Chapter 4

# Feedforward Neural Networks

## 4.1 Introduction

This chapter presents a detailed analysis of the pattern recognition tasks that can be performed by a feedforward artificial neural network. As mentioned earlier, a feedforward artificial neural network consists of layers of processing units, each layer feeding input to the next layer in a feedforward manner through a set of connection strengths or weights. The simplest such network is a two layer network.

By a suitable choice of architecture for a feedforward network, it is possible to perform several pattern recognition tasks. The simplest task is a pattern association task, which can be realized by a two layer feedforward network with linear processing units. A detailed analysis of the linear association network shows that the network is limited in its capabilities. In particular, the number of input-output pattern pairs to be associated are limited to the dimensionality of the input pattern, and also the set of input patterns must be linearly independent. The constraint on the number of input patterns is overcome by using a two layer feedforward network with nonlinear processing units in the output layer. This modification automatically leads to the consideration of pattern classification problems. While this modification overcomes the constraints of number and linear independence on the input patterns, it introduces the limitations of linear separability of the functional relation between input and output patterns. Classification problems which are not linearly separable are called hard problems. In order to overcome the constraint of linear separability for pattern classification problems, a multilayer feedforward network with nonlinear processing units in all the intermediate hidden layers and in the output layer is proposed. While a multilayer feedforward architecture could solve representation of the hard problems in a network, it introduces the problem of hard learning, i.e., the difficulty in adjusting the weights of the network to capture the implied functional relationship between the given input-output pattern pairs. The hard learning problem is solved by using the backpropagation learning algorithm. The resulting network provides a solution to the pattern mapping problems. The generalization (ability to learn a mapping function) feature of a multilayer feedforward network with the backpropagation learning law depends on several factors such as the architectural details of the network, the learning rate parameter of the training process and the training samples themselves.

Table 4.1 shows the summary of the topics to be discussed in this chapter. The pattern association problem is discussed in Section 4.2.

Table 4.1 Pattern Recognition Tasks by Feedfoward Neural Networks

| | |
|---|---|
| Pattern association | |
| • *Architecture:* | Two layers, linear processing units, single set of weights |
| • *Learning:* | Hebb's (orthogonal) rule, Delta (linearly independent) rule |
| • *Recall:* | Direct |
| • *Limitation:* | Linear independence, number of patterns restricted to input dimensionality |
| • *To overcome:* | Nonlinear processing units, leads to a pattern classification problem |
| Pattern classification | |
| • *Architecture:* | Two layers, nonlinear processing units, geometrical interpretation |
| • *Learning:* | Perceptron learning |
| • *Recall:* | Direct |
| • *Limitation:* | Linearly separable functions, cannot handle hard problems |
| • *To overcome:* | More layers, leads to a hard learning problem |
| Pattern mapping or classification | |
| • *Archztecture:* | Multilayer (hidden), nonlinear processing units, geometrical interpretation |
| • *Learning:* | Generalized delta rule (backpropagation) |
| • *Recall:* | Direct |
| • *Limitation:* | Slow learning, does not guarantee convergence |
| • *To overcome:* | More complex architecture |

This section gives a detailed analysis of a linear associative network, and shows the limitations of the network for pattern association problems. Section 4.3 describes the pattern classification problem. An analysis of a two layer feedforward network with nonlinear processing units in the output layer brings out the limitations of the network for pattern classification task. The section also discusses the problems of classification, representation, learning and convergence in the context of perceptron networks. In Section 4.4 the problem of pattern mapping by a multilayer neural network is discussed. The chapter concludes with a discussion on the backpropagation learning law and its implications for generalization in a pattern mapping problem.

## 4.2 Analysis of Pattern Association Networks

### 4.2.1 Linear Associative Network

The objective in pattern association is to design a network that can represent the association in the pairs of vectors $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \ldots, L$, through a set of weights to be determined by a learning law. The given set of input-output pattern pairs is called training data. The input **patterns** are typically generated synthetically, like machine printed characters. The input patterns used for recall may be corrupted by external noise.

The following vector and matrix notations are used for the analysis of a linear associative network:

| | |
|---|---|
| Input vector | $\mathbf{a}_l = [a_{l1}, a_{l2}, \ldots, a_{lM}]^T$ |
| Activation vector of input layer | $x = [x_1, x_2, \ldots, x_M]^T$ |
| Activation vector of output layer | $\mathbf{y} = [y_1, y_2, \ldots, y_N]^T$ |
| Output **vector** | $\mathbf{b}_l = [b_{l1}, b_{l2}, \ldots, b_{lN}]^T$ |
| Input matrix | $A = [\mathbf{a}_1\ \mathbf{a}_2 \ldots \mathbf{a}_L]$ is an $M \times L$ matrix |
| Output matrix | $B = [\mathbf{b}_1\ \mathbf{b}_2 \ldots \mathbf{b}_L]$ is an $N \times L$ matrix |
| Weight matrix | $W = [\mathbf{w}_1\ \mathbf{w}_2 \ldots \mathbf{w}_N]^T$ **is an** $N \times M$ **matrix** |
| Weight vector for $j$th unit of output layer | $\mathbf{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jM}]^T$ |

The network consists of a set of weights connecting two layers of processing units as shown in Figure *4.1.* The output function of each

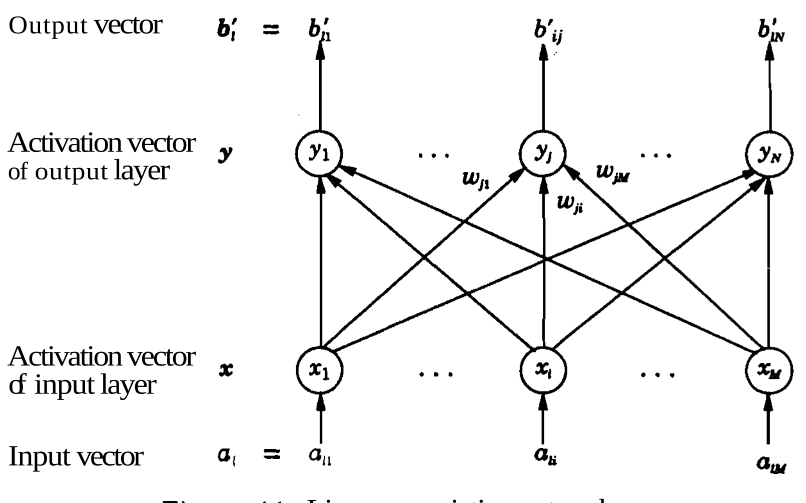


Figure 4.1 Linear associative network.

