13

The Hopfield Model

One of the milestones for the current renaissance in the field of neural networks was the associative model proposed by Hopfield at the beginning of the 1980s. Hopfield's approach illustrates the way theoretical physicists like to think about ensembles of computing units. No synchronization is required, each unit behaving as a kind of elementary system in complex interaction with the rest of the ensemble. An energy function must be introduced to harness the theoretical complexities posed by such an approach. The next two sections deal with the structure of Hopfield networks. We then proceed to show that the model converges to a stable state and that two kinds of learning rules can be used to find appropriate network weights.

13.1 Synchronous and asynchronous networks

A relevant issue for the correct design of recurrent neural networks is the adequate synchronization of the computing elements. In the case of McCulloch-Pitts networks we solved this difficulty by assuming that the activation of each computing element consumes a unit of time. The network is built taking this delay into account and by arranging the elements and their connections in the necessary pattern. When the arrangement becomes too contrived, additional units can be included which serve as delay elements. What happens when this assumption is lifted, that is, when the synchronization of the computing elements is eliminated?

13.1.1 Recursive networks with stochastic dynamics

We discussed the design and operation of associative networks in the previous chapter. The synchronization of the output was achieved by requiring that all computing elements evaluate their inputs and compute their output simultaneously. Under this assumption the operation of the associative memory can

be described with simple linear algebraic methods. The excitation of the output units is computed using vector-matrix multiplication and evaluating the sign function at each node.

The methods we have used before to avoid dealing explicitly with the synchronization problem have the disadvantage, from the point of view of both biology and physics, that global information is needed, namely a global time. Whereas in conventional computers synchronization of the digital building blocks is achieved using a clock signal, there is no such global clock in biological systems. In a more biologically oriented simulation, global synchronization should thus be avoided. In this chapter we deal with the problem of identifying the properties of neural networks lacking global synchronization.

Networks in which the computing units are activated at different times and which provide a computation after a variable amount of time are stochastic automata. Networks built from this kind of units behave like *stochastic dynamical systems*.

13.1.2 The bidirectional associative memory

Before we start analyzing asynchronous networks we will examine another kind of synchronous associative model with bidirectional edges. We will arrive at the concept of the *energy function* in a very natural way.

We have already discussed recurrent associative networks in which the output of the network is fed back to the input units using additional feedback connections (Figure 12.3). In this way we designed recurrent dynamical systems and tried to determine their fixpoints. However, there is another way to define a recurrent associative memory made up of two layers which send information recursively between them. The input layer contains units which receive the input to the network and send the result of their computation to the output layer. The output of the first layer is transported by bidirectional edges to the second layer of units, which then return the result of their computation back to the first layer using the same edges. As in the case of associative memory models, we can ask whether the network achieves a stable state in which the information being sent back and forth does not change after a few iterations [258]. Such a network (shown in Figure 13.1) is known as a resonance network or *bidirectional associative memory* (BAM). The activation function of the units is the sign function and information is coded using bipolar values.

The network in Figure 13.1 maps an n-dimensional row vector $\mathbf{x}_0$ to a k-dimensional row vector $\mathbf{y}_0$. We denote the $n \times k$ weight matrix of the network by $\mathbf{W}$ so that the mapping computed in the first step can be written as

$$\mathbf{y}_0 = \text{sgn}(\mathbf{x}_0 \mathbf{W}).$$

In the feedback step $\mathbf{y}_0$ is treated as the input and the new computation is

$$\mathbf{x}_1^{\text{T}} = \text{sgn}(\mathbf{W}\mathbf{y}_0^{\text{T}}).$$

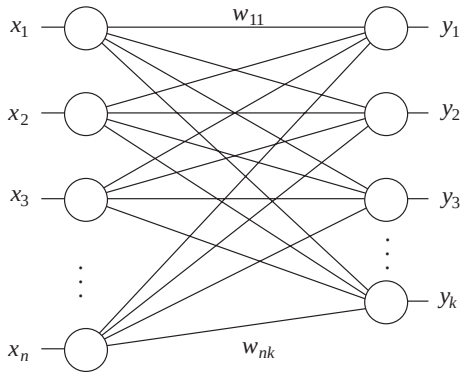

Fig. 13.1. Example of a resonance network (BAM)

A new computation from left to right produces

$$\mathbf{y}_1 = \text{sgn}(\mathbf{x}_1\mathbf{W}).$$

After m iterations the system has computed a set of $m+1$ vector pairs $(\mathbf{x}_0, \mathbf{y}_0), \ldots, (\mathbf{x}_m, \mathbf{y}_m)$ which fulfill the conditions

$$\mathbf{y}_i = \text{sgn}(\mathbf{x}_i\mathbf{W}) \tag{13.1}$$

and

$$\mathbf{x}_{i+1}^{\text{T}} = \text{sgn}(\mathbf{W}\mathbf{y}_i^{\text{T}}). \tag{13.2}$$

The question is whether after some iterations a fixpoint $(\mathbf{x}, \mathbf{y})$ is found. This is the case when both

$$\mathbf{y} = \text{sgn}(\mathbf{x}\mathbf{W}) \quad \text{and} \quad \mathbf{x}^{\text{T}} = \text{sgn}(\mathbf{W}\mathbf{y}^{\text{T}}) \tag{13.3}$$

hold. The BAM is thus a generalization of a unidirectional associative memory. An input vector, the "key", can be presented to the network from the left or from the right and, after some iterations, the BAM finds the corresponding complementary vector. As can be seen, no external feedback connections are necessary. The same edges are used for the transmission of information back and forth.

It can be immediately deduced from (13.3) that if a vector pair $(\mathbf{x}, \mathbf{y})$ is given and we want to condition a BAM to accept this pair as a fixed point, Hebbian learning can be used to compute an adequate matrix $\mathbf{W}$. If $\mathbf{W}$ is defined as $\mathbf{W} = \mathbf{x}^{\text{T}}\mathbf{y}$, as prescribed by Hebbian learning, then

$$\mathbf{y} = \text{sgn}(\mathbf{x}\mathbf{W}) = \text{sgn}(\mathbf{x}\mathbf{x}^{\text{T}}\mathbf{y}) = \text{sgn}(\|\mathbf{x}\|^2\mathbf{y}) = \mathbf{y}$$

and also

$$\mathbf{x}^{\text{T}} = \text{sgn}(\mathbf{W}\mathbf{y}^{\text{T}}) = \text{sgn}(\mathbf{x}^{\text{T}}\mathbf{y}\mathbf{y}^{\text{T}}) = \text{sgn}(\mathbf{x}^{\text{T}}\|\mathbf{y}\|^2) = \mathbf{x}^{\text{T}}.$$

If we want to store several vector pairs $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)$ in a BAM, then Hebbian learning works better if the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and $\mathbf{y}_1, \ldots, \mathbf{y}_m$ are pairwise orthogonal within their respective groups, because in that case the perturbation term becomes negligible (refer to Chap. 12).

For a set of m vector pairs the matrix $\mathbf{W}$ is set to

$$\mathbf{W} = \mathbf{x}_1^{\mathrm{T}}\mathbf{y}_1 + \mathbf{x}_2^{\mathrm{T}}\mathbf{y}_2 + \cdots + \mathbf{x}_m^{\mathrm{T}}\mathbf{y}_m.$$

BAMs can be used to build autoassociative networks because the matrices produced by the Hebb rule or by computing the pseudoinverse are symmetric. To see this, define $\mathbf{X}$ as the matrix, each of whose m rows is an n-dimensional vector, so that if $\mathbf{W}$ denotes the connection matrix of an autoassociative memory for those m vectors, then it is true that

$$\mathbf{X} = \mathbf{X}\mathbf{W} \quad \text{and} \quad \mathbf{X}^{\mathrm{T}} = \mathbf{W}\mathbf{X}^{\mathrm{T}},$$

because $\mathbf{W}$ is symmetric. This is just another way of writing the type of computation performed by a BAM.

13.1.3 The energy function

With the BAM we can motivate and explore the concept of an *energy function* in a simple setting. Assume that a BAM is given for which the vector pair $(\mathbf{x}, \mathbf{y})$ is a stable state. If the initial vector presented to the network from the left is $\mathbf{x}_0$, the network will converge to $(\mathbf{x}, \mathbf{y})$ after some iterations. The vector $\mathbf{y}_0$ is computed according to $\mathbf{y}_0 = \mathrm{sgn}(\mathbf{x}_0\mathbf{W})$. If $\mathbf{y}_0$ is now used for a new iteration from the right, excitation of the units in the left layer can be summarized in an excitation vector $\mathbf{e}$ computed according to

$$\mathbf{e}^{\mathrm{T}} = \mathbf{W}\mathbf{y}_0.$$

The vector pair $(\mathbf{x}_0, \mathbf{y}_0)$ is a stable state of the network if $\mathrm{sgn}(\mathbf{e}) = \mathbf{x}_0$. All vectors $\mathbf{e}$ close enough to $\mathbf{x}_0$ fulfill this condition. These vectors differ from $\mathbf{x}_0$ by a small angle and therefore the product $\mathbf{x}_0\mathbf{e}^{\mathrm{T}}$ is larger than for other vectors of the same length but further away from $\mathbf{x}_0$. The product

$$E = -\mathbf{x}_0\mathbf{e}^{\mathrm{T}} = -\mathbf{x}_0\mathbf{W}\mathbf{y}_0^{\mathrm{T}}$$

is therefore smaller (because of the minus sign) if the vector $\mathbf{W}\mathbf{y}_0^{\mathrm{T}}$ lies closer to $\mathbf{x}_0$. The scalar value E can be used as a kind of index of convergence to the stable states of an associative memory. We call E the *energy function* of the network.

