# 2

# Deterministic Hopfield Networks

## 2.1 Pattern Recognition

As an example of a pattern-recognition task, consider $p$ images (*patterns*), for instance the digits shown in Figure 2.1. The different patterns are indexed by $\mu = 1, \ldots, p$. Here $p$ is the number of patterns ($p = 5$ in Figure 2.1). The bits of pattern $\mu$ are denoted by $x_i^{(\mu)}$. The index $i = 1, \ldots, N$ labels the different bits of a given pattern, and $N$ is the number of bits per pattern ($N = 160$ in Figure 2.1). The bits are *binary*: they can take only the values $-1$ and $+1$. To determine the generic properties of the algorithm, one often turns to *random patterns* where each bit $x_i^{(\mu)}$ is chosen randomly, taking either value with probability equal to $\frac{1}{2}$. It is convenient to gather the bits of a pattern in a column vector like this:

$$\boldsymbol{x}^{(\mu)} = \begin{bmatrix} x_1^{(\mu)} \\ x_2^{(\mu)} \\ \vdots \\ x_N^{(\mu)} \end{bmatrix} . \tag{2.1}$$

In this book, vectors are written in a bold math font, as in Equation (2.1).

The task of the neural network is to recognise distorted patterns, to determine for instance that the pattern on the right in Figure 2.2 is a perturbed version of the digit on the left of this figure. To this end, one *stores* $p$ patterns in the network, presents it with a distorted version of one of these patterns, and asks the network to find the stored pattern that is most similar to the distorted one.

The formulation of the problem makes it necessary to define how similar a distorted pattern $\boldsymbol{x}$ is to any of the stored patterns, say $\boldsymbol{x}^{(\nu)}$. One possibility is to quantify the distance $d_\nu$ between the patterns $\boldsymbol{x}$ and $\boldsymbol{x}^{(\nu)}$ by the mean squared error:

$$d_\nu = \frac{1}{4N} \sum_{i=1}^{N} \left( x_i - x_i^{(\nu)} \right)^2 . \tag{2.2}$$



https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

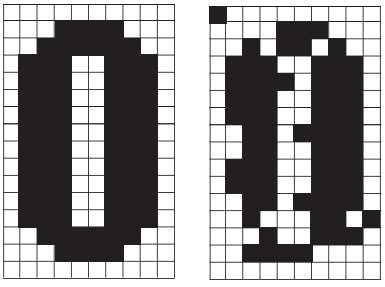

Figure 2.2 Binary image ($N = 160$) of the digit 0, and a distorted version of the same image. There are $N = 160$ bits $x_i$, $i = 1, \ldots, N$, and ■ stands for $x_i = +1$ while □ denotes $x_i = -1$

The prefactor is chosen so that the distance equals the fraction of bits by which two $\pm 1$-patterns differ. Note that the distance (2.2) does not refer to distortions by translations, rotations, or shearing. An improved measure of distance might take the minimum distance between the patterns subject to all possible translations, rotations, and so forth.

## 2.2 Hopfield Networks and Hebb's Rule

Hopfield networks [13, 24] are networks of McCulloch-Pitts neurons designed to solve the pattern-recognition task described in the previous section. The bits of the patterns to be recognised are encoded in a particular choice of weights called Hebb's rule, as explained in the following.

All possible states of the McCulloch-Pitts neurons in the network,

$$\boldsymbol{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_N \end{bmatrix}, \tag{2.3}$$

form the *configuration space* of the network. The components of the states $\boldsymbol{s}$ are updated either with the synchronous rule (1.2):

$$s_i(t+1) = \operatorname{sgn}\big[b_i(t)\big] \quad \text{with local field} \quad b_i(t) = \sum_{j=1}^{N} w_{ij} s_j(t) - \theta_i, \tag{2.4}$$

or with the asynchronous rule

$$s_i(t+1) = \begin{cases} \operatorname{sgn}\big[b_m(t)\big] & \text{for } i = m, \\ s_i(t) & \text{otherwise.} \end{cases} \tag{2.5}$$

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

To recognise a distorted pattern, one feeds its bits $x_i$ into the network by assigning the initial states of the neurons to the pattern bits,

$$s_i(t=0) = x_i \,. \tag{2.6}$$

The idea is to choose a set of weights $w_{ij}$ so that the network dynamics (2.4) or (2.5) converges to the correct stored pattern. The choice of weights must depend on all $p$ patterns, $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(p)}$. We say that these patterns are *stored* in the network by assigning appropriate weights. For example, if $\boldsymbol{x}$ is a distorted version of $\boldsymbol{x}^{(\nu)}$ (Figure 2.2), then we want the network to converge to this pattern:

$$\text{if} \quad \boldsymbol{s}(t=0) = \boldsymbol{x} \approx \boldsymbol{x}^{(\nu)} \quad \text{then} \quad \boldsymbol{s}(t) \to \boldsymbol{x}^{(\nu)} \quad \text{as} \quad t \to \infty \,. \tag{2.7}$$

Equation (2.7) means that the network corrects the errors in the distorted pattern $\boldsymbol{x}$. If this works, the stored pattern $\boldsymbol{x}^{(\nu)}$ is said to be an *attractor* of the network dynamics.

But convergence is not guaranteed. If the initial distortion is too large, the network may converge to another pattern, or to some other state that bears no or little relation to the stored patterns, or it may not converge at all. The region around pattern $\boldsymbol{x}^{(\nu)}$ in which all patterns $\boldsymbol{x}$ converge to $\boldsymbol{x}^{(\nu)}$ is called the *region of attraction* of $\boldsymbol{x}^{(\nu)}$. The size of this region depends in an intricate way upon the ensemble of stored patterns, and there is no general convergence proof.