unit in these layers is linear. Each output unit **receives inputs** from the M input units corresponding to the M-dimensional input vectors. The number ($N$) of output units corresponds to the dimensionality of the output vectors. Due to linearity of the output function, the activation values ($x_i$) and the signal values of the units in the input layer are the same as the input data values $a_{li}$. The activation value of the $j$th unit in the output layer is given by

$$y_j = \sum_{i=1}^{M} w_{ji} a_{li} = \mathbf{w}_j^T \mathbf{a}_l, \qquad j = 1, 2, ..., N. \qquad (4.1)$$

The output ($b'_{lj}$) of the $j$th unit is the same as its activation value $y_j$, since the output function of the unit is linear, i.e., $b'_{lj} = y_j$. The network is called linear since the output of the network is simply a linear weighted sum of the component values of the input pattern.

The objective is to determine a set of weights $\{w_{ji}\}$ in such a way that the actual output $b'_{lj}$ is equal to the desired output $b_{lj}$ for all the given L pattern pairs. The weights are determined by using the criterion that the total mean squared error between the desired output and the actual output is to be minimized. The weights can be determined either by computing them from the training data set or by learning. Computation of weights makes use of all the training set data together. On the other hand, in learning, the weights are updated after presentation of each of the input-output pattern pairs in the training set.

### 4.2.2 Determination of Weights by Computation

For a linear associative network Mecht-Nielsen, 19901,

$$x_i = a_{li}, \; i = 1, 2, ..., M \qquad (4.2)$$

$$y_j = \sum_{i=1}^{M} w_{ji} x_i, \; j = 1, 2, ..., N \qquad (4.3)$$

$$b'_{lj} = y_j = \mathbf{w}_j^T \mathbf{x} = \mathbf{w}_j^T \mathbf{a}_l, \; j = 1, 2, ..., N \qquad (4.4)$$

Actual output vector

$$\mathbf{b}'_l = \mathbf{y} = W\mathbf{x} = W\mathbf{a}_l \qquad (4.5)$$

Error in the output is given by the distance between the desired output vector and the actual output vector. The total error $E(W)$ over all the L input-output pattern pairs is given by

$$E(W) = \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{N} (b_{lj} - b'_{lj})^2$$

$$= \frac{1}{L} \sum_{l=1}^{L} \| \mathbf{b}_l - W\mathbf{a}_l \|^2 \qquad (4.6)$$

We can write

$$E(W) = \frac{1}{L} \| B - WA \|^2 \tag{4.7}$$

where the square norm

$$\| B - WA \|^2 = \sum_{l=1}^{L} \sum_{j=1}^{N} (b_{lj} - \mathbf{w}_j^T \mathbf{a}_l)^2 \tag{4.8}$$

Using the definition that the trace of a square matrix S is the sum of the main diagonal entries of S, it is easy to see that

$$E(W) = \frac{1}{L} \mathrm{tr}(S), \tag{4.9}$$

where the matrix S is given by

$$S = (B - WA)(B - WA)^T \tag{4.10}$$

and $\mathrm{tr}(S)$ is the trace of the matrix $S$.

Using the definition for pseudoinverse of a matrix [Penrose, 19551, i.e., $A^+ = A^T(AA^T)^{-1}$, we get the matrix identities $A^+AA^T = A^T$ and $AA^T(A^+)^T = A$. Using these matrix identities we get

$$S = (W - BA^+)AA^T(W - BA^+)^T + B(I - A^+A)B^T \tag{4.11}$$

It can be seen that the trace of the first term in Eq. (4.11) is always nonnegative, as it is in a quadratic form of the real symmetric matrix $AA^T$. It becomes zero for $W = BA^+$. The trace of the second term is a constant, independent of W. Since the trace of sum of matrices is the sum of traces of the individual matrices, the error $E(W)$ is minimum when $W = BA^+$. The minimum error is obtained by substituting W = BA+ in Eq. (4.7), and is given by

$$\begin{aligned} E_{\min} &= \frac{1}{L} \| B - BA^+A \|^2 \\ &= \frac{1}{L} \mathrm{tr}[(B(I - A^+A))(B(I - A^+A))^T] \\ &= \frac{1}{L} \mathrm{tr}[B(I - A^+A)B^T]. \end{aligned} \tag{4.12}$$

where $I$ is an L x L identity matrix. The above simplification is obtained by using the following matrix identities: $(A^+A)^T = A^T(A^+)^T$ and $AA^T(A^+)^T = A$.

The following singular value decomposition (SVD) of an $M \times L$ matrix A is used to compute the pseudoinverse and to evaluate the minimum error. Assuming $L \leq M$, the SVD of a matrix A is given by [Strang, 19801

$$A = \sum_{i=1}^{L} \lambda_i^{1/2} \mathbf{p}_i \mathbf{q}_i^T, \tag{4.13}$$

where $AA^T\mathbf{p}_i = \lambda_i\mathbf{p}_i$, $A^TA\mathbf{q}_i = \lambda_i\mathbf{q}_i$, and the sets $(\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_M)$ and $\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_L)$ are each orthogonal. The eigenvalues $\lambda_i$ of the matrices $AA^T$ and $A^TA$ will be real and nonnegative, since the matrices are symmetric. The eigenvalues are ordered, **i.e.**, I, $\geq \lambda_{i+1}$. Note that the $\mathbf{p}_i$'s are *M*-**dimensional** vectors and $\mathbf{q}_i$'s are L-dimensional vectors.

The pseudoinverse $A^+$ of A is given by

$$A^+ = \sum_{i=1}^{r} \lambda_i^{-1/2}\,\mathbf{q}_i\mathbf{p}_i^T, \tag{4.14}$$

where $r$ is the rank (maximum number of linearly independent columns) of A. Also $r$ turns out to be the number of nonzero eigenvalues **I,**. Note that if $r$ = L, then all the L column vectors are linearly independent.

