Poisson summation[1]). Integrating the Gaussian in (5.12), one gets an error function. Taking the limit of $N \to \infty$ at fixed α gives Equation (5.38) . 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.

[1] E. W. Weisstein, *Poisson Sum Formula*, `mathworld.wolfram.com/PoissonSumFormula.html`

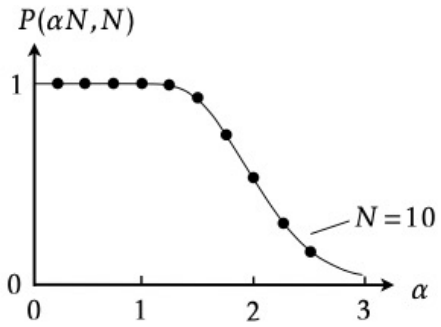


Figure 5.9: $P(\alpha N, N)$ versus α for $N = 10$ (schematic). Numerical evaluation of Equation (**5.29**), symbols. Approximate expression (**5.38**), solid line. Exercise 5.12.

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 5.13. For $n = 4$ and $n = 5$, 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. Numerical results obtained by Ludvig Storm. 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 5.2.
The McCulloch-Pitts neuron has weights $\boldsymbol{w}_j$ and a threshold θ. The weights are updated using the learning rule (**5.18**). For a single output unit, this rule simplifies to

$$\delta w_n^{(\mu)} = \eta(t^{(\mu)} - O^{(\mu)})x_n^{(\mu)}. \tag{5.13}$$

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 (5.13):

$$\delta\theta^{(\mu)} = -\eta(t^{(\mu)} - O^{(\mu)}). \tag{5.14}$$

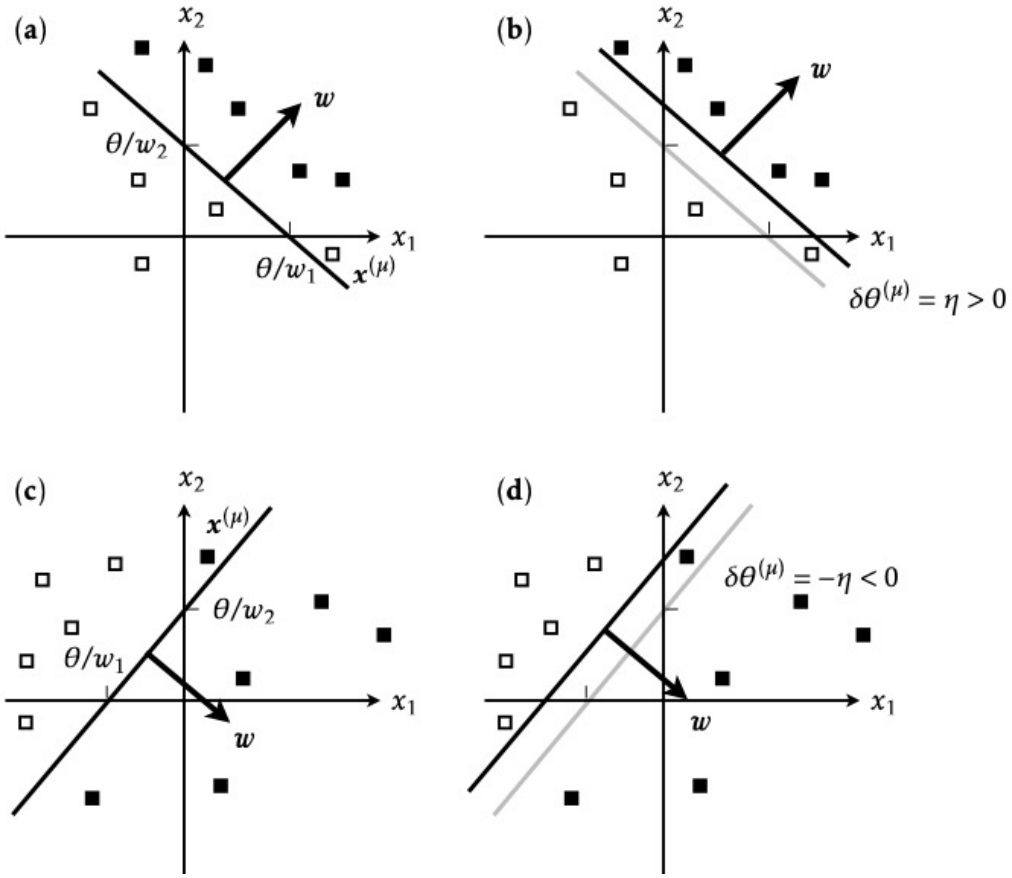


Figure 5.10: Geometrical interpretation of the learning rule (5.14) for the threshold of a binary threshold neuron. Exercise 5.13.

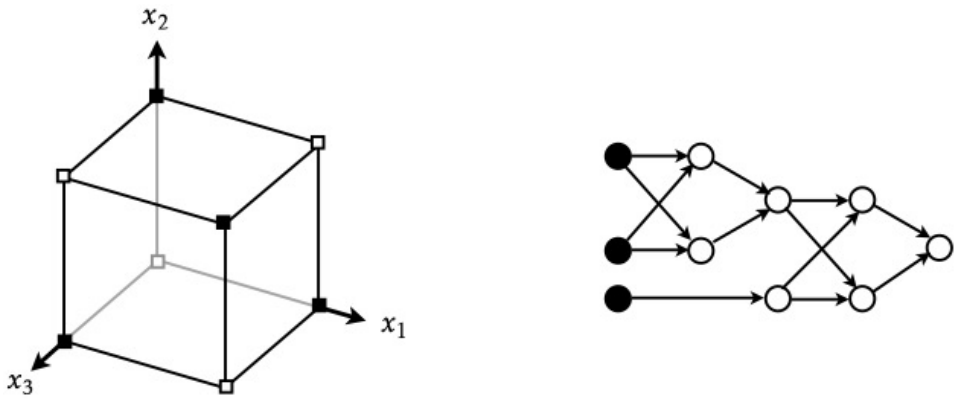


Figure 5.11: Input space and network layout for the Boolean function shown in Figure **5.25**. The network is made out of two XOR units. Exercise 5.14.

To see that this rule does the trick, consider its geometric interpretation, shown in Figure 5.10. Panel (**a**) depicts 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, by adding $\delta\theta^{(\mu)} = \eta > 0$ as shown in panel (**b**). Since □ 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 ■ stands for $t^{(\mu)} = 1$, this rule can again be written as $\delta\theta = -\eta\, t^{(\mu)}$. Both conclusions are consistent with Equation (5.14), 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 algorithm finds a solution. If the algorithm does not, 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 [**74**], 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 estimate the fraction $\mathcal{N}_n/2^{(2^n)}$. 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 quickly 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 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). See also Exercise 7.8.

6 Solutions for exercises in Chapter 6

Answer (6.1) — The fact that the eigenvalues of the data covariance matrix (6.24) are real is demonstrated on page 105. The argument goes as follows. Since the 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 $\mathbb{C} = \sum_\alpha \lambda_\alpha \boldsymbol{u}_\alpha \boldsymbol{u}_\alpha^\mathsf{T}$ 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 conclude 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

$$\mathbb{C} = \frac{1}{4}\begin{bmatrix} 10 & \frac{5}{2} \\ \frac{5}{2} & \frac{5}{8} \end{bmatrix}. \tag{6.2}$$

The largest eigenvalue is $\lambda_1 = 85/32$, with eigenvector $\boldsymbol{u}_1 \propto [4,1]^\mathsf{T}$. The second eigenvalue vanishes, $\lambda_2 = 0$, 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:

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

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

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

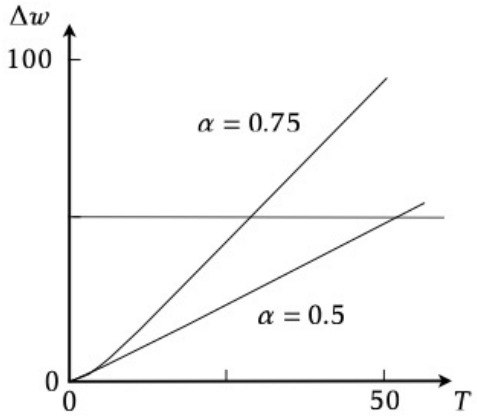


Figure 6.1: Weight change $\Delta w = w_B - w_A$ versus iteration number T. Exercise 6.4.

where η_t is a learning rate that may depend on the time-step 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

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

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 $\Delta w = w_\mathrm{B} - w_\mathrm{A}$. Figure 6.1 shows Δw versus T for different values of α. We see that Δw is larger for larger values of 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

$$T \approx 2(1-\alpha)(w_\mathrm{B} - w_\mathrm{A}) + \frac{1}{1-\alpha}. \tag{6.13}$$

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

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

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

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

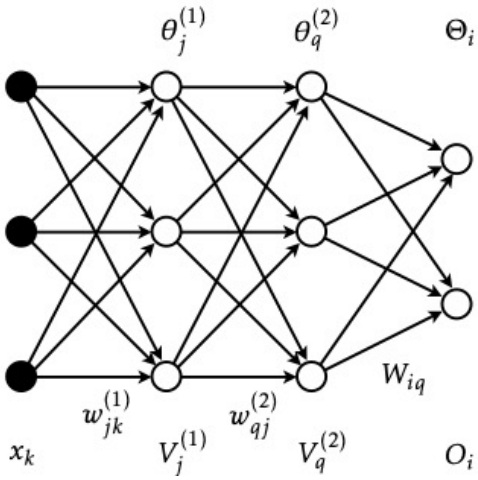


Figure 6.2: Network layout for Exercise 6.6. See also Figure **6.13**.

Using Equations (6.16) and (6.17) gives

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

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 $\Delta_i^{(\mu)} = t_i^{(\mu)} - O_i^{(\mu)}$. Note that 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. To obtain δW_{mn} we take the derivative of H with respect to W_{mn} and multiply with $-\eta$,

$$\delta W_{mn} = \eta (t_m - O_m) g'(B_m) V_n^{(2)}. \tag{6.20}$$

This expression corresponds to Equation (**6.6a**), but here and in the following we leave out the sum over the pattern index μ, to get the stochastic gradient-descent algorithm. The learning rule for Θ_m is obtained from Equation (6.20) by setting $V_n^{(2)} = -1$ [Equation (**6.11a**)].
To find the learning rule for $w_{mn}^{(2)}$, we need to calculate

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

Using $\partial O_i/\partial V_q^{(2)} = g'(B_i)W_{iq}$ and $\partial V_q^{(2)}/\partial w_{mn}^{(2)} = g'(b_q^{(2)})\delta_{qm}V_n^{(1)}$, we find

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

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) in Exercise 6.7. Finally, 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 minimise $H = \frac{1}{2}\sum_i (t_i - O_i)^2$ (leaving out the sum over μ),

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

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

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

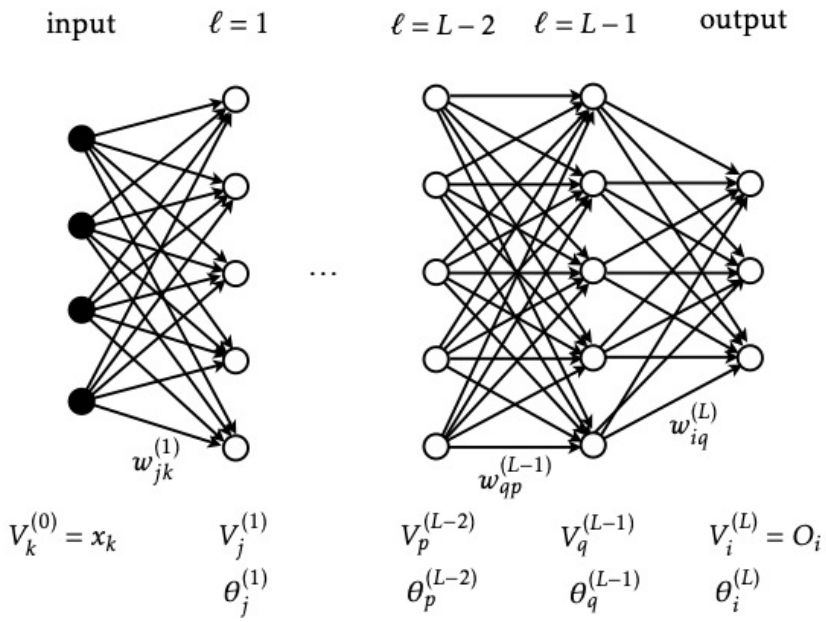


Figure 6.3: Network layout for Exercise 6.7.

