12

Associative Networks

12.1 Associative pattern recognition

The previous chapters were devoted to the analysis of neural networks without feedback, capable of mapping an input space into an output space using only feed-forward computations. In the case of backpropagation networks we demanded continuity from the activation functions at the nodes. The neighborhood of a vector $\mathbf{x}$ in input space is therefore mapped to a neighborhood of the image $\mathbf{y}$ of $\mathbf{x}$ in output space. It is this property which gives its name to the *continuous mapping networks* we have considered up to this point.

In this chapter we deal with another class of systems, known generically as associative memories. The goal of learning is to *associate* known input vectors with given output vectors. Contrary to continuous mappings, the neighborhood of a known input vector $\mathbf{x}$ should also be mapped to the image $\mathbf{y}$ of $\mathbf{x}$, that is if $B(\mathbf{x})$ denotes all vectors whose distance from $\mathbf{x}$ (using a suitable metric) is lower than some positive constant ε, then we expect the network to map $B(\mathbf{x})$ to $\mathbf{y}$. Noisy input vectors can then be associated with the correct output.

12.1.1 Recurrent networks and types of associative memories

Associative memories can be implemented using networks with or without feedback, but the latter produce better results. In this chapter we deal with the simplest kind of feedback: the output of a network is used repetitively as a new input until the process converges. However, as we will see in the next chapter not all networks converge to a stable state after having been set in motion. Some restrictions on the network architecture are needed.

The function of an associative memory is to recognize previously learned input vectors, even in the case where some noise has been added. We have already discussed a similar problem when dealing with clusters of input vectors (Chap. 5). The approach we used there was to find cluster centroids in input space. For an input $\mathbf{x}$ the weight vectors produced a maximal excitation at the unit representing the cluster assigned to $\mathbf{x}$, whereas all other units remained silent. A simple approach to inhibit the other units is to use non-local information to decide which unit should fire.

The advantage of associative memories compared to this approach is that only the local information stream must be considered. The response of each unit is determined exclusively by the information flowing through its own weights. If we take the biological analogy seriously or if we want to implement these systems in VLSI, locality is always an important goal. And as we will see in this chapter, a learning algorithm derived from biological neurons can be used to train associative networks: it is called *Hebbian learning.*

The associative networks we want to consider should not be confused with conventional associative memory of the kind used in digital computers, which consists of content addressable memory chips. With this in mind we can distinguish between three overlapping kinds of associative networks [294, 255]:

- *Heteroassociative networks* map m input vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ in n-dimensional space to m output vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m$ in k-dimensional space, so that $\mathbf{x}^i \mapsto \mathbf{y}^i$. If $\|\tilde{\mathbf{x}} - \mathbf{x}^i\|^2 < \varepsilon$ then $\tilde{\mathbf{x}} \mapsto \mathbf{y}^i$. This should be achieved by the learning algorithm, but becomes very hard when the number m of vectors to be learned is too high.
- *Autoassociative networks* are a special subset of the heteroassociative networks, in which each vector is associated with itself, i.e., $\mathbf{y}^i = \mathbf{x}^i$ for $i = 1, \ldots, m$. The function of such networks is to correct noisy input vectors.
- *Pattern recognition networks* are also a special type of heteroassociative networks. Each vector $\mathbf{x}^i$ is associated with the scalar i. The goal of such a network is to identify the 'name' of the input pattern.

Figure 12.1 shows the three kinds of networks and their intended use. They can be understood as automata which produce the desired output whenever a given input is fed into the network.

12.1.2 Structure of an associative memory

The three kinds of associative networks discussed above can be implemented with a single layer of computing units. Figure 12.2 shows the structure of a heteroassociative network without feedback. Let w_{ij} be the weight between input site i and unit j. Let $\mathbf{W}$ denote the $n \times k$ weight matrix $[w_{ij}]$. The row vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ produces the excitation vector $\mathbf{e}$ through the computation

$$\mathbf{e} = \mathbf{x}\mathbf{W}.$$

The activation function is computed next for each unit. If it is the identity, the units are just linear associators and the output $\mathbf{y}$ is just $\mathbf{x}\mathbf{W}$. In general m different n-dimensional row vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ have to be associated

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

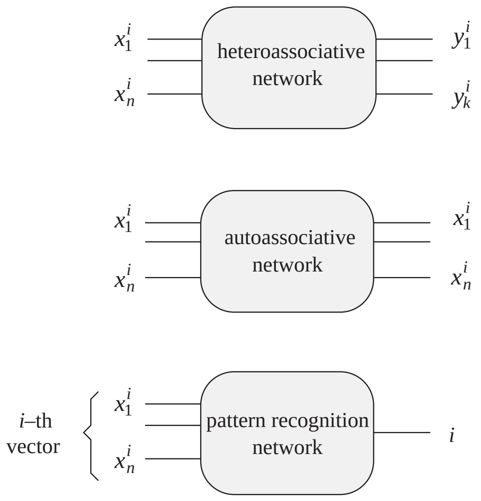

Fig. 12.1. Types of associative networks

with m k-dimensional row vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m$. Let $\mathbf{X}$ be the $m \times n$ matrix whose rows are each one of the input vectors and let $\mathbf{Y}$ be the $m \times k$ matrix whose rows are the output vectors. We are looking for a weight matrix $\mathbf{W}$ for which

$$\mathbf{XW} = \mathbf{Y} \tag{12.1}$$

holds. In the autoassociative case we associate each vector with itself and equation (12.1) becomes

$$\mathbf{XW} = \mathbf{X} \tag{12.2}$$

If $m = n$, then $\mathbf{X}$ is a square matrix. If it is invertible, the solution of (12.1) is

$$\mathbf{W} = \mathbf{X}^{-1}\mathbf{Y},$$

which means that finding $\mathbf{W}$ amounts to solving a linear system of equations.

What happens now if the output of the network is used as the new input? Figure 12.3 shows such a recurrent autoassociative network. We make the assumption that all units compute their outputs simultaneously and we call such networks *asynchronous*. In each step the network is fed an input vector $\mathbf{x}(i)$ and produces a new output $\mathbf{x}(i+1)$. The question with this class of networks is whether there is a fixed point $\vec{\xi}$ such that

$$\vec{\xi}\,\mathbf{W} = \vec{\xi}\ .$$

The vector $\vec{\xi}$ is an eigenvector of $\mathbf{W}$ with eigenvalue 1. The network behaves as a first-order dynamical system, since each new state $\mathbf{x}(i+1)$ is completely determined by its most recent predecessor [30].

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

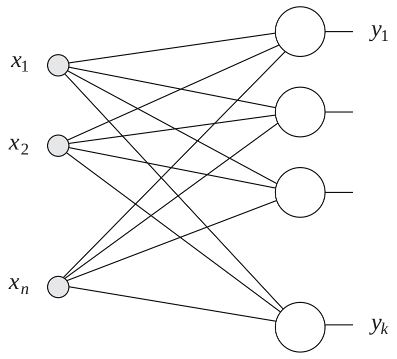

Fig. 12.2. Heteroassociative network without feedback

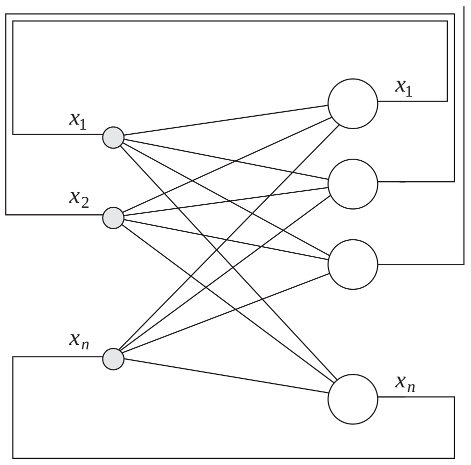

Fig. 12.3. Autoassociative network with feedback

12.1.3 The eigenvector automaton

Let **W** be the weight matrix of an autoassociative network, as shown in Figure 12.3. The individual units are linear associators. As we said before, we are interested in fixed points of the dynamical system but not all weight matrices lead to convergence to a stable state. A simple example is a rotation by 90 degrees in two-dimensional space, given by the matrix

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

Such a matrix has no non-trivial eigenvectors. There is no fixed point for the dynamical system, but infinitely many cycles of length four. By changing the angle of rotation, arbitrarily long cycles can be produced and by picking an irrational angle, even a non-cyclic succession of states can be produced.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

Quadratic matrices with a complete set of eigenvectors are more useful for storage purposes. An $n \times n$ matrix $\mathbf{W}$ has at most n linear independent eigenvectors and n eigenvalues. The eigenvectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n$ satisfy the set of equations

$$\mathbf{x}^i \mathbf{W} = \lambda_i \mathbf{x}^i \qquad \text{for } i = 1, \ldots, n,$$

where $\lambda_1, \ldots, \lambda_n$ are the matrix eigenvalues.