Using the SVD, it can be shown that $A^+A = I_{L\times L}$, if L is the number of linearly independent columns of A. In such a case $I - A^+A = 0$ (null **vector**), and hence $E_{\min} = 0$ (See **Eq.** (4.12)). Therefore, for a linearly independent set of input pattern vectors, the error in the recall of the associated output pattern vector is zero, if the optimum choice of W = $BA^+$ is used.

If the rank $r$ of the matrix A is less than L, then the input vectors are linearly dependent. In this case also the choice of the weight matrix as W = BA+still results in the least **error** $E_{\min}$. But this least error is not zero in this case. The value of the error depends on the rank $r$ of the matrix. The matrix $A^+A$ will have a sub-matrix $I_{r\times r}$, and all the other four sub-matrices will be zero. That is

$$[A^+A]_{L\times L} = \begin{bmatrix} I_{r\times r} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{4.15}$$

The expression for minimum error is given from **Eq.** (4.12) as

$$\begin{aligned} E_{\min} &= \frac{I}{L}\,\mathrm{tr}[B(I_{L\times L} - A^+A)B^T] \\ &= \frac{I}{L}\sum_{i=r+1}^{L} \|B\mathbf{q}_i\|^2 \end{aligned} \tag{4.16}$$

The **next** issue is how to achieve the minimum error retrieval from the linear associative network, when there is noise $\mathbf{n}_l$ added to the input vectors. The noisy input vectors are

$$\mathbf{c}_l = \mathbf{a}_l + \mathbf{n}_l \tag{4.17}$$

It is assumed that each component of the noise vector $\mathbf{n}_l$ is uncorrelated with the other components and also with the components of the pattern vectors, and has the same standard deviation $\sigma$. Let C be an M x L matrix of the noisy input vectors. The objective is to find a W that minimizes

$$E(W) = \frac{1}{L} \| B - WC \|^2 \tag{4.18}$$

Murakami has shown that, if W = $BA^+$, then the expression for error $E(W)$ is given by [Murakami and Aibara, **19871**

$$E(W) = \frac{1}{L} \left[ \sum_{i=r+1}^{L} \| B\mathbf{q}_i \|^2 + L\sigma^2 \sum_{i=1}^{r} \lambda_i^{-1} \| B\mathbf{q}_i \|^2 \right] \tag{4.19}$$

The first term in the square brackets can be attributed to the linear dependency part of the column vectors of A. If the rank of the matrix A is L, then $r$ = L, and the first term will be zero. Therefore, the error is determined by the noise power $\sigma^2$. If in addition, there is no noise, **i.e.,** $\sigma = 0$, then the error $E(W) = 0$. In that case error-free recall is possible.

To minimize $E(W)$ **in** the presence of noise in the input pattern vectors, choose W = B $\hat{A}^+$, where

$$\hat{A}^+ = \sum_{i=1}^{s} \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T. \tag{4.20}$$

The value of $s$ is determined in such a way that

$$\frac{L\sigma^2}{\lambda_s} \leq 1 \leq \frac{L\sigma^2}{\lambda_{s+1}} \tag{4.21}$$

That is, the noise power $\sigma^2$ will decide how many terms should be considered in the SVD expression for the pseudoinverse. If the eigenvalue $\lambda_i$ is so small that the noise power dominates, then that eigenvector could as well be included in the first term of the expression in Eq. **(4.19)** for the error corresponding to the linear dependence. This will reduce the error.

Note that this analysis is valid only if the legal inputs are corrupted by noise. It is not valid if the input consists of only noise. The expression for the error $E(W)$ is applicable for the closed set of the column vectors in A [**Murakami** and Aibara, **19871.**

## 4.23 Determination of Weights by Learning

It is desirable to determine the weights of a network in an incremental manner, as and when a new training input-output pattern pair is available. This is called ***learning.*** Each update of the weights with a new input data can be interpreted as network learning. Computationally also learning is preferable because it does not require information of all the training set data at the same time. As will be seen later in this section, it is also preferable to have learning confined to a local operation. That is, the update of a weight connecting two processing units depends only on the connection weight and the activations of the units on either side of the connection. Two learning laws and their variations, as applicable to a linear associative network, are discussed in this section.

**Hebb's law:** Let the input pattern vector $\mathbf{a}_l$ and the corresponding desired output pattern vector $\mathbf{b}_l$ be applied to the linear associative network. According to the Hebb's law, the updated weight value of a connection depends only on the activations of the processing units on either side of the connecting link. That is

$$\begin{aligned} w_{ji}(l) &= w_{ji}(l-1) + x_i y_j \\ &= w_{ji}(l-1) + a_{li}\, b_{lj}, \\ &\quad \text{for } i = 1, 2, \ldots, M; \quad j = 1, 2, \ldots, N \end{aligned} \tag{4.22}$$

Note that the computation of the increment $x_i y_j = a_{li} b_{lj}$ is purely local for the processor unit and the input-output pattern pair. The updated weight matrix for the application of the $l$th pair $(\mathbf{a}_l, \mathbf{b}_l)$ is given by

$$W(l) = W(l-1) + \mathbf{b}_l \mathbf{a}_l^T, \tag{4.23}$$

where $W(l-1)$ refers to the weight matrix after presentation of the first $(l-1)$ pattern pairs, and $W(l)$ refers to the weight matrix after presentation of the first $l$ pattern pairs. Note that $\mathbf{b}_l \mathbf{a}_l^T$ is the outer product of the two vectors, which results in an $N \times M$ matrix. Each element of this matrix is an increment of the corresponding element in the weight matrix.