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

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

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

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

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,

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

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 $\langle \boldsymbol{x} \rangle$.[1] 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.

[1] Hinton, G., *A bag of tricks for mini batch gradient descent. Neural Networks for Machine Learning* (2012).

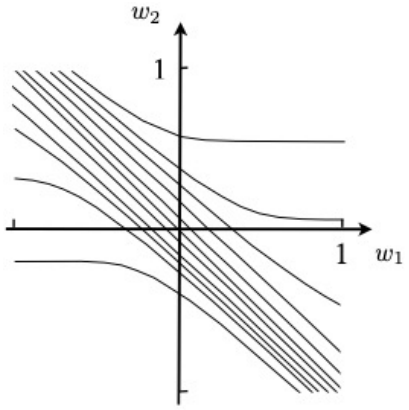


Figure 6.4: Contour plot of the energy function $H(w_1, w_2, \theta = 0)$ for Exercise 6.9.

Answer (6.10) — To find the global minimum $\boldsymbol{x}^*$ of $f(\boldsymbol{x})$ subject to the constraint $g(\boldsymbol{x}) = 0$, one determines the singular points of the Lagrangian $\mathscr{L} = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$,

$$\boldsymbol{\nabla} f(\boldsymbol{x}^*) = -\lambda \boldsymbol{\nabla} g(\boldsymbol{x}^*). \tag{6.39}$$

Here λ is the Lagrange multiplier. Equation (6.39) says that the gradients of f and g are proportional to each other.
Now consider the following example. Minimise $f(x_1, x_2) = x_1$ subject to the constraint $g(x_1, x_2) = x_1^3 - x_2^2 = 0$. The global minimum of this constrained optimisation problem is at the origin, $\boldsymbol{x}^* = [0,0]^\mathsf{T}$. But at this point we have:

$$\boldsymbol{\nabla} f(\boldsymbol{x}^*) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \text{while} \quad \boldsymbol{\nabla} g(\boldsymbol{x}^*) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{6.40}$$

This is not a solution of Equation (6.39). But note that this is quite a special case. The problem occurs because $\boldsymbol{\nabla} f(\boldsymbol{x}^*) \neq 0$, but the gradient of g vanishes at $\boldsymbol{x}^*$ since the constraint has a cusp at this point.[2]

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

$$\frac{\partial \mathscr{L}}{\partial x_1} = (2\lambda x_1 + 1), \quad \frac{\partial \mathscr{L}}{\partial x_2} = (2\lambda - 1)x_2, \quad \frac{\partial \mathscr{L}}{\partial \lambda} = x_1^2 + x_2^2 - 1. \tag{6.41}$$

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

$$x_1^* = \mp 1, \quad x_2^* = 0, \quad \lambda^* = \pm\frac{1}{2}. \tag{6.42}$$

[2] Nonnenmacher, *Lagrange multipliers can fail to determine extrema*, College Mathematics Journal, January 2003, Taylor & Francis.

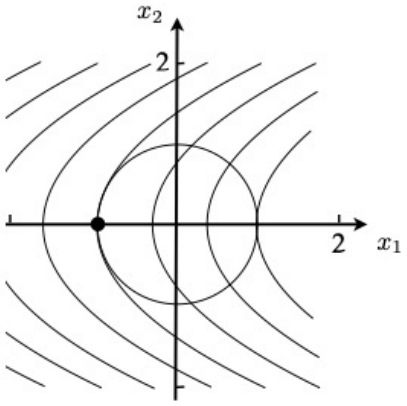


Figure 6.5: Contours of the function $f(x_1, x_2)$ and the constraint $x_1^2 + x_2^2 = 1$. Exercise 6.11.

One finds the same solutions if one starts instead from $\mathscr{L} = x_1 - x_2^2/2 - \lambda(x_1^2 + x_2^2 - 1)$. Substituting these solutions into $f(\boldsymbol{x})$, one finds 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.
Concerning the geometric picture underlying the method,[3] note that Equation (6.41) ensures that the gradients of $f(\boldsymbol{x}) = x_1 - x_2^2/2$ and of $x_1^2 + x_2^2 - 1$ w.r.t. $\boldsymbol{x}$ are proportional when the constraint is satisfied, at $x_1^2 + x_2^2 - 1 = 0$. The Lagrange multiplier is the constant of proportionality between the two gradients.

[3] Nonnenmacher, *Lagrange multipliers can fail to determine extrema*, College Mathematics Journal, January 2003, Taylor & Francis.

7 Solutions for 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 $[0,\ldots,0,1]^{\mathsf{T}}$ and $[0,\ldots,0,1]$ 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 minimum for our problem, 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 $\boldsymbol{w}_j$ ($j = 0,1,2,3$) 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, equal to $2(\delta - 1)$ (see Section 7.1). The intersections of the decision boundaries with the x_2-axis are determined by Equation (5.13). The 4-digit codes shown in Figure 7.1 describe the output of the hidden neurons. Inserting these into the function $O = \mathrm{sgn}(-V_0 + V_1 + V_2 - V_3)$, shows that $W_0 = -1$, $W_1 = 1$, $W_2 = 1$, $W_3 = -1$ and $\Theta = 0$ solve the classification problem.

Answer (7.3) — Figure 7.2(**a**) shows the r.m.s. error $\delta_{\mathrm{rms}}^{(\ell)} = \frac{1}{N}\sum_{j=1}^{N}[\delta_j^{(\ell)}]^2$ for different layers ℓ as a function of the number of training epochs, for the network from Figure 7.9, with $N = 5$ neurons per layer. All activation functions are $g(b) = \tanh(b)$. The network is trained on the iris data set [70]. The weights were initialised as independent Gaussian random numbers with mean zero and variance $\sigma_w^2 = 0.1/N$. During the first 500 epochs, the gradients are small. Their magnitude decreases exponentially as one moves away from the output layer because the maximal Lyapunov exponent (7.21) is negative for the given weight initialisation.

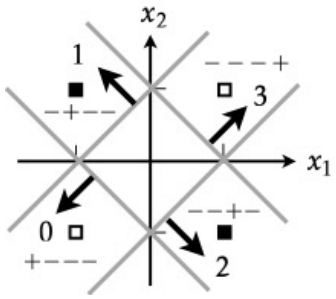


Figure 7.1: Shows the solution of XOR problem discussed in Exercise 7.2, for $\delta = 2$ [Equation (7.3)]. Exercise 7.2.

After 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 becomes better at classifying the training set.
The vanishing-gradient phase appears in Figure 7.2(**a**) because the initial random weights were too small, giving rise to a very negative maximal Lyapunov exponent. For a slightly larger variance σ_w^2, the vanishing-gradient problem is less severe, Figure 7.2(**b**).
However, increasing the variance σ_w^2 too much causes the gradients to explode. Ideally, one should determine the variance σ_w^2 such that the maximal Lyapunov exponent of the network vanishes. For $N \to \infty$, this can be ensured using Equation (**7.25**). But for small values of N, fluctuations of the singular values $\Lambda_j^{(t)}$ [Equation (**7.20**)] may still cause convergence problems.

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

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

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 $\ell \leq L-1$. This recursion reflects that every neuron $\ell \leq L-2$ 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 $\delta^{(L-3)}$:

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

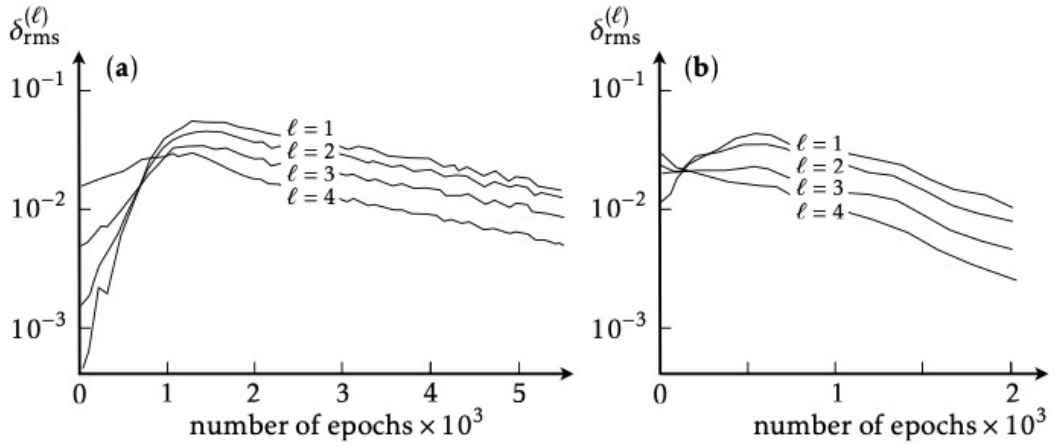


Figure 7.2: Vanishing-gradient problem for the network shown in Figure 7.9 with activation functions $g(b) = \tanh(b)$. (**a**) Root-mean-square error $\delta^{(\ell)}_{\text{rms}}$ in layer ℓ as a function of the number of training epochs. The weights were initialised as independent Gaussian random numbers with mean zero and variance $\sigma^2_w = 0.1/N$ with $N = 5$. (**b**) Same, but for $\sigma^2_w = 1/N$. Schematic, after simulations results obtained by Ludvig Storm. Exercise 7.3.

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. They visit only one 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 $-\log z \geq -(z-1)$, we have

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

Now note that $-1 \leq O_i^{(\mu)} \leq 1$ and $t_i^{(\mu)} = \pm 1$. This implies that $1 - t_i^{(\mu)}O_i^{(\mu)} \geq 0$, and therefore $H \geq 0$.

Answer (7.7) — To simplify the notation, leave out the superscript 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 unity, $\sum_i t_i^{(\mu)} = 1$.

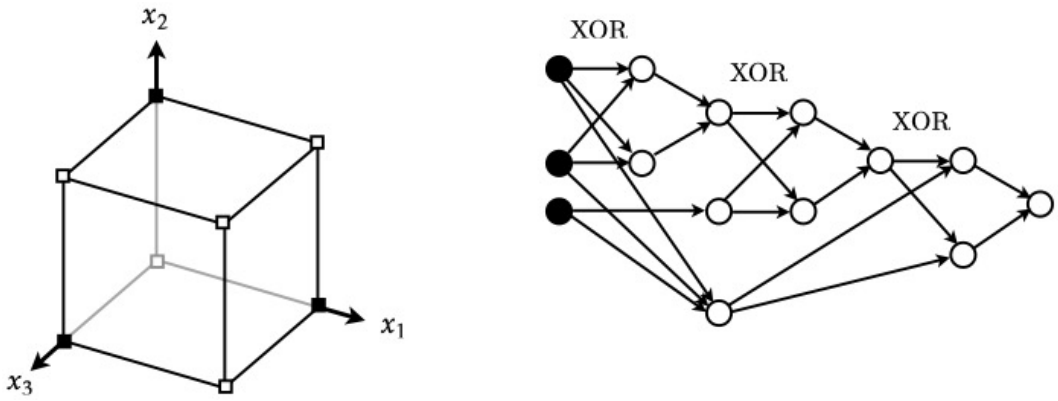


Figure 7.3: Input space and network layout for the three-dimensional exclusive XOR problem. Exercise 7.8.