Therefore we ask a simpler question first: if one feeds one of the undistorted patterns, for instance $\boldsymbol{x}^{(\nu)}$, does the network recognise it as one of the stored, undistorted ones? The network should not make any changes to $\boldsymbol{x}^{(\nu)}$ because all bits are correct:

$$\text{if} \quad \boldsymbol{s}(t=0) = \boldsymbol{x}^{(\nu)} \quad \text{then} \quad \boldsymbol{s}(t) = \boldsymbol{x}^{(\nu)} \quad \text{for all} \quad t = 0, 1, 2, \ldots \,. \tag{2.8}$$

How can we choose weights and thresholds to ensure that Equation (2.8) holds? Let us consider a simple case first, where there is only one pattern, $\boldsymbol{x}^{(1)}$, to recognise. In this case, a suitable choice of weights and thresholds is

$$w_{ij} = \frac{1}{N} x_i^{(1)} x_j^{(1)} \quad \text{and} \quad \theta_i = 0 \,. \tag{2.9}$$

This *learning rule* reminds of a relation between activation patterns of neurons and their coupling, postulated by Hebb [9] more than 70 years ago:

*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*

This is a mechanism for learning through establishing connections: the coupling between neurons tends to increase if they are active at the same time. Equation (2.25), expresses an analogous principle. Together with Equation (2.7), it says that

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

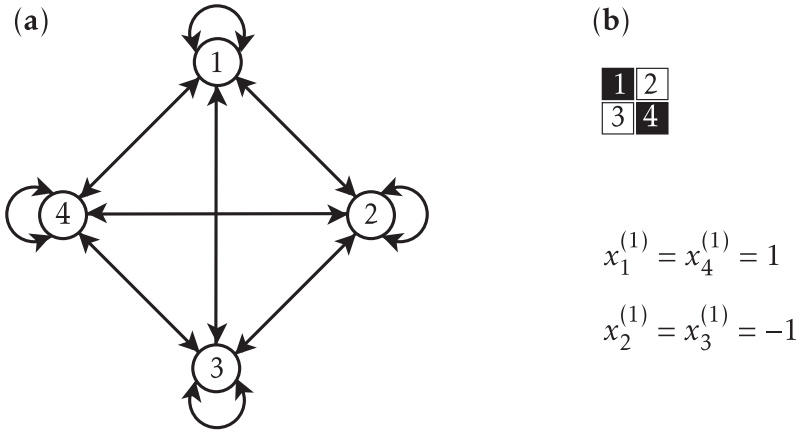


Figure 2.3 Hopfield network with $N = 4$ neurons. (**a**) Network layout. Neuron $i$ is represented as $\textcircled{i}$. Arrows indicate symmetric connections. (**b**) Pattern $\boldsymbol{x}^{(1)}$

the coupling $w_{ij}$ between two neurons is positive if they are both active ($s_i = s_j = 1$); if their states differ, the coupling is negative. Therefore the rule (2.25) is called Hebb's rule. Hopfield networks are networks of McCulloch-Pitts neurons with weights determined by Hebb's rule.

Does a Hopfield network recognise the pattern $\boldsymbol{x}^{(1)}$ stored in this way? To check that the rule (2.9) does the trick, we feed the pattern to the network by assigning $s_j(t = 0) = x_j^{(1)}$, and evaluate Equation (2.4):

$$\sum_{j=1}^{N} w_{ij} x_j^{(1)} = \frac{1}{N} \sum_{j=1}^{N} x_i^{(1)} x_j^{(1)} x_j^{(1)} = \frac{1}{N} \sum_{j=1}^{N} x_i^{(1)} = x_i^{(1)} . \tag{2.10}$$

The second equality follows because $x_j^{(1)}$ can only take the values $\pm 1$, so that $[x_j^{(1)}]^2 = 1$. Using $\mathrm{sgn}(x_i^{(1)}) = x_i^{(1)}$, we obtain

$$\mathrm{sgn}\left( \sum_{j=1}^{N} w_{ij} x_j^{(1)} \right) = x_i^{(1)} . \tag{2.11}$$

Comparing Equation (2.11) with the update rule (2.4) shows that the bits $x_j^{(1)}$ of the pattern $\boldsymbol{x}^{(1)}$ remain unchanged under the update, as required by Equation (2.8). The network recognises the pattern as a stored one, so Equation (2.9) does what we asked for. Note that we obtain the same result if we leave out the factor of $N^{-1}$ in Equation (2.9).

But does the network correct errors? In other words, is the pattern $\boldsymbol{x}^{(1)}$ an attractor [Equation (2.7)]? This question cannot be answered in general. Yet, in

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

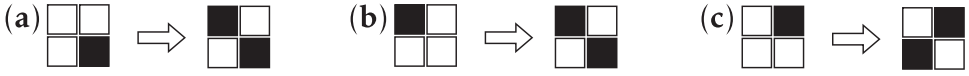


Figure 2.4 Reconstruction of a distorted pattern. Under synchronous updating (2.4), the first two distorted patterns (**a**) and (**b**) converge to the stored pattern $\boldsymbol{x}^{(1)}$, but pattern (**c**) does not

practice, Hopfield networks work often quite well. It is a fundamental insight that neural networks may perform well although no proof exists that their dynamics converges on the correct solution.

To illustrate the difficulties, consider an example: a Hopfield network with $N = 4$ neurons (Figure 2.3). Store the pattern $\boldsymbol{x}^{(1)}$ shown in Figure 2.3 by assigning the weights $w_{ij}$ using Equation (2.9). Now consider a distorted pattern $\boldsymbol{x}$ that has a non-zero distance to $\boldsymbol{x}^{(1)}$ [Figure 2.4 (**a**)]:

$$d_1 = \frac{1}{16}\sum_{i=1}^{4}\left(x_i - x_i^{(1)}\right)^2 = \frac{1}{4}\,. \tag{2.12}$$

To feed the pattern to the network, we set $s_j(t = 0) = x_j$. Then we iterate the dynamics using the synchronous rule (2.4). Results for different distorted patterns are shown in Figure 2.4. We see that the first two distorted patterns converge to the stored pattern, cases (**a**) and (**b**). But the third distorted pattern does not [case (**c**)].

To understand this behaviour, we analyse the synchronous dynamics (2.4) using the *weight matrix*

$$\mathbb{W} = \frac{1}{N}\boldsymbol{x}^{(1)}\boldsymbol{x}^{(1)\mathsf{T}}\,. \tag{2.13}$$

Here $\boldsymbol{x}^{(1)\mathsf{T}}$ denotes the *transpose* of the column vector $\boldsymbol{x}^{(1)}$, so that $\boldsymbol{x}^{(1)\mathsf{T}}$ is a row vector. The standard rules for matrix multiplication apply also to column and row vectors, they are just $N \times 1$ and $1 \times N$ matrices. So the product on the r.h.s. of Equation (2.13) is an $N \times N$ matrix. In the following, matrices with elements $A_{ij}$ or $B_{ij}$ are written as $\mathbb{A}$, $\mathbb{B}$, and so forth. The product in Equation (2.13) is also referred to as an *outer product*. If we change the order of $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(1)\mathsf{T}}$ in the product, we obtain a number instead:

$$\boldsymbol{x}^{(1)\mathsf{T}}\boldsymbol{x}^{(1)} = \sum_{j=1}^{N}\left[x_j^{(1)}\right]^2 = N\,. \tag{2.14}$$