If the initial values of the elements of the weight matrix are assumed to be zero, then the weight matrix resulting after application of the L input-output pattern vector pairs $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \ldots, L$, is given by

$$W = \sum_{l=1}^{L} \mathbf{b}_l \mathbf{a}_l^T = BA^T, \tag{4.24}$$

where the element $w_{ji}$ of W is given by

$$w_{ji} = \sum_{l=1}^{L} a_{li} b_{lj} \tag{4.25}$$

To verify whether the network has learnt the association of the given set of input-output pattern vector pairs, apply the input pattern $\mathbf{a}_k$ and determine the actual output vector $\mathbf{b}_k'$.

$$\begin{aligned} \mathbf{b}_k' &= W\mathbf{a}_k = \sum_{l=1}^{L} \mathbf{b}_l \mathbf{a}_l^T \mathbf{a}_k \\ &= \mathbf{b}_k(\mathbf{a}_k^T \mathbf{a}_k) + \sum_{l \neq k} \mathbf{b}_l(\mathbf{a}_l^T \mathbf{a}_k) \end{aligned} \tag{4.26}$$

It is obvious from the above equation that the actual output $\mathbf{b}'_k$ is not the same as the desired output $\mathbf{b}_k$. Only if some restrictions are imposed on the set of input pattern vectors $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_L\}$, we can get the recall of the correct output pattern $\mathbf{b}_k$ for the input pattern $\mathbf{a}_k$. The restriction is that the set of input vectors must be **orthonormal**. That is

$$\begin{aligned} \mathbf{a}_k^T\mathbf{a}_l = \mathbf{a}_l^T\mathbf{a}_k = \delta_{kl} &= 1 \quad \text{if } l = k \\ &= 0 \quad \text{if } l \neq k \end{aligned} \qquad (4.27)$$

In such a case the first term in the expression for $\mathbf{b}'_k$ in Eq. (4.26) becomes $\mathbf{b}_k$, and the second term becomes zero, as each of the products $\mathbf{a}_l^T\mathbf{a}_k$ is zero for $l \neq k$. The restriction of orthogonality limits the total number ($L$) of the input patterns in the set A to M, i.e., the dimensionality of the input vectors, as there can be only M or less than M mutually orthogonal vectors in an M-dimensional space.

If the restriction of orthogonality on the set of input vectors is relaxed to mere linear independence, then the expression in Eq. (4.26) for recall reduces to

$$\mathbf{b}'_k = \mathbf{b}_k + \mathbf{e} \qquad (4.28)$$

where it is assumed that the vectors are of unit magnitude, so that $\mathbf{a}_k^T\mathbf{a}_k = 1$. This leaves an error. term e indicating that the recall is not perfect, if the weight matrix is derived using the Hebb's law.

However, it was shown in the previous Section 4.2.2 that, for linearly independent set of input vectors, exact recall can be achieved if the weight matrix W is chosen as W = $BA^+$, where $A^+$ is the pseudoinverse of the matrix A. If the set of input vectors are not linearly independent, then still the best choice of W is $BA^+$, as this yields, on the average, the least squared error in the recall of the associated pattern. The error is defined as the difference between the desired and the actual output patterns from the associative network. If there is noise in the input, the best choice of the weight matrix W is $B\hat{A}^+$, where $\hat{A}^+$ includes fewer ($s$) terms in the singular value decomposition expansion of A than the rank ($r$) of the matrix, the choice of s being dictated by the level of the noise (See Eq. (4.21)).

For all these best choices of W, the weight values have to be computed from the knowledge of the complete input pattern matrix A, since all of them need the SVD of A to compute the pseudoinverse $A^+$. However, it is possible, at least in some cases, to develop learning algorithms which can approach the best choices for the weight matrices. The purpose of these learning algorithms is to provide a procedure for incremental update of the weight matrix when an input-output pattern pair is presented to the network. Most of these learning algorithms are based on gradient descent along an error surface (See Appendix C). The most basic among them is Widrow and

**Hoff's** least mean square (**LMS**) algorithm [**Widrow** and Hoff, 19601. The gradient descent algorithms are discussed in detail later in the section on pattern mapping tasks.

**Widrow's law:** A form of **Widrow** learning can be used to obtain W= BA+recursively. Let W(l – 1)be the weight matrix after presentation of (1– 1)samples. Then $W(l-1) = B(l-1)A^+(l-1)$, where the matrices $B(l-1)$ and $A(l-1)$ are composed of the first $(l-1)$ vectors of $\mathbf{b}_k$ and the first (1– 1)vectors of $\mathbf{a}_k$, respectively. When the pair $(\mathbf{a}_l, \mathbf{b}_l)$ is given to the network, then the updated matrix is given by (See [Hecht-Nielsen, 19901)

$$W(l) = W(l-1) + (\mathbf{b}_l - W(l-1)\mathbf{a}_l)\mathbf{p}_l^T \tag{4.29}$$

where

$$\mathbf{p}_l = \frac{[I - A(l-1)A^+(l-1)]\mathbf{a}_l}{|[I - A(l-1)A^+(l-1)]\mathbf{a}_l|^2}, \quad \text{if the denominator is} \neq 0$$

$$= \frac{A^T(l-1)A^+(l-1)\mathbf{a}_l}{1 + |A^+(l-1)\mathbf{a}_l|^2}, \quad \text{otherwise} \tag{4.30}$$

By starting with zero initial values for all the weights, and successively adding the pairs $(\mathbf{a}_1, \mathbf{b}_1), (\mathbf{a}_2, \mathbf{b}_2), \ldots, (\mathbf{a}_L, \mathbf{b}_L)$, we can obtain the final pseudoinverse-based weight matrix $W = BA^+$. The problem with this approach is that the recursive procedure **cannot** be implemented locally because of the need to calculate $\mathbf{p}_l$ in Eq. (4.29).

The same eventual effect can be approximately realized using the following variation of the above learning law,

$$W(l) = W(l-1) + \eta\,(\mathbf{b}_l - W(l-1)\mathbf{a}_l)\mathbf{a}_l^T, \tag{4.31}$$

where $\eta$ is a small positive constant called the learning rate parameter. This **Widrow's** learning law can be implemented locally by means of the following equation,

$$w_{ji}(l) = w_{ji}(l-1) + \eta\,(b_{lj} - \mathbf{w}_j^T(l-1)\mathbf{a}_l)\,a_{li}, \tag{4.32}$$

where $\mathbf{w}_j(l-1)$ is the weight vector associated with the jth processing unit in the output layer of the linear associative network at the (1– 1)**th** iteration. With this scheme, it is **often** necessary to apply the pairs $(\mathbf{a}_l, \mathbf{b}_l)$ of the training set data several times, with each pair chosen at random.