Answer (7.8) — The input-space representation of the three-dimensional generalised XOR function is shown in Figure 7.3. The problem 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. In Figure 7.3, 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.

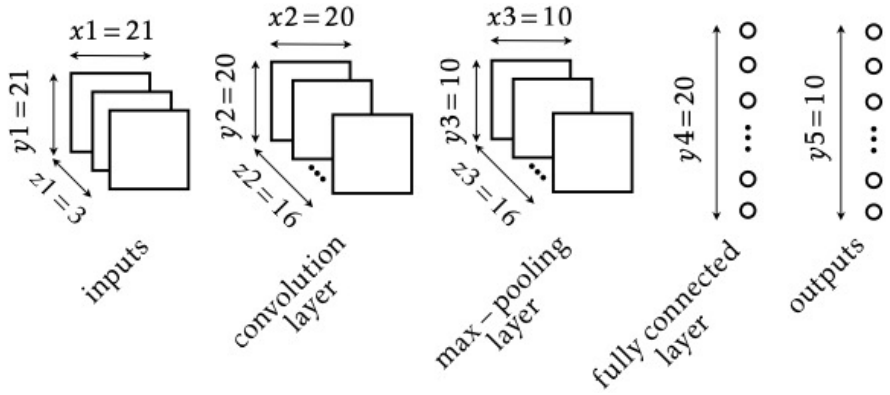


Figure 8.1: Parameter values for the network described in Exercise 8.1.

8 Solutions for exercises in Chapter 8

Answer (8.1) — The roles of the different layers of the network shown in Figure **8.13** are described in Sections **8.1** and **8.2**.
The second part of the solution is summarised in Figure 8.1. The input dimensions x1= 21, y1= 21, and z1= 3 are given in the question. To obtain the remaining dimensions, note that the convolution layers have dimensions 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.
Finally, 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) — The answer 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:

$$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 3 & 3 \\ 3 & 2 & 2 \\ 3 & 2 & 2 \\ 3 & 2 & 1 \end{bmatrix}, \quad \begin{bmatrix} 3 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 2 & 3 \\ 3 & 3 & 3 \\ 3 & 2 & 2 \end{bmatrix} \tag{8.1}$$

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

$$\begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}, \quad \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix}. \tag{8.2}$$

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

$$\mathbb{W} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} 10 \\ -10 \end{bmatrix}. \tag{8.3}$$

For input '2', this gives

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} - \begin{bmatrix} 10 \\ -10 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}. \tag{8.4}$$

For input '8'one finds:

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \\ 4 \end{bmatrix} - \begin{bmatrix} 10 \\ -10 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}. \tag{8.5}$$

The first output component is positive for input '8', while the second output component is positive for input '2'. We conclude that the network manages to distinguish the two inputs.

Answer (8.4) — The network layout used for this task 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 $\eta = 0.01$. 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.

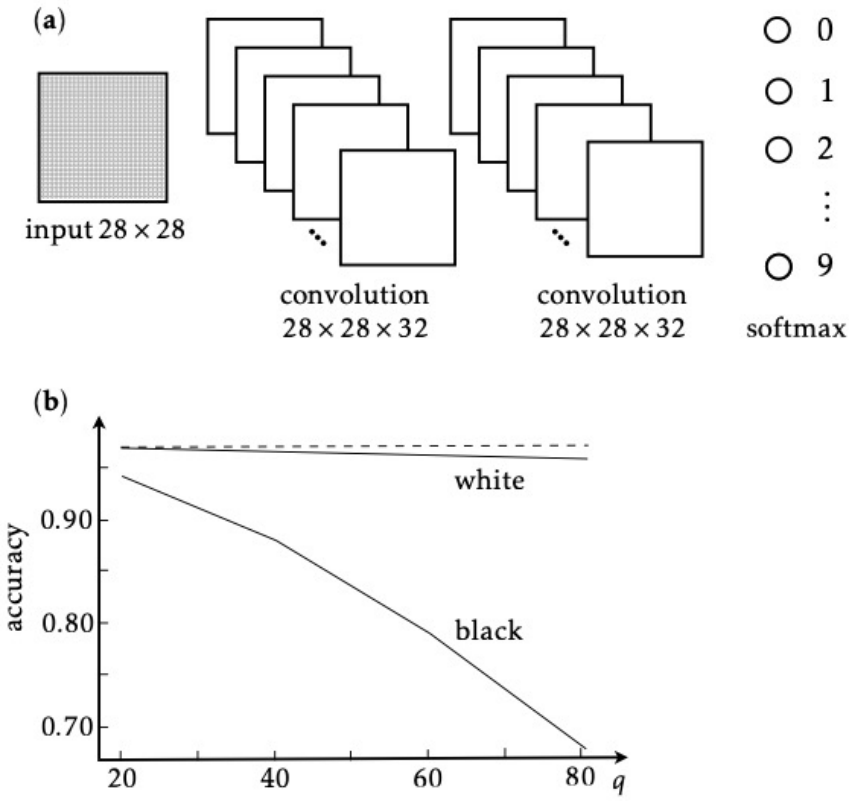


Figure 8.2: (**a**) Layout of the convolutional network for Exercise 8.4. (**b**) Accuracy as a function of the noise strength q. Also shown is the accuracy without noise (dashed line), as a benchmark. Schematic, after numerical results obtained by Hampus Linander. Exercise 8.4.

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**).

However, if one compares accuracy for similar numbers of changed pixels, there is still a significant difference (not shown), possibly because the black noise introduces black pixels in the border region, which may confuse 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

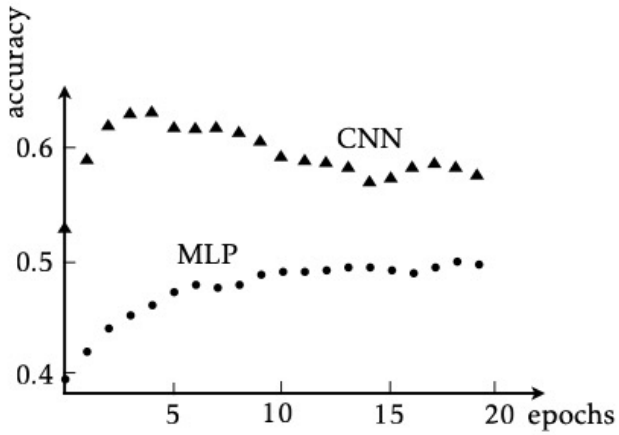


Figure 8.3: Validation accuracy for the CIFAR-10 data set [132] for a multi-layer perceptron (MLP, •) with two hidden layers with 105 neurons each, and a softmax-output layer with 10 units, and for a convolutional network (CNN, ▲) with two convolutional layers with 3×3 kernels and a fully connected softmax-output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

10 units (337834 trainable parameters). Both networks were trained on CIFAR-10 using adaptive learning rate with momentum[2] without any regularisation, using learning rate $\eta = 0.001$ and mini-batch size $m_{\mathrm{B}} = 128$.[3]
Without regularisation, the convolutional network overfits after around five epochs of training. Still, the convolutional 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

$$\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}. \tag{8.6}$$

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. Note that the bits of bar-and-stripe patterns have the values ± 1 (Figure 4.4). 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

[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] These values of η and m_{B} were chosen by trial and error, compromising between reasonable training times and accuracy.

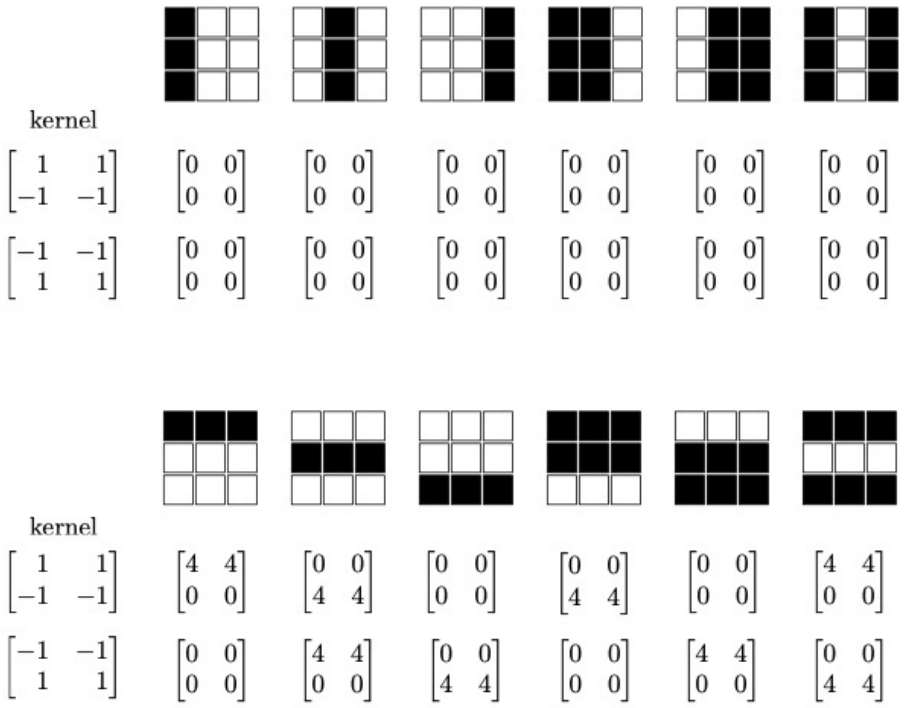


Figure 8.4: Result of applying the first two kernels in Equation (8.6) to bar and to stripe patterns, using ReLU activation functions (**1.8**). Exercise 8.6.

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, $O = \text{sgn}(V_1 + V_2 - V_3 - V_4)$.

Answer (8.7) — Starting from a trained convolutional model (same as network architecture and training parameters as in Exercise 8.5), the feature map in convolution layer ℓ is extracted as $V^{(\ell)}_{ijk}(\boldsymbol{x})$ for an input image $\boldsymbol{x}$. Starting with a uniform random input $\boldsymbol{x}$ of size $32 \times 32 \times 3$ (the CIFAR RGB-image tensor size), stochastic gradient ascent is used to maximize the average activation $V^{(\ell)}_k(\boldsymbol{x}) = \sum_{ij} V^{(\ell)}_{ijk}(\boldsymbol{x})$ with respect to the input $\boldsymbol{x}$. Since this optimisation procedure does not know about the limits of pixel values (e.g. $[0,1]$ or $[0,255]$), it is necessary to clip the components of the new $\boldsymbol{x}$ after each step.

Using an initial learning rate of $\eta = 0.01$ and performing 500 steps of gradient ascent using the Adam optimiser[4], the process is repeated for different choices of layer ℓ and feature map k. A similar procedure is used to maximise a certain component of the softmax-output layer.

Figure 8.5 shows the results of this procedure. Panel (**a**) shows the optimised input

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

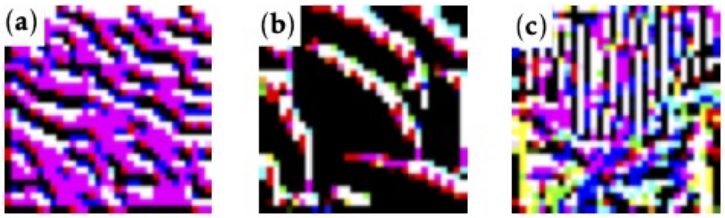


Figure 8.5: Input images maximising activations in different layers of the trained network shown in Figure 8.2(**a**), for feature map $k = 5$ in layer $\ell = 1$ [panel (**a**)], for $k = 5$ and $\ell = 2$ [panel (**b**)], and for the 9:th softmax output [panel (**c**)]. Numerical results obtained by Hampus Linander. Exercise 8.7.

for feature map $k = 5$ of the first convolution layer. We see that this feature map looks for straight patterns in the top-left to bottom-right direction. Panel (**b**) shows the optimised input for feature map $k = 5$ of the second convolution layer. This feature map looks for patterns with varying orientations. We see that second convolution layer can utilise features found in the first layer to build more expressive representations (varying orientations in this case). Panel (**c**) shows an image that maximises the output for the 9:th softmax-output layer, corresponding to the class *ship* of CIFAR-10. Even though this image does not look like a ship at all, it is still classified as such by this network (see Section **8.6** and Refs. [**5,128**]).