Each weight matrix with a full set of eigenvectors defines a kind of "eigenvector automaton". Given an initial vector, the eigenvector with the largest eigenvalue can be found (if it exists). Assume without loss of generality that λ_1 is the eigenvalue of $\mathbf{w}$ with the largest magnitude, that is, $|\lambda_1| > |\lambda_i|$ for $i \neq j$. Let $\lambda_1 > 0$ and pick randomly a non-zero n-dimensional vector $\mathbf{a}_0$. This vector can be expressed as a linear combination of the n eigenvectors of the matrix $\mathbf{W}$:

$$\mathbf{a}_0 = \alpha_1 \mathbf{x}^1 + \alpha_2 \mathbf{x}^2 + \cdots + \alpha_n \mathbf{x}^n.$$

Assume that all constants α are non-zero. After the first iteration with the weight matrix $\mathbf{W}$ we get

$$\begin{aligned} \mathbf{a}_1 &= \mathbf{a}_0 \mathbf{W} \\ &= (\alpha_1 \mathbf{x}^1 + \alpha_2 \mathbf{x}^2 + \cdots + \alpha_n \mathbf{x}^n) \mathbf{W} \\ &= \alpha_1 \lambda_1 \mathbf{x}^1 + \alpha_2 \lambda_2 \mathbf{x}^2 + \cdots + \alpha_n \lambda_n \mathbf{x}^n. \end{aligned}$$

After t iterations the result is

$$\mathbf{a}_t = \alpha_1 \lambda_1^t \mathbf{x}^1 + \alpha_2 \lambda_2^t \mathbf{x}^2 + \cdots + \alpha_n \lambda_n^t \mathbf{x}^n.$$

It is obvious that the eigenvalue λ_1, the one with the largest magnitude, dominates this expression for a big enough t. The vector $\mathbf{a}_t$ can be brought arbitrarily close to the eigenvector $\mathbf{x}^1$ (with respect to the direction and without considering the relative lengths). In each iteration of the associative network, the vector $\mathbf{x}^1$ attracts any other vector $\mathbf{a}_0$ whose component α_1 is non-zero. An example is the following weight matrix:

$$\mathbf{W} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

Two eigenvectors of this matrix are $(1,0)$ and $(0,1)$ with respective eigenvalues 2 and 1. After t iterations any initial vector (x_1, x_2) with $x_1 \neq 0$ is transformed into the vector $(2^t x_1, x_2)$, which comes arbitrarily near to the vector $(1,0)$ for a large enough t. In the language of the theory of dynamical systems, the vector $(1,0)$ is an *attractor* for all those two-dimensional vectors whose first component does not vanish.

12.2 Associative learning

The simple examples of the last section illustrate our goal: we want to use associative networks as dynamical systems, whose attractors are exactly those

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

vectors we would like to store. In the case of the linear eigenvector automaton, unfortunately just one vector absorbs almost the whole of input space. The secret of associative network design is locating as many attractors as possible in input space, each one of them with a well-defined and bounded influence region. To do this we must introduce a nonlinearity in the activation of the network units so that the dynamical system becomes nonlinear.

Bipolar coding has some advantages over binary coding, as we have already seen several times, but this is still more noticeable in the case of associative networks. We will use a step function as nonlinearity, computing the output of each unit with the sign function

$$\mathrm{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

The main advantage of bipolar coding is that bipolar vectors have a greater probability of being orthogonal than binary vectors. This will be an issue when we start storing vectors in the associative network.

12.2.1 Hebbian learning – the correlation matrix

Consider a single-layer network of k units with the sign function as activation function. We want to find the appropriate weights to map the n-dimensional input vector $\mathbf{x}$ to the k-dimensional output vector $\mathbf{y}$. The simplest learning rule that can be used in this case is Hebbian learning, proposed in 1949 by the psychologist Donald Hebb [182]. His idea was that two neurons which are simultaneously active should develop a degree of interaction higher than those neurons whose activities are uncorrelated. In the latter case the interaction between the elements should be very low or zero [308].

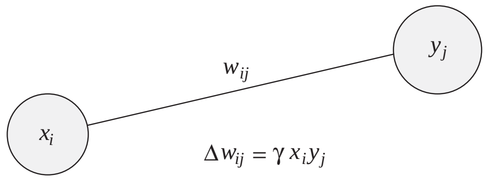

Fig. 12.4. The Hebb rule

In the case of an associative network, Hebbian learning is applied, updating the weight w_{ij} by Δw_{ij}. This increments measures the correlation between the input x_i at site i and the output y_j of unit j. During learning, both input and output are clamped to the network and the weight update is given by

$$\Delta w_{ij} = \gamma\, x_i x_j.$$

The factor γ is a learning constant in this equation. The weight matrix $\mathbf{W}$ is set to zero before Hebbian learning is started. The learning rule is applied

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

to all weights clamping the n-dimensional row vector $\mathbf{x}^1$ to the input and the k-dimensional row vector $\mathbf{y}^1$ to the output. The updated weight matrix $\mathbf{W}$ is the correlation matrix of the two vectors and has the form

$$\mathbf{W} = [w_{ij}]_{n\times k} = \left[x_i^1 y_j^1\right]_{n\times k}.$$

The matrix $\mathbf{W}$ maps the non-zero vector $\mathbf{x}^1$ exactly to the vector $\mathbf{y}^1$, as can be proved by looking at the excitation of the k output units of the network, which is given by the components of

$$\begin{aligned}\mathbf{x}^1\mathbf{W} &= (y_1^1 \sum_{i=1}^n x_i^1 x_i^1, y_2^1 \sum_{i=1}^n x_i^1 x_i^1, ..., y_k^1 \sum_{i=1}^n x_i^1 x_i^1) \\ &= \mathbf{y}^1(\mathbf{x}^1 \cdot \mathbf{x}^1).\end{aligned}$$

For $\mathbf{x}^1 \neq \mathbf{0}$ it holds $\mathbf{x}^1 \cdot \mathbf{x}^1 > 0$ and the output of the network is

$$\mathrm{sgn}(\mathbf{x}^1\mathbf{W}) = (y_1^1, y_2^1, ..., y_k^1) = \mathbf{y}^1,$$

where the sign function is applied to each component of the vector of excitations.

In general, if we want to associate m n-dimensional non-zero vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ with m k-dimensional vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m$, we apply Hebbian learning to each input-output pair and the resulting weight matrix is

$$\mathbf{W} = \mathbf{W}^1 + \mathbf{W}^2 + \cdots + \mathbf{W}^m, \tag{12.3}$$

where each matrix $\mathbf{W}^\ell$ is the $n \times k$ correlation matrix of the vectors $\mathbf{x}^\ell$ and $\mathbf{y}^\ell$, i.e.,

$$\mathbf{W}^\ell = \left[x_i^\ell y_j^\ell\right]_{n\times k}.$$

If the input to the network is the vector $\mathbf{x}^p$, the vector of unit excitations is

$$\begin{aligned}\mathbf{x}^p\mathbf{W} &= \mathbf{x}^p(\mathbf{W}^1 + \mathbf{W}^2 + \cdots + \mathbf{W}^m) \\ &= \mathbf{x}^p\mathbf{W}^p + \sum_{\ell\neq p}^m \mathbf{x}^p\mathbf{W}^\ell \\ &= \mathbf{y}^p(\mathbf{x}^p \cdot \mathbf{x}^p) + \sum_{\ell\neq p}^m \mathbf{y}^\ell(\mathbf{x}^\ell \cdot \mathbf{x}^p).\end{aligned}$$

The excitation vector is equal to $\mathbf{y}^p$ multiplied by a positive constant plus an additional perturbation term $\sum_{\ell\neq p}^m \mathbf{y}^\ell(\mathbf{x}^\ell \cdot \mathbf{x}^p)$ called the *crosstalk*. The network produces the desired vector $\mathbf{y}^p$ as output when the crosstalk is zero. This is the case whenever the input patterns $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ are pairwise orthogonal. Yet, the perturbation term can be different from zero and the mapping can still work. The crosstalk should only be sufficiently smaller than $\mathbf{y}^p(\mathbf{x}^p \cdot \mathbf{x}^p)$. In that case the output of the network is

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

$$\mathrm{sgn}(\mathbf{x}^p\mathbf{W}) = \mathrm{sgn}\left(\mathbf{y}^p(\mathbf{x}^p\cdot\mathbf{x}^p) + \sum_{\ell\neq p}^{m}\mathbf{y}^\ell(\mathbf{x}^\ell\cdot\mathbf{x}^p)\right).$$