The convergence of the **Widrow's** learning law in Eq. (4.32) depends on the choice of the learning rate parameter $\eta$. For sufficiently low values of $\eta$, a linear associative network can adaptively form only an approximation to the desired weight matrix $W = BA^+$. There is no known method for adaptively learning the best choice of the weight matrix W = $BA^+$. Note also that no method is known to adaptively learn even an approximation to the best choice of the weight matrix W = $B\hat{A}^+$ in the case of additive noise in the input pattern vectors. Therefore, in the case of noisy patterns, the best weight matrix has to be computed using the expressions for $\hat{\mathbf{A}}^+$ in terms of the components of the singular value decomposition of **A,** depending on the estimated noise level in the input patterns. This is obvious from the fact that noise effects can be reduced only when its statistics are observed over several patterns.

### 4.2.4 Discussion on Pattern Association Problem

Table 4.2 gives a summary of the results of linear associative networks.

Table 4.2 Summary of Results of Linear Associative Networks

Pattern association **problem**

Given a set $\{(\mathbf{a}_l, \mathbf{b}_l)\}$ of $L$ pattern pairs, the objective is to determine the weights of a linear associative network so as to minimize the error between the desired and actual outputs. If A = $[\mathbf{a}_1\ \mathbf{a}_2 \ldots \mathbf{a}_L]$, $B = (\mathbf{b}_1\ \mathbf{b}_2 \ldots \mathbf{b}_L]$ and Ware the input, output and weight matrices, respectively, then the optimum weights are given by

| | | |
|---|---|---|
| (a) | W = $BA^T$ | for orthogonal set of input vectors |
| (b) | W = $BA^{-1}$ | for linearly independent set of input vectors (full rank square matrix: $r = L = M$) |
| (c) | W = $BA^+$ | for linearly independent set of input vectors (full rank matrix: $r = L < M$) |
| (d) | W = $BA^+$ | for linearly dependent set of input vectors (reduced rank: $r < L \le M$) |
| (e) | W = $B\hat{A}^+$ | for noisy input vectors |

For the cases (a), (b) and (c), the minimum error is zero. For the case (d) the minimum error is determined by the rank of the input matrix. For the case (e) the minimum error is determined by both the rank of the input matrix and the noise power.

Determination of weights by learning

(a) For orthogonal input vectors the optimum weights W = $BA^T$ can be obtained using Hebb's learning law.

(b) For linearly independent or dependent input vectors an approximation to the optimum weights W = $BA^+$ can be learnt using a form of Widrow's learning law.

(c) For noisy input vectors there is no known learning law that can provide even an approximation to the optimum weights W = $B\hat{A}^+$.

It is often useful to allow the processing units in the output layer of the network to have a bias input. In such a case the input matrix $\boldsymbol{A}$ to this layer is augmented with an additional column vector, whose values are always $-1$. Addition of this bias term results in a weight matrix W that performs an ***affine transformation.*** With the affine transformation, any arbitrary rotation, scaling and translation operation on patterns can be handled, whereas linear transformations of the previous associative network can carry out only arbitrary rotation and scaling operations on the input patterns [Hecht-Nielsen, 19901.

In many applications the linkage between the dimensionality ($M$) of the input data and the number ($L$) of data items that can be associated and recalled is an unacceptable restriction. By means of coding schemes, the dimeriionality of the input data can sometimes be increased artificially, thus allowing more ($L > M$) pairs of items to be associated [Pao, 19891.

But, as we will see in the next section, the dependence of the number of input patterns on the dimensionality of the pattern vector can be removed completely by using nonlinear processing units in the output layer. Thus the ***artificial neural networks*** can capture the association among the pairs $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, ..., L$, even when the number of input patterns is greater than the dimensionality of the input vectors, i.e., $L > M$. While the constraint of dimensionality on the number of input patterns is removed in the artificial neural networks, some other restrictions will be placed which involve the functional relation between an input and the corresponding output. In particular, the implied mapping between the input and output pattern pairs can be captured by a two layer artificial neural network, provided the mapping function belongs to a linearly separable class. But the number of linearly separable functions decrease rapidly as the dimensionality of the input and output pattern vectors increases. These issues will be discussed in the following section.

## 4.3 Analysis of Pattern Classification Networks

In an $M$-dimensional space if a set of points could be considered as input patterns, and if an output pattern, not necessarily distinct from one another, is assigned to each of the input patterns, then the number of distinct output patterns can be viewed as distinct classes or class labels for the input patterns. There is no restriction on the number of input patterns. The input-output pattern vector pairs $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, ..., L$, in this case can be considered as a training set for ***a pattern classification*** problem. Typically, for pattern classification problems, the output patterns are points in a discrete (normally binary) N-dimensional space. The input patterns are usually from natural sources like speech and hand-printed characters. The input patterns may be corrupted by external noise. Even a noisy input will be mapped onto one of the distinct pattern classes, and hence the recall displays an accretive behaviour.

### 4.3.1 Pattern Classification Network: Perceptron

**A** two layer **feedforward** network with nonlinear **(hard-limiting)** output functions for the units in the output layer can be used to perform the task of pattern classification. The number of units in the input layer corresponds to the dimensionality of the input pattern vectors. The units in the input layer are all linear, as the input layer merely contributes to fan out the input to each of the output units. The number of output units depends on the number of distinct classes in the pattern classification task. We assume for this discussion that the output units are binary. Each output unit is connected to all the input units, and a weight is associated with each connection. Since the output function of a unit is a hard-limiting threshold function, for a given set of input-output patterns, the weighted sum of the input values is compared with the threshold for the unit to determine whether the sum is greater or less than the threshold. Thus in this case a set of inequalities are generated with the given data. Thus there is no unique solution for the weights in this case, as in the case of linear associative network. It is necessary to determine a set of weights to satisfy all the inequalities. Determination of such weights is usually. accompanied by means of incremental adjustment of the weights using a learning law.

**A** detailed analysis of pattern classification networks is presented here assuming M input units and a single binary output unit. The output unit uses a hard-limiting threshold function to decide whether the output signal should be 1 or 0. **Typically,** if the weighted sum of the input values to the output unit exceeds the threshold, the output signal is labelled as **1,** otherwise as 0. Extension of the analysis for a network consisting of multiple binary units in the output layer is trivial [**Zurada**, 19921. Multiple binary output units are needed if the number of pattern classes exceeds 2.