The product (2.14) is called the *scalar product*. It is also denoted by $\boldsymbol{a}\cdot\boldsymbol{b} = \boldsymbol{a}^{\mathsf{T}}\boldsymbol{b}$ and equals $|\boldsymbol{a}||\boldsymbol{b}|\cos\varphi$, where $\varphi$ is the angle between the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, and $|\boldsymbol{a}|$ is the magnitude of $\boldsymbol{a}$. We use the same notation for multiplying a transposed vector with a matrix from the left: $\boldsymbol{x}\cdot\mathbb{A} = \boldsymbol{x}^{\mathsf{T}}\mathbb{A}$. An excellent source for those not familiar with

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

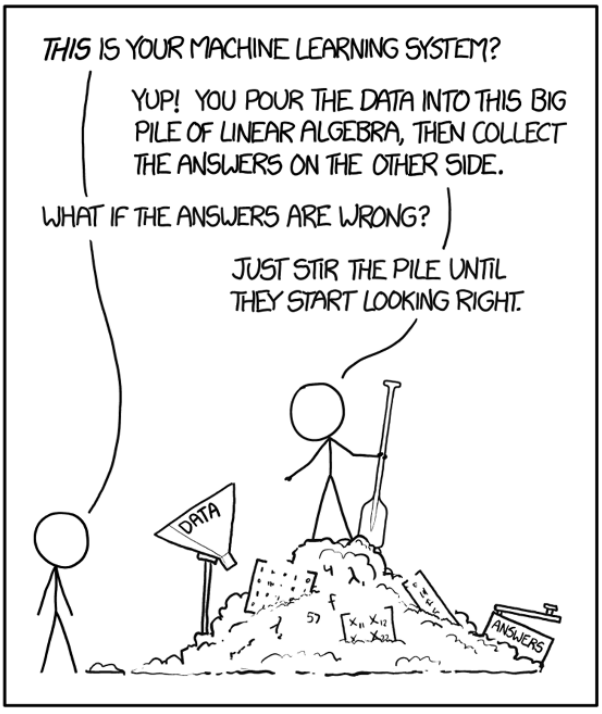


Figure 2.5 Reproduced from xkcd.com/1838 under the creative commons attribution-noncommercial 2.5 license

these terms from linear algebra (Figure 2.5) is Chapter 6 of *Mathematical Methods of Physics* by Mathews and Walker [29].

Using Equation (2.14), we see that $\mathbb{W}$ *projects* onto the vector $\boldsymbol{x}^{(1)}$:

$$\mathbb{W}\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(1)} . \tag{2.15}$$

In addition, it follows from Equation (2.14) that the matrix (2.13) is *idempotent*:

$$\mathbb{W}^t = \mathbb{W} \quad \text{for} \quad t = 1, 2, 3, \ldots . \tag{2.16}$$

Equations (2.15) and (2.16) together with $\operatorname{sgn}\left(x_i^{(1)}\right) = x_i^{(1)}$ imply that the network recognises the pattern $\boldsymbol{x}^{(1)}$ as the stored one. This example illustrates the general proof, Equations (2.10) and (2.11).

Now consider the distorted pattern (**a**) in Figure 2.4. We feed this pattern to the network by assigning

$$\boldsymbol{s}(t=0) = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} . \tag{2.17}$$

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

To compute one step in the synchronous dynamics (2.4), we apply $\mathbb{W}$ to $\boldsymbol{s}(t=0)$. This is done in two steps, using the outer-product form (2.13) of the weight matrix. We first multiply $\boldsymbol{s}(t=0)$ with $\boldsymbol{x}^{(1)\mathsf{T}}$ from the left

$$\boldsymbol{x}^{(1)\mathsf{T}}\boldsymbol{s}(t=0) = \begin{bmatrix}1, & -1, & -1, & 1\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\\1\end{bmatrix} = 2\,, \tag{2.18}$$

and then we multiply this result with $\boldsymbol{x}^{(1)}$. This results in

$$\mathbb{W}\boldsymbol{s}(t=0) = \tfrac{1}{2}\boldsymbol{x}^{(1)}\,. \tag{2.19}$$

The signum of the $i$-th component of the vector $\mathbb{W}\boldsymbol{s}(t=0)$ yields $s_i(t=1)$:

$$s_i(t=1) = \operatorname{sgn}\Big(\sum_{j=1}^{N} w_{ij}s_j(t=0)\Big) = x_i^{(1)}. \tag{2.20}$$

We conclude that the state of the network converges to the stored pattern, in one synchronous update. Since $\mathbb{W}$ is idempotent, the network stays there: the pattern $\boldsymbol{x}^{(1)}$ is an attractor. Case (**b**) in Figure 2.4 works in a similar way.

Now look at case (**c**), where the network fails to converge to the stored pattern. We feed this pattern to the network by assigning $\boldsymbol{s}(t=0) = [-1, 1, -1, -1]^{\mathsf{T}}$. For one iteration of the synchronous dynamics, we evaluate

$$\boldsymbol{x}^{(1)\mathsf{T}}\boldsymbol{s}(0) = \begin{bmatrix}1, & -1, & -1, & 1\end{bmatrix}\begin{bmatrix}-1\\1\\-1\\-1\end{bmatrix} = -2\,. \tag{2.21}$$

It follows that

$$\mathbb{W}\boldsymbol{s}(t=0) = -\tfrac{1}{2}\boldsymbol{x}^{(1)}\,. \tag{2.22}$$

Using the update rule (2.4), we find

$$\boldsymbol{s}(t=1) = -\boldsymbol{x}^{(1)}\,. \tag{2.23}$$

Equation (2.16) implies that

$$\boldsymbol{s}(t) = -\boldsymbol{x}^{(1)} \quad \text{for} \quad t \geq 1\,. \tag{2.24}$$

Thus the network shown in Figure 2.3 has two attractors, the pattern $\boldsymbol{x}^{(1)}$ as well as the *inverted* pattern $-\boldsymbol{x}^{(1)}$. As we shall see in Section 2.5, this is a general property of Hopfield networks: if $\boldsymbol{x}^{(1)}$ is an attractor, then the pattern $-\boldsymbol{x}^{(1)}$ is an attractor too. In the next section, we discuss what happens when more than one pattern is stored in the Hopfield network.

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

### 2.3 The Cross-Talk Term

When there are more patterns than just one, we need to generalise Equation (2.9). One possibility is to simply sum Equation (2.9) over the stored patterns [13]:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} x_i^{(\mu)} x_j^{(\mu)} \quad \text{and} \quad \theta_i = 0\,. \tag{2.25}$$

Equation (2.25) generalises Hebb's rule to $p$ patterns. Because of the sum over $\mu$, the relation to Hebb's learning hypothesis is less clear, but we nevertheless refer to Equation (2.25) as Hebb's rule. At any rate, we see that the weights are proportional to the second moments of the pattern bits. It is plausible that a neural network based upon the rule (2.25) can recognise properties of the patterns $\boldsymbol{x}^{(\mu)}$ that are encoded in two-point correlations of their bits.