Definition 16. *The energy function E of a BAM with weight matrix $\mathbf{W}$, in which the output $\mathbf{y}_i$ of the right layer of units is computed in the i-th iteration according to equation* (13.1) *and the output $\mathbf{x}_i$ of the left layer is computed according to* (13.2) *is given by*

$$E(\mathbf{x}_i, \mathbf{y}_i) = -\frac{1}{2}\mathbf{x}_i\mathbf{W}\mathbf{y}_i^{\mathrm{T}}. \tag{13.4}$$

The factor 1/2 will be useful later and is just a scaling constant for the energy function. In the following sections we show that the energy function assumes locally minimal values at stable states. The energy function can also be generalized to arbitrary vectors $\mathbf{x}$ and $\mathbf{y}$.

Up to this point we have only considered units with the sign function as activation nonlinearity in the type of associative memories we have discussed. If we now consider units with a threshold and the step function as its activation function, we must use a more general expression for the energy function. This can be done by extending the input vectors with an additional constant component. Each n-dimensional vector $\mathbf{x}$ will be transformed into the vector $(x_1, \ldots, x_n, 1)$. We proceed in a similar way with the k-dimensional vector $\mathbf{y}$. The weight matrix $\mathbf{W}$ must be extended to a new matrix $\mathbf{W}'$ with an additional row and column. The negative thresholds of the units in the right layer of the BAM are included in row $n+1$ of $\mathbf{W}'$, whereas the negative thresholds of the units in the left are used as the entries of the column $k+1$ of the weight matrix. The entry $(n+1, k+1)$ of the weight matrix can be set to zero. This transformation is equivalent to the introduction of an additional unit with constant output 1 into each layer. The weight of each edge from a constant unit to each one of the others is the negative threshold of the connected unit. It is straightforward to deduce that the energy function of the extended network can be written as

$$E(\mathbf{x}_i, \mathbf{y}_i) = -\frac{1}{2}\mathbf{x}_i \mathbf{W} \mathbf{y}_i^{\mathrm{T}} + \frac{1}{2}\theta_r \mathbf{y}_i^{\mathrm{T}} + \frac{1}{2}\mathbf{x}_i \theta_\ell^{\mathrm{T}}. \tag{13.5}$$

The row vector of thresholds of the k units in the left layer is denoted in the above expression by θ_ℓ. The row vector of thresholds of the n units in the right layer is denoted by θ_r.

13.2 Definition of Hopfield networks

So far we have considered only conventional or bidirectional associative memories working with synchronized units. Dropping the assumption of simultaneous firing of the computing elements leads to the appearance of novel network properties.

13.2.1 Asynchronous networks

In an asynchronous network each unit computes its excitation at random times and changes its state to 1 or -1 independently of the others and according to the sign of its total excitation. The probability of two units firing simultaneously is zero. Consequently, the same dynamics can be obtained by selecting one unit randomly, computing its excitation and updating its state accordingly. There will not be any delay between computation of the excitation and state update. We adopt the additional simplification that the state of a unit

is not changed if the total excitation is zero. This means that we leave the sign function undefined for the argument zero. Asynchronous networks are of course more realistic models of biological networks, although the assumption of zero delay in the computation and transmission of signals lacks any biological basis.

Using the energy function it can be shown that a BAM arrives at a stable state after a finite number of iterations. A stable state is a vector pair $(\mathbf{x}, \mathbf{y})$ which fulfills the conditions (13.3). When a BAM reaches this state pair, no component of the bipolar vectors $\mathbf{x}$ and $\mathbf{y}$ can be changed without contradicting (13.3). The vector pair $(\mathbf{x}, \mathbf{y})$ is therefore also a stable state for an asynchronous network.

Proposition 19. *A bidirectional associative memory with an arbitrary weight matrix* $\mathbf{W}$ *reaches a stable state in a finite number of iterations using either synchronous or asynchronous updates.*

Proof. For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, a vector $\mathbf{y} = (y_1, y_2, \ldots, y_k)$ and an $n \times k$ weight matrix $\mathbf{W} = \{w_{ij}\}$ the energy function is the bilinear form

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(x_1, x_2, \ldots, x_n) \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nk} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}.$$

The value of $E(\mathbf{x}, \mathbf{y})$ can be computed by multiplying first $\mathbf{W}$ by $\mathbf{y}^{\mathrm{T}}$ and the result with $-\mathbf{x}/2$. The product of the i-th row of $\mathbf{W}$ and $\mathbf{y}^{\mathrm{T}}$ represents the excitation of the i-th unit in the left layer. If we denote these excitations by $g_1, g_2, \ldots, g_n$ the above expression transforms to

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(x_1, x_2, \ldots, x_n) \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}.$$

We can also compute $E(\mathbf{x}, \mathbf{y})$ multiplying first $\mathbf{x}$ by $\mathbf{W}$. The product of the i-th column of $\mathbf{W}$ with $\mathbf{x}$ corresponds to the excitation of unit i in the right layer. If we denote these excitations by $e_1, e_2, \ldots, e_k$, the expression for $E(\mathbf{x}, \mathbf{y})$ can be written as

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(e_1, e_2, \ldots, e_k) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix}.$$

Therefore, the energy function can be written in the two equivalent forms

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}\sum_{i=1}^{k} e_i y_i \quad \text{and} \quad E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}\sum_{i=1}^{n} g_i x_i.$$

In asynchronous networks at each time t we randomly select a unit from the left or right layer. The excitation is computed and its sign is the new activation of the unit. If the previous activation of the unit remains the same after this operation, then the energy of the network has not changed.

The state of unit i on the left layer will change only when the excitation g_i has a different sign than x_i, the present state. The state is updated from x_i to x'_i, where x'_i now has the same sign as g_i. Since the other units do not change their state, the difference between the previous energy $E(\mathbf{x}, \mathbf{y})$ and the new energy $E(\mathbf{x}', \mathbf{y})$ is

$$E(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}', \mathbf{y}) = -\frac{1}{2} g_i (x_i - x'_i).$$

Since both x_i and $-x_i$ have a different sign than g_i it follows that

$$E(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}', \mathbf{y}) > 0.$$

The new state $(\mathbf{x}', \mathbf{y})$ has a lower energy than the original state $(\mathbf{x}, \mathbf{y})$. The same argument can be made if a unit on the right layer has been selected, so that for the new state $(\mathbf{x}, \mathbf{y}')$ it holds that

$$E(\mathbf{x}, \mathbf{y}) - E(\mathbf{x}, \mathbf{y}') > 0,$$

whenever the state of a unit in the right layer has been flipped.

Any update of the network state reduces the total energy. Since there are only a finite number of possible combinations of bipolar states, the process must stop at some point, that is, a state $(\mathbf{a}, \mathbf{b})$ is found whose energy cannot be further reduced. The network has fallen into a local minimum of the energy function and the state $(\mathbf{a}, \mathbf{b})$ is an attractor of the system. □

The above proposition also holds for synchronous networks, since these can be considered as a special case of asynchronous dynamics. Note that the proposition puts conditions on the matrix $\mathbf{W}$. This means that any given real matrix $\mathbf{W}$ possesses bidirectional stable bipolar states.

13.2.2 Examples of the model

In 1982 the American physicist John Hopfield proposed an asynchronous neural network model which made an immediate impact in the AI community. It is a special case of a bidirectional associative memory, but chronologically it was proposed before the BAM.

In the Hopfield model it is assumed that the individual units preserve their individual states until they are selected for a new update. The selection is made randomly. A Hopfield network consists of n totally coupled units, that is, each unit is connected to all other units except itself. The network is symmetric because the weight w_{ij} for the connection between unit i and

unit j is equal to the weight w_{ji} of the connection from unit j to unit i. This can be interpreted as meaning that there is a single bidirectional connection between both units. The absence of a connection from each unit to itself avoids a permanent feedback of its own state value [198].

Figure 13.2 shows an example of a network with three units. Each one of them can assume the state 1 or -1. A Hopfield network can also be interpreted as an asynchronous BAM in which the left and right layers of units have fused to a single layer. The connections in a Hopfield network with n units can be represented using an $n \times n$ weight matrix $\mathbf{W} = \{w_{ij}\}$ with a zero diagonal.

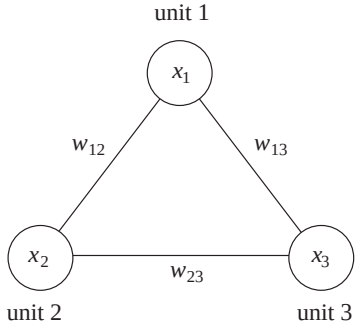

Fig. 13.2. A Hopfield network of three units

It is easy to show that if the weight matrix does not contain a zero diagonal, the network dynamics does not necessarily lead to stable states. The weight matrix

$$\mathbf{W} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix},$$

for example, transforms the state vector $(1, 1, 1)$ into the state vector $(-1, -1, -1)$ and conversely. In the case of asynchronous updating, the network chooses randomly among the eight possible network states.

A connection matrix with a zero diagonal can also lead to oscillations in the case where the weight matrix is not symmetric. The weight matrix

$$\mathbf{W} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

describes the network of Figure 13.3. It transforms the state vector $(1, -1)$ into the state vector $(1, 1)$ when the network is running asynchronously. After this transition the state $(-1, 1)$ can be updated to $(-1, -1)$ and finally to $(1, -1)$. The state vector changes cyclically and does not converge to a stable state.

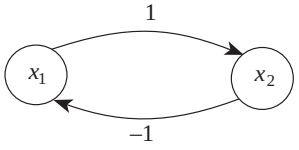

Fig. 13.3. Network with asymmetric connections

The symmetry of the weight matrix and a zero diagonal are thus *necessary conditions* for the convergence of an asynchronous totally connected network to a stable state. These conditions are also sufficient, as we show later.

The units of a Hopfield network can be assigned a threshold θ different from zero. In this case each unit selected for a state update adopts the state 1 if its total excitation is greater than θ, otherwise the state -1. This is the activation rule for perceptrons, so that we can think of Hopfield networks as asynchronous recurrent networks of perceptrons.

The energy function of a Hopfield network composed of units with thresholds different from zero can be defined in a similar way as for the BAM. In this case the vector $\mathbf{y}$ of equation (13.5) is $\mathbf{x}$ and we let $\theta = \theta_\ell = \theta_r$.