Since $\mathbf{x}^p\cdot\mathbf{x}^p$ is a positive constant, it holds that

$$\mathrm{sgn}(\mathbf{x}^p\mathbf{W}) = \mathrm{sgn}\left(\mathbf{y}^p + \sum_{\ell\neq p}^{m}\mathbf{y}^\ell\frac{(\mathbf{x}^\ell\cdot\mathbf{x}^p)}{(\mathbf{x}^p\cdot\mathbf{x}^p)}\right).$$

To produce the output $\mathbf{y}^p$ the equation

$$\mathbf{y}^p = \mathrm{sgn}\left(\mathbf{y}^p + \sum_{\ell\neq p}^{m}\mathbf{y}^\ell\frac{(\mathbf{x}^\ell\cdot\mathbf{x}^p)}{(\mathbf{x}^p\cdot\mathbf{x}^p)}\right)$$

must hold. This is the case when the absolute value of all components of the perturbation term

$$\sum_{\ell\neq p}^{m}\mathbf{y}^\ell\frac{(\mathbf{x}^\ell\cdot\mathbf{x}^p)}{(\mathbf{x}^p\cdot\mathbf{x}^p)}$$

is smaller than 1. This means that the scalar products $\mathbf{x}^\ell\cdot\mathbf{x}^p$ must be small in comparison to the quadratic length of the vector $\mathbf{x}^p$ (equal to n for n-dimensional bipolar vectors). If randomly selected bipolar vectors are associated with other also randomly selected bipolar vectors, the probability is high that they will be nearly pairwise orthogonal, as long as not many of them are selected. The crosstalk will be small. In this case Hebbian learning will lead to an efficient set of weights for the associative network.

In the autoassociative case, in which a vector $\mathbf{x}^1$ is to be associated with itself, the matrix $\mathbf{W}^1$ can also be computed using Hebbian learning. The matrix is given by

$$\mathbf{W}^1 = (\mathbf{x}^1)^{\mathrm{T}}\mathbf{x}^1.$$

Some authors define the autocorrelation matrix $\mathbf{W}^1$ as $\mathbf{W}^1 = (1/n)(\mathbf{x}^1)^{\mathrm{T}}\mathbf{x}^1$. Since the additional positive constant does not change the sign of the excitation, we will not use it in our computations.

If a set of m row vectors $\mathbf{x}^1,\mathbf{x}^2,\ldots,\mathbf{x}^m$ is to be autoassociated, the weight matrix $\mathbf{W}$ is given by

$$\begin{aligned}\mathbf{W} &= (\mathbf{x}^1)^{\mathrm{T}}\mathbf{x}^1 + (\mathbf{x}^2)^{\mathrm{T}}\mathbf{x}^2 + \cdots + (\mathbf{x}^m)^{\mathrm{T}}\mathbf{x}^m\\ &= \mathbf{X}^{\mathrm{T}}\mathbf{X},\end{aligned}$$

where $\mathbf{X}$ is the $m\times n$ matrix whose rows are the m given vectors. The matrix $\mathbf{W}$ is now the autocorrelation matrix for the set of m given vectors. It is expected from $\mathbf{W}$ that it can lead to the reproduction of each one of the vectors $\mathbf{x}^1,\mathbf{x}^2,\ldots,\mathbf{x}^m$ when used as weight matrix of the autoassociative network. This means that

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

$$\operatorname{sgn}(\mathbf{x}^i\mathbf{W}) = \mathbf{x}^i \quad \text{for } i = 1, \ldots, m, \tag{12.4}$$

or alternatively

$$\operatorname{sgn}(\mathbf{X}\mathbf{W}) = \mathbf{X}. \tag{12.5}$$

The function $\operatorname{sgn}(\mathbf{x}\mathbf{W})$ can be considered to be a nonlinear operator. Equation (12.4) expresses the fact that the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ are the eigenvectors of the nonlinear operator. The *learning problem for associative networks* consists in constructing the matrix $\mathbf{W}$ for which the nonlinear operator $\operatorname{sgn}(\mathbf{x}\mathbf{W})$ has these eigenvectors as fixed points. This is just a generalization of the eigenvector concept to nonlinear functions. On the other hand, we do not want to simply use the identity matrix, since the objective of the whole exercise is to correct noisy input vectors. Those vectors located near to stored patterns should be attracted to them. For $\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$, (12.5) demands that

$$\operatorname{sgn}(\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{X}) = \mathbf{X}. \tag{12.6}$$

If the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ are pairwise orthogonal $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is the identity matrix multiplied by n and (12.6) is fulfilled. If the patterns are nearly pairwise orthogonal $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is nearly the identity matrix (times n) and the associative network continues to work.

12.2.2 Geometric interpretation of Hebbian learning

The matrices $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^m$ in equation (12.3) can be interpreted geometrically. This would provide us with a hint as to the possible ways of improving the learning method. Consider first the matrix $\mathbf{W}^1$, defined in the autoassociative case as

$$\mathbf{W}^1 = (\mathbf{x}^1)^{\mathrm{T}}\mathbf{x}^1.$$

Any input vector $\mathbf{z}$ fed to the network is *projected* into the linear subspace L_1 spanned by the vector $\mathbf{x}^1$, since

$$\mathbf{z}\mathbf{W}^1 = \mathbf{z}(\mathbf{x}^1)^{\mathrm{T}}\mathbf{x}^1 = (\mathbf{z}\,(\mathbf{x}^1)^{\mathrm{T}})\mathbf{x}^1 = c_1\mathbf{x}^1,$$

where c_1 represents a constant. The matrix $\mathbf{W}^1$ represents, in general, a non-orthogonal projection operator that projects the vector $\mathbf{z}$ in L_1. A similar interpretation can be derived for each weight matrix $\mathbf{W}^2$ to $\mathbf{W}^m$. The matrix $\mathbf{W} = \sum_{i=1}^{m} \mathbf{W}^i$ is therefore a linear transformation which projects a vector $\mathbf{z}$ into the linear subspace spanned by the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$, since

$$\begin{aligned}\mathbf{z}\mathbf{W} &= \mathbf{z}\mathbf{W}^1 + \mathbf{z}\mathbf{W}^2 + \cdots + \mathbf{z}\mathbf{W}^m \\ &= c_1\mathbf{x}^1 + c_2\mathbf{x}^2 + \cdots + c_m\mathbf{x}^m\end{aligned}$$

with constants $c_1, c_2, \ldots, c_m$. In general $\mathbf{W}$ does not represent an orthogonal projection.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

12.2.3 Networks as dynamical systems – some experiments

How good is Hebbian learning when applied to an associative network? Some empirical results can illustrate this point, as we proceed to show in this section.

Since associative networks can be considered to be dynamical systems, one of the important issues is how to identify their attractors and how extended the basins of attraction are, that is, how large is the region of input space mapped to each one of the fixed points of the system. We have already discussed how to engineer a nonlinear operator whose eigenvectors are the patterns to be stored. Now we investigate this matter further and measure the changes in the basins of attraction when the patterns are learned one after the other using the Hebb rule. In order to measure the size of the basins of attraction we must use a suitable metric. This will be the *Hamming distance* between bipolar vectors, which is just the number of different components which both contain.

Table 12.1. Percentage of 10-dimensional vectors with a Hamming distance (H) from 0 to 4, which converge to a stored vector in a single iteration. The number of stored vectors increases from 1 to 10.

	Number of stored vectors (dimension 10)									
H	1	2	3	4	5	6	7	8	9	10
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1	100.0	100.0	90.0	85.0	60.0	60.0	54.3	56.2	45.5	33.0
2	100.0	86.7	64.4	57.2	40.0	31.8	22.5	23.1	17.0	13.3
3	100.0	50.0	38.6	25.4	13.5	8.3	4.8	5.9	3.1	2.4
4	100.0	0.0	9.7	7.4	4.5	2.7	0.9	0.8	0.3	0.2

Table 12.1 shows the results of a computer experiment. Ten randomly chosen vectors in 10-dimensional space were stored in the weight matrix of an associative network using Hebbian learning. The weight matrix was computed iteratively, first for one vector, then for two, etc. After each update of the weight matrix the number of vectors with a Hamming distance from 0 to 4 to the stored vectors, and which could be mapped to each one of them by the network, was counted. The table shows the percentage of vectors that were mapped in a single iteration to a stored pattern. As can be seen, each new stored pattern reduces the size of the average basins of attraction, until by 10 stored vectors only 33% of the vectors with a single different component are mapped to one of them.