Note that the weights are symmetric, $w_{ij} = w_{ji}$. Also, note that the prefactor $N^{-1}$ in Equation (2.25) is not important. It is chosen to simplify the large-$N$ analysis of the model (Chapter 3). An alternative version of Hebb's rule [2, 13] sets the diagonal weights to zero:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} x_i^{(\mu)} x_j^{(\mu)} \quad \text{for} \quad i \neq j, \quad w_{ii} = 0, \quad \text{and} \quad \theta_i = 0. \tag{2.26}$$

In this section, we use the form (2.26) of Hebb's rule. If we assign the weights in this way, does the network recognise the stored patterns? The question is whether

$$\underbrace{\operatorname{sgn}\Big(\frac{1}{N} \sum_{j\neq i} \sum_{\mu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)}\Big)}_{=b_i^{(\nu)}} = x_i^{(\nu)} \tag{2.27}$$

holds or not. To check this, we repeat the calculation described in the previous section. As a first step, we evaluate the local field

$$b_i^{(\nu)} = \big(1 - \tfrac{1}{N}\big) x_i^{(\nu)} + \frac{1}{N} \sum_{j\neq i} \sum_{\mu\neq\nu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)}\,. \tag{2.28}$$

Here we split the sum over the patterns into two contributions. The first term corresponds to $\mu = \nu$, where $\nu$ refers to the pattern that was fed to the network, the one we want the network to recognise. Condition (2.27) is satisfied if the second term in (2.28) does not affect the sign of the r.h.s. of this equation. This second term is called the *cross-talk* term.

Whether adding the cross-talk term to $\boldsymbol{x}^{(\nu)}$ affects $\operatorname{sgn}(b_i^{(\nu)})$ or not depends on the stored patterns. Since the cross-talk term contains a sum over $\mu = 1, \ldots, p$, we expect that this term does not matter if $p$ is small enough. The fewer patterns we

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

store, the more likely it is that all of them are recognised. Furthermore, by analogy with the example described in the previous section, it is plausible that the stored patterns are then also attractors, so that slightly distorted patterns converge to the correct stored pattern.

For a more quantitative analysis of the effect of the cross-talk term, we store patterns with random bits (*random patterns*). Different bits are assigned $\pm 1$ independently with equal probability:

$$\text{Prob}(x_i^{(\nu)} = \pm 1) = \tfrac{1}{2} \,. \tag{2.29}$$

This means in particular that different patterns are *uncorrelated*, because their *covariance* vanishes:

$$\langle x_i^{(\mu)} x_j^{(\nu)} \rangle = \delta_{ij} \delta_{\mu\nu} \,. \tag{2.30}$$

Here $\langle \cdots \rangle$ denotes an average over many realisations of random patterns, and $\delta_{ij}$ is the *Kronecker delta*, equal to unity if $i = j$ but zero otherwise. Note that $\langle x_j^{(\mu)} \rangle = 0$. This follows from Equation (2.29).

Given an ensemble of random patterns, what is the probability that the cross-talk term changes $\text{sgn}(b_i^{(\nu)})$? In other words, what is the probability that the network produces a wrong bit in one asynchronous update if all bits were initially correct? The magnitude of the cross-talk term does not matter when it has the same sign as $x_i^{(\nu)}$. If it has a different sign, then the cross-talk term may matter. To determine when this is the case, one defines

$$C_i^{(\nu)} \equiv -x_i^{(\nu)} \underbrace{\frac{1}{N} \sum_{j \neq i} \sum_{\mu \neq \nu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)}}_{\text{cross-talk term}} \,. \tag{2.31}$$

If $C_i^{(\nu)} < 0$, then the cross-talk term has same sign as $x_i^{(\nu)}$, so that the cross-talk term does not make a difference: adding this term does not change the sign of $x_i^{(\nu)}$. If $0 < C_i^{(\nu)} < 1$, it does not matter either, only when $C_i^{(\nu)} > 1$. The network produces an error in bit $i$ of pattern $\nu$ if $C_i^{(\nu)} > 1$ (we approximated $1 - 1/N \approx 1$ in Equation (2.28), assuming that $N$ is large).

## 2.4 One-Step Error Probability

The one-step *error probability* $P_{\text{error}}^{t=1}$ is defined as the probability that an error occurs in one attempt to update a bit, given that initially all bits were correct:

$$P_{\text{error}}^{t=1} = \text{Prob}(C_i^{(\nu)} > 1) \,. \tag{2.32}$$

Since patterns and bits are identically distributed, $\text{Prob}(C_i^{(\nu)} > 1)$ does not depend on $i$ or $\nu$. Therefore $P_{\text{error}}^{t=1}$ does not carry any indices.

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

How does $P_{\text{error}}^{t=1}$ depend on the parameters of the problem, $p$ and $N$? When both $p$ and $N$ are large, we can use the *central-limit theorem* [29, 30] to answer this question. Since different bits/patterns are independent, we can think of $C_i^{(\nu)}$ as a sum of independent random numbers $c_m$ that take the values $-1$ and $+1$ with equal probabilities,

$$C_i^{(\nu)} = -\frac{1}{N}\sum_{j\neq i}\sum_{\mu\neq\nu} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\nu)} x_i^{(\nu)} = -\frac{1}{N}\sum_{m=1}^{(N-1)(p-1)} c_m \,. \tag{2.33}$$

There are $M = (N-1)(p-1)$ terms in the sum on the r.h.s. because terms with $\mu = \nu$ are excluded, and also those with $j = i$ [Equation (2.26)]. If we use the rule (2.25) instead, then there is a correction to Equation (2.33) from the diagonal weights. For $p \ll N$, this correction is small.

When $p$ and $N$ are large, the sum $\sum_{m=1}^{M} c_m$ contains a large number of independently identically distributed random numbers with mean zero and variance unity. It follows from the central-limit theorem [29, 30] that $\sum_{m=1}^{M} c_m$ is Gaussian distributed with mean zero and variance $M$.