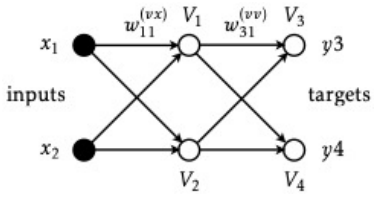


Figure 9.1: Multi-layer feedforward network obtained by removing the feedback connection in Figure 9.1. Exercise 9.1.

9 Solutions for 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 in Figure 9.1. 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 equal to zero otherwise). Writing out the matrix $\mathbb{L}$ for the network shown in Figure 9.1, we find

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

The inverse of $\mathbb{L}$ is

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

(this can be checked by matrix multiplication). Inserting $\mathbb{L}^{-1}$ into Equation (9.3), we find

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

Substituting this result into Equation (**9.14**) yields

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

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 **6.1**.
In a similar fashion one obtains learning rules for the hidden weights in the feed-forward network shown in Figure 9.1. Substituting Equation (9.6) into (**9.15**) yields

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

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

for the hidden weights. This is the update rule (**6.9**) for the hidden weights. Since all eigenvalues of $\mathbb{L}$ are equal to 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 in the steady state. This yields Equation (**9.6**),

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

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 eigenvalues of $\mathbb{L}$ are positive. If this is not the case, the steady state is a saddle point and training fails. See also Exercise 2.10.

Answer (9.3) — We begin by deriving the learning rule (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 the learning rule (9.27). We find:

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

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:

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

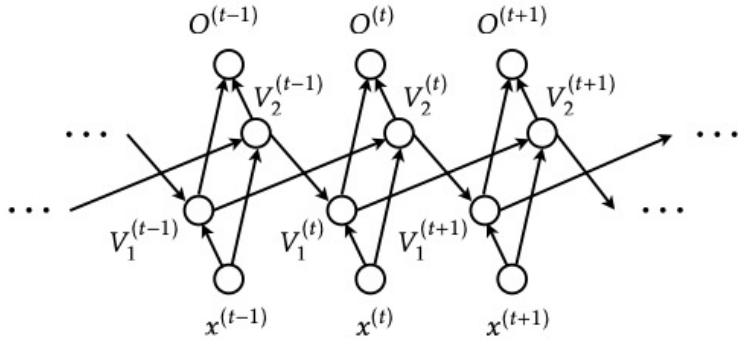


Figure 9.2: Network from Figure 9.8 unfolded in time.

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)].

Answer (9.5) — The unfolded network is shown in Figure 9.2. The update rule for the output is given by Equation (9.19b):

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

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

$$V_1^{(t)} = g\left(w_1^{(vx)} x^{(t)} + w_{12}^{(vv)} V_2^{(t-1)} - \theta_1^{(v)}\right), \tag{9.19}$$

where $x^{(t)}$ is the input at time t, and $w_{ij}^{(vv)}$ are the hidden weights. Note that the hidden neurons have no self connections in Figure 9.2, $w_{ii}^{(vv)} = 0$.
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. The gradient-descent learning rule reads:

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

Here, $B^{(t)} = \sum_{j=1}^{2} w_j^{(ov)} V_j^{(t)} - \theta^{(o)}$ 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 $w_{mn}^{(vv)}$:

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

The next step is to derive a recursion formula for $\partial V_j^{(t)} / \partial w_{mn}^{(vv)}$:

$$\begin{aligned} \frac{\partial V_j^{(t)}}{\partial w_{mn}^{(vv)}} &= g'(b_j^{(t)}) \frac{\partial}{\partial w_{mn}^{(vv)}} \left(\sum_k w_{jk}^{(vv)} V_k^{(t-1)} + \sum_l w_{jl}^{(vx)} x_l^{(t)} \right) \\ &= g'(b_j^{(t)}) \left(\delta_{jm} V_n^{(t-1)} + \sum_k w_{jk}^{(vv)} \frac{\partial V_k^{(t-1)}}{\partial w_{mn}^{(vv)}} \right). \end{aligned} \tag{9.24}$$

Here $b_i^{(t)} = w_i^{(vx)} x^{(t)} + \sum_{j=1}^{2} w_{ij}^{(vv)} V_j^{(t-1)} - \theta_i^{(v)}$ is the local field of $V_i^{(t)}$. We iterate this recursion from $t = 1$ to see whether any pattern emerges. The first iteration yields:

$$\frac{\partial V_j^{(1)}}{\partial w_{mn}^{(vv)}} = g'(b_j^{(1)}) \delta_{jm} V_n^{(0)}. \tag{9.25}$$

Here we used $\partial V_i^{(0)}/\partial w_{mn}^{(vv)} \equiv 0$. Iterating, we find:

$$\frac{\partial V_j^{(2)}}{\partial w_{mn}^{(vv)}} = g'(b_j^{(2)})\delta_{jm} V_n^{(1)} + g'(b_j^{(2)}) w_{jm}^{(vv)} g'(b_m^{(1)}) V_n^{(0)}, \tag{9.26a}$$

$$\begin{aligned}\frac{\partial V_j^{(3)}}{\partial w_{mn}^{(vv)}} &= g'(b_j^{(3)})\delta_{jm} V_n^{(2)} + g'(b_j^{(3)}) w_{jm}^{(vv)} g'(b_m^{(2)}) V_n^{(1)} \\ &\quad + g'(b_j^{(3)}) \sum_k w_{jk}^{(vv)} g'(b_k^{(2)}) w_{km}^{(vv)} g'(b_m^{(1)}) V_n^{(0)},\end{aligned} \tag{9.26b}$$

$$\begin{aligned}\frac{\partial V_j^{(4)}}{\partial w_{mn}^{(vv)}} &= g'(b_j^{(4)})\delta_{jm} V_n^{(3)} + g'(b_j^{(4)}) w_{jm}^{(vv)} g'(b_m^{(3)}) V_n^{(2)} \\ &\quad + g'(b_j^{(4)}) \sum_k w_{jk}^{(vv)} g'(b_k^{(3)}) w_{km}^{(vv)} g'(b_m^{(2)}) V_n^{(1)} \\ &\quad + g'(b_j^{(4)}) \sum_k w_{jk}^{(vv)} g'(b_k^{(3)}) \sum_l w_{kl}^{(vv)} g'(b_l^{(2)}) w_{lm}^{(vv)} g'(b_m^{(1)}) V_n^{(0)},\end{aligned} \tag{9.26c}$$

and so forth. In the sum over t in Equation (9.23), we regroup the terms as described on page **164**. We start from

$$\frac{\partial H}{\partial w_{mn}^{(vv)}} = -\sum_j w_j^{(ov)} \left[\Delta^{(1)} \frac{\partial V_j^{(1)}}{\partial w_{mn}^{(vv)}} + \Delta^{(2)} \frac{\partial V_j^{(2)}}{\partial w_{mn}^{(vv)}} + \Delta^{(3)} \frac{\partial V_j^{(3)}}{\partial w_{mn}^{(vv)}} + \ldots \right], \tag{9.27}$$

and collect terms multiplied by $V_n^{(T-1)}$, $V_n^{(T-2)}$, $V_n^{(T-3)}$,...:

$$\begin{aligned}\delta w_{mn}^{(vv)} = \eta \Bigg\{ & w_m^{(ov)} \Delta^{(T)} g'(b_m^{(T)}) V_n^{(T-1)} \\ & + \left[w_m^{(ov)} \Delta^{(T-1)} g'(b_m^{(T-1)}) + \sum_j w_j^{(ov)} \Delta^{(T)} g'(b_j^{(T)}) w_{jm}^{(vv)} g'(b_m^{(T-1)}) \right] V_n^{(T-2)} \\ & + \Bigg[w_m^{(ov)} \Delta^{(T-2)} g'(b_m^{(T-2)}) + \sum_j \Big[w_j^{(ov)} \Delta^{(T-1)} g'(b_j^{(T-1)}) w_{jm}^{(vv)} g'(b_m^{(T-2)}) \\ & \qquad + \sum_j w_j^{(ov)} \Delta^{(T)} g'(b_j^{(T)}) \sum_k w_{jk}^{(vv)} g'(b_k^{(T-1)}) w_{km}^{(vv)} g'(b_m^{(T-2)}) \Big] V_n^{(T-3)} \\ & + \ldots \Bigg\}.\end{aligned} \tag{9.28}$$

This learning rule can be written recursively as Equation (**9.29**).
A similar recursion holds for the errors in the learning rule for the input weights $w_{mn}^{(vx)}$. The only difference is that $V_n^{(t-1)}$ is replaced by $x_n^{(t)}$.

Finally, consider the specific case of updating the weights leading to hidden neuron $j=1$ (note that there are no self connections in Figure 9.2, so the diagonal weights vanish, $w_{ii}^{(vv)}=0$). We find:

$$\delta_1^{(t)} = \begin{cases} \Delta^{(T)} w_1^{(ov)} g'(b_1^{(T)}) & \text{for } t=T, \\ \Delta^{(T)} w_1^{(ov)} g'(b_1^{(t)}) + \delta_2^{(t+1)} w_{21}^{(vv)} g'(b_1^{(t)}) & \text{for } 0<t<T\,, \end{cases} \tag{9.29a}$$

and

$$\delta_2^{(t)} = \begin{cases} \Delta^{(T)} w_2^{(ov)} g'(b_2^{(T)}) & \text{for } t=T, \\ \Delta^{(T)} w_2^{(ov)} g'(b_2^{(t)}) + \delta_1^{(t+1)} w_{12}^{(vv)} g'(b_2^{(t)}) & \text{for } 0<t<T\,. \end{cases} \tag{9.29b}$$

For $0<t<T$, the current error $\delta_1^{(2)}$ only depends only on $\delta_2^{(t+1)}$, and vice versa. This follows from the fact that there are no self connections. The recursions (9.29) can be written as two-step recursions. For $\delta_1^{(t)}$, for example, we use that that for $t<T-1$, $\delta_2^{(t+1)}$ depends on $\delta_1^{(t+2)}$. This gives the following recursion for $\delta_1^{(t)}$:

$$\delta_1^{(t)} = \begin{cases} \Delta^{(T)} w_1^{(ov)} g'(b_1^{(T)}) & \text{for } t=T, \\ \Delta^{(T-1)} w_1^{(ov)} g'(b_1^{(T-1)}) + \Delta^{(T)} w_2^{(ov)} g'(b_2^{(T)}) w_{21}^{(vv)} g'(b_1^{(T-1)}) & \text{for } t=T-1, \\ \Delta^{(t)} w_1^{(ov)} g'(b_1^{(t)}) + \Big[\Delta^{(t+1)} w_2^{(ov)} g'(b_2^{(t+1)}) & \\ \qquad + \delta_1^{(t+2)} w_{12}^{(vv)} g'(b_2^{(t+1)})\Big] w_{21}^{(vv)} g'(b_1^{(t)}) & \text{for } 0<t<T-1. \end{cases}$$

A corresponding two-step recursion for $\delta_2^{(t)}$ can be obtained. While the more general recursion relation (**9.29**) is easier to implement, the somewhat more complicated recursion (9.30) reveals the temporal structure of the network: an error takes two steps to return.