The table shows the average results for all stored patterns. When only one vector has been stored, all vectors up to Hamming distance 4 converge to it. If two vectors have been stored, only 86.7% of the vectors with Hamming

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

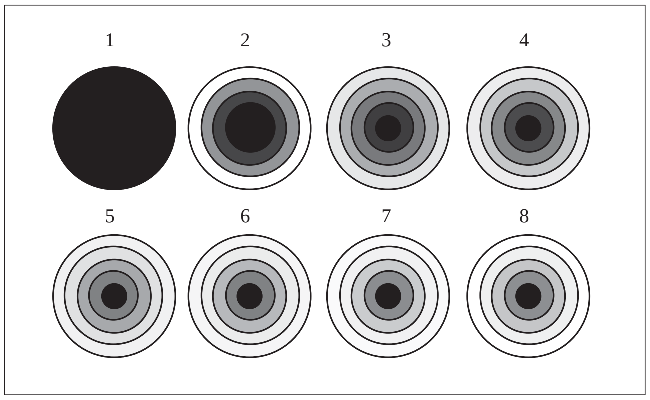

Fig. 12.5. The grey shading of each concentric circle represents the percentage of vectors with Hamming distance from 0 to 4 to stored patterns and which converge to them. The smallest circle represents the vectors with Hamming distance 0. Increasing the number of stored patterns from 1 to 8 reduces the size of the basins of attraction.

distance 2 converge to the stored patterns. Figure 12.5 is a graphical representation of the changes to the basins of attraction after each new pattern has been stored. The basins of attraction become continuously smaller and less dense.

Table 12.1 shows only the results for vectors with a maximal Hamming distance of 4, because vectors with a Hamming distance greater than 5 are mapped not to $\mathbf{x}$ but to $-\mathbf{x}$, when $\mathbf{x}$ is one of the stored patterns. This is so because

$$\mathrm{sgn}(-\mathbf{xW}) = -\mathrm{sgn}(\mathbf{xW}) = -\mathbf{x}.$$

These additional stored patterns are usually a nuisance. They are called *spurious* stable states. However, they are inevitable in the kind of associative networks considered in this chapter.

The results of Table 12.1 can be improved using a recurrent network of the type shown in Figure 12.3. The operator $\mathrm{sgn}(\mathbf{xW})$ is used repetitively. If an input vector does not converge to a stored pattern in one iteration, its image is nevertheless rotated in the direction of the fixed point. An additional iteration can then lead to convergence.

The iterative method was described before for the eigenvector automaton. The disadvantage of that automaton was that a single vector attracts almost the whole of input space. The nonlinear operator $\mathrm{sgn}(\mathbf{xW})$ provides a solution for this problem. On the one hand, the stored vectors can be reproduced, i.e., $\mathrm{sgn}(\mathbf{xW}) = \mathbf{x}$. There are no “eigenvalues” different from 1. On the other hand, the nonlinear sign function does not allow a single vector to attract most of input space. Input space is divided more regularly into basins of attraction for the different stored vectors. Table 12.2 shows the convergence

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

of an autoassociative network when five iterations are used. Only the first seven vectors used in Table 12.1 were considered again for this table.

Table 12.2. Percentage of 10-dimensional vectors with a Hamming distance (H) from 0 to 4, which converge to a stored vector in five iterations. The number of stored vectors increases from 1 to 7.

Number of stored vectors (dimension 10)

H	1	2	3	4	5	6	7
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1	100.0	100.0	90.0	85.0	60.0	60.0	54.3
2	100.0	100.0	72.6	71.1	41.8	34.8	27.9
3	100.0	80.0	48.6	47.5	18.3	10.8	8.5
4	100.0	42.8	21.9	22.3	6.8	3.7	2.0

The table shows an improvement in the convergence properties of the network, that is, an expansion of the average size and density of the basins of attraction. After six stored vectors the results of Table 12.2 become very similar to those of Table 12.1.

The results of Table 12.1 and Table 12.2 can be compared graphically. Figure 12.6 shows the percentage of vectors with a Hamming distance of 0 to 5 from a stored vector that converge to it. The broken line shows the results for the recurrent network of Table 12.2. The continuous line summarizes the results for a single iteration. Comparison of the "single shot" method and the recurrent computation show that the latter is more effective at least as long as not too many vectors have been stored. When the *capacity* of the weight matrix begins to be put under strain, the differences between the two methods disappear.

The sizes of the basins of attraction in the feed-forward and the recurrent network can be compared using an index I for their size. This index can be defined as

$$I = \sum_{h=0}^{5} h p_h$$

where p_h represents the percentage of vectors with Hamming distance h from a stored vector which converge to it. Two indices were computed using the data in Table 12.1 and Table 12.2. The results are shown in Figure 12.7. Both indices are clearly different up to five stored vectors.

As all these comparisons show, the size of the basins of attraction falls very fast. This can be explained by remembering that five stored vectors imply at least ten actually stored patterns, since together with $\mathbf{x}$ the vector $-\mathbf{x}$ is also stored. Additionally, distortion of the basins of attraction produced by

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

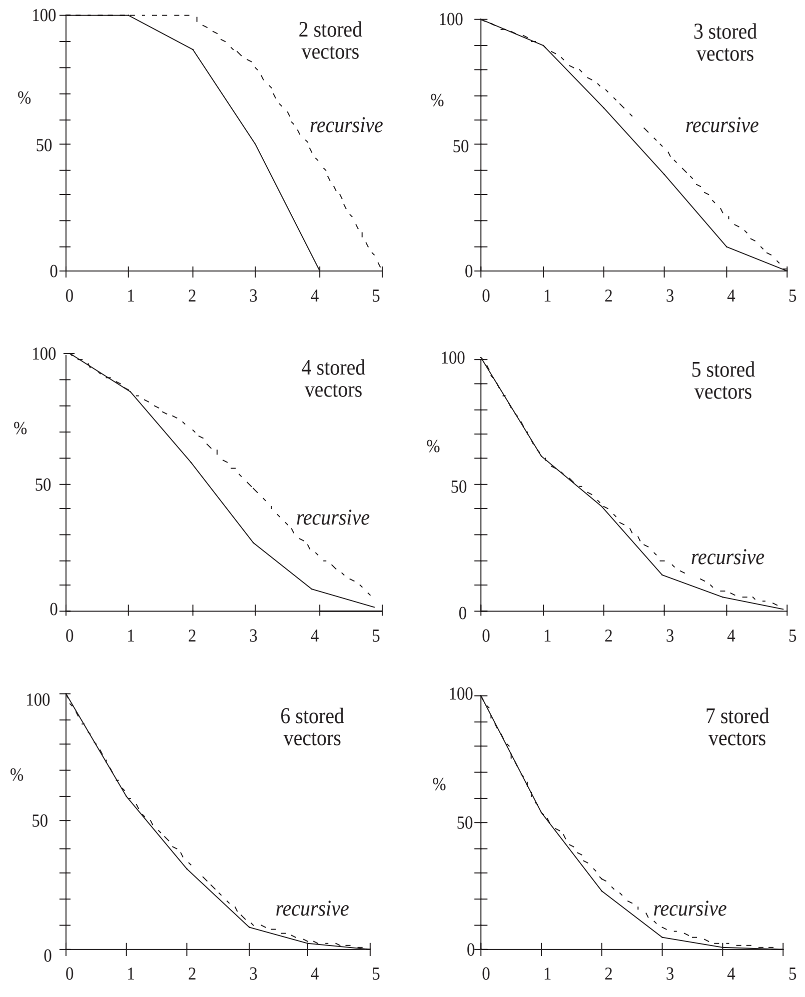

Fig. 12.6. Comparison of the percentage of vectors with a Hamming distance from 0 to 5 from stored patterns and which converge to them

Hebbian learning leads to the appearance of other spurious stable states. A computer experiment was used to count the number of spurious states that appear when 2 to 7 random bipolar vectors are stored in a 10×10 associative matrix. Table 12.3 shows the fast increase in the number of spurious states.