Since the central-limit theorem plays an important role in the analysis of neural-network algorithms, it is worth discussing this theorem in a little more detail. To begin with, note that the sum $\sum_{m=1}^{M} c_m$ equals $2k - M$, where $k$ is the number of occurrences of $c_m = +1$ in the sum. Choosing $c_m$ randomly to equal either $-1$ or $+1$ is called a *Bernoulli trial* [30], and the probability $P_{k,M}$ of drawing $k$ times $+1$ and $M - k$ times $-1$ is given by the *binomial* distribution [30]. In our case, the probability of $c_m = \pm 1$ equals $\frac{1}{2}$, so that

$$P_{k,M} = \binom{M}{k} \left(\tfrac{1}{2}\right)^k \left(\tfrac{1}{2}\right)^{M-k} . \tag{2.34}$$

Here $\binom{M}{k} = M!/[k!\,(M-k)!]$ denotes the number of ways in which $k$ occurrences of $+1$ can be distributed over $M$ places.

We want to show that $P_{k,M}$ approaches a Gaussian distribution for large $M$, with mean zero and with variance $M$. Since the variance diverges as $M \to \infty$, it is convenient to use the variable $z = (2k - M)/\sqrt{M}$. The central-limit theorem implies that $z$ is Gaussian with mean zero and unit variance in the limit of large $M$. To prove that this is the case, we substitute $k = \frac{M}{2} + \frac{\sqrt{M}}{2} z$ into Equation (2.34) and take the limit of large $M$ using Stirling's approximation

$$n! \approx \mathrm{e}^{n\log n - n + \frac{1}{2}\log 2\pi n} . \tag{2.35}$$

Expanding $P_{k,M}$ to leading order in $M^{-1}$ assuming that $z$ remains of order unity gives $P_{k,M} = \sqrt{2/(\pi M)}\exp(-z^2/2)$. Now one changes variables from $k$ to $z$. This stretches local neighbourhoods $\mathrm{d}k$ to $\mathrm{d}z$. Conservation of probability implies that $P(z)\mathrm{d}z = P(k)\mathrm{d}k$. It follows that $P(z) = (\sqrt{M}/2)P(k)$, so that $P(z) = (2\pi)^{-1/2}$

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

$\exp(-z^2/2)$. In other words, the distribution of $z$ is Gaussian with zero mean and unit variance, as we intended to show.

Returning to Equation (2.33), we conclude that $C_i^{(\nu)}$ is Gaussian distributed

$$P(C) = (2\pi\sigma_C^2)^{-1/2} \exp[-C^2/(2\sigma_C^2)]\,, \tag{2.36}$$

with zero mean, as illustrated in Figure 2.6, and with variance

$$\sigma_C^2 = \frac{M}{N^2} \approx \frac{p}{N}\,. \tag{2.37}$$

Here we used $M \approx Np$ for large $N$ and $p$.

Another way to compute this variance is to square Equation (2.33) and to average over random patterns:

$$\sigma_C^2 = \frac{1}{N^2}\left\langle \left(\sum_{m=1}^{M} c_m\right)^2 \right\rangle = \frac{1}{N^2}\sum_{n=1}^{M}\sum_{m=1}^{M}\langle c_n c_m\rangle. \tag{2.38}$$

Here $\langle\cdots\rangle$ denotes the average over random realisations of $c_m$. Since the random numbers $c_m$ are independent for different indices and because $\langle c_m^2\rangle = 1$, we have that $\langle c_n c_m\rangle = \delta_{nm}$. So only the diagonal terms in the double sum contribute, summing to $M \approx Np$. This yields Equation (2.37).

To determine $P_{\text{error}}^{t=1}$ [Equation (2.32)], we must integrate the distribution of $C$ from 1 to $\infty$:

$$P_{\text{error}}^{t=1} = \frac{1}{\sqrt{2\pi}\sigma_C}\int_1^\infty \mathrm{d}C\ \mathrm{e}^{-\frac{C^2}{2\sigma_C^2}} = \frac{1}{2}\left[1 - \operatorname{erf}\left(\sqrt{\frac{N}{2p}}\right)\right]. \tag{2.39}$$

Here erf is the *error function* defined as [31]

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z \mathrm{d}x\ \mathrm{e}^{-x^2}\,. \tag{2.40}$$

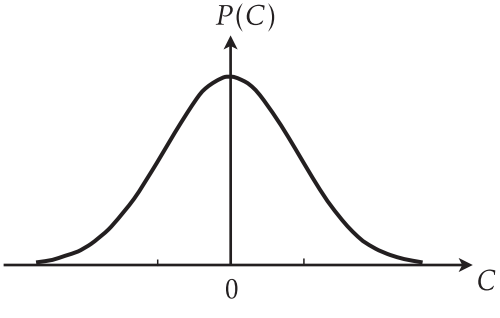


Figure 2.6 Gaussian distribution of the quantity $C$ defined in Equation (2.31)

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

Since $\mathrm{erf}(z)$ increases monotonically as $z$ increases, we conclude that $P_{\text{error}}^{t=1}$ increases as $p$ increases, or as $N$ decreases. This is expected: it is more difficult for the network to distinguish stored patterns when there are more of them. On the other hand, it is easier to differentiate stored patterns if they have more bits. We also see that the one-step error probability depends on $p$ and $N$ only through the combination

$$\alpha \equiv \frac{p}{N} . \tag{2.41}$$

The parameter $\alpha$ is called the *storage capacity* of the network. Figure 2.7 shows how $P_{\text{error}}^{t=1}$ depends on the storage capacity. For $\alpha = 0.2$, for example, the one-step error probability is slightly larger than 1%.

In the derivation of Equation (2.39), we assumed that the stored patterns are random with independent bits. Realistic patterns are not random. We nevertheless expect that $P_{\text{error}}^{t=1}$ describes the typical one-step error probability of the Hopfield network when $p$ and $N$ are large. However, it is straightforward to construct counterexamples. Consider for example *orthogonal patterns*:

$$\boldsymbol{x}^{(\mu)} \cdot \boldsymbol{x}^{(\nu)} = 0 \quad \text{for} \quad \mu \neq \nu . \tag{2.42}$$

For such patterns, the crosstalk term vanishes in the limit of large $N$ (Exercise 2.2), so that $P_{\text{error}}^{t=1} = 0$.

More importantly, the error probability defined in this section refers only to the initial update, the first iteration. What happens in the next iteration, and after many iterations? Numerical experiments show that the error probability can be much higher in later iterations, because an error tends to increase the probability of making another error later on. So the estimate $P_{\text{error}}^{t=1}$ is only a lower bound for the probability of observing errors in the long run.