Definition 17. *Let* $\mathbf{W}$ *denote the weight matrix of a Hopfield network of* n *units and let* θ *be the* n*-dimensional row vector of units' thresholds. The energy* $E(\mathbf{x})$ *of a state* $\mathbf{x}$ *of the network is given by*

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}\mathbf{W}\mathbf{x}^{\mathrm{T}} + \theta\mathbf{x}^{\mathrm{T}}.$$

The energy function can also be written in the form

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i x_i.$$

The factor $1/2$ is used because the identical terms $w_{ij}x_i x_j$ and $w_{ji}x_j x_i$ are present in the double sum.

The energy function of a Hopfield network is a quadratic form. A Hopfield network always finds a local minimum of the energy function. It is thus interesting to look at an example of the shape of such an energy function. Figure 13.4 shows a network of just two units with threshold zero. It is obvious that the only stable states are $(1,-1)$ and $(-1,1)$. In any other state, one of the units forces the other to change its state to stabilize the network. Such a network is a flip-flop, a logic component with two outputs which assume complementary logic values.

The energy function of a flip-flop with weights $w_{12} = w_{21} = -1$ and two units with threshold zero is given by

$$E(x_1, x_2) = x_1 x_2,$$

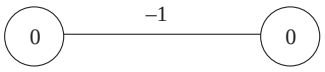

Fig. 13.4. A flip-flop

where x_1 and x_2 denote the states of the first and second units respectively. Figure 13.5 shows the energy function for the so-called continuous Hopfield model [199] in which the unit's states can assume all real values between 0 and 1. In the network of Figure 13.4 only the four discrete states $(1,1)$, $(1,-1)$, $(-1,1)$ and $(-1,-1)$ are allowed. The energy function has local minima at $(1,-1)$ and $(-1,1)$. A flip-flop can therefore be interpreted as a network capable of storing one of the states $(1,-1)$ or $(-1,1)$.

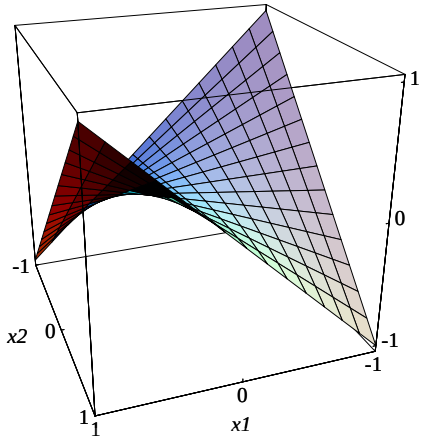

Fig. 13.5. Energy function of a flip-flop

Hopfield networks can also be used to compute logical functions. Conjunction, for example, can be implemented with a network of three units. The states of two units are set and remain fixed during the computation (clamping their states). Only the third unit can change its state. If the network weights and the unit thresholds have the appropriate values, the unconstrained unit will assume a state which corresponds to the conjunction of the two clamped states.

Figure 13.6 shows a network for the computation of the logical disjunction of two Boolean values x_1 and x_2. The input is clamped and after some time the network settles to a state which corresponds to the disjunction of x_1 and x_2. The constants "true" and "false" correspond to the numerical values 1 and -1. In this network the thresholds of the clamped units and their mutual connections play no role in the computation.

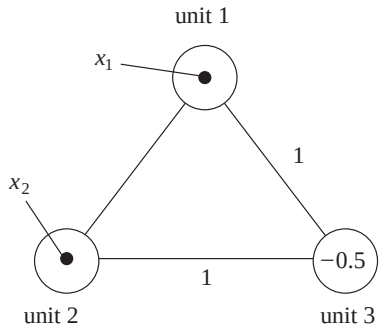

Fig. 13.6. Network for the computation of the OR function

Since the individual units of the network are perceptrons, the question of whether there are logic functions which cannot be computed by a Hopfield network of a given size arises. This is the case in our next example. Assume that a Hopfield network of three units should store the set of stable states given by the following table:

unit	1	2	3
state 1	−1	−1	−1
state 2	1	−1	1
state 3	−1	1	1
state 4	1	1	−1

From the point of view of the third unit (third column) this is the XOR function. If the four vectors shown above are to become stable states of the network, the third unit cannot change state when any of these four vectors has been loaded in the network. In this case the third unit should be capable of linearly separating the vectors $(-1,-1)$ and $(1,1)$ from the vectors $(-1,1)$ and $(1,-1)$, which we know is impossible. The same argument is valid for any of the three units, since the table given above remains unchanged after a permutation of the units' labels. This shows that no Hopfield network of three units can have these stable states. However, the XOR problem can be solved if the network is extended to four units. The network of Figure 13.7 can assume the following stable states, if adequate weights and thresholds are selected:

unit	1	2	3	4
state 1	−1	−1	−1	1
state 2	1	−1	1	1
state 3	−1	1	1	1
state 4	1	1	−1	−1

The third column represents the XOR function of the two first columns. The fourth column corresponds to an auxiliary unit, whose state can be set from

outside. The unknown weights can be found using the learning algorithms described in the next sections.

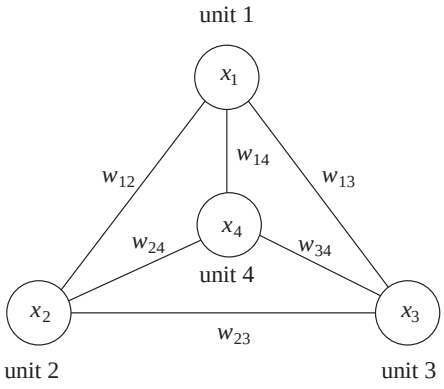

Fig. 13.7. Network for the computation of XOR

13.2.3 Isomorphism between the Hopfield and Ising models

Physicists have analyzed the Hopfield model in such exquisite detail because it is isomorphic to the *Ising model* of magnetism (at temperature zero) [25]. Ising proposed the model which now bears his name more than 70 years ago in order to describe some properties of ensembles of elementary magnets [214].

In general, the Ising model can be used to describe those systems made of particles capable of adopting one of two states. In the case of ferromagnetic materials, their atoms can be modeled as particles of spin $1/2$ (*up*) or spin $-1/2$ (*down*). The spin points in the direction of the magnetic field. All tiny magnets interact with each other. This causes some of the atoms to flip their spin until equilibrium is reached and the total magnetization of the material reaches a constant level, which is the sum of the individual spins. With these few assumptions we can show that the energy function deduced from the Ising model has the same form as the energy function of Hopfield networks.

The total magnetic field h_i sensed by the atom i in an ensemble of particles is the sum of the fields induced by each atom and the external field h^* (if present), that is

$$h_i = \sum_{j=1}^{n} w_{ij} x_j + h^*, \tag{13.6}$$

where w_{ij} represents the magnitude of the magnetic coupling between the atoms labeled i and j. The magnetic coupling changes according to the distance between atoms and the magnetic permeability of the environment. The

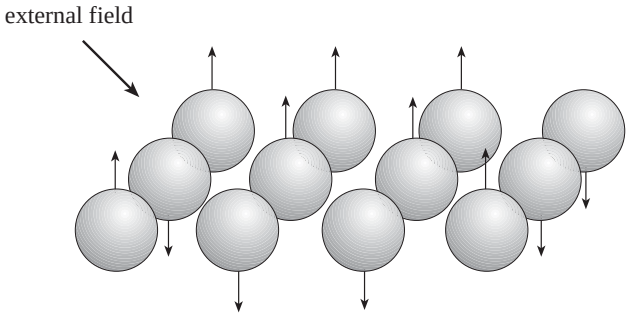

Fig. 13.8. Particles with two possible spins

potential energy E of a certain state $(x_1, x_2, \ldots, x_n)$ of an Ising material can be derived from (13.6) and has the form

$$E = -\frac{1}{2}\sum_{i,j}^{n} w_{ij}x_i x_j + \sum_{i}^{n} -h^* x_i. \tag{13.7}$$

In paramagnetic materials the coupling constants are zero. In ferromagnetic materials the constants w_{ij} are all positive, which leads in turn to a significant coupling of the spin states.

Equation (13.7) is isomorphic to the energy function of Hopfield networks. This is why the term *energy function* is used in the first place. Both systems are dynamically equivalent, but only in the case of zero temperature, since the system behaves deterministically at each state update. Later on, when we consider *Boltzmann machines*, we will accept a time-varying temperature and stochastic state updates as in the full Ising model.

13.3 Converge to stable states

It is easy to show that Hopfield models always converge to stable states. The proof of this fact relies on analysis of the new value of the energy function after each state update.

13.3.1 Dynamics of Hopfield networks

Before going into the details of the convergence proof, we analyze two simple examples and compute the energy levels of all their possible states. Figure 13.9 shows a network composed of three units with arbitrarily chosen weights and thresholds. The network can adopt any of eight possible states whose transitions we want to visualize. Figure 13.10 shows a diagram of all possible state transitions for the network of Figure 13.9. The vertical axis represents the energy of the network defined in the usual way. Each state of the network is

represented by an oval located at its precise *energy level*. The arrows show the state transitions allowed. Each transition has the same probability because the probability of selecting one of the three units for a state transition is uniform and equal to 1/3. Note that the diagram does not show the few transitions in which a state returns to itself.

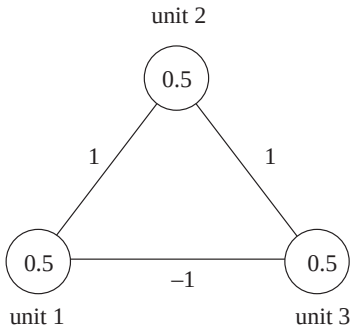

Fig. 13.9. Example of a Hopfield network

We can make other interesting observations in the transition diagram. The state $(1, -1, 1)$, for example, is extremely unstable. The probability of leaving it at the next iteration is 1, because three different transitions to other states are possible, each with probability 1/3. The state $(-1, 1, 1)$ is relatively stable because the probability of leaving it at the next iteration is just 1/3. There is only a single stable state, namely the vector $(-1, -1, -1)$, as the reader can readily verify. The only two states without a predecessor are shown in gray. In the theory of cellular automata, such "urstates" are called *garden of Eden* configurations. They cannot be arrived at, they can only be induced from the outside before the automaton starts working.