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

$$\delta\theta_m^{(v)} = -\eta \sum_{t=1}^{T} \delta_m(t) \quad \text{and} \quad \delta\theta_m^{(o)} = -\eta \sum_{t=1}^{T} \Delta_m(t). \tag{9.30}$$

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

$$Y_j^* - \sum_k Y_k^* g'(b_k^*)] w_{kj}^{(vv)} = E_j^*, \tag{9.31}$$

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

$$\sum_k Y_k^* \mathbb{L}_{kj} = E_j^* . \tag{9.32}$$

According to Equation (**9.9**), $\mathbb{L} = \delta_{kj} - g'(b_k^*) w_{kj}^{(vv)}$. This shows that (9.31) and (9.32) 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 $\Lambda_\alpha = \sqrt{2} = |\nu_\alpha|$. So the singular values equal the absolute values of the eigenvalues. 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:

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

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,[1] $\mathbb{B}(t) = \sum_\alpha \nu_\alpha^t \boldsymbol{R}_\alpha \overline{\boldsymbol{L}}_\alpha^\mathsf{T}$. Here $\overline{\boldsymbol{L}}^\mathsf{T}$ denotes the Hermitian transpose of $\boldsymbol{L}$. One finds $\mathbb{B}(t)^\mathsf{T}\mathbb{B}(t) = \sum_{\alpha,\beta} \nu_\alpha^t \overline{\nu}_\beta^t \boldsymbol{L}_\alpha (\overline{\boldsymbol{R}}_\alpha^\mathsf{T} \boldsymbol{R}_\alpha) \overline{\boldsymbol{L}}_\alpha$. 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 $\lambda_1^{(\mathrm{Ikeda})}$ [**151**]. It is difficult to predict the Ikeda time series for much longer than $1/\lambda_1^{(\mathrm{Ikeda})}$. For the parameters specified in the question, $\lambda_1^{(\mathrm{Ikeda})} \approx 0.24$. 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 $\lambda_1^{(\mathrm{Ikeda})} t \approx 5$.
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 $w_i^{(\mathrm{in})}$ were sampled from a normal distribution with mean zero and variance $\sigma_{\mathrm{in}}^2 = 1$. 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

[1] Chalker & Mehlig, Phys. Rev. Lett. **81** (1998) 3367, and references cited in this paper.

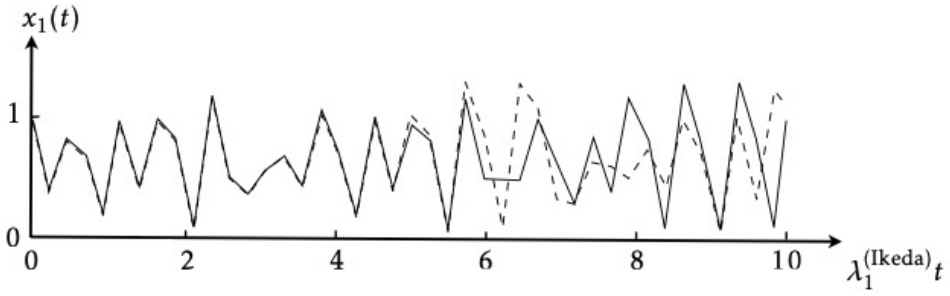


Figure 9.3: Time series generated using the Ikeda map (**9.34**), solid line. Prediction of the time series using a reservoir computer, dashed line. The maximal Lyapunov exponent of the Ikeda map is denoted by $\lambda_1^{(\mathrm{Ikeda})}$. Schematic, after numerical results obtained by Ludvig Storm. Exercise 9.9.

first 100 iterations were discarded. From the next T iterations, the energy function is computed. Using Equation (**9.35b**), it can be written as

$$H = \tfrac{1}{2}\sum_{t=0}^{T-1}[y(t)-O(t)]^2 = \tfrac{1}{2}\left\|\boldsymbol{y}-\mathbb{R}^{\mathsf{T}}\boldsymbol{w}^{(\mathrm{out})}\right\|^2 \tag{9.34}$$

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 $\boldsymbol{w}^{(\mathrm{out})}$. It was solved with ridge regression[2] with ridge parameter 0.001. Storm *et al.*[3] discuss general rules for choosing the parameters for successful reservoir computing.
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 $\boldsymbol{r}(T-1)$, as well as the predicted value $O(T) = \boldsymbol{w}^{(\mathrm{out})}\cdot\boldsymbol{r}(T)$, using Equation (**9.35b**). Figure 9.3 was obtained by iterating Equation (**9.35a**), always with $x(t) = O(t)$ and fixed output weights $\boldsymbol{w}^{(\mathrm{out})}$.

[2] See Section 1.5 of [W. N. van Wieringen, *Lecture notes on ridge regression*, arxiv:1509.09169].
[3] L. Storm, K. Gustavsson & B. Mehlig, *Constraints on parameter choices for successful time-series prediction with echo-state networks*, MLST **3** (2022) 045021.

10 Solutions for exercises in Chapter 10

Answer (10.1) — Write $\boldsymbol{q} = \alpha \boldsymbol{w}$ and assume that $\boldsymbol{w}$ 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). Note that Equation (10.5) describes the dynamics of the normalised orientation vector of a small rod in turbulence,[1] where $\mathbb{A}(t)$ is the matrix of fluid-velocity gradients.

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

$$\mathbb{C} = \frac{1}{9}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \tag{10.3}$$

This matrix has eigenvalues and eigenvectors

$$\lambda_1 = \tfrac{1}{9}, \quad \boldsymbol{u}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}, \quad \text{and} \quad \lambda_2 = \tfrac{1}{3}, \quad \boldsymbol{u}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}-1\\1\end{bmatrix} \tag{10.4}$$

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

Answer (10.3) — The solution to this exercise 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

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

[1] Wilkinson, Bezuglyy & Mehlig, Phys. Fluids **21** (2009) 043304.

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**) implies

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

We conclude that

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

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) — From a uniform distribution over the gray triangle in Figure 10.1, 2000 input points were independently sampled. The learning rule for a self-organising map with neighbourhood function (**10.18**) with a 10×10 output array was iterated for 2000 epochs (one epoch consists of feeding all 2000 input points). The algorithm was run with an iteration-dependent learning rate, and also the width of the neighbourhood function depended on the iteration number:

$$\eta_k = \eta_0 \exp(-k d_\eta), \quad \sigma_k = \sigma_0 \exp(-k d_\sigma). \tag{10.10}$$

Here the integer k counts the epochs, and $\eta_0 = 0.1, d_\eta = 0.005, \sigma_0 = 10$, as well as $d_\sigma = 0.005$. The result is shown in Figure 10.1, see also Figure **10.9**.

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 (10.10)]. The result is shown in Fig. 10.2. 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

$$SSR = \sum_{i=1}^{N} \big(y_i - \hat{y}_i\big)^2, \tag{10.11}$$

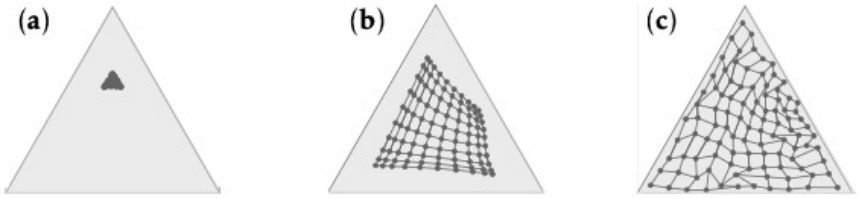


Figure 10.1: Convergence of self-organising map. (**a**) initial weights (•), (**b**) snapshot of weights at an intermediate stage, (**b**) weights after 4×10^6 iterations. The network of lines indicates the location of the corresponding neurons in the output array. The data distribution is $P_{\text{data}} = \mathscr{A}^{-1}$ inside the triangle and $P_{\text{data}} = 0$ outside, and $\mathscr{A}$ is the area of the triangle. Schematic, after numerical results obtained by Ludvig Storm. Exercise 10.6.

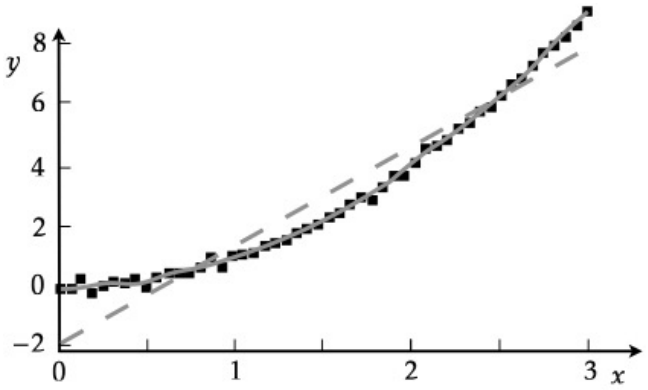


Figure 10.2: Principal component and principal manifold fitted to data described in Exercise 10.7. Shown is the principal component (gray dashed line) as well as the principal manifold (solid gray line). Schematic, after numerical results obtained by Ludvig Storm. Exercise 10.7.

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 10.3 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

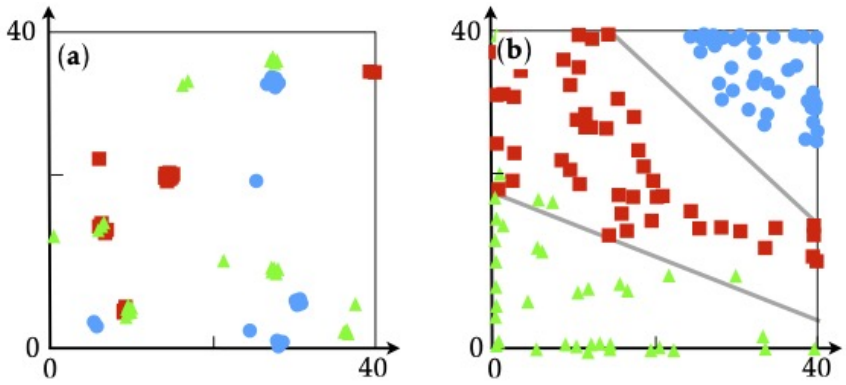


Figure 10.3: Classification of Iris data set (Figure 5.1) using a self-organising map with a 40×40 output array, and a perceptron with one hidden layer with two neurons and and an output layer with three neurons. (**a**) Initial output the self-organising map. Shown are the locations of the winning neurons in the output array. The corresponding classes are indicated by symbols, *iris setosa* (•), *iris versicolor* (■), and *iris virginica* (▲). (**b**) Final output of the self-organising map, and decision boundaries of the hidden neurons of the perceptron (gray lines). Schematic, after numerical results obtained by Ludvig Storm. Exercise 10.8.

and the width σ of the neighbourhood function were reduced during the learning,

$$\eta_k = \eta_0 \exp(-k d_\eta), \quad \sigma_k = \sigma_0 \exp(-k d_\sigma). \tag{10.12}$$

Here k counts epochs, and $\eta_0 = 0.1, d_\eta = 10^{-2}, \sigma_0 = 10$, as well as $d_\sigma = 0.05$. 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 10.3 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):

$$0 = \int \mathrm{d}\boldsymbol{r}_0 |\det \mathbb{J}| Q(\boldsymbol{r}_0) h(\boldsymbol{r}-\boldsymbol{r}_0)\big[\boldsymbol{w}^*(\boldsymbol{r}_0) - \boldsymbol{w}^*(\boldsymbol{r})\big]. \tag{10.13}$$

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

$$\int \mathrm{d}\delta\boldsymbol{r}\,\delta r_i \delta r_j\, h(\delta\boldsymbol{r})\Big[\partial_i \boldsymbol{w}^* \partial_j (JQ) + \tfrac{1}{2} JQ \partial_i \partial_j \boldsymbol{w}^*\Big], \tag{10.14}$$

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

$$\int \mathrm{d}\delta\boldsymbol{r}\,\delta r_i \delta r_j\, h(\delta\boldsymbol{r}) = \sigma^2 \delta_{ij}\,, \tag{10.15}$$

Equation (10.14) is equivalent to

$$\partial_i \boldsymbol{w}^* \left(\frac{\partial_i Q}{Q} + \frac{\partial_i J}{J} \right) = -\tfrac{1}{2} \partial_i^2 \boldsymbol{w}^*\,. \tag{10.16}$$

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, and defining $J_i \equiv \partial_i w_i^*$, Equation (10.16) splits into two conditions

$$\partial_1 w_1^* \left(\frac{\partial_1 Q_1}{Q_1} + \frac{\partial_1 J_1}{J_1} \right) = -\tfrac{1}{2} \partial_1^2 w_1^*\,, \quad \partial_2 w_2^* \left(\frac{\partial_2 Q_2}{Q_2} + \frac{\partial_2 J_2}{J_2} \right) = -\tfrac{1}{2} \partial_2^2 w_2^*\,, \tag{10.17}$$

where $Q = Q_1 Q_2$. 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 (10.16) vanishes. As a consequence, it follows from Equation (10.16) that the distributions are equal, subject to normalisation: $\varrho = |\det \mathbb{J}|^{-1} \propto P_{\text{data}}$.