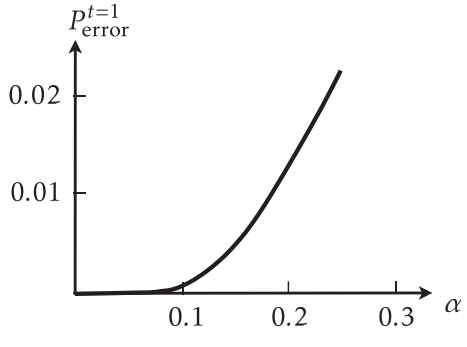


Figure 2.7 Dependence of the one-step error probability on the storage capacity $\alpha$ according to Equation (2.39)

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

## 2.5 Energy Function

Consider the long-time limit $t \to \infty$. Does the Hopfield dynamics converge, as required by Equation (2.7)? This is an important question in the analysis of neural-network algorithms, because an algorithm that does not converge to a meaningful solution is useless.

The standard way of analysing the convergence of neural-network algorithms is to define an *energy function* $H(\boldsymbol{s})$ that has a minimum at the desired solution, say $\boldsymbol{s} = \boldsymbol{x}^{(\nu)}$. We monitor how the energy function changes as we iterate, and keep track of the smallest values of $H$ encountered, to find the minimum. If we store only one pattern, $p = 1$, then a suitable energy function is

$$H = -\frac{1}{2N}\left(\sum_{i=1}^{N} s_i x_i^{(1)}\right)^2 . \tag{2.43}$$

This function is minimal when $\boldsymbol{s} = \boldsymbol{x}^{(1)}$, *i.e.*, when $s_i = x_i^{(1)}$ for all $i$. It is customary to insert the factor $1/(2N)$; this does not change the fact that $H$ is minimal at $\boldsymbol{s} = \boldsymbol{x}^{(1)}$.

A crucial point is that the asynchronous McCulloch-Pitts dynamics (2.5) *converges* to the minimum [13]. This follows from the fact that $H$ cannot increase under the update rule (2.5). To prove this important property, we begin by evaluating the expression on the r.h.s. of Equation (2.43):

$$H = -\frac{1}{2}\sum_{ij}^{N}\left(\frac{1}{N}x_i^{(1)}x_j^{(1)}\right)s_i s_j . \tag{2.44}$$

Using Hebb's rule (2.9), we find that the energy function (2.43) becomes

$$H = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j . \tag{2.45}$$

This function has the same form as the energy function (or *Hamiltonian*) for certain physical models of magnetic systems consisting of interacting spins [32], where the interaction energy between spins $s_i$ and $s_j$ is $\frac{1}{2}(w_{ij} + w_{ji})s_i s_j$. Note that Hebb's rule (2.9) yields symmetric weights: $w_{ij} = w_{ji}$, and $w_{ii} > 0$. Note also that setting the diagonal weights to zero does not change the fact that $H$ is minimal at $\boldsymbol{s} = \boldsymbol{x}^{(1)}$, because $s_i^2 = 1$. The diagonal weights just give a constant contribution to $H$, independent of $\boldsymbol{s}$.

The second step is to show that $H$ cannot increase under the asynchronous McCulloch-Pitts dynamics (2.5). In this case, we say that the energy function is a *Lyapunov function*, or *loss function*. To demonstrate that the energy function is a Lyapunov function, choose a neuron $m$ and update it according to Equation (2.5).

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

We denote the updated state of neuron $m$ by $s'_m$:

$$s'_m = \operatorname{sgn}\Big(\sum_j w_{mj} s_j\Big)\,. \tag{2.46}$$

All other neurons remain unchanged. There are two possibilities, either $s'_m = s_m$ or $s'_m = -s_m$. In the first case, $H$ remains the same, $H' = H$. Here $H'$ refers to the value of the energy function after the update (2.46). When $s'_m = -s_m$, by contrast, the energy function changes by the amount

$$\begin{aligned} H' - H &= -\frac{1}{2}\sum_{j\neq m}(w_{mj} + w_{jm})(s'_m s_j - s_m s_j) - \frac{1}{2} w_{mm}(s'_m s'_m - s_m s_m) \\ &= \sum_{j\neq m}(w_{mj} + w_{jm}) s_m s_j\,. \end{aligned} \tag{2.47}$$

The sum goes over all neurons $j$ that are connected to the neuron $m$, the one to be updated in Equation (2.46). Now if the weights are symmetric, $H' - H$ equals

$$H' - H = 2\sum_{j\neq m} w_{mj} s_m s_j = 2\sum_j w_{mj} s_m s_j - 2 w_{mm}\,. \tag{2.48}$$

Since the sign of $\sum_j w_{mj} s_j$ is that of $s'_m = -s_m$ and if $w_{mm} \geq 0$, it follows that

$$H' - H < 0\,. \tag{2.49}$$

In other words, the value of $H$ must decrease when the state of neuron $m$ changes, $s'_m \neq s_m$. In summary,[1] $H$ either remains constant under the asynchronous McCulloch-Pitts dynamics ($s'_m = s_m$) or its value decreases ($s'_m \neq s_m$). Note that this does not hold for the synchronous dynamics (2.4); see Exercise 2.9. Since the energy function cannot increase under the asynchronous McCulloch-Pitts dynamics, it must converge to a minimum of the energy function. For the energy function (2.43) this implies that the dynamics must either converge to the stored pattern or to its inverse. Both are attractors.

We assumed the thresholds to vanish, but the proof also works when the thresholds are not zero, in this case for the energy function

$$H = -\frac{1}{2}\sum_{ij} w_{ij} s_i s_j + \sum_i \theta_i s_i \tag{2.50}$$

in conjunction with the update rule $s'_m = \operatorname{sgn}\Big(\sum_j w_{mj} s_j - \theta_m\Big)$.

[1] The derivation outlined here did not use the specific form of Hebb's rule (2.9), only that the weights are symmetric, and that $w_{mm} \geq 0$. However, the derivation fails when $w_{mm} < 0$. In this case, it is still true that $H$ assumes a minimum at $\boldsymbol{s} = \boldsymbol{x}^{(1)}$, but $H$ can increase under the update rule, so that convergence is not guaranteed. We therefore require that the diagonal weights are not negative.

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

Up until now, we considered only one stored pattern, $p = 1$. If we store more than one pattern [Hebb's rule (2.25)], the proof that (2.45) cannot increase under the McCulloch-Pitts dynamics works in the same way because no particular form of the weights $w_{ij}$ was assumed, only that they must be symmetric, and that the diagonal weights must not be negative. Therefore it follows in this case too that the minima of the energy function must correspond to attractors, as illustrated schematically in Figure 2.8. The configuration space of the network, corresponding to all possible choices of $\boldsymbol{s} = [s_1, \ldots s_N]^\mathsf{T}$, is drawn as a single axis, the $x$-axis. But when $N$ is large, the configuration space is really very high dimensional.