The network in Figure 13.11 has the same structure as the network considered previously, but the weights and thresholds have the opposite sign. The diagram of state transitions (Figure 13.12) is the inversion of the diagram in Figure 13.10. The new network has two stable states and just one state without predecessors. As can be seen from the diagrams, the dynamic of the Hopfield model is always the same: the energy of the system eventually reaches a local minimum and the state of the network can no longer change.

13.3.2 Convergence proof

We can now proceed to prove that, in general, Hopfield models behave in the way shown in the last two examples.

Proposition 20. *A Hopfield network with n units and asynchronous dynamics, which starts from any given network state, eventually reaches a stable state at a local minimum of the energy function.*

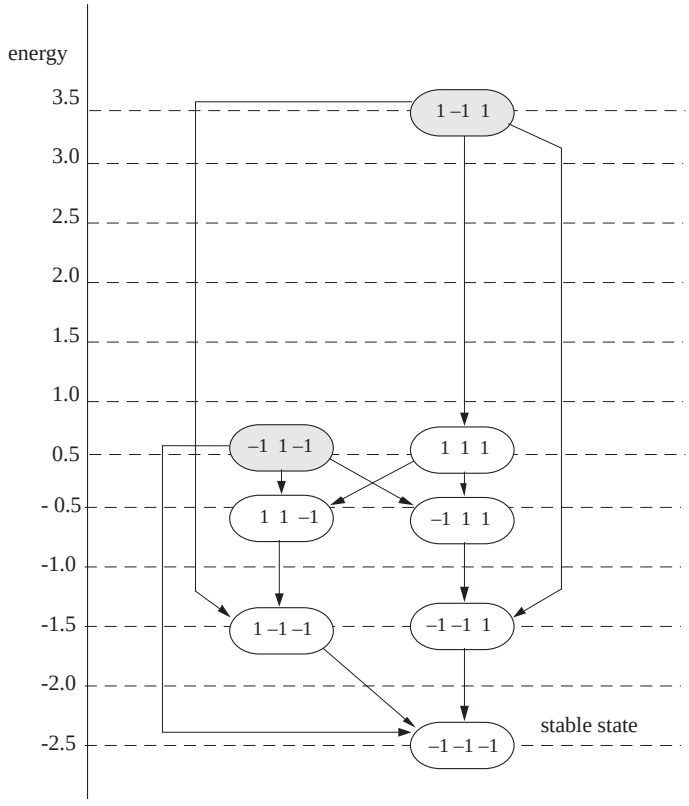

Fig. 13.10. State transitions for the network of Figure 13.9

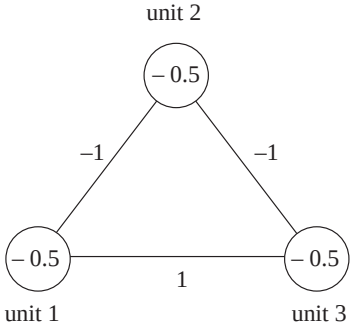

Fig. 13.11. Second example of a Hopfield network

Proof. The energy function of a state $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of a Hopfield network with n units is given by

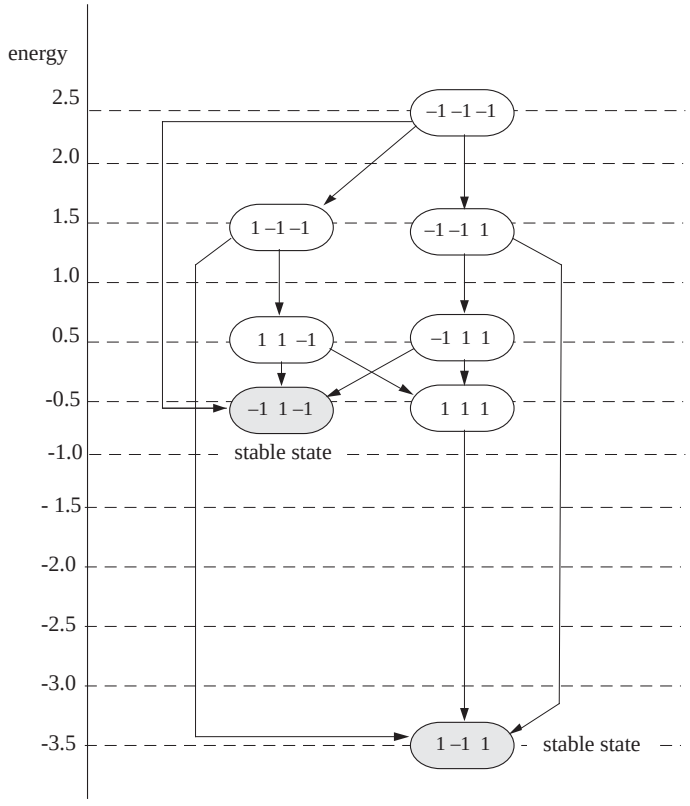

Fig. 13.12. State transitions for the network of Figure 13.11

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i x_i, \tag{13.8}$$

where the terms involved are defined as usual. If during the current iteration unit k is selected and does not change its state, then the energy of the system does not change either. If the state of the unit is changed in the update operation, the network reaches a new global state $\mathbf{x}' = (x_1, \ldots, x'_k, \ldots, x_n)$ for which the new energy is $E(\mathbf{x}')$. The difference between $E(\mathbf{x})$ and $E(\mathbf{x}')$ is given by all terms in the summation in (13.8) which contain x_k and x'_k, that is

$$E(\mathbf{x}) - E(\mathbf{x}') = (-\sum_{j=1}^{n} w_{kj}x_k x_j + \theta_k x_k) - (-\sum_{j=1}^{n} w_{kj}x'_k x_j + \theta_k x'_k).$$

The factor $1/2$ disappears from the computation because the terms $w_{kj}x_k x_j$ appear twice in the double sum of (13.8). Since $w_{kk} = 0$ we can rewrite the above equation as

$$E(\mathbf{x}) - E(\mathbf{x}') = -(x_k - x'_k)\sum_{j=1}^{n} w_{kj}x_j + \theta_k(x_k - x'_k)$$
$$= -(x_k - x'_k)(\sum_{j=1}^{n} w_{kj}x_j - \theta_k),$$

from which we finally obtain

$$E(\mathbf{x}) - E(\mathbf{x}') = -(x_k - x'_k)e_k,$$

where e_k denotes the total excitation of unit k (including subtraction of the threshold). The excitation e_k has a different sign from x_k and $-x'_k$, because otherwise the unit state would not have been changed. This means that the product $-(x_k - x'_k)e_k$ is positive and therefore

$$E(\mathbf{x}) - E(\mathbf{x}') > 0.$$

This shows that every time the state of a unit is altered, the total energy of the network is reduced. Since there is only a finite set of possible states, the network must eventually reach a state for which the energy cannot be reduced further. It is a stable state of the network, as we wanted to prove. □

There is a simpler proof of the last proposition, which has the advantage of offering a nice visualization of the dynamics of a Hopfield network [74]. Assume that we classify the units of a network according to their state: the first set contains the units with state 1, the second set the units with state -1. There are edges linking every unit with all the others, so that some edges go from one set to the other. We now randomly select one of the units and compute its "attraction" by the units in its own set and the attraction by the units in the other set. The "attraction" is the sum of the weights of all edges between a unit and the units in its set or in the other one. If the attraction from the outside is greater than the attraction from its own set, the unit changes sides by altering its state. If the external attraction is lower than the internal, the unit keeps its current state. This procedure is repeated several times, each time selecting one of the units randomly. It corresponds to the updating strategy of a Hopfield network. Figure 13.13 shows an example in which the attraction from the outside is greater than the internal one. The selected unit must change sides. It is clear that the network must eventually reach a stable state, because the sum of the weights of all edges connecting one set to the other can only become lower in the course of time. Since the number of possible network states is finite, a global state must be reached in which the attraction of one set by the other cannot be further reduced. This is the task known in combinatorics as the *minimal cut* problem, in which we want to find a cut of minimal flow in a graph. The procedure described always finds a locally minimal cut.

The wording of Proposition 20 has been carefully chosen. That the network "eventually" settles in a stable state, means that the probability of not

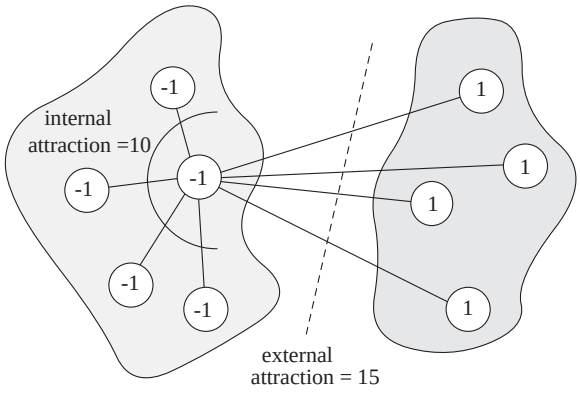

Fig. 13.13. Attraction from the inside and from the outside of a unit's class

reaching such a state approaches zero as the number of iterations increases. It would be possible to select always one and the same unit for computation of the excitation, and in this case the network would stay in deadlock. Since the units are selected randomly, the probability of such pathological behavior falls to zero as time progresses.

In the proof of Proposition 20 only the symmetry and the zero diagonal of the weight matrix were used. The proof of convergence is very similar to the proof of convergence for the BAM. However, in the case of a BAM the decisive property was the independence of a unit's state from its own excitation. This is also the case for Hopfield networks, since no unit feeds its own state back into itself, i.e., the diagonal of the weight matrix is zero.

13.3.3 Hebbian learning

A Hopfield network can be used as an associative memory. If we want to "imprint" m different stable states in the network we have to find adequate weights for the connections. In the case of the BAM we already mentioned that Hebbian learning is a possible alternative. Since Hopfield networks are a specialization of BAM networks, we also expect Hebbian learning to be applicable in this case. Let us first discuss the case of a Hopfield network with n units and threshold zero.

Hebbian learning is implemented by loading the m selected n-dimensional stable states $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ on the network and by updating the network's weights (initially set to zero) after each presentation according to the rule

$$w_{ij} \leftarrow w_{ij} + x_i^k x_j^k, \qquad i, j = 1, \ldots, n \quad \text{and} \quad i \neq j.$$

The symbols x_i^k and x_j^k denote the i-th and j-th component respectively of the vector $\mathbf{x}_k$. The only difference from an autoassociative memory is the

requirement of a zero diagonal. After presentation of the first vector $\mathbf{x}_1$ the weight matrix is given by the expression

$$\mathbf{W}_1 = \mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 - \mathbf{I},$$

where $\mathbf{I}$ denotes the $n \times n$ identity matrix. Subtraction of the identity matrix guarantees that the diagonal of $\mathbf{W}$ becomes zero, since for any bipolar vector $\mathbf{x}_i$ it holds that $x_k^i x_k^i = 1$. Obviously $\mathbf{W}_1$ is a symmetric matrix.

The minimum of the energy function of a Hopfield network with the weight matrix $\mathbf{W}_1$ is located at $\mathbf{x}_1$ because

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}\mathbf{W}_1\mathbf{x}^{\mathrm{T}} = -\frac{1}{2}(\mathbf{x}\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1\mathbf{x}^{\mathrm{T}} - \mathbf{x}\mathbf{x}^{\mathrm{T}})$$

and $\mathbf{x}\mathbf{x}^{\mathrm{T}} = n$ for bipolar vectors. This means that the function

$$E(\mathbf{x}) = -\frac{1}{2}\|\mathbf{x}\mathbf{x}_1^{\mathrm{T}}\|^2 + \frac{n}{2}$$

has a local minimum at $\mathbf{x} = \mathbf{x}_1$. In this case it holds that

$$E(\mathbf{x}) = -\frac{n^2}{2} + \frac{n}{2}.$$

This shows that $\mathbf{x}_1$ is a stable state of the network.

In the case of m different vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ the matrix $\mathbf{W}$ is defined as

$$\mathbf{W} = (\mathbf{x}_1\mathbf{x}_1^{\mathrm{T}} - \mathbf{I}) + (\mathbf{x}_2^{\mathrm{T}}\mathbf{x}_2 - \mathbf{I}) + \cdots + (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_m - \mathbf{I}),$$

or equivalently

$$\mathbf{W} = \mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 + \mathbf{x}_2^{\mathrm{T}}\mathbf{x}_2 + \cdots + \mathbf{x}_m^{\mathrm{T}}\mathbf{x}_m - m\mathbf{I}.$$

If the network is initialized with the state $\mathbf{x}_1$, the vector $\mathbf{e}$ of the excitation of the units is

$$\begin{aligned}\mathbf{e} &= \mathbf{x}_1\mathbf{W} \\ &= \mathbf{x}_1\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_1 + \mathbf{x}_1\mathbf{x}_2^{\mathrm{T}}\mathbf{x}_2 + \cdots + \mathbf{x}_1\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_m - m\mathbf{x}_1\mathbf{I} \\ &= (n-m)\mathbf{x}_1 + \sum_{j=2}^{m}\alpha_{1j}\mathbf{x}_j.\end{aligned}$$

The constants $\alpha_{12}, \alpha_{13}, \ldots, \alpha_{1m}$ represent the scalar products of the first vector with each one of the other $m-1$ vectors $\mathbf{x}_2, \ldots, \mathbf{x}_m$. The state $\mathbf{x}_1$ is stable when $m < n$ and the perturbation term $\sum_{j=2}^{m}\alpha_{1j}\mathbf{x}_j$ is small. In this case it holds that

$$\mathrm{sgn}(\mathbf{e}) = \mathrm{sgn}(\mathbf{x}_1)$$

as desired. The same argumentation can be used for any of the other vectors. The best results are achieved with Hebbian learning when the vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ are orthogonal or close to orthogonal, just as in the case of any other associative memory.

13.4 Equivalence of Hopfield and perceptron learning

Hebbian learning is a simple rule which is useful for the computation of the weight matrix in Hopfield networks. However, sometimes Hebbian learning cannot find a weight matrix for which m given vectors are stable states, although such a matrix exists. If the vectors to be stored lie near each other, the perturbation term can grow so large as to preclude a solution by Hebbian learning. In this case another learning rule is needed, which is a variant of perceptron learning.

13.4.1 Perceptron learning in Hopfield networks

Let us consider Hopfield networks composed of units with a non-zero threshold and the step function as activation function. The units adopt state 1 when the excitation is greater than the threshold and otherwise the state -1. The units are just perceptrons and it is straightforward to assume that perceptron learning could be used for determination of the weights and thresholds of the network for a given learning problem.

Let n denote the number of units in a Hopfield network, let $\mathbf{W} = \{w_{ij}\}$ be the $n \times n$ weight matrix, and let θ_i denote the threshold of unit i. If a vector $\mathbf{x} = (x_1, \ldots, x_n)$ is given to be "imprinted" on the network, this vector will be a stable state only when, if loaded in the network, the network global state does not change. This is the case if for every unit its excitation minus its threshold has the same sign as the current state (the value zero is assigned the minus sign). This means that the following n inequalities must hold:

$$
\begin{array}{lccccccc}
\text{For unit 1} : \operatorname{sgn}(x_1)(0 & + & x_2 w_{12} & + & x_3 w_{13} & + & \cdots \\
 & + & x_n w_{1n} & - & \theta_1) & < & 0 \\
\text{For unit 2} : \operatorname{sgn}(x_2)(x_1 w_{21} & + & 0 & + & x_3 w_{23} & + & \cdots \\
 & + & x_n w_{2n} & - & \theta_2) & < & 0 \\
\vdots & & & & & & \\
\text{For unit n} : \operatorname{sgn}(x_n)(x_1 w_{n1} & + & x_2 w_{n2} & + & \cdots & + & x_{n-1} w_{nn-1} \\
 & + & 0 & - & \theta_n) & < & 0
\end{array}
$$

The factor $\operatorname{sgn}(x_i)$ is used in each inequality to obtain always the same inequality operator ("less than"). Only the $n(n-1)/2$ non-zero entries of the weight matrix as well as the n thresholds of the units appear in these inequalities. Let $\mathbf{v}$ denote a vector of dimension $n + n(n-1)/2$ whose components are the non-diagonal entries w_{ij} of the weight matrix $\mathbf{W}$ (with $i < j$ so as to consider each weight only once) and the n thresholds with minus sign. The vector $\mathbf{v}$ is given by

$$
\mathbf{v} = (\underbrace{w_{12}, w_{13}, \ldots, w_{1n}}_{n-1}, \underbrace{w_{23}, w_{24}, \ldots, w_{2n}}_{n-2}, \ldots, \underbrace{w_{n-1n}}_{1}, \underbrace{-\theta_1, \ldots, -\theta_n}_{n}).
$$

The vector $\mathbf{x}$ is transformed into n auxiliary vectors $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$ of dimension $n + n(n-1)/2$ given by the expression

$$\begin{aligned}
\mathbf{z}_1 &= (\underbrace{x_2, x_3, \ldots, x_n}_{n-1}, 0, 0, \ldots, \underbrace{1, 0, \ldots, 0}_{n}) \\
\mathbf{z}_2 &= (\underbrace{x_1, 0, \ldots, 0}_{n-1}, \underbrace{x_3, \ldots, x_n}_{n-2}, 0, 0, \ldots, \underbrace{0, 1, \ldots, 0}_{n}) \\
&\vdots \\
\mathbf{z}_n &= (\underbrace{0, 0, \ldots, x_1}_{n-1}, \underbrace{0, 0, \ldots, x_2}_{n-2}, 0, 0, \ldots, \underbrace{0, 0, \ldots, 1}_{n}).
\end{aligned}$$

The components of the vectors $\mathbf{z}_1, \ldots, \mathbf{z}_n$ were defined so that the previous inequalities for each unit can be written in the equivalent form

$$\begin{aligned}
&\text{unit } 1 \ \operatorname{sgn}(x_1)\mathbf{z}_1 \cdot \mathbf{v} > 0 \\
&\text{unit } 2 \ \operatorname{sgn}(x_2)\mathbf{z}_2 \cdot \mathbf{v} > 0 \\
&\vdots \\
&\text{unit } n \ \operatorname{sgn}(x_n)\mathbf{z}_n \cdot \mathbf{v} > 0
\end{aligned}$$

The vectors $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$ can always be defined in this way. We will not write down the exact transformation rule here because it is rather involved.

The last set of inequalities shows that the solution to the original problem is found by computing a linear separation of the vectors $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$. The vectors which belong to the positive half-space are those for which $\operatorname{sgn}(x_i)$ holds. The vectors which belong to the negative half-space are those for which $\operatorname{sgn}(x_i) = -1$. This problem can be solved using perceptron learning, which allows us to compute the vector $\mathbf{v}$ of weights needed for the linear separation, and from this we can deduce the weight matrix $\mathbf{W}$.

In the case where m vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ are given to be imprinted in the Hopfield network, we have to use the above transformation for every one of them. Each vector is transformed into n auxiliary vectors, so that at the end we have nm different auxiliary vectors which must be linearly separated. If they are actually linearly separable, perceptron learning will find the solution to the problem, coded in the vector $\mathbf{v}$ of the transformed perceptron.

The analysis performed above shows that it is possible to transform a learning problem in a Hopfield network with n units into a learning problem for a perceptron of dimension $n + n(n-1)/2$, that is, $n(n+1)/2$. Figure 13.14 shows an example of a Hopfield network that can be transformed into the equivalent perceptron to the right. The three-dimensional Hopfield problem is transformed in this way into a learning problem for a six-dimensional perceptron.

Each iteration of the perceptron learning algorithm updates only the weights of the edges attached to a single unit and its threshold. For example, if a correction is needed because of the sign of $\mathbf{z}_1 \cdot \mathbf{v}$, then only the weights

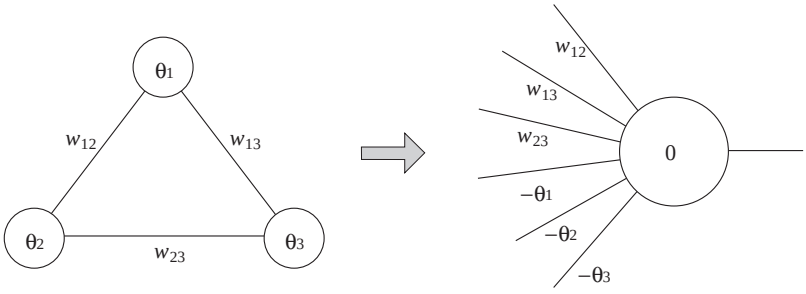

Fig. 13.14. Transformation of a Hopfield network into a perceptron

$w_{12}, w_{13}, \ldots, w_{1n}$ and the threshold θ_1 must be updated. This means that it is possible to use perceptron learning or the delta rule locally. During training all units are set to the desired stable states. If the sign of a unit's excitation is incorrect for the desired state, then the weights and threshold of this individual perceptron are corrected in the usual manner. It is not necessary to transform the Hopfield states into the $n(n+1)/2$-dimensional perceptron states every time we want to start the learning algorithm. This is only needed to prove the equivalence of Hopfield and perceptron learning.

13.4.2 Complexity of learning in Hopfield models

The interesting result which can immediately be inferred from the equivalence of Hopfield networks and perceptrons is that every learning algorithm for perceptrons can be transformed into a learning method for Hopfield networks. The delta rule or algorithms that proceed by finding inner points of solution polytopes can also be used to train Hopfield networks.

We have already shown in Chap. 10 that learning problems for multilayer networks are in general *NP*-complete. However, some special architectures can be trained in polynomial time. We saw in Chap. 4 that the learning problem for Hopfield networks can be solved in polynomial time, because there are learning algorithms for perceptrons whose complexity grows polynomially with the number of training vectors and their dimension (for example, Karmarkar's algorithm). Since the transformation described in the previous section converts m desired stable states into nm vectors to be linearly separated, and since this can be done in polynomial time, it follows that the learning problem for Hopfield networks can be solved in polynomial time. In Chap. 6 we also showed how to compute an upper bound for the number of linearly separable functions. This upper bound, valid for perceptrons, is also valid for Hopfield networks, since the stable states must be linearly separable (for the equivalent perceptron). This equivalence simplifies computation of the capacity of a Hopfield network when it is used as an associative memory.

13.5 Parallel combinatorics

The networks analyzed in the previous sections can be used either to compute Boolean functions or as associative memories. Those recurrent networks for which an energy function of a certain form exists can be used to solve some difficult problems in the fields of combinatorics and optimization theory. Hopfield networks have been proposed for these kinds of tasks.

13.5.1 *NP*-complete problems and massive parallelism

Many complex problems can be solved in a reasonable length of time using multiprocessor systems and parallel algorithms. This is easier for tasks that can be divided into independent subproblems, which are then assigned to different processors. The solution to the original problem is obtained by collecting the partial results after they have been computed. However, many well-known and important problems cannot be split in this manner. The parallel processes must cooperate and exchange information, so that the programmer must include some synchronization primitives in the system. If synchronization consumes too many resources and too much time, the parallel system may become only marginally faster than a sequential one.

Hopfield networks do not need any kind of synchronization; they guarantee that a local minimum of the energy function will be reached. If an optimization problem can be written in an analytical form isomorphic to the Hopfield energy function, it can be solved by a Hopfield network. We can assume that every unit in the network is simulated by a small processor. The states of the units can be computed asynchronously by transmitting the current unit states from processor to processor. There is no need for expensive synchronization and the task is solved by a massively parallel system. This strategy can be applied to all those combinatorial problems for whose solution large mainframes have traditionally been used.

We now show how to "load" an optimization problem on a Hopfield network discussing some progressively complicated examples. In the next subsections we will use the usual coding (with 0 and 1) for binary vectors and not the bipolar coding used in the previous examples.

13.5.2 The multiflop problem

Assume that we are looking for a binary vector of dimension n whose components are all zero with the exception of a single 1. The Hopfield network that solves this problem when $n = 4$ is depicted in Figure 13.15. Whenever a unit is set to 1, it inhibits the other units through the edges with weight -2. If the network is started with all units set to zero, then the excitation of every unit is zero, which is greater than the threshold and therefore the first unit to be asynchronously selected will flip its state to 1. No other unit can change its state after this first unit has been set to 1. A stable state has been reached.

One may think of this network as a generalization of the flip-flop network for two-dimensional vectors.

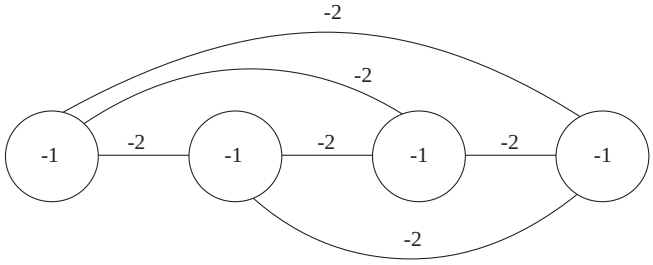

Fig. 13.15. A multiflop network

The weights for this network can be deduced from the following considerations. Let $x_1, x_2, \ldots, x_n$ denote the binary states of the individual units. Our task is to find a minimum of

$$E(x_1, \ldots, x_n) = (\sum_{i=1}^{n} x_i - 1)^2.$$

This expression can also be written as

$$E(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i^2 + \sum_{i \neq j}^{n} x_i x_j - 2 \sum_{i=1}^{n} x_i + 1.$$

For binary states it holds that $x_i = x_i^2$ and therefore

$$E(x_1, \ldots, x_n) = \sum_{i \neq j}^{n} x_i x_j - \sum_{i=1}^{n} x_i + 1$$

which can be rewritten as

$$E(x_1, \ldots, x_n) = -\frac{1}{2} \sum_{i \neq j}^{n} (-2) x_i x_j + \sum_{i=1}^{n} (-1) x_i + 1.$$

This expression is isomorphic to the energy function of the Hopfield network of Figure 13.15 (not considering the constant 1, which is irrelevant for the optimization problem). The network solves the multiflop problem in an automatic way by following its inherent dynamics.

13.5.3 The eight rooks problem

We make the optimization problem a notch more complicated: n rooks must be positioned in an $n \times n$ chess board so that no one figure can take another.

It is thus necessary to position each rook in a different row and column to the others. This problem can be thought of as a two-dimensional generalization of the multiflop problem. Each row is a chain of cells and only one of them can be set to 1. The same holds for each column.

The network of Figure 13.16 can solve this problem for a 4×4 board. Each field is represented by a unit. Only the connections of the first unit in the board are shown to avoid cluttering the diagram. The connections of each unit to all elements in the same row or column have the weight -2, all others have a weight zero. All units have the threshold -1. Any unit set to 1 inhibits any other units in the same row or column. If a row or column is all set to 0, when one of its elements is selected it will immediately switch its state to 1, since the total excitation (zero) is greater than the threshold -1.

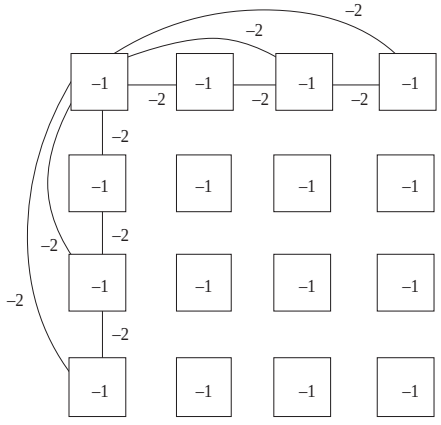

Fig. 13.16. Network for the solution of a four rooks problem

The weights for the network are derived from the following considerations: Let x_{ij} represent the state of the unit corresponding to the square ij in the $n \times n$ board. The number of ones in column j is given by $\sum_{i=1}^{n} x_{ij}$. If in each column only a single 1 is allowed, the following function must be minimized:

$$E_1(x_{11}, \ldots, x_{nn}) = \sum_{j=1}^{n} (\sum_{i=1}^{n} x_{ij} - 1)^2.$$

The minimum of the function corresponds to the situation in which just one rook has been positioned in every column. Similarly, for the rows of the board we define the function E_2 according to

$$E_2(x_{11}, \ldots, x_{nn}) = \sum_{i=1}^{n} (\sum_{j=1}^{n} x_{ij} - 1)^2.$$

We want to minimize the function $E = E_1 + E_2$. The general strategy is to reduce its analytical expression to a Hopfield form. The necessary algebraic steps can be avoided by noticing that the expression for E_1 is the sum of n independent functions (one per column). The term $(\sum_{i=1}^{n} x_{ij} - 1)^2$ corresponds to a multiflop problem. The weights for the edges in each column can be set to -2, as was done before in the multiflop problem. The same is done for each row: the weights between any unit and its row partners are set to -2. Only the thresholds must be selected with a little more care. The simple juxtaposition of a row-multiflop with a column-multiflop at each field will give us a threshold of $-1 + (-1) = -2$. This would mean that each row or column can contain up to two elements whose state is 1. This is avoided by setting the thresholds of the units to -1. The resulting network is the one shown in Figure 13.16. Each field will be forced to adopt the state zero whenever another unit is set to 1 in its own row or its own column.

13.5.4 The eight queens problem

The well-known eight queens problem can also be solved with a Hopfield network. It is just a generalization of the rooks problem, since now the diagonals of the board will also be considered. Each diagonal can be occupied at most once by a queen. As before with the rooks problem, we solve this task by overlapping multiflop problems at each square. Figure 13.17 shows how three multiflop chains have to be considered for each field. The diagram shows a 4×4 board and the overlapping of multiflop problems for the upper left square on the board. This overlapping provides us with the necessary weights, which are set to $w_{ij} = -2$, when unit i is different from unit j and belongs to the same row, column or diagonal as unit j. Otherwise we set w_{ij} to zero. The thresholds of all units are set to -1.

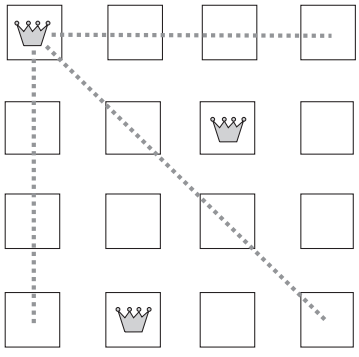

Fig. 13.17. The eight queens problem

A computer simulation shows, however, that this simple connection pattern does not always provide a correct solution for the n-queens problem. The

proposed connection weights produce an energy function in which local minima with less than n queens are possible. The only alternative if such a stable state is found is to start the simulation again.

It is not possible to set the weights of the network of Figure 13.17 in such a way as to obtain only correct solutions. The difficulty is that diagonals can be occupied by a queen but they can also be unoccupied. Simple overlapping of multiflop problems does not work any more. The energy function has become much more complex than in the previous cases and we now have to achieve compromises between the weights which we choose for rows and columns and for diagonals.

13.5.5 The traveling salesman

The Traveling Salesman Problem (TSP) is one of the most celebrated benchmarks in combinatorics. It is simple to state and visualize and it is *NP*-complete. If we can solve the TSP efficiently, we can provide an efficient solution for other problems in the class *NP*. Hopfield and Tank [200] were the first to try to solve the TSP using a neural network.

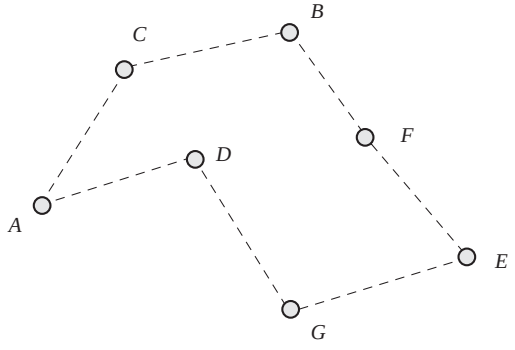

Fig. 13.18. A TSP and its solution

In the TSP we are looking for a path through n cities $S_1, S_2, \ldots, S_n$, such that every city is visited at least once and the length of a round trip is minimal. Let d_{ij} denote the distance between the cities S_i and S_j. A round trip can be represented using a matrix. Each of the n rows of the matrix is associated with a city. The columns are labeled 1 through n and they correspond to the n necessary visits. The following matrix shows a path going through the cities S_1, S_2, S_3 and S_4 in that same order:

	1	2	3	4
S_1	1	0	0	0
S_2	0	1	0	0
S_3	0	0	1	0
S_4	0	0	0	1

A single 1 is allowed in each row and each column, otherwise the salesman would visit a city twice or two cities simultaneously. This matrix must fulfill the same conditions as in the rooks problem and we can again use a network in which a unit is used to represent each entry in the matrix (the new "chess board").

Solving the TSP requires minimizing the length of the round trip, that is of

$$L = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{ik} x_{j,k+1},$$

where x_{ik} represents the state of the unit corresponding to the entry ik in the matrix. When x_{ik} and $x_{j,k+1}$ are both 1, this means that the city S_i is visited in the k-th step and the city S_j in the step $k+1$. The distance between both cities must be added to the total length of the round trip. We use the convention that the column $n+1$ is equal to the first column of the matrix, so that we always obtain a round trip.

In the minimization problems we must include the constraints for a valid round trip. It is necessary to add the energy function of the rooks problem to the length L to obtain the new energy function E, which is given by

$$E = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{ik} x_{j,k+1} + \frac{\gamma}{2} (\sum_{j=1}^{n} (\sum_{i=1}^{n} x_{ij} - 1)^2 + \sum_{i=1}^{n} (\sum_{j=1}^{n} x_{ij} - 1)^2),$$

where the constant γ regulates how much weight is given to the minimization of the length or to the generation of legal paths. The first summation to the right of the equal sign already has the form of a Hopfield energy function; the expression in parentheses has it too, since it is the energy function of a rooks problem. The weights for the network can be deduced immediately from this expression: the weights of edges between units in the same row or column must be set to $-\gamma$ and the thresholds of the units to $-\gamma/2$. The weights must be modified by including the length between states, so that the weight of the edge between unit ik and unit $j, k+1$ becomes

$$w_{ik,jk+1} = -d_{ij} + t_{ik,jk+1}$$

where $t_{ik,jk+1} = -\gamma$ whenever the units belong to the same row or column, otherwise $t_{ik,jk+1}$ is zero.

With this network we can try to find solutions of the TSP. A simulation shows that the generated routes are sometimes not legal, because one city is not visited or more than one city is visited in a single step. We can always

force the network to generate legal tours: it is only necessary to set γ to a very large value so as to obliterate the contribution of the cities' distances. If γ is zero, we do not care whether the tour is a legal one and only the total length is minimized (by choosing an "empty" tour). Since the value of γ is not prescribed, it can be found by trial and error. In general the network will not be capable of finding the global minimum and because large TSPs (with more than 100 cities) have so many local minima, it is difficult to decide whether the local minimum that has been found is a good approximation to the optimal result. The whole approach is dependent on the existence of real, massively parallel systems, since the number of units required to solve a TSP increases quadratically with the number n of cities (and the number of weights increases proportionally to n^4).

13.5.6 The limits of Hopfield networks

The first articles of Hopfield and Tank on parallel solutions to combinatorial problems received a lot of attention [200, 201]. The theoretical question was whether this could be a method to solve *NP*-hard problems or at least to get an approximate solution in polynomial time. In the following years many other researchers tried to extend the range of combinatorial problems that could be solved using Hopfield's technique, trying to improve the quality of the results at the same time. It emerged that well-behaved average problems could be solved efficiently. However, these average results should be compared to the expected running time for the worst case. Bruck and Goodman [74] showed that a polynomially bounded network (on the size of the problem) is unable to find the global minimum of the energy function of *NP*-complete problems (encoded as Hopfield networks) with a 100% guarantee. Stated in another way: if we try to transform all local minima of the Hopfield network into an optimal solution of the combinatorial problem, the size of the network explodes exponentially. We proceed to prove the result of Bruck and Goodman, but we must first introduce an additional complexity class: the complement of the class *NP*.

The class *NP* of nondeterministic problems solvable in polynomial time is different from the class P of problems solvable in polynomial time not only in the way exposed already in Chap. 10. If a problem is a member of the class P, the same is true for the complementary problem. The complement of the decision problem "For the problem instance I, is X true for I?"is just "For the problem instance I, is X false for I?". A deterministic polynomial time algorithm terminates on each of the two questions. It is only necessary to substitute "true" for "false" to transform a polynomial time algorithm for a problem in P in an algorithm for its complement. But this is not necessarily so for problems in *NP*. A solution for the *Traveling Salesman Decision Problem* (TSDP), that is, the computation of the tour's length and the comparison with the decision's threshold, can be verified in polynomial time. However, the complementary problem has the wording "Is there no tour with a total

length smaller than R?". If the answer is "yes", no polynomial time algorithm is known that could verify this assertion. It would be necessary to propose a data structure on which to perform some computations which could convince us of the truth of the assertion. Theoreticians assume that the complement of the TSDP probably does not belong to the class *NP*. The class which contains the complement of all *NP* problems is called co-*NP*. It is generally assumed that $NP \neq$ co-NP. This inequality is somewhat strong, since it implies that $P \neq NP$. Otherwise we would have co-P = co-$NP = P = NP$, i.e., the equality $NP =$ co-NP would be valid. Yet theoreticians expect that eventually it will be proved that $NP \neq$ co-NP. Figure 13.19 illustrates the expected containments of the classes *NP*, P and co-*NP*.

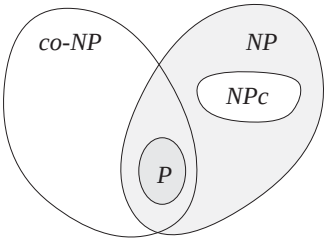

Fig. 13.19. The classes *NP* and co-*NP*

The following lemma determines under what conditions equality of the classes *NP* and co-*NP* would be possible. We can assume that this condition cannot be fulfilled.

Lemma 1. *If there is an* NP*-complete problem X whose complement X^c belongs to* NP*, then $NP =$ co-NP.*

The lemma is true because any problem Y in *NP* can be reduced in polynomial time into X. The complement of Y can therefore be transformed in polynomial time into X^c. Since a solution of X^c can be verified in polynomial time, the same is true for any solution of Y^c. This and some additional technical details would imply that $NP =$ co-NP.

Neural networks are just a subset of the algorithmic world. Since it is suspected that there is no polynomial time algorithm for the problems in the class *NP*, it should be possible to prove that Hopfield networks of bounded size are subjected to the same limitations. The following proposition settles this question [75].

Proposition 21. *Let L be an* NP*-complete decision problem and H a Hopfield network with a number of weights bounded by a polynomial on the size of the problem. If H can solve L (100% success rate) then $NP =$ co-NP.*

Proof. The problem L has a certain size defined by an appropriate coding. Since we must compute the energy function and from it derive the necessary

weights for H, a polynomial bound on the total number of weights is necessary. A Hopfield network always finds a local minimum of its energy function. In our case, a 100% hit rate means that *all* local minima of the energy function should make possible a decision on the truth or falsity of the decision problem L. The Hopfield network can be considered a data structure that makes possible the verification of the found solution. It is only necessary to verify whether the solution found by the network is indeed a local minimum of the energy function. The polynomial size of the net makes this verification possible in polynomial time. The decision problem and the complement are, in this case, completely symmetric. The TSDP can be answered with "yes" if the tour found by the network is shorter than the decision threshold. But the complement of the TSDP can be decided also just by comparing the length of the optimal tour found with the decision threshold. Therefore the complement of L is a member of the class NP and it follows from Lemma 1 that $NP = \text{co-}NP$. Since it is generally assumed that this cannot be so, there should be a contradiction in the premises. The network H does not exist unless $NP = \text{co-}NP$. □

Even if we content ourselves with a polynomially bounded network that can provide approximate solutions (for example, TSP round-trips not larger by a given ε than the optimal tour), no such network can be built. It is because of this inherent limitation that some authors have sought to introduce stochastic factors into the networks, as we will discuss when we deal with *Boltzmann machines* in the next chapter.

13.6 Implementation of Hopfield networks

Hopfield networks as massively parallel systems are only interesting if they can be implemented in hardware and not just simulated in a sequential computer. Some proposals have been made for special chips capable of simulating Hopfield networks but the most promising approach are optical computers, capable of solving the connectivity problem of neural networks.

13.6.1 Electrical implementation

In 1984 Hopfield proposed an electrical realization of his model which uses a circuit very similar to the ones used by Steinbuch (Chap. 18) [199]. Figure 13.20 shows a diagram of the circuit. The outputs of the amplifiers $x_1, x_2, \ldots, x_n$ are interpreted as the states of the Hopfield units. Their complements $\neg x_1, \neg x_2, \ldots, \neg x_n$ are produced by inverters. All states and their complements are fed back to the input of the electrical circuit. An electrical contact is represented by a small circle. A resistance is present at each contact point. The connection r_{13}, for example, contains a resistor with resistance $r_{13} = 1/w_{13}$. The constants w_{ij} represent the weights of the Hopfield network

between the units i and j. Inhibiting connections between one unit and another (that is, connections with negative weights) are simulated by connecting one inverted output of a unit to the other one. In Figure 13.20, for example, the connection points in the upper row all come from the inverted output of unit 1.

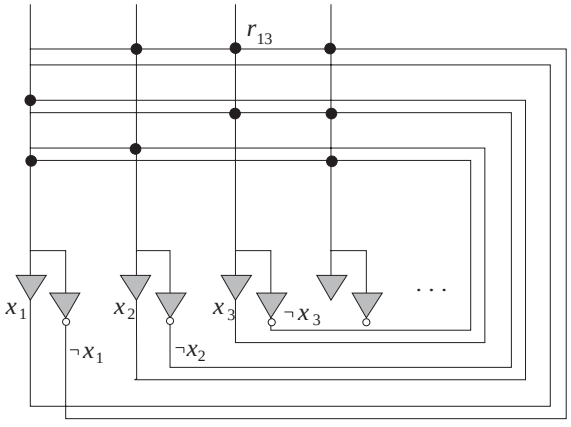

Fig. 13.20. Electrical implementation of a Hopfield network

In a network with n amplifiers the current I_i at the input to the i-th amplifier is given by

$$I_i = \sum_{j=1}^{n} \frac{x_j}{r_{ij}} = \sum_{j=1}^{n} x_j w_{ij},$$

where we have used the convention that r_{ij} is negative if the inverted value of x_j has been connected to the input of the amplifier x_i. The current I_i represents the excitation of unit i. The amplifier transforms the total excitation into 0 or 1 according to a certain electrical threshold, which can be set to an arbitrary positive value.

This simple circuit can be used to simulate Hopfield networks in a fraction of the time needed by a sequential computer. If the circuit is provided with variable resistors it is then possible to implement some learning algorithms directly in hardware.

13.6.2 Optical implementation

The most important computation that must be accelerated for a fast simulation of Hopfield networks is the vector matrix multiplication. Computation of the excitation of a node requires such an operation every time a unit's state is to be updated. Optical methods can be used to perform this numerical operation faster. Figure 13.21 shows an optical realization of the Hopfield model [132].

The logical structure is in principle the same as in the network of Figure 13.20, but the vector matrix multiplication is computed analogically using optical techniques. The n binary values which represent the network's state are projected through the vertical lens to the left of the arrangement. The lens projects each value x_i onto the corresponding row of an optical mask. Each row i in the mask is divided into fields which represent the n weights $w_{i1}, w_{i2}, \ldots, w_{in}$. Each field is partially darkened according to the value of the corresponding weight. The individual unit states are projected using light emitting diodes and the luminosity is proportional to the corresponding x_i value. The light going through the mask is collected by another lens in such a way that all the incoming light from a column is collected at a single position. The amount of light that goes through the mask is proportional to the product of x_i and w_{ij} at each position ij of the mask. The incoming light at the j-th detector represents the total excitation of unit j, which is equal to

$$s_j = \sum_{i=1}^{n} w_{ij} x_i.$$

The total excitation of the unit j can now be processed by an analog or digital circuit to produce the unit state which is used again in a new iteration of the network.

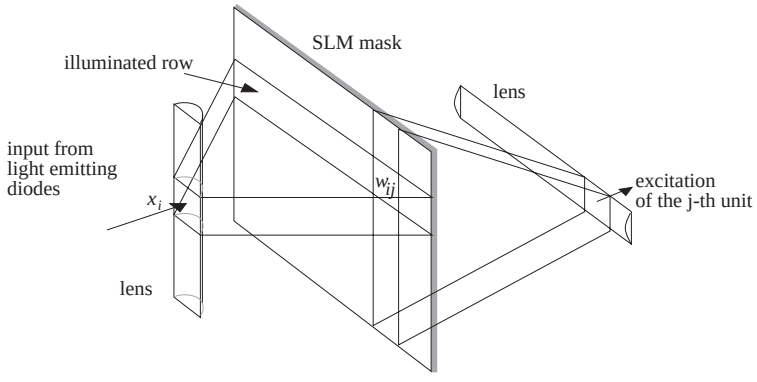

Fig. 13.21. Optical implementation of a Hopfield network

The difference compared with the electrical model is that the weights and signals must be normalized and scaled to fit the kind of optical processing being done. The most significant difference is the absence of direct connections. The light paths do not affect each other, so that it is possible to implement much larger networks than in the purely electrical realization. We will come back to the topic of optical implementations when we discuss neural hardware in Chap. 18.

13.7 Historical and bibliographical remarks

With the introduction in 1982 of the model named after him, John Hopfield established the connection between neural networks and physical systems of the type considered in statistical mechanics. This in turn gave computer scientists a whole new arsenal of mathematical tools for the analysis of neural networks. Other researchers had already considered more general associative memory models in the 1970s, but by restricting the architecture of the network to a symmetric connection matrix with a zero diagonal, it was possible to design recurrent networks with stable states. With the introduction of the concept of the energy function, the convergence properties of the networks could be more easily analyzed.

The Hopfield network also has the advantage, in comparison with other models, of a simple technical implementation using electronic or optical devices [132]. The computing strategy used when updating the network states corresponds to the relaxation methods traditionally used in physics [92].

The properties of Hopfield networks have been investigated since 1982 using the theoretical tools of statistical mechanics [322]. Gardner [155] published a classical treatise on the capacity of the perceptron and its relation to the Hopfield model. The total field sensed by particles with a spin can be computed using the methods of mean field theory. This simplifies a computation which is hopeless without the help of some statistical assumptions [189]. Using these methods Amit et al. [24] showed that the number of stable states in a Hopfield network of n units is bounded by $0.14n$. A *recall* error is tolerated only 5% of the time. This upper bound is one of the most cited numbers in the theory of Hopfield networks.

In 1988 Kosko proposed the BAM model, which is a kind of "missing link" between conventional associative memories and Hopfield networks. Many other variations have been proposed since, some of them with asynchronous, others with synchronous dynamics [231]. Hopfield networks have also been studied from the point of view of dynamical systems. In this respect spin glass models play a relevant role. These are materials composed of particles with a spin and mutual interactions [412].

Combinatorial problems have a long tradition, but a really systematic theory capable of unifying the myriad of heuristic methods developed in the past was first developed in the 1960s and 1970s [361]. The important point was the increasingly important role played by computers and the emergence of a new attitude which tried to reach whole classes of problems and not just individual cases. An important research theme which remains is how to split a combinatorial problem into subtasks that can be assigned to different processors [160].

The efforts of Hopfield and Tank with the TSP led to many other similar experiments in related fields. Wilson and Pawley [456] repeated their experiments but they could not confirm the optimistic results of the former authors. The main difficulty is that complex combinatorial problems produce an expo-

nential number of local minima of the energy function. In sequential computers, Hopfield models cannot compete with conventional methods [224]. Many heuristics have been proposed for the TSP, starting with the classical work by Kernighan and Lin [274]. The only way to make Hopfield models competitive is through the use of special hardware. Sheu et al. have obtained interesting results and significant speedup in comparison with sequential computers by using a technique they call *hardware annealing*.

One of the first to deal with the intrinsic limits of the Hopfield model for the solution of the TSP was Abu-Mostafa [3], who nevertheless considered only the case of networks of constant size. Bruck and Goodman [75] considered networks of variable but polynomially bounded size and obtained the same negative results. Although this almost meant the "death of the traveling salesman" [322], the Hopfield model and its stochastic variants have been applied in many other fields, such as psychology, simulation of ensembles of biological neural networks, and chaotic behavior of neural circuits.

The optical implementation of Hopfield networks is a promising field of research. Other than masks, holograms can also be used to store the network weights [352]. The main technical problem is still the size reduction of the optical components, which could make them a viable alternative to conventional electronic systems.

Exercises

1. Train a Hopfield network in the computer using the perceptron learning algorithm.
2. Solve a TSP of 10 cities with a Hopfield network. How many weights do you need for the network?
3. Compute the energy of the different possible states of the network shown in Figure 13.6. Do the same for Figure 13.7 assigning some values to the weights and thresholds.