Spurious states other than the negative stored patterns appear because the crosstalk becomes too large. An alternative way to minimize the crosstalk term is to use another kind of learning rule, as we discuss in the next section.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

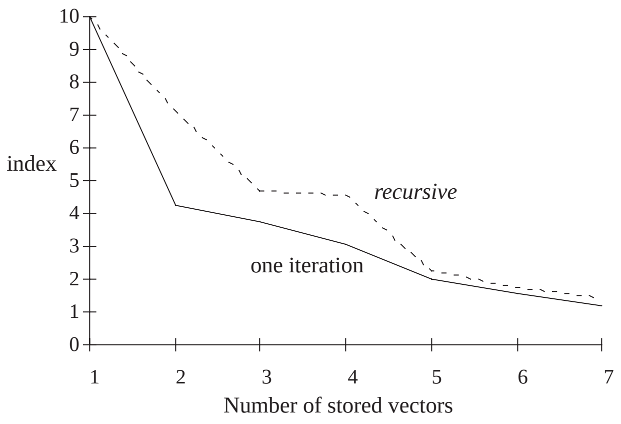

Fig. 12.7. Comparison of the indices of attraction for an associative network with and without feedback

Table 12.3. Increase in the number of spurious states when the number of stored patterns increases from 2 to 7 (dimension 10)

stored vectors	2	3	4	5	6	7
negative vectors	2	3	4	5	6	7
spurious fixed points	2	4	8	16	24	45
total	4	7	12	21	30	52

12.2.4 Another visualization

A second kind of visualization of the basins of attraction allows us to perceive their gradual fragmentation in an associative network. Figure 12.8 shows the result of an experiment using input vectors in a 100-dimensional space. Several vectors were stored and the size of the basins of attraction of one of them and its bipolar complement were monitored using the following technique: the 100 components of each vector were divided into two groups of 50 components using random selection. The Hamming distance of a vector to the stored vector was measured by counting the number of different components in each group. In this way the vector $\mathbf{z}$ with Hamming distances 20 and 30 to the vector $\mathbf{x}$, according to the chosen partition in two groups of components, is a vector with total Hamming distance 50 to the stored vector. The chosen partition allows us to represent the vector $\mathbf{z}$ as a point at the position $(20, 30)$ in a two-dimensional grid. This point is colored black if $\mathbf{z}$ is associated with $\mathbf{x}$ by the associative network, otherwise it is left white. For each point in the grid a vector with the corresponding Hamming distance to $\mathbf{x}$ was generated.

As Figure 12.8 shows, when only 4 vectors have been stored, the vector $\mathbf{x}$ and its complement $-\mathbf{x}$ attract a considerable portion of their neighborhood.

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

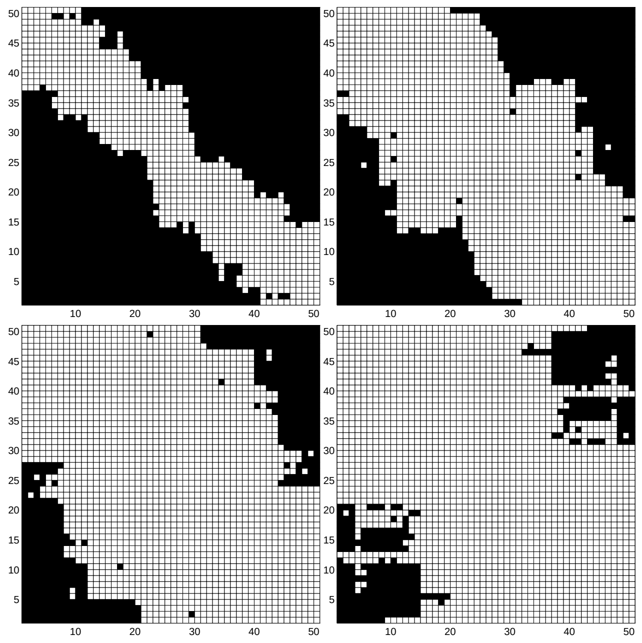

Fig. 12.8. Basin of attraction in 100-dimensional space of a stored vector. The number of stored patterns is 4, 6, 10 and 15, from top to bottom, left to right.

As the number of stored vectors increases, the basins of attraction become ragged and smaller, until by 15 stored vectors they are starting to become unusable. This is near to the theoretical limit for the capacity of an associative network.

12.3 The capacity problem

The experiments of the previous sections have shown that the basins of attraction of stored patterns deteriorate every time new vectors are presented to an associative network. If the crosstalk term becomes too large, it can even happen that previously stored patterns are lost, because when they are presented to the network one or more of their bits are flipped by the associative computation. We would like to keep the probability that this could happen low, so that stored patterns can always be recalled. Depending on the upper bound that we want to put on this probability some limits to the number of patterns m that can be stored safely in an autoassociative network with an $n \times n$ weight matrix $\mathbf{W}$ will arise. This is called the maximum *capacity* of the network and an often quoted rule of thumb is $m \approx 0.18n$. It is actually easy to derive this rule [189].

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

The crosstalk term for n-dimensional bipolar vectors and m patterns in the autoassociative case is

$$\frac{1}{n}\sum_{\ell \neq p}^{m} \mathbf{x}^\ell(\mathbf{x}^\ell \cdot \mathbf{x}^p).$$

This can flip a bit of a stored pattern if the magnitude of this term is larger than 1 and if it is of opposite sign. Assume that the stored vectors are chosen randomly from input space. In this case the crosstalk term for bit i of the input vector is given by

$$\frac{1}{n}\sum_{\ell \neq p}^{m} x_i^\ell(\mathbf{x}^\ell \cdot \mathbf{x}^p).$$

This is a sum of $(m-1)n$ bipolar bits. Since the components of each pattern have been selected randomly we can think of mn random bit selections (for large m and n). The expected value of the sum is zero. It can be shown that the sum has a binomial distribution and for large mn we can approximate it with a normal distribution. The standard deviation of the distribution is $\sigma = \sqrt{m/n}$. The probability of error P that the sum becomes larger than 1 or -1 is given by the area under the Gaussian from 1 to ∞ or from -1 to $-\infty$. That is,

$$P = \frac{1}{\sqrt{2\pi}\sigma}\int_1^\infty \mathrm{e}^{-x^2/(2\sigma^2)}dx$$

If we set the upper bound for one bit failure at 0.01, the above expression can be solved numerically to find the appropriate combination of m and n. The numerical solution leads to $m \approx 0.18n$ as mentioned before.

Note that these are just probabilistic results. If the patterns are correlated, even $m < 0.18n$ can produce problems. Also, we did not consider the case of a recursive computation and only considered the case of a false bit. A more precise computation does not essentially change the order of magnitude of these results: an associative network remains statistically safe only if fewer than $0.18n$ patterns are stored. If we demand perfect recall of all patterns more stringent bounds are needed [189]. The experiments in the previous section are on such a small scale that no problems were found with the recall of stored patterns even when m was much larger than $0.18n$.

12.4 The pseudoinverse

Hebbian learning produces good results when the stored patterns are nearly orthogonal. This is the case when m bipolar vectors are selected randomly from an n-dimensional space, n is large enough and m much smaller than n. In real applications the patterns are almost always correlated and the crosstalk in the expression

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

$$\mathbf{x}^p\mathbf{W} = \mathbf{y}^p(\mathbf{x}^p \cdot \mathbf{x}^p) + \sum_{\ell \neq p}^{m} \mathbf{y}^\ell(\mathbf{x}^\ell \cdot \mathbf{x}^p)$$

affects the recall process because the scalar products $\mathbf{x}^\ell \cdot \mathbf{x}^p$, for $\ell \neq p$, are not small enough. This causes a reduction in the *capacity* of the associative network, that is, of the number of patterns that can be stored and later recalled. Consider the example of scanned letters of the alphabet, digitized using a 16×16 grid of pixels. The pattern vectors do not occupy input space homogeneously but concentrate around a small region. We must therefore look for alternative learning methods capable of minimizing the crosstalk between the stored patterns. One of the preferred methods is using the pseudoinverse of the pattern matrix instead of the correlation matrix.

12.4.1 Definition and properties of the pseudoinverse

Let $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ be n-dimensional vectors and associate them with the m k-dimensional vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m$. The matrix $\mathbf{X}$ is, as before, the $m \times n$ matrix whose rows are the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$. The rows of the $m \times k$ matrix $\mathbf{Y}$ are the vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m$. We are looking for a weight matrix $\mathbf{W}$ such that

$$\mathbf{XW} = \mathbf{Y}.$$

Since in general $m \neq n$ and the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ are not necessarily linearly independent, the matrix $\mathbf{X}$ cannot be inverted. However, we can look for a matrix $\mathbf{W}$ which minimizes the quadratic norm of the matrix $\mathbf{XW} - \mathbf{Y}$, that is, the sum of the squares of all its elements. It is a well-known result of linear algebra that the expression $\|\mathbf{XW} - \mathbf{Y}\|^2$ is minimized by the matrix $\mathbf{W} = \mathbf{X}^+\mathbf{Y}$, where $\mathbf{X}^+$ is the so-called pseudoinverse of the matrix $\mathbf{X}$. In some sense the pseudoinverse is the best "approximation" to an inverse that we can get, and if $\mathbf{X}^{-1}$ exists, then $\mathbf{X}^{-1} = \mathbf{X}^+$.