However, some stored patterns may not be attractors when $p > 1$. This follows from our analysis of the cross-talk term in Section 2.2. If the cross-talk term causes errors for a certain stored pattern, then this pattern is not located at a minimum of the energy function. Another way to see this is to combine Equations (2.25) and (2.45) to give

$$H = -\frac{1}{2N} \sum_{\mu=1}^{p} \left( \sum_{i=1}^{N} s_i x_i^{(\mu)} \right)^2 . \tag{2.51}$$

While the energy function defined in Equation (2.43) has a minimum at $\boldsymbol{x}^{(1)}$, Equation (2.51) need not have a minimum at $\boldsymbol{x}^{(1)}$ (or at any other stored pattern), because a maximal value of $\left( \sum_{i=1}^{N} s_i x_i^{(1)} \right)^2$ may be compensated by terms stemming from other patterns. This happens rarely when $p$ is small (Section 2.2).

Conversely, there may be minima that do not correspond to stored patterns. Such states are referred to as *spurious states*. The network may converge to spurious states. This is undesirable, but it occurs even when there is only one stored pattern, as we saw in Section 2.2: the McCulloch-Pitts dynamics may converge to the inverted pattern. This follows also from Equation (2.51): if $\boldsymbol{s} = \boldsymbol{x}^{(1)}$ is a local

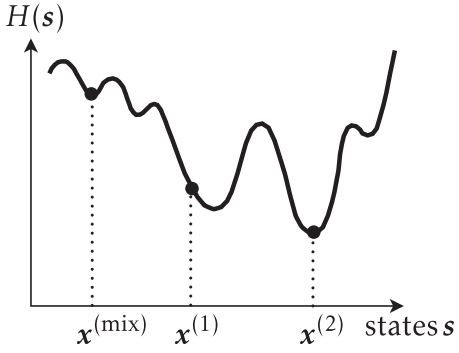


Figure 2.8 Minima of the energy function are attractors. Not all minima correspond to stored patterns ($\boldsymbol{x}^{(\text{mix})}$ is a mixed state; see the text), and stored patterns need not correspond to minima

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

minimum of $H$, then so is $\boldsymbol{s} = -\boldsymbol{x}^{(1)}$. This is a consequence of the invariance of $H$ under $\boldsymbol{s} \to -\boldsymbol{s}$. There are other types of spurious states besides inverted patterns. An example are *mixed states*, *superpositions* of an odd number $2n + 1$ of patterns [1]. For $n = 1$, for example, the bits of a mixed state read:

$$x_i^{(\mathrm{mix})} = \operatorname{sgn}\left(\pm x_i^{(1)} \pm x_i^{(2)} \pm x_i^{(3)}\right). \tag{2.52}$$

The number of mixed states increases as $n$ increases. There are $2^{2n+1}\binom{p}{2n+1}$ mixed states that are superpositions of $2n + 1$ out of $p$ patterns, for $n = 1, 2, \ldots$ (Exercise 2.4). Mixed states such as (2.52) are sometimes recognised by the network (Exercise 2.5); therefore it may happen that the network converges to these states. Finally, there are spurious states that are not related in any way to the stored patterns $x_j^{(\mu)}$. Such *spin-glass* states are discussed in detail in Refs. [27, 33, 34], and also by Hertz, Krogh, and Palmer [1].

## 2.6 Summary

Hopfield networks are networks of McCulloch-Pitts neurons that recognise patterns (Algorithm 1). Their layout is defined by connection strengths, or weights, chosen according to Hebb's rule. The weights $w_{ij}$ are symmetric, and the network is in general fully connected. Hebb's rule ensures that stored patterns are recognised, at least most of the time if the number of patterns is not too large. Convergence of the McCulloch-Pitts dynamics is analysed in terms of an energy function, which cannot increase under the asynchronous McCulloch-Pitts dynamics.

A single-step estimate for the error probability of the network dynamics was derived in Section 2.2. If one iterates several steps, the error probability is usually much larger, but it is difficult to evaluate in general. For stochastic Hopfield networks, the steady-state error probability can be estimated more easily, because the dynamics converges to a steady state (Chapter 3).

**Algorithm 1** Pattern recognition with a deterministic Hopfield network

store patterns $\boldsymbol{x}^{(\mu)}$ using Hebb's rule;
feed distorted pattern $\boldsymbol{x}$ into network by assigning $\boldsymbol{s}(t = 0) \leftarrow \boldsymbol{x}$;
**for** $t = 1, \ldots, T$ **do**
  choose a value of $m$ and update $s_m(t) \leftarrow \operatorname{sgn}\left(\sum_{j=1}^{N} w_{mj} s_j(t-1)\right)$;
**end for**
read out pattern $\boldsymbol{s}(T)$;

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

## 2.7 Exercises

**2.1 Modified Hebb's rule.** Show that the modified Hebb's rule (2.26) satisfies Equation (2.8) if we store only one pattern, $p = 1$.

**2.2 Orthogonal patterns.** For Hebb's rule (2.25), show that the cross-talk term vanishes for orthogonal patterns, so that $P_{\text{error}}^{t=1} = 0$. For the modified Hebb's rule (2.26), the cross-talk term is non-zero for orthogonal patterns. Show that it becomes negligible in the limit of large $N$.

**2.3 Cross-talk term.** Expression (2.33) for the cross-talk term was derived using modified Hebb's rule, Equation (2.26). How does Equation (2.33) change if you use the rule (2.25) instead? Show that the distribution of $C_i^{(\nu)}$ then acquires a non-zero mean, obtain an estimate for this mean value, and compute the one-step error probability. Show that your result approaches (2.39) for small values of $\alpha$. Explain why your result is different from (2.39) for large $\alpha$.

**2.4 Mixed states.** Explain why there are no mixed states that are superpositions of an even number of stored patterns. Show that there are $2^{2n+1}\binom{p}{2n+1}$ mixed states that are superpositions of $2n+1$ out of $p$ patterns, for $n = 1, 2, \ldots$.

**2.5 Recognising mixed states.** Store $p$ random patterns in a Hopfield network with $N = 50$ and 100 neurons using Hebb's rule (2.25). Using computer simulations, determine the probability that the network recognises bit $x_i^{(\text{mix})}$ of the mixed state $\boldsymbol{x}^{(\text{mix})}$ with bits

$$x_i^{(\text{mix})} = \text{sgn}\big(x_i^{(1)} + x_i^{(2)} + x_i^{(3)}\big). \tag{2.53}$$

Show that the one-step error probability tends to zero as $\alpha \to 0$ for large $N$, by analysing how often $\text{sgn}\big(\frac{1}{N}\sum_{\mu=1}^{p}\sum_{j=1}^{N} x_i^{(\mu)} x_j^{(\mu)} x_j^{(\text{mix})}\big) = x_i^{(\text{mix})}$ holds. *Hint:* Think of $\frac{1}{N}\sum_j$ as an average of $x_j^{(\mu)} x_j^{(\text{mix})}$ over random bits and evaluate this average. Then apply the signum function.