Answer (10.10) — See Ref. [**2**]. Figure 10.4 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

$$\boldsymbol{u}^{(1)} \approx \begin{bmatrix} 1 \\ 0.14 \end{bmatrix}, \quad \boldsymbol{u}^{(2)} \approx \begin{bmatrix} 0.37 \\ 0.37 \end{bmatrix}, \quad \boldsymbol{u}^{(3)} \approx \begin{bmatrix} 0.37 \\ 0.37 \end{bmatrix}, \quad \boldsymbol{u}^{(4)} \approx \begin{bmatrix} 0.14 \\ 1 \end{bmatrix}. \tag{10.18}$$

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.

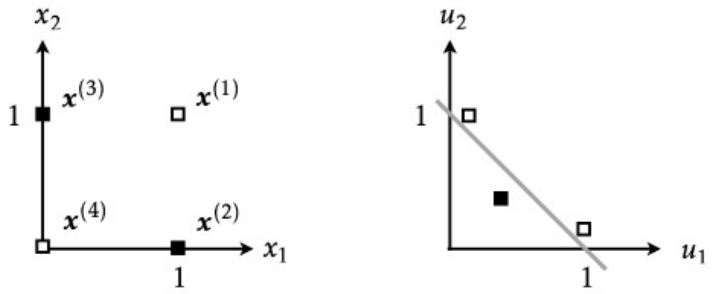


Figure 10.4: Left: input plane for Exercise 10.10. Right: mapped problem in the u_1-u_2-plane and corresponding decision boundary for Exercise 10.10. After Fig. 5.2 in Ref. [2].

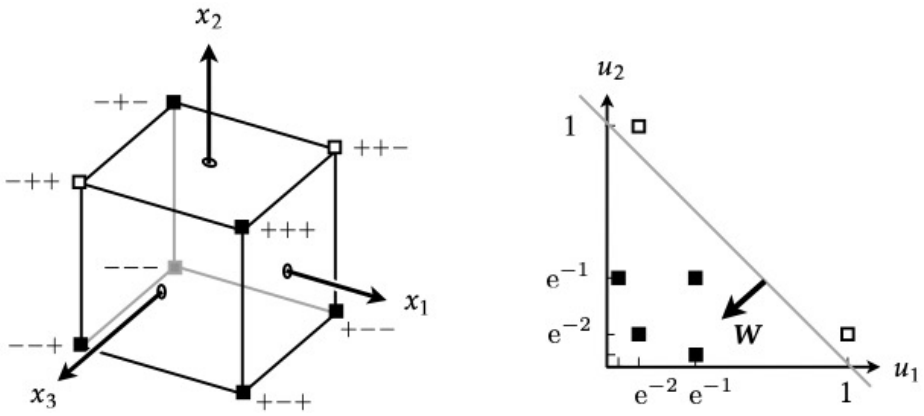


Figure 10.5: Left: input plane for Exercise 10.11. Right: Right: mapped problem in the u_1-u_2-plane, weight vector $\boldsymbol{W}$, and corresponding decision boundary for Exercise 10.11.

Answer (10.11) — Figure 10.5 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 10.5. In the new coordinates, the problem is linearly separable with $\boldsymbol{W} = [-1,-1]^{\mathsf{T}}$ and $\Theta = -1$.

Answer (10.12) — Figure 10.6 shows the decision boundaries in the x_1-x_2-plane for for radial basis-function networks with different numbers of radial basis functions. The resulting classification accuracies are shown in Table 10.1. To obtain these results, 1000 inputs were generated. Algorithm **10** was iterated for 100 steps with learning rate $\eta = 10^{-3}$. Figure 10.6(**d**) shows that the algorithm overfits for 100 radial basis functions.

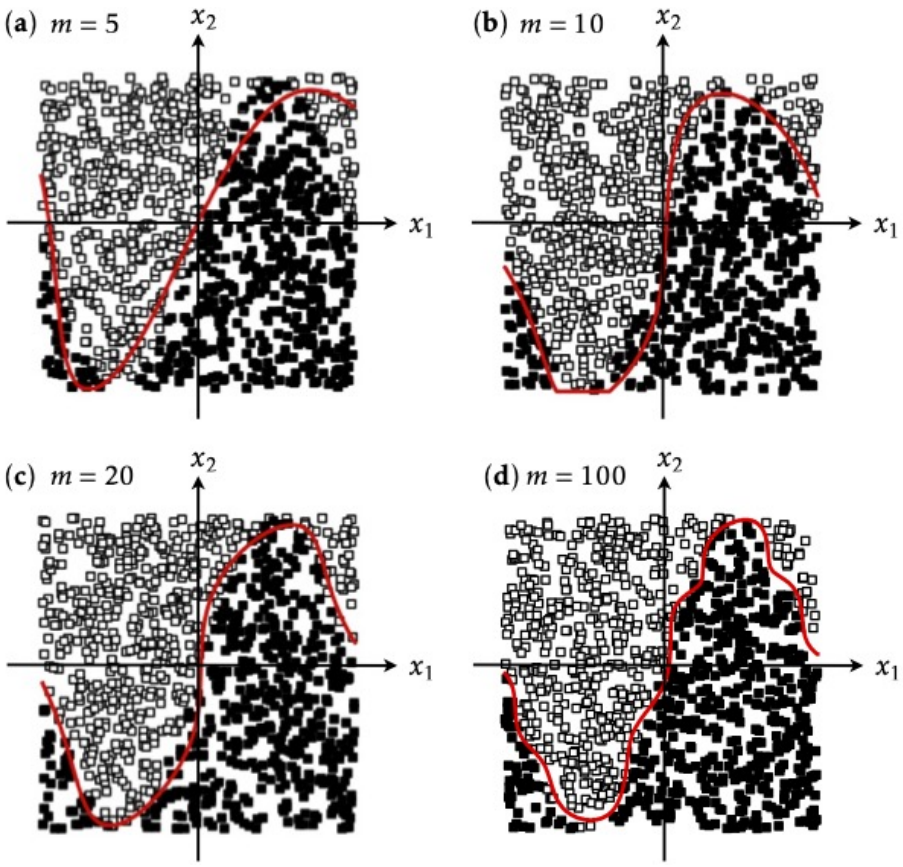


Figure 10.6: Decision boundaries (solid red lines) for Exercise 10.12, for different numbers m of radial basis functions. The corresponding classification accuracies are shown in Table 10.1. Schematic, after numerical results obtained by Ludvig Storm. Exercise 10.12.

Table 10.1: Classification accuracy for Exercise 10.12 for different numbers of radial basis functions. Numerical results obtained by Ludvig Storm.

m	5	10	20	100
Accuracy in %	90.6	93.1	95.6	98.6

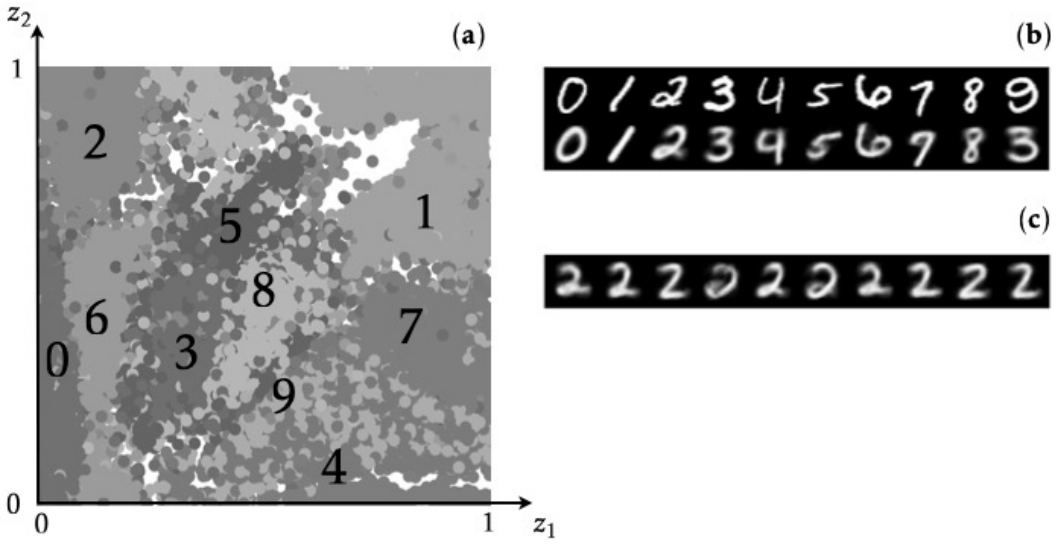


Figure 10.7: (**a**) Representation of the MNIST digits in the latent plane. (**b**) Examples of input-output pairs. The inputs are in the top row, the corresponding outputs in the bottom row. (**c**) Artificial digits generated by sampling points from a cluster in latent space (here: top left corner), and then applying the decoder. Numerical results obtained by Anshuman Dubey. Exercise 10.13.

Answer (10.13) — The result of training the autoencoder from Figure **10.18** on the MNIST data set is shown in Figure 10.7. All neurons have 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.[2]

Figure 10.7(**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 10.7(**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.

This is precisely what variational autoencoders are designed to do (page **199**). Numerical simulations performed by A. Wenzel Wartenberg show that for roughly equal numbers of trainable parameters, the variational autoencoder tends to achieve better separation of the clusters in latent space, compared with the deterministic autoencoder considered in this exercise.

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

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

$$H = \tfrac{1}{2} \left\| \mathbb{X} - \mathbb{W}_\mathrm{d} \mathbb{W}_\mathrm{e} \mathbb{X} \right\|^2 \tag{10.19}$$

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 at most 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 $\mathbb{X}$,

$$\mathbb{X} = \mathbb{U} \mathbb{S} \mathbb{V}^\mathsf{T} , \tag{10.20}$$

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

$$\mathbb{X}^{(2)} = \mathbb{U} \mathbb{S}^{(2)} \mathbb{V}^\mathsf{T} , \tag{10.21}$$

where $\mathbb{S}^{(2)}$ is obtained from $\mathbb{S}$ by setting all singular values to zero except the largest two. Taking together Equations (10.20) and (10.21), $\mathbb{X}^{(2)} = \mathbb{W}_\mathrm{d} \mathbb{W}_\mathrm{e} \mathbb{X}$ 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,

$$\mathbb{W}_\mathrm{d} \mathbb{W}_\mathrm{e} = \mathbb{U} \mathbb{S}^{(2)} \mathbb{V}^\mathsf{T} \mathbb{V} \mathbb{S}' \mathbb{U}^\mathsf{T} , \tag{10.22}$$

where $\mathbb{S}'$ is obtained from $\mathbb{S}$ by taking the reciprocal of all non-zero diagonal elements. Equation (10.22) 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 entries equal to unity in the first two diagonal elements, and zero for all other matrix elements. If we choose $\mathbb{W}_\mathrm{d} = \mathbb{U} \mathbb{1}^{(2)}$ and $\mathbb{W}_\mathrm{e} = \mathbb{1}^{(2)} \mathbb{U}^\mathsf{T}$, 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. These are the two first columns of $\mathbb{U}$.