Definition 14. *The pseudoinverse of a real $m \times n$ matrix is the real matrix $\mathbf{X}^+$ with the following properties:*

a) $\mathbf{XX}^+\mathbf{X} = \mathbf{X}$.
b) $\mathbf{X}^+\mathbf{XX}^+ = \mathbf{X}^+$.
c) $\mathbf{X}^+\mathbf{X}$ *and* $\mathbf{XX}^+$ *are symmetrical.*

It can be shown that the pseudoinverse of a matrix $\mathbf{X}$ always exists and is unique [14]. We can now prove the result mentioned before [185].

Proposition 18. *Let $\mathbf{X}$ be an $m \times n$ real matrix and $\mathbf{Y}$ be an $m \times k$ real matrix. The $n \times k$ matrix $\mathbf{W} = \mathbf{X}^+\mathbf{Y}$ minimizes the quadratic norm of the matrix $\mathbf{XW} - \mathbf{Y}$.*

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

Proof. Define $E = \|\mathbf{XW} - \mathbf{Y}\|^2$. The value of E can be computed from the equation

$$E = \mathrm{tr}(\mathbf{S}),$$

where $\mathbf{S} = (\mathbf{XW} - \mathbf{Y})^{\mathrm{T}}(\mathbf{XW} - \mathbf{Y})$ and $\mathrm{tr}(\cdot)$ denotes the function that computes the trace of a matrix, that is, the sum of all its diagonal elements. The matrix $\mathbf{S}$ can be rewritten as

$$\mathbf{S} = (\mathbf{X}^+\mathbf{Y} - \mathbf{W})^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}(\mathbf{X}^+\mathbf{Y} - \mathbf{W}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y}. \tag{12.7}$$

This can be verified by bringing $\mathbf{S}$ back to its original form. Equation (12.7) can be written as

$$\mathbf{S} = (\mathbf{X}^+\mathbf{Y} - \mathbf{W})^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{XX}^+\mathbf{Y} - \mathbf{X}^{\mathrm{T}}\mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y}.$$

Since the matrix $\mathbf{XX}^+$ is symmetrical (from the definition of $\mathbf{X}^+$), the above expression transforms to

$$\mathbf{S} = (\mathbf{X}^+\mathbf{Y} - \mathbf{W})^{\mathrm{T}}((\mathbf{XX}^+\mathbf{X})^{\mathrm{T}}\mathbf{Y} - \mathbf{X}^{\mathrm{T}}\mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y}.$$

Since from the definition of $\mathbf{X}^+$ we know that $\mathbf{XX}^+\mathbf{X} = \mathbf{X}$ it follows that

$$\begin{aligned}
\mathbf{S} &= (\mathbf{X}^+\mathbf{Y} - \mathbf{W})^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{Y} - \mathbf{X}^{\mathrm{T}}\mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y} \\
&= (\mathbf{X}^+\mathbf{Y} - \mathbf{W})^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(\mathbf{Y} - \mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y} \\
&= (\mathbf{XX}^+\mathbf{Y} - \mathbf{XW})^{\mathrm{T}}(\mathbf{Y} - \mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y} \\
&= (-\mathbf{XW})^{\mathrm{T}}(\mathbf{Y} - \mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}\mathbf{XX}^+(\mathbf{Y} - \mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y} \\
&= (-\mathbf{XW})^{\mathrm{T}}(\mathbf{Y} - \mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}(-\mathbf{XW}) + \mathbf{Y}^{\mathrm{T}}\mathbf{Y} \\
&= (\mathbf{Y} - \mathbf{XW})^{\mathrm{T}}(\mathbf{Y} - \mathbf{XW}).
\end{aligned}$$

This confirms that equation (12.7) is correct. Using (12.7) the value of E can be written as

$$E = \mathrm{tr}\left((\mathbf{X}^+\mathbf{Y} - \mathbf{W})^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}(\mathbf{X}^+\mathbf{Y} - \mathbf{W})\right) + \mathrm{tr}\left(\mathbf{Y}^{\mathrm{T}}(\mathbf{I} - \mathbf{XX}^+)\mathbf{Y}\right),$$

since the trace of an addition of two matrices is the addition of the trace of each matrix. The trace of the second matrix is a constant. The trace of the first is positive or zero, since the trace of a matrix of the form $\mathbf{AA}^{\mathrm{T}}$ is always positive or zero. The minimum of E is reached therefore when $\mathbf{W} = \mathbf{X}^+\mathbf{Y}$, as we wanted to prove. □

An interesting corollary of the above proposition is that the pseudoinverse of a matrix $\mathbf{X}$ minimizes the norm of the matrix $\mathbf{XX}^+ - \mathbf{I}$. This is what we mean when we say that the pseudoinverse is the second best choice if the inverse of a matrix is not defined.

The quadratic norm E has a straightforward interpretation for our associative learning task. Minimizing $E = \|\mathbf{XW} - \mathbf{Y}\|^2$ amounts to minimizing

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

the sum of the quadratic norm of the vectors $\mathbf{x}^i\mathbf{W} - \mathbf{y}^i$, where $(\mathbf{x}^i, \mathbf{y}^i)$, for $i = 1, \ldots, m$, are the vector pairs we want to associate. The identity

$$\mathbf{x}^i\mathbf{W} = \mathbf{y}^i + (\mathbf{x}^i\mathbf{W} - \mathbf{y}^i)$$

always holds. Since the desired result of the computation is $\mathbf{y}^i$, the term $\mathbf{x}^i\mathbf{W} - \mathbf{y}^i$ represents the deviation from the ideal situation, that is, the crosstalk in the associative network. The quadratic norm E defined before can also be written as

$$E = \sum_{i=1}^{m} \left\| \mathbf{x}^i\mathbf{W} - \mathbf{y}^i \right\|^2 .$$

Minimizing E amounts to minimizing the deviation from perfect recall. The pseudoinverse can thus be used to compute an optimal weight matrix $\mathbf{W}$.

Table 12.4. Percentage of 10-dimensional vectors with a Hamming distance (H) from 0 to 4, which converge to a stored vector in five iterations. The number of stored vectors increases from 1 to 7. The weight matrix is $\mathbf{X}^+\mathbf{X}$.

Number of stored vectors (dimension 10)

H	1	2	3	4	5	6	7
0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1	100.0	100.0	90.0	85.0	50.0	50.0	30.0
2	100.0	86.7	77.8	60.0	22.7	10.4	1.3
3	100.0	50.8	40.8	30.4	3.2	0.0	0.0
4	100.0	1.9	7.6	8.6	0.0	0.0	0.0

An experiment similar to the ones performed before (Table 12.4) shows that using the pseudoinverse to compute the weight matrix produces better results than when the correlation matrix is used. Only one iteration was performed during the associative recall. The table also shows that if more than five vectors are stored the quality of the results does not improve. This can be explained by noting that the capacity of the associative network has been overflowed. The stored vectors were also randomly chosen. The pseudoinverse method performs better than Hebbian learning when the patterns are correlated.

12.4.2 Orthogonal projections

In Sect. 12.2.2 we provided a geometric interpretation of the effect of the correlation matrix in the associative computation. We arrived at the conclusion that when an n-dimensional vector $\mathbf{x}^1$ is autoassociated, the weight matrix $(\mathbf{x}^1)^{\mathrm{T}}\mathbf{x}^1$ projects the whole of input space into the linear subspace spanned by

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

$\mathbf{x}^1$. However, the projection is not an orthogonal projection. The pseudoinverse can be used to construct an operator capable of projecting the input vector orthogonally into the subspace spanned by the stored patterns.

What is the advantage of performing an orthogonal projection? An autoassociative network must associate each input vector with the nearest stored pattern and must produce this as the result. In nonlinear associative networks this is done in two steps: the input vector $\mathbf{x}$ is projected first onto the linear subspace spanned by the stored patterns, and then the nonlinear function sgn$(\cdot)$ is used to find the bipolar vector nearest to the projection. If the projection falsifies the information about the distance of the input to the stored patterns, then a suboptimal selection could be the result. Figure 12.9 illustrates this problem. The vector $\mathbf{x}$ is projected orthogonally and non-orthogonally in the space spanned by the vectors $\mathbf{x}^1$ and $\mathbf{x}^2$. The vector $\mathbf{x}$ is closer to $\mathbf{x}^2$ than to $\mathbf{x}^1$. The orthogonal projection $\tilde{\mathbf{x}}$ is also closer to $\mathbf{x}^2$. However, the non-orthogonal projection $\breve{\mathbf{x}}$ is closer to $\mathbf{x}^1$ as to $\mathbf{x}^2$. This is certainly a scenario we should try to avoid.