**2.6 Energy function.** Figure 2.9 shows a network with two neurons with asymmetric weights, $w_{12} = 2$ and $w_{21} = -1$. Show that the energy function $H = -\frac{w_{12}+w_{21}}{2} s_1 s_2$ can increase under the asynchronous McCulloch-Pitts rule.

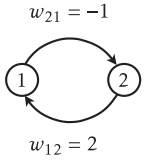


Figure 2.9 Two neurons with asymmetric connections (Exercise 2.6)

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press

**2.7 Higher-order Hopfield networks.** Determine under which conditions the energy function $H = -\frac{1}{2}\sum_{ij} w_{ij}^{(2)} s_i s_j - \frac{1}{6}\sum_{ijk} w_{ijk}^{(3)} s_i s_j s_k$ is a Lyapunov function for the asynchronous dynamics $s'_m = \mathrm{sgn}(b_m)$ with $b_m = \partial H / \partial s_m$.

**2.8 Hebb's rule and energy function for 0/1 units.** Suppose that the state of a neuron takes the values 0 (inactive) and 1 (active). The corresponding asynchronous update rule is $n'_m = \theta_{\mathrm{H}}\left(\sum_j w_{mj} n_j - \mu_m\right)$ with threshold $\mu_m$. The activation function $\theta_{\mathrm{H}}(b)$ is the Heaviside function, equal to 0 if $b < 0$ and equal to 1 if $b \geq 0$ (Figure 2.10). Write down Hebb's rule for such 0/1 units and show that if one stores only one pattern, then this pattern is recognised. Show that $H = -\frac{1}{2}\sum_{ij} w_{ij} n_i n_j + \sum_i \mu_i n_i$ cannot increase under the asynchronous update rule (it is assumed that the weights are symmetric, and that $w_{ii} \geq 0$). See Ref. [13].

**2.9 Energy function and synchronous dynamics.** Analyse how the energy function (2.45) changes under the synchronous dynamics (2.4). Show that the energy function can increase, even though the weights are symmetric and the diagonal weights are zero.

**2.10 Continuous Hopfield network.** Hopfield [35] also analysed a version of his model with continuous-time dynamics. Consider $\tau \frac{\mathrm{d}}{\mathrm{d}t} n_i = -n_i + g\left(\sum_j w_{ij} n_j - \theta_i\right)$ with $g(b) = (1 + \mathrm{e}^{-b})^{-1}$ (this dynamical equation is slightly different from the one used by Hopfield [35]). Show that the energy function $E = -\frac{1}{2}\sum_{ij} w_{ij} n_i n_j + \sum_i \theta_i n_i + \sum_i \int_0^{n_i} \mathrm{d}n\, g^{-1}(n)$ cannot increase under the network dynamics if the weights are symmetric. It is not necessary to assume that $w_{ii} \geq 0$.

**2.11 Hopfield network with four neurons.** The pattern shown in Figure. 2.11 is stored in a Hopfield network using Hebb's rule $w_{ij} = \frac{1}{N} x_i^{(1)} x_j^{(1)}$. There are $2^4$ four-bit patterns. Apply each of these to the Hopfield network, and perform one synchronous update. List the patterns you obtain and discuss your results.

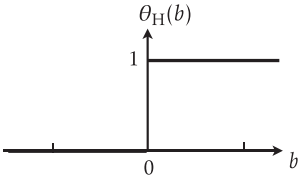


Figure 2.10 Heaviside function (Exercise 2.8)

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press


Figure 2.11 The pattern $\boldsymbol{x}^{(1)}$ has $N = 4$ bits, $x_1^{(1)} = 1$, and $x_i^{(1)} = -1$ for $i = 2, 3, 4$. See Exercise 2.11

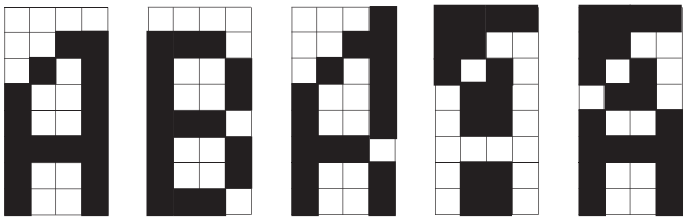


Figure 2.12 Each of the five patterns consists of 32 bits $x_i^{(\mu)}$. A black pixel $i$ in pattern $\mu$ corresponds to $x_i^{(\mu)} = 1$, a white one to $x_i^{(\mu)} = -1$. See Exercise 2.12

**2.12 Recognising letters with a Hopfield network.** The five patterns in Figure 2.12 each have $N = 32$ bits. Store the patterns $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ in a Hopfield network using Hebb's rule $w_{ij} = \frac{1}{N}\sum_{\mu=1}^{2} x_i^{(\mu)} x_j^{(\mu)}$. Which of the patterns in Figure 2.12 remain unchanged after one synchronous update with $s_i' = \operatorname{sgn}\left(\sum_{j=1}^{N} w_{ij} s_j\right)$? *Hint*: read off $\sum_{j=1}^{N} x_j^{(\mu)} x_j^{(\nu)}$ from the *Hamming* distance between the two patterns, equal to the number of bits by which the patterns differ. Use this quantity to express the local fields $b_i^{(\mu)}$ as linear combinations of $x_i^{(1)}$ and $x_i^{(2)}$.

**2.13 XOR function.** The Boolean XOR function takes two binary inputs. For the inputs $[-1, -1]$ and $[1, 1]$ the function evaluates to $-1$, for the other two inputs to $+1$. Try to encode the XOR function in a Hopfield network with three neurons by storing the patterns $[-1, -1, -1]$, $[1, 1, -1]$, $[-1, 1, 1]$, and $[1, -1, 1]$ using Hebb's rule. Test whether the patterns are recognised or not. Discuss your findings.

**2.14 Distance as a measure of convergence.** The distance $d = \frac{1}{4N}\sum_i (s_i - x_i^{(1)})^2$ [Equation (2.2)] has a minimum at $\boldsymbol{s} = \boldsymbol{x}^{(1)}$. How are $d$ and $H$ [Equation (2.43)] related? What is the advantage of using $H$ instead of $d$ as a measure of convergence?

https://doi.org/10.1017/9781108860604.002 Published online by Cambridge University Press