The derivation summarised above assumes zero thresholds. The corresponding calculations for non-zero thresholds are a bit more involved. They are described in Ref. [164].

11 Solutions for 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 $b_i^{(\mu)} = \sum_j w_{ij} x_j^{(\mu)}$. It follows that

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

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: $\langle H \rangle = \sum_{i\mu} \left(1 - t_i^{\mu} \langle y_i^{(\mu)} \rangle\right)$. 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 $\sum_{mn} \frac{\partial \langle H \rangle}{\partial w_{mn}} \delta w'_{mn}$. 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) — The calculation is described in Section 7.4 of Ref. [1]. In summary, one starts from Equation (11.5) for the average immediate reward given an input $\boldsymbol{x}$:

$$\frac{\partial \langle r \rangle}{\partial w_{mn}} = \sum_{y_1 = \pm 1, \ldots, y_M = \pm 1} \langle r(\boldsymbol{x}, \boldsymbol{y}) P(\boldsymbol{y}|\boldsymbol{x}) \rangle_{\text{reward}} = \sum_{y_1 = \pm 1, \ldots, y_M = \pm 1} \langle r(\boldsymbol{x}, \boldsymbol{y}) \frac{\partial}{\partial w_{mn}} P(\boldsymbol{y}|\boldsymbol{x}) \rangle_{\text{reward}} . \tag{11.5}$$

According to Equation (11.6),

$$P(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{M} \begin{cases} p(b_i) & \text{for} \quad y_i = 1 , \\ 1 - p(b_i) & \text{for} \quad y_i = -1 , \end{cases} \tag{11.6}$$

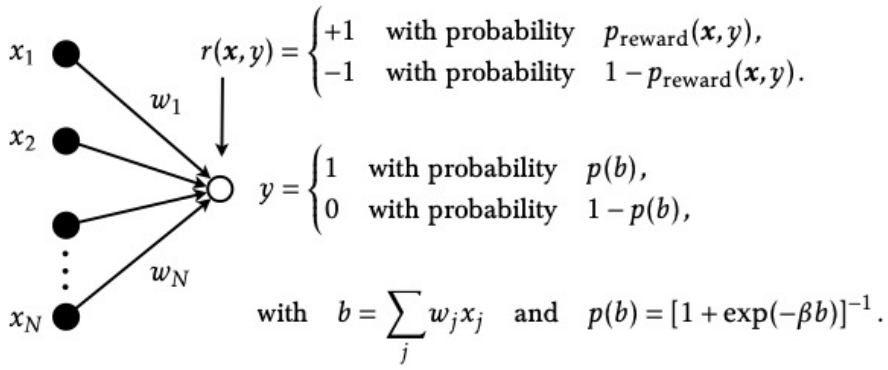


Figure 11.1: Stochastic binary neuron with 0/1 outputs. The learning rule is the same as in Exercise 4.2. Exercise 11.3.

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

$$\frac{\partial}{\partial w_{mn}} P(\boldsymbol{y}|\boldsymbol{x}) = \sum_i \left(\prod_{k\neq i} \begin{cases} p(b_k) & \text{for} \quad y_k = 1 \\ 1-p(b_k) & \text{for} \quad y_k = -1 \end{cases} \right) \begin{cases} p'(b_i)\delta_{im} x_n & \text{for} \quad y_i = 1, \\ -p'(b_i)\delta_{im} x_n & \text{for} \quad y_i = -1, \end{cases} \tag{11.7}$$

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

$$= P(\boldsymbol{y}|\boldsymbol{x}) x_n \begin{cases} p'(b_m)/p(b_m) & \text{for} \quad y_m = 1, \\ -p'(b_m)/[1-p(b_m)] & \text{for} \quad y_m = -1. \end{cases} \tag{11.8}$$

Now one makes use of the relations $p'/p = \beta(1-\tanh\beta b)$ and $-p'/(1-p) = -\beta(1+\tanh\beta b)$ to obtain

$$\frac{\partial}{\partial w_{mn}} P(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{y}|\boldsymbol{x})\beta[y_m - \tanh(\beta b_m)] x_n . \tag{11.9}$$

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**),

$$\frac{\partial \langle r \rangle}{\partial w_n} = \sum_{y=\pm 1} \langle r(\boldsymbol{x},y) P(y|\boldsymbol{x}) \rangle_{\text{reward}} = \sum_{y=\pm 1} \langle r(\boldsymbol{x},y) \frac{\partial}{\partial w_n} P(y|\boldsymbol{x}) \rangle_{\text{reward}}, \tag{11.10}$$

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

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

$$\frac{\partial\,\langle r\rangle}{\partial\, w_n} = \sum_{y=\pm 1} \langle r(\boldsymbol{x}, y) P(y|\boldsymbol{x}) \beta [y - p(b)]\rangle_{\text{reward}}\, x_n\,. \tag{11.11}$$

So the following learning rule increases the expected immediate reward,

$$\delta\, w_n = \alpha\, r[y - p(b)]x_n\,, \tag{11.12}$$

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. This is shown in Figure 11.2 for the embedding

$$\begin{bmatrix}0\\0\end{bmatrix} \to \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \quad \begin{bmatrix}1\\1\end{bmatrix} \to \begin{bmatrix}1\\1\\0\\1\end{bmatrix}, \quad \begin{bmatrix}0\\1\end{bmatrix} \to \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \quad \text{and} \quad \begin{bmatrix}1\\0\end{bmatrix} \to \begin{bmatrix}1\\0\\0\\0\end{bmatrix}. \tag{11.13}$$

The weights were initialised randomly, from a uniform distribution in $[0, 1]$.
Figure 11.2 demonstrates that the asymmetry parameter δ |Equation (**11.10**)] affects the convergence. The algorithm converges to $r_{\text{max}} = 0.6$ [Equation (**11.4**)] for $\delta = 0.3$. For $\delta = 0.1$, convergence is much slower. Simulations (not shown) indicate that the algorithm fails for $\delta = 0$.

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, so training fails. When $\varepsilon > 0$, the algorithm explores suboptimal state-action pairs that leads to higher rewards in the long run (exploitation-exploration dilemma, page **206**). The maximal expected reward $r_{\text{max}} = 15$ is shown as a dashed line in Figure 11.3. We see that the algorithm converges to r_{max} for $\varepsilon = 0.01$ and $\varepsilon = 0.1$, but convergence is faster for $\varepsilon = 0.1$.

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.

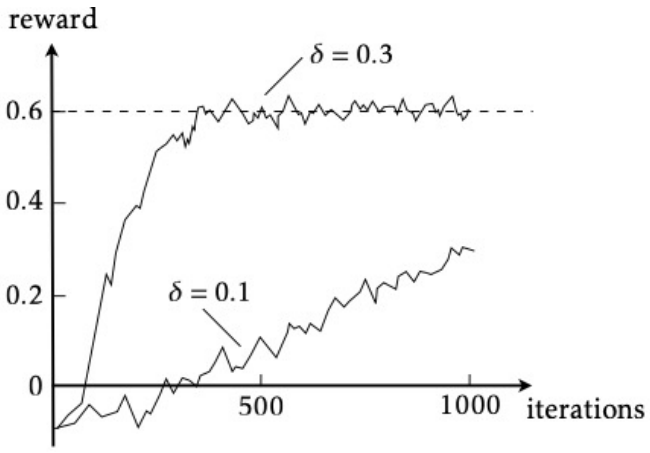


Figure 11.2: Associative reward-penalty algorithm for the XOR problem. The dashed line shows the maximal expected reward $r_{\max} = 0.6$, computed from Table **11.1**. The solid line shows the reward as a function of the number of iterations of Equation (**11.10**), obtained after embedding the four patterns from Table **11.1**, within four-dimensional input space so that they become linearly independent. Parameters: $\delta = 0.1, 0.3$ and $\alpha = 0.1$. Schematic, after data obtained by Navid Mousavi, averaged over independent realisations. Exercise 11.4.

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

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

where $k_{\max}$ is the total number of games played (number of episodes), and r_k is the reward received in game k. 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**).

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. It is necessary to take into account the outcome of the previous round because the decision tree of the opponent depends on this outcome (Figure 11.4).

One finds that $\langle R \rangle = 1$ when $p = q = 0$ [bottom left corner in Figure 11.4(**b**)], where the opponent always plays 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 in this case, we expect to receive $+1, 0, -1$ with probability $\frac{1}{2}$. So the reward averages to zero in this corner, $\langle R \rangle = 0$.

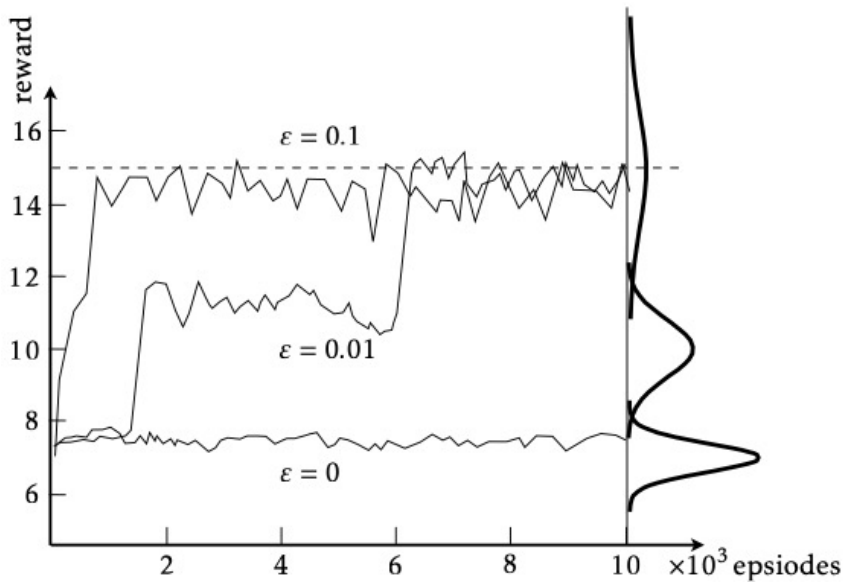


Figure 11.3: Three-armed bandit problem. The dashed line shows the maximal expected reward, $r_{\max} = 15$. The thick solid lines show the reward distributions from Figure **11.8**. The thin solid lines show the expected reward versus the number of learning episodes for three different values of ε defining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

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. As mentioned above, this is accounted for by defining the states to include your opponent's last move 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 Figure 11.4(**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

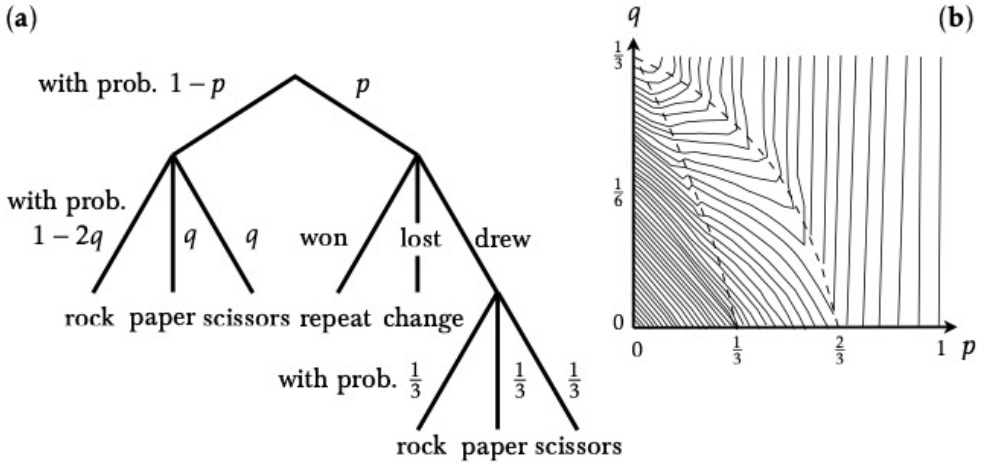


Figure 11.4: Rock-paper-scissors. (**a**) Decision tree of the opponent, as described in the problem formulation. (**b**) Shows contour lines of the average reward (11.14) as a function of p and q. Also shown are the critical values of the parameter q given by Eq. (11.16), dashed lines. The numerical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

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. Crowley & Siegler state that one should always take the action that is highest up in the table. 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. Comparing the result of the Q-learning algorithm with the rules proposed by Crowley and Siegler, we see that the results are broadly consistent, the Q element corresponding to the proposed action is the largest.
However, the Q-values quoted in Table 11.2 are not perfectly converged, although the algorithm has converged to the optimal strategy. Examples are the very small negative values in 'play empty corner'. That those Q-values are negative means that the player loses and receives reward -1. The Q-values do not equal -1 because the algorithm explores unfavourable actions less frequently (exploitation-exploration

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 11.6.