**Pattern classification problem:** If a subset of the input patterns belong to one class (say class A,) and the remaining subset of the input patterns to another class (say class A,), then the objective in a pattern classification problem is to determine a set of weights $w_1, w_2, ..., w_M$ **such** that if the weighted sum

$$\sum_{i=1}^{M} w_i a_i > 8, \text{ then } \mathbf{a} = (a_1, a_{,,} ..., a_M)^T \text{ belongs to class A,} \quad (4.33)$$

and if

$$\sum_{i=1}^{M} w_i a_i \leq 8, \text{ then } \mathbf{a} = (a_1, a_{,,} ..., a_M)^T \text{ belongs to class A,} \quad (4.34)$$

Note that the dividing surface between the two classes is given by

$$\sum_{i=1}^{M} w_i a_i = \theta \tag{4.35}$$

This equation represents a linear hyperplane in the $M$-dimensional space. The hyperplane becomes a point if M = 1, a straight line if M = 2, and a plane if M = 3.

Since the solution of the classification problem involves determining the weights and the threshold value, the classification network can be depicted as shown in Figure 4.2, where the input $a_0$

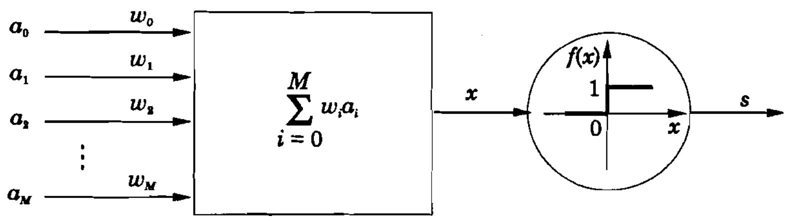


**Figure 4.2** A single unit pattern classification network (perceptron).

to the connection involving the threshold value $w_0 = \theta$ is always –1. Defining the augmented input and weight vectors as $a = (-1, a_1, ..., a_M)^T$ and $w = (w_0, w_1, ..., w_M)^T$, respectively, the perceptron classification problem can be stated as follows:

If $w^T a > 0$, then a belongs to class $A_1$, and
if $w^T a \leq 0$, then a belongs to class $A_2$.

The equation for the dividing linear hyperplane is $w^T a = 0$.

**Perceptron learning law:** In the above perceptron classification problem, the input space is an $M$-dimensional space and the number of output patterns are two, corresponding to the two classes. Note that we use the $(M + 1)$-dimensional vector to denote a point in the M-dimensional space, as the $a_0$ component of the vector is always –1. Suppose the subsets $A_1$ and $A_2$ of points in the M-dimensional space contain the sample patterns belonging to the classes $A_1$ and $A_2$, respectively. The objective in the perceptron learning is to systematically adjust the weights for each presentation of an input vector belonging to $A_1$ or $A_2$ along with its class identification. The perceptron learning law for the two-class problem may be stated as follows:

$$\begin{aligned} \mathbf{w}(m+1) &= \mathbf{w}(m) + \eta\, a, \text{ if } a \in A_1 \text{ and } w^T(m)a \leq 0 \\ &= \mathbf{w}(m) - \eta\, a, \text{ if } a \in A_2 \text{ and } w^T(m)a > 0 \end{aligned} \tag{4.36}$$

where the index m is used to denote the learning process at the mth step. The vectors a and $\mathbf{w}(m)$ are the input and weight vectors, respectively, at the mth step, and $\eta$ is a positive learning rate parameter. $\eta$ can be varying at each learning step, although it is assumed as constant in the perceptron learning. Note that no adjustment of weights is made when the input vector is correctly classified. That is,

$$\begin{aligned} \mathbf{w}(m+1) &= \mathbf{w}(m), \text{ if } \mathbf{a} \in A_1 \text{ and } \mathbf{w}^T(m)\mathbf{a} > 0 \\ &= \mathbf{w}(m), \text{ if } \mathbf{a} \in A_2 \text{ and } \mathbf{w}^T(m)\mathbf{a} \leq 0 \end{aligned} \qquad (4.37)$$

The initial value of the weight vector $\mathbf{w}(0)$ could be random. Figure 4.3 shows an example of the decision boundaries at different

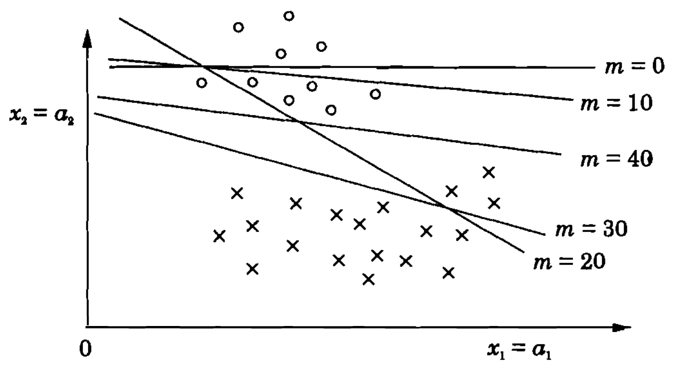


**Figure 4.3 Illustration of decision boundaries formed during implementation of perceptron learning for linearly separable classes.**

times for a 2-dimensional input vector. The equation of the straight line is given by

$$w_1 a_1 + w_2 a_2 = \theta \qquad (4.38)$$

For different values of the weights during learning, the position of the line changes. Note that in this example the two classes can be separated by a straight line, and hence they are called linearly separable classes. On the other hand consider the example of the pattern classification problem in Figure 4.4. In this case the straight line wanders in the plane during learning, and the weights do not converge to a final stable value, as the two classes cannot be separated by a single straight line.