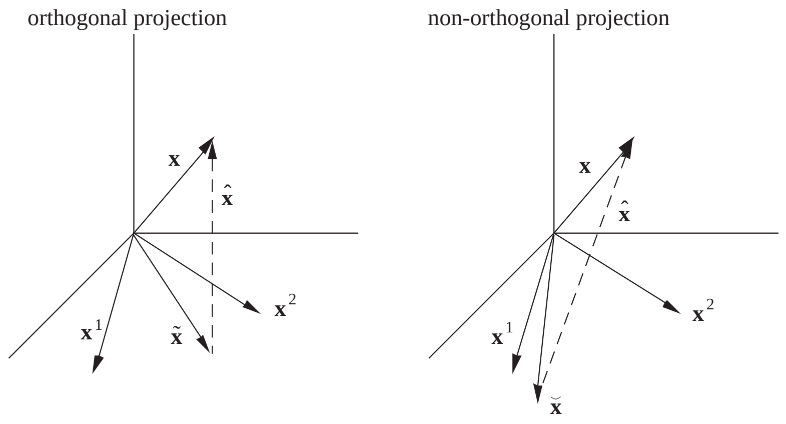

Fig. 12.9. Projections of a vector $\mathbf{x}$

In what follows we consider only the orthogonal projection. If we use the scalar product as the metric to assess the distance between patterns, the distance between the input vector $\mathbf{x}$ and the stored vector $\mathbf{x}^i$ is given by

$$d = \mathbf{x} \cdot \mathbf{x}^i = (\tilde{\mathbf{x}} + \hat{\mathbf{x}}) \cdot \mathbf{x}^i = \tilde{\mathbf{x}} \cdot \mathbf{x}^i, \tag{12.8}$$

where $\tilde{\mathbf{x}}$ represents the orthogonal projection onto the vector subspace spanned by the stored vectors, and $\hat{\mathbf{x}} = \mathbf{x} - \tilde{\mathbf{x}}$. Equation (12.8) shows that the original distance d is equal to the distance between the projection $\tilde{\mathbf{x}}$ and the vector $\mathbf{x}^i$. The orthogonal projection does not falsify the original information. We must therefore only show that the pseudoinverse provides a simple way

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

to find the orthogonal projection to the linear subspace defined by the stored patterns.

Consider m n-dimensional vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$, where $m < n$. Let $\tilde{\mathbf{x}}$ be the orthogonal projection of an arbitrary input vector $\mathbf{x} \neq \mathbf{0}$ on the subspace spanned by the m given vectors. In this case

$$\mathbf{x} = \tilde{\mathbf{x}} + \hat{\mathbf{x}},$$

where $\hat{\mathbf{x}}$ is orthogonal to the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$. We want to find an $n \times n$ matrix $\mathbf{W}$ such that

$$\mathbf{x}\mathbf{W} = \tilde{\mathbf{x}}.$$

Since $\hat{\mathbf{x}} = \mathbf{x} - \tilde{\mathbf{x}}$, the matrix $\mathbf{W}$ must fulfill

$$\hat{\mathbf{x}} = \mathbf{x}(\mathbf{I} - \mathbf{W}). \tag{12.9}$$

Let $\mathbf{X}$ be the $m \times n$ matrix whose rows are the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$. Then it holds for $\hat{\mathbf{x}}$ that

$$\mathbf{X}\hat{\mathbf{x}}^{\mathrm{T}} = \mathbf{0},$$

since $\hat{\mathbf{x}}$ is orthogonal to all vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$. A possible solution for this equation is

$$\hat{\mathbf{x}}^{\mathrm{T}} = (\mathbf{I} - \mathbf{X}^+\mathbf{X})\mathbf{x}^{\mathrm{T}}, \tag{12.10}$$

because in this case

$$\mathbf{X}\hat{\mathbf{x}}^{\mathrm{T}} = \mathbf{X}(\mathbf{I} - \mathbf{X}^+\mathbf{X})\mathbf{x}^{\mathrm{T}} = (\mathbf{X} - \mathbf{X})\mathbf{x}^{\mathrm{T}} = \mathbf{0}.$$

Since the matrix $\mathbf{I} - \mathbf{X}^+\mathbf{X}$ is symmetrical it holds that

$$\hat{\mathbf{x}} = \mathbf{x}(\mathbf{I} - \mathbf{X}^+\mathbf{X}). \tag{12.11}$$

A direct comparison of equations (12.9) and (12.11) shows that $\mathbf{W} = \mathbf{X}^+\mathbf{X}$, since the orthogonal projection $\hat{\mathbf{x}}$ is unique. This amounts to a proof that the orthogonal projection of a vector $\mathbf{x}$ can be computed by multiplying it with the matrix $\mathbf{X}^+\mathbf{X}$.

What happens if the number of patterns m is larger than the dimension of the input space? Assume that the m n-dimensional vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ are to be associated with the m real numbers $y_1, y_2, \ldots, y_m$. Let $\mathbf{X}$ be the matrix whose rows are the given patterns and let $\mathbf{y} = (y_1, y_2, \ldots, y_m)$. We are looking for the $n \times 1$ matrix $\mathbf{W}$ such that

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

In general there is no exact solution for this system of m equations in n variables. The best approximation is given by $\mathbf{W} = \mathbf{X}^+\mathbf{y}^{\mathrm{T}}$, which minimizes the quadratic error $\|\mathbf{X}\mathbf{W} - \mathbf{y}^{\mathrm{T}}\|^2$. The solution $\mathbf{X}^+\mathbf{X}^{\mathrm{T}}$ is therefore the same as that we find if we use *linear regression.*

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

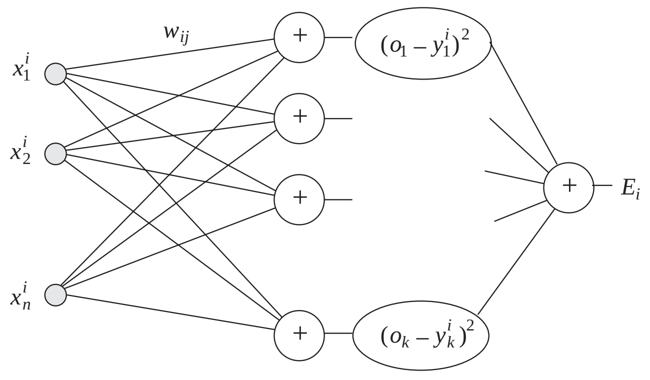

Fig. 12.10. Backpropagation network for a linear associative memory

The only open question is how best to compute the pseudoinverse. We already discussed in Chap. 9 that there are several algorithms which can be used, but that a simple approach is to compute an approximation using a backpropagation network. A network, such as the one shown in Figure 12.10, can be used to find the appropriate weights for an associative memory. The units in the first layer are linear associators. The learning task consists in finding the weight matrix $\mathbf{W}$ with elements w_{ij} that produces the best mapping from the vectors $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^m$ to the vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^m$. The network is extended as shown in Figure 12.10. For the i-th input vector, the output of the network is compared to the vector $\mathbf{y}^i$ and the quadratic error E_i is computed. The total quadratic error $E = \sum_{i=1}^m E_i$ is the quadratic norm of the matrix $\mathbf{XW} - \mathbf{Y}$. Backpropagation can find the matrix $\mathbf{W}$ which minimizes the expression $\|\mathbf{XW} - \mathbf{Y}\|^2$. If the input vectors are linearly independent and $m < n$, then a solution with zero error can be found. If $m \geq n$ and $\mathbf{Y} = \mathbf{I}$, the network of Figure 12.10 can be used to find the pseudoinverse $\mathbf{X}^+$ or the inverse $\mathbf{X}^{-1}$ (if it exists). This algorithm for the computation of the pseudoinverse brings backpropagation networks in direct contact with the problem of associative networks. Strictly speaking, we should minimize $\|\mathbf{XW} - \mathbf{Y}\|^2$ and the quadratic norm of $\mathbf{W}$. This is equivalent to introducing a *weight decay term* in the backpropagation computation.

12.4.3 Holographic memories

Associative networks have found many applications for pattern recognition tasks. Figure 12.11 shows the "canonical" example. The faces were scanned and encoded as vectors. A white pixel was encoded as -1 and a black pixel as 1. Some images were stored in an associative network. When later a *key*, that is, an incomplete or noisy version of one of the stored images is presented to the network, this completes the image and finds the one with the greatest similarity. This is called an *associative recall.*

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

Fig. 12.11. Associative recall of stored patterns

This kind of application can be implemented in conventional workstations. If the dimension of the images becomes too large (for example 10^6 pixels for a 1000×1000 image, the only alternative is to use innovative computer architectures, like so-called *holographic memories*. They work with similar methods to those described here, but computation is performed in parallel using optical elements. We leave the discussion of these architectures for Chap. 18.