		$\langle R\rangle_{\text{rock}}$	$\langle R\rangle_{\text{paper}}$	$\langle R\rangle_{\text{scissors}}$
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	$(1-p)(1-3q)$	$(1-p)(3q-1)$

dilemma). Another example is 'play center'. The player receives a small negative reward when playing any other move, indicating that the player loses. But the expected future reward does not equal -1.

Note that the example 'play empty side' in Table 11.2 does not convey any strategic information. The purpose of the last four rows is to allow the player to make a move if the first five choices are unavailable. However, the Q-learning algorithm did not find any example for 'play opposite corner' that was not covered by the first five rows in in Table 11.2.

Finally, Crowley & Siegler designed their rules to take advantage of a weaker player. Therefore they suggest to 'play center' in the beginning of the game when the first five rules do not apply. This increases the probability that a weaker opponent makes an error (Newell & Herbert[1] discuss a slightly different decision table, also designed to increase the probability to win against a weaker player). In our case, the Q-values for the empty board are all equal zero,

$$\begin{array}{|c|c|c|}\hline \text{o} & \text{o} & \text{o} \\ \hline \text{o} & \text{o} & \text{o} \\ \hline \text{o} & \text{o} & \text{o} \\ \hline \end{array}\,, \tag{11.17}$$

because any game between expert players must end in draw.

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

[1] A. Newell & H. A. Simon, *Human problem solving*, Prentice-Hall, London (1972).

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 Navid Mousavi using Q-learning. Exercise 11.7.

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)	x o x x o - o x o	— — — — — 0 — — —

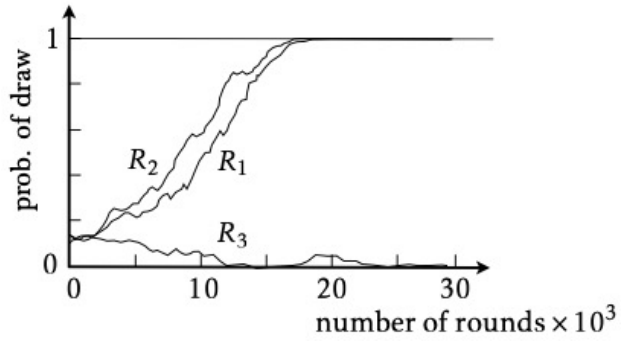


Figure 11.5: Learning to play tic-tac-toe using Q-learning, for three different reward functions, R_1: $r = 1$ (win), $r = 0$ (draw), $r = -1$ (lose); R_2: $r = 2$ (win), $r = 0$ (draw), $r = -1$ (lose); R_3: $r = 1$ (win), $r = -1$ (draw, lose). Initially $\varepsilon = 1$. During training, the parameter ε was iteratively decreased, $\varepsilon' = 0.95\varepsilon$ after 100 epochs. All elements of the Q-matrix were initialised to zero, the learning rate was $\alpha = 0.1$. Schematic, after simulation results obtained by Navid Mousavi, averaged over 20 independent training runs. Exercise 11.8.

function R_1 (see Figure caption) is the same as in 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 in this case the reward is the same for 'draw' or 'lose'. Therefore the second player cannot learn to prevent the first one from winning. Since the first player has the advantage of placing the first piece, this player learns to always win.

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**).
There are many more states to consider in connect four on a 6×6 playing field. The latter allows for 3^{36} board configurations, as opposed to 3^9 for tic-tac-toe. Note, however, that not all board configurations are allowed states. For example, the configuration

x	x	x
o	o	o
-	-	x

(11.18)

is never encountered. For tic-tac-toe, the algorithm visits around 2000 states for each player.
For connect four, the number of states which Q-learning may encounter is much larger, which can slow down the learning. Figure 10.7 shows how two players learn to play connect four on a 6×6 board by competing against each other, using Q-learning.

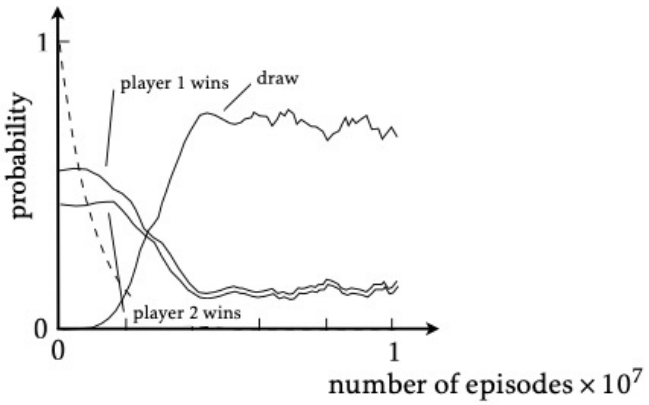


Figure 11.6: Q-learning for the board game *connect four* on a 6×6 playing field. Initially $\varepsilon = 1$. During training, the parameter ε was iteratively decreased, as shown by the dashed line. All elements of the Q-matrix were initialised to zero, the learning rate was $\alpha = 0.1$. The results shown were averaged using a moving average with window size 5×10^5 epochs. Schematic, adapted from numerical results obtained by Navid Mousavi. Exercise 11.9.

Compared with Q-learning for tic-tac-toe (Exercise **11.8**), the parameter ε decreases much more slowly, in order to allow the algorithm to explore the much larger state space (counting the states visited, one finds that the algorithm saw 5×10^6 states after 4×10^6 epochs). We see that the game most often ends in a draw, but the probability of obtaining a draw is smaller than unity. Ref. [**196**] says, by contrast, that expert players always draw on a 6×6 board. The reason for this difference is that the algorithm has not explored sufficiently many states, even though ε decreases so slowly. The problem is that, as ε becomes small, the players explore only if they receive a negative reward: by losing, one learns how to block the opponent or win the game. Since most games end in a draw, exploration takes many epochs.
In summary, the algorithm should explore even more. However, the tabular Q-learning as described in Section **11.3** becomes quite inefficient for this purpose, as one needs a a large amount of memory to save all state-action pairs encountered. In this case it may be more efficient to replace the Q-table with a Q-function and use function approximators such as neural networks (Sections **11.2** and **11.4**).
At any rate, it is proven in Ref. **196** that the game always ends in a draw for expert players on a 6×6 board. The situation is different when the playing field has seven columns instead of six. 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 [**196**].

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

$$\langle s_{t+1} \rangle = \begin{cases} \big(\langle s_t \rangle + 1\big)(1-p) & \text{save} \\ 0 & \text{eat} \end{cases} \tag{11.19}$$

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 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

$$\langle R_T \rangle_{\text{eat}} = 2T\,. \tag{11.20}$$

The second strategy is to save the chocolate for $T-1$ days, and to eat what is left on day T. The reward for this strategy is

$$\langle R_T \rangle_{\text{save}} = \left[\sum_{t=1}^{T-1} \langle s_{t-1} \rangle + 1 \right] + 2\big(\langle s_{T-1} \rangle + 1\big). \tag{11.21}$$

To evaluate this expression further, one needs the solution of the recursion (11.19):

$$\langle s_t \rangle = \frac{1}{p}(1-p)\big[1-(1-p)^t\big]. \tag{11.22}$$

Inserting this expression into Equation (11.21) yields

$$\langle R_T \rangle_{\text{save}} = \frac{T}{p} + \frac{2p-1}{p^2}\big[1-(1-p)^T\big]. \tag{11.23}$$

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: slightly below $\frac{1}{2}$, the reward for keeping is still higher than for eating, but at the same time there is a high chance of losing all chocolate.

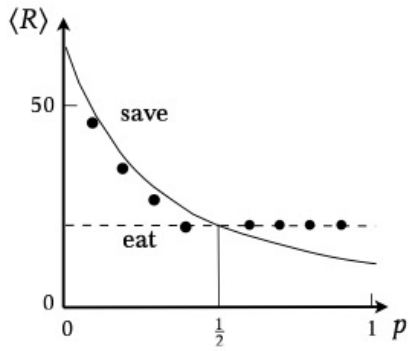


Figure 11.7: Eat or save the chocolate? Expected reward obtained by by Q learning for $T = 10$ days (•). The expected rewards for two strategies are also shown: to eat the chocolate immediately (Equation (11.20), 'eat'), and to wait and eat on the last day (Equation (11.23), 'save'). The optimal strategy changes for $p = \frac{1}{2}$ (vertical line). Schematic, after simulation results by Navid Mousavi. Exercise 11.10.

Bibliography

[1] Chalker, J.T. & Mehlig, B., *Eigenvector Statistics in Non-Hermitian Random Matrix Ensembles*, Phys. Rev. Lett. **81** (1998) 3367. Page 66.

[2] Ferrenberg, A. M. & Landau, D. P., *Critical behavior of the three-dimensional Ising model: A high-resolution Monte Carlo study*, Phys. Rev. B **44** (1991) 5081. Page 16.

[3] Gaunt, D. S., Sykes, M. F. & McKenzie,S., *Susceptibility and fourth-field derivative of the spin-$\frac{1}{2}$ Ising model for $T > T_c$ and $d = 4$*, J. Phys. A **12** (1979) 871. Page 16.

[4] Hinton, G., *A bag of tricks for mini batch gradient descent. Neural Networks for Machine Learning* (2012), `youtube.com/watch?v=Xjtu1L7RwVM`. Page 45.

[5] Horner, H., *Statistische Physik*, Institut für theoretische Physik, Universität Heidelber (2004). Page 12.

[6] Kingma, D.P. & Ba, J., *Adam: a method for stochastic optimization*, arXiv:1412.6980. Pages 54, 56, 57, and 75.

[7] Luijten, E., Binder, K. & Blöte, H. W. J., *Finite-size scaling above the upper critical dimension revisited: the case of the five-dimensional Ising model*, Eur. Phys. J. B **9** (1999) 289. Page 16.

[8] Mehlig, B., *Machine learning with neural networks. An introduction for scientists and engineers*, Cambridge University Press, Cambridge (2021). Page 1.

[9] Newell, G. F. & Montroll, E. W., *On the Theory of the Ising Model of Ferromagnetism*, Rev. Mod. Phys. **25** (1953) 353. Page 16.

[10] Newell, A. & Simon, H. A., *Human problem solving*, Prentice-Hall, London (1972). Page 83.

[11] Nonnenmacher, J., *Lagrange multipliers can fail to determine extrema*, College Mathematics Journal, January 2003, Taylor & Francis. Pages 46 and 47.

[12] Storm, L., Gustavsson, K. & Mehlig, B., *Constraints on parameter choices for successful time-series prediction with echo-state networks*, MLST **3** (2022) 045021. Page 67.

[13] Van Wieringen, W. N., *Lecture notes on ridge regression*, arxiv:1509.09169. Page 67.

[14] Weisstein, E. W., *Poisson Sum Formula*, `mathworld.wolfram.com`. Page 34.

[15] Wilkinson, M., Bezuglyy, V. & Mehlig, B., *Fingerprints of random flows?* Phys. Fluids **21** (2009) 043304. Page 68.

[16] Wilkinson, M. & Mehlig, B., *Path coalescence transition and its applications*, Phys. Rev. E **68** (2003) 040101(R). Page 20.