**Perceptron convergence theorem:** This theorem states that the perceptron learning law converges to a final set of weight values in a finite number of steps, if the classes are linearly separable. The proof of this theorem is as follows:

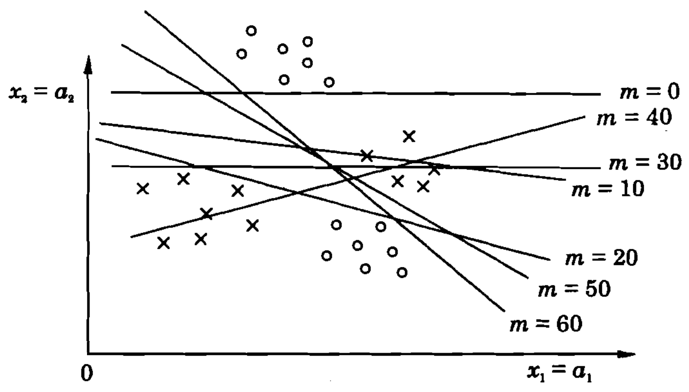


**Figure 4.4 Illustration of decision boundaries formed during implementation of perceptron learning for linearly inseparable classes.**

Let $a$ and $w$ be the augmented input and weight vectors, respectively. Assuming that there exists a solution $w^*$ for the classification problem, we have to show that $w^*$ can be approached in a finite number of steps, starting from some initial random weight values. We know that the solution $w^*$ satisfies the following inequality as per the Eq. *(4.37)*:

$$\mathbf{w}^{*T}\mathbf{a} > \alpha > 0, \quad \text{for each } a \in A_1 \qquad (4.39)$$

where

$$\alpha = \min_{\mathbf{a} \in A_1} (\mathbf{w}^{*T}\mathbf{a})$$

The weight vector is updated if $w^T(m)a \leq 0$, for $a \in A_1$. That is,

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta\, \mathbf{a}(m), \quad \text{for } \mathbf{a}(m) = a \in A_1, \qquad (4.40)$$

where $\mathbf{a}(m)$ is used to denote the input vector at step $m$. If we start with $\mathbf{w}(0) = \mathbf{0}$, where $\mathbf{0}$ is an all zero column vector, then

$$\mathbf{w}(m) = \eta \sum_{i=0}^{m-1} \mathbf{a}(i) \qquad (4.41)$$

Multiplying both sides of Eq. *(4.41)* by $\mathbf{w}^{*T}$, we get

$$\mathbf{w}^{*T}\mathbf{w}(m) = \eta \sum_{i=0}^{m-1} \mathbf{w}^{*T}\mathbf{a}(i) > \eta m \alpha \qquad (4.42)$$

since $\mathbf{w}^{*T}\mathbf{a}(i) > \alpha$ according to Eq. *(4.39)*. Using the Cauchy-Schwartz inequality

$$\| w^{*T} \|^2 . \| \mathbf{w}(m) \|^2 \geq [\mathbf{w}^{*T}\mathbf{w}(m)]^2 \qquad (4.43)$$

we get from Eq. *(4.42)*

$$\| \mathbf{w}(m) \|^2 > \eta^2 m^2 \alpha^2 / \| \mathbf{w}^{*T} \|^2 \qquad (4.44)$$

We also have from Eq. *(4.40)*

$$\begin{aligned} \| \mathbf{w}(m+1) \|^2 &= (\mathbf{w}(m) + \eta\, \mathbf{a}(m))^T (\mathbf{w}(m) + \eta\, \mathbf{a}(m)) \\ &= \| \mathbf{w}(m) \|^2 + \eta^2 \| \mathbf{a}(m) \|^2 + 2\eta\, \mathbf{w}^T(m)\, \mathbf{a}(m) \\ &\leq \| \mathbf{w}(m) \|^2 + \eta^2 \| \mathbf{a}(m) \|^2 \end{aligned} \tag{4.45}$$

since for learning $w^T(m)a(m) \leq 0$ when $\mathbf{a}(m) \in A_1$. Therefore, starting from $\mathbf{w}(0) = \mathbf{0}$, we get from Eq. *(4.45)*

$$\| \mathbf{w}(m) \|^2 \leq \eta^2 \sum_{i=0}^{m-1} \| \mathbf{a}(i) \|^2 < \eta^2 m\, \beta \tag{4.46}$$

where $\beta = \max\limits_{\mathbf{a}(i) \in A_1} \| \mathbf{a}(i) \|^2$. Combining Eqs. *(4.44)*and (4.46), we obtain the optimum value of $m$ by solving

$$\frac{m^2 \alpha^2}{\| \mathbf{w}^{*T} \|^2} = \beta m \tag{4.47}$$

or,

$$m = \frac{\beta}{\alpha^2} \| \mathbf{w}^{*T} \|^2 = \frac{\beta}{\alpha^2} \| \mathbf{w}^* \|^2 \tag{4.48}$$

Since $\beta$ is positive, Eq. *(4.48)*shows that the optimum weight value can be approached in a finite number of steps using the perceptron learning law.

**Alternate proof of the convergence theorem:** Assume that a solution vector $w^*$ exists. Then using the following perceptron behaviour

$$\mathbf{w}^{*T}\mathbf{a} > \alpha > 0 \quad \text{for } a \in A_1 \tag{4.49}$$

and

$$\mathbf{w}^{*T}\mathbf{a} < -\alpha < 0 \quad \text{for } a \in A_2 \tag{4.50}$$

we can show that the magnitude of the cosine of the angle $\phi$ between the weight vectors $w^*$ and $\mathbf{w}(m)$ is given by

$$|\cos \phi| = \frac{|\mathbf{w}^{*T}\mathbf{w}(m)|}{\| \mathbf{w}^* \| \, \| \mathbf{w}(m) \|} > \frac{\sqrt{m}\,\alpha}{\| \mathbf{w}^* \| \sqrt{\beta}} \tag{4.51}$$

where

$$\alpha = \min_{\mathbf{a}} |\mathbf{w}^{*T}\mathbf{a}| \tag{4.52}$$

and

$$\beta = \max_{\mathbf{a}} \| \mathbf{a} \|^2 \tag{4.53}$$

Using the perceptron learning law in Eq. (4.36), and Eqs. *(4.49)* and (4.52), we get the following:

$$\begin{aligned}\mathbf{w}^{*T}\mathbf{w}(m+1) &= \mathbf{w}^{*T}(\mathbf{w}(m) + \eta\, \mathbf{a}(m)) \\ &> \mathbf{w}^{*T}\mathbf{w}(m) + qa, \text{ for } w^T(m)a(m) \leq 0 \qquad (4.54)\end{aligned}$$

Starting **from** $\mathbf{w}(0) = \mathbf{0}$, we get

$$\mathbf{w}^{*T}\mathbf{w}(m) > m\eta\alpha, \text{ for } w^T(m)a(m) \leq 0, \text{ and } \mathbf{a}(m) \in A_1 \qquad (4.55)$$

Likewise, using the perceptron learning law in Eq. (4.36), and Eqs. (4.50) and (4.52), we get

$$\begin{aligned}\mathbf{w}^{*T}\mathbf{w}(m+1) &= \mathbf{w}^{*T}(\mathbf{w}(m) - \eta\, \mathbf{a}(m)) \\ &< \mathbf{w}^{*T}\mathbf{w}(m) - \eta\alpha, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 0\end{aligned}$$

Starting from $\mathbf{w}(0) = \mathbf{0}$, we get

$$\mathbf{w}^{*T}\mathbf{w}(m) < -mqa, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 0, \text{ and } \mathbf{a}(m) \in A_2 \qquad (4.56)$$

That is

$$|\mathbf{w}^{*T}\mathbf{w}(m)| > mqa, \text{ for } w^T(m)a(m) > 0, \text{ and } \mathbf{a}(m) \in A_2 \qquad (4.57)$$

Therefore from Eqs. (4.55) and (4.57), we get

$$|\mathbf{w}^{*T}\mathbf{w}(m)| > mqa, \text{ for all } a \qquad (4.58)$$

Similarly, using Eq. (4.36), we get

$$\begin{aligned}\|\mathbf{w}(m+1)\|^2 &= (\mathbf{w}(m) + \eta\, \mathbf{a}(m))^T(\mathbf{w}(m) + \eta\, \mathbf{a}(m)) \\ &< \|\mathbf{w}(m)\|^2 + 2\eta\, \mathbf{w}^T(m)\mathbf{a}(m) + \eta^2\beta \\ &< \|\mathbf{w}(m)\|^2 + \eta^2\beta, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) \leq 0 \qquad (4.59)\end{aligned}$$

and

$$\begin{aligned}\|\mathbf{w}(m+1)\|^2 &= (\mathbf{w}(m) - \eta\, \mathbf{a}(m))^T(\mathbf{w}(m) - \eta\, \mathbf{a}(m)) \\ &< \|\mathbf{w}(m)\|^2 - 2\eta\, \mathbf{w}^T(m)\mathbf{a}(m) + \eta^2\beta \\ &< \|\mathbf{w}(m)\|^2 + \eta^2\beta, \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 0 \qquad (4.60)\end{aligned}$$

Starting from $\mathbf{w}(0) = \mathbf{0}$, we get for both (4.59) and (4.60)

$$\|\mathbf{w}(m)\|^2 < m\eta^2\beta \qquad (4.61)$$

Therefore, from Eqs. (4.51), (4.58) and (4.61), we get

$$1 \geq \frac{|\mathbf{w}^{*T}\mathbf{w}(m)|}{\|\mathbf{w}^*\|\,\|\mathbf{w}(m)\|} > \frac{\sqrt{m}\,\alpha}{\|\mathbf{w}^*\|\sqrt{\beta}} \qquad (4.62)$$

$$\frac{\sqrt{m}\,\alpha}{\|\mathbf{w}^*\|\sqrt{\beta}} < 1 \qquad (4.63)$$

$$m < \frac{\beta}{\alpha^2}\|\mathbf{w}^*\|^2 \qquad (4.64)$$

**Discussion on the convergence theorem:** The number of iterations for convergence depends on the relation between $\mathbf{w}(0)$ and $w^*$. Normally the initial value $\mathbf{w}(0)$ is set to 0. The initial setting of the weight values does not affect the proof of the perceptron convergence theorem.

The working of the perceptron learning can be viewed as follows: At the $(m + 1)$th iteration we have

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta\, \mathbf{a}(m), \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) \leq 0 \text{ and } \mathbf{a}(m) \in A_1 \tag{4.65}$$

From this we get

$$\mathbf{w}^T(m+1)\mathbf{a}(m) = \mathbf{w}^T(m)\mathbf{a}(m) + \eta\, \mathbf{a}^T(m)\mathbf{a}(m) \tag{4.66}$$

Notice that if $\mathbf{w}^T(m)\mathbf{a}(m) < 0$, then $\mathbf{w}^T(m+1)\mathbf{a}(m) > 0$, provided $\eta$ is chosen as the smallest positive real number ($< 1$) such that

$$\eta\, \mathbf{a}^T(m)\mathbf{a}(m) > |\mathbf{w}^T(m)\mathbf{a}(m)| \tag{4.67}$$

Thus the given pattern $\mathbf{a}(m)$ is classified correctly if it is presented to the perceptron with the new weight vector $\mathbf{w}(m + 1)$. The weight vector is adjusted to enable the pattern to be classified correctly.

The perceptron convergence theorem for the two class problem is applicable for both binary $\{0, 1\}$ and bipolar $\{-1, +1\}$ input and output data. By considering a two-class problem each time, the perceptron convergence theorem can be proved for a multiclass problem as well. The perceptron learning law and its proof of convergence are applicable for a single layer of nonlinear processing units, also called a single layer perceptron. Note that convergence takes place provided an optimal solution $w^*$ exists. Such a solution exists for a single layer perceptron, only if the given classification problem is ***linearly separable.*** In other words, the perceptron learning law converges to a solution only if the class boundaries are separable by linear hyperplanes in the M-dimensional input pattern space.

**Perceptron learning as gradient descent:** The perceptron learning law in Eq. (4.36) can also be written as

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta\, (b(m) - s(m))\, \mathbf{a}(m) \tag{4.68}$$

where $b(m)$ is the desired output, which for the binary case is given by

$$b(m) = 1, \text{ for } \mathbf{a}(m) \in A_1, \tag{4.69}$$

$$= 0, \text{ for } \mathbf{a}(m) \in A_2 \tag{4.70}$$

and $s(m)$ is the actual output for the input vector $\mathbf{a}(m)$ to the perceptron. The actual output is given by

$$s(m) = 1, \text{ if } \mathbf{w}^T(m)\mathbf{a}(m) > 0 \tag{4.71}$$

$$= 0, \text{ if } \mathbf{w}^T(m)\mathbf{a}(m) \leq 0 \tag{4.72}$$

From Eq. (4.68) we note that if $s(m) = b(m)$, then $\mathbf{w}(m+1) = \mathbf{w}(m)$, **i.e.**, no correction takes place. On the other hand, if there is an error, $s(m) \neq b(m)$, then the update rule given by (4.68) is same as the