12.4.4 Translation invariant pattern recognition

It would be nice if the process of associative recall could be made robust in the sense that those patterns which are very similar, except for a translation in the plane, could be identified as being, in fact, similar. This can be done using the two-dimensional Fourier transform of the scanned images. Figure 12.12 shows an example of two identical patterns positioned with a small displacement from each other.

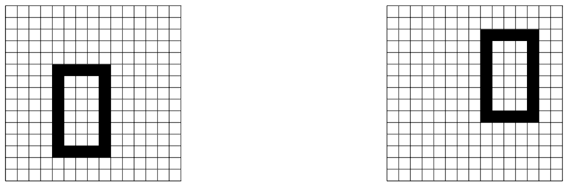

Fig. 12.12. The pattern "0" at two positions of a grid

The two-dimensional discrete Fourier transform of a two-dimensional array is computed by first performing a one-dimensional Fourier transform of the rows of the array and then a one-dimensional Fourier transform of the columns of the results (or vice versa). It is easy to show that the absolute value of the Fourier coefficients does not change under a translation of the two-dimensional pattern. Since the Fourier transform also has the property that it preserves angles, that is, similar patterns in the original domain are also similar in

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

the Fourier domain, this preprocessing can be used to implement translation-invariant associative recall. In this case the vectors which are stored in the network are the absolute values of the Fourier coefficients for each pattern.

Definition 15. *Let $a(x,y)$ denote an array of real values, where x and y represent integers such that $0 \leq x \leq N-1$ and $0 \leq y \leq N-1$. The two-dimensional discrete Fourier transform $F_a(f_x,f_y)$ of $a(x,y)$ is given by*

$$F_a(f_x,f_y) = \frac{1}{N}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1} a(x,y)\mathrm{e}^{\frac{2\pi}{N}ixf_x}\mathrm{e}^{\frac{2\pi}{N}iyf_y}$$

where $0 \leq f_x \leq N-1$ and $0 \leq f_y \leq N-1$.

Consider two arrays $a(x,y)$ and $b(x,y)$ of real values. Assume that they represent the same pattern, but with a certain displacement, that is $b(x,y) = a(x+d_x, y+d_y)$, where d_x and d_y are two given integers. Note that the additions $x+d_x$ and $y+d_y$ are performed modulo N (that is, the displaced patterns wrap around the two-dimensional array). The two-dimensional Fourier transform of the array $b(x,y)$ is therefore

$$F_b(f_x,f_y) = \frac{1}{N}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1} a(x+d_x,y+d_y)\mathrm{e}^{\frac{2\pi}{N}ixf_x}\mathrm{e}^{\frac{2\pi}{N}iyf_y}.$$

With the change of indices $x' = (x+d_x) \bmod N$ and $y' = (y+d_y) \bmod N$ the above expression transforms to

$$F_b(f_x,f_y) = \frac{1}{N}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1} a(x',y')\mathrm{e}^{\frac{2\pi}{N}ix'f_x}\mathrm{e}^{\frac{2\pi}{N}iy'f_y}\mathrm{e}^{\frac{-2\pi}{N}id_xf_x}\mathrm{e}^{\frac{-2\pi}{N}id_yf_y}.$$

The two sums can be rearranged and the constant factors taken out of the double sum to yield

$$F_b(f_x,f_y) = \frac{1}{N}\mathrm{e}^{\frac{-2\pi}{N}id_xf_x}\mathrm{e}^{\frac{-2\pi}{N}id_yf_y}\sum_{y'=0}^{N-1}\sum_{x'=0}^{N-1} a(x',y')\mathrm{e}^{\frac{2\pi}{N}ix'f_x}\mathrm{e}^{\frac{2\pi}{N}iy'f_y}.$$

which can be written as

$$F_b(f_x,f_y) = \mathrm{e}^{\frac{-2\pi}{N}id_xf_x}\mathrm{e}^{\frac{-2\pi}{N}id_yf_y}F_a(f_x,f_y).$$

This expression tells us that the Fourier coefficients of the array $b(x,y)$ are identical to the coefficients of the array $a(x,y)$, except for a phase factor. Since taking the absolute values of the Fourier coefficients eliminates any phase factor, it holds that

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

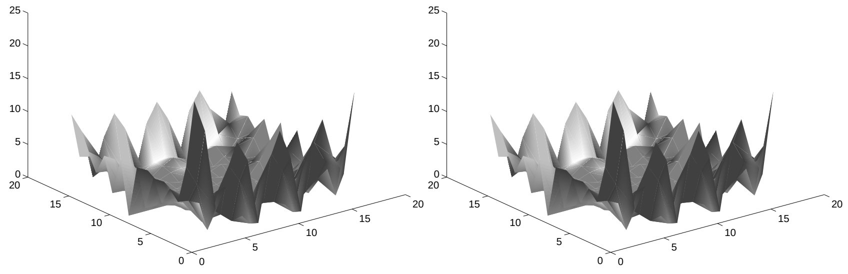

Fig. 12.13. Absolute values of the two-dimensional Fourier transform of the patterns of Figure 12.12

$$\begin{aligned}\|F_b(f_x, f_y)\| &= \|\mathrm{e}^{\frac{-2\pi}{N} i d_x f_x} \mathrm{e}^{\frac{-2\pi}{N} i d_y f_y}\| \|F_a(f_x, f_y)\| \\ &= \|F_a(f_x, f_y)\|,\end{aligned}$$

and this proves that the absolute values of the Fourier coefficients for both patterns are identical.

Figure 12.13 shows the absolute values of the two-dimensional Fourier coefficients for the two patterns of Figure 12.12. They are identical, as expected from the above considerations.

Note that this type of comparison holds as long as the background noise is low. If there is a background with a certain structure it should be displaced together with the patterns, otherwise the analysis performed above does not hold. This is what makes the translation-invariant recognition of patterns a difficult problem when the background is structured. Recognizing faces in real photographs can become extremely difficult when a typical background (a forest, a street, for example) is taken into account.

12.5 Historical and bibliographical remarks

Associative networks have been studied for a long time. Donald Hebb considered in the 1950s how neural assemblies can self-organize into feedback circuits capable of recognizing patterns [308]. Hebbian learning has been interpreted in different ways and several modifications of the basic algorithm have been proposed, but they all have three aspects in common: Hebbian learning is a local, interactive, and time-dependent mechanism [73]. A synaptic phenomenon in the hippocampus, known as long-term potentiation, is thought to be produced by Hebbian modification of the synaptic strength.

There were some other experiments with associative memories in the 1960s, for example the hardware implementations by Karl Steinbuch of his "learning matrix". A preciser mathematical description of associative networks was given by Kohonen in the 1970s [253]. His experiments with many different

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

classes of associative memories contributed enormously to the renewed surge of interest in neural models. Some researchers tried to find biologically plausible models of associative memories following his lead [89]. Some discrete variants of correlation matrices were analyzed in the 1980s, as done for example by Kanerva who considered the case of weight matrices with only two classes of elements, 0 or 1 [233].

In recent times much work has been done on understanding the dynamical properties of recurrent associative networks [231]. It is important to find methods to describe the changes in the basins of attraction of the stored patterns [294]. More recently Haken has proposed his model of a *synergetic computer*, a kind of associative network with a continuous dynamics in which synergetic effects play the crucial role [178]. The fundamental difference from conventional models is the continuous, instead of discrete, dynamics of the network, regulated by some differential equations.

The Moore–Penrose pseudoinverse has become an essential instrument in linear algebra and statistics [14]. It has a simple geometric interpretation and can be computed using standard linear algebraic methods. It makes it possible to deal efficiently with correlated data of the kind found in real applications.

Exercises

1. The Hamming distance between two n-dimensional binary vectors $\mathbf{x}$ and $\mathbf{y}$ is given by $h = \sum_{i=1}^{n}(x_i(1-y_i)+y_i(1-x_i))$. Write h as a function of the scalar product $\mathbf{x}\cdot\mathbf{y}$. Find a similar expression for bipolar vectors. Show that measuring the similarity of vectors using the Hamming distance is equivalent to measuring it using the scalar product.
2. Implement an associative memory capable of recognizing the ten digits from 0 to 9, even in the case of a noisy input.
3. Preprocess the data, so that the pattern recognition process is made translation invariant.
4. Find the inverse of an invertible matrix using the backpropagation algorithm.
5. Show that the pseudoinverse of a matrix is unique.
6. Does the backpropagation algorithm always find the pseudoinverse of any given matrix?
7. Show that the two-dimensional Fourier transform is a composition of two one-dimensional Fourier transforms.
8. Express the two-dimensional Fourier transform as the product of three matrices (see the expression for the one-dimensional FT in Chap. 9).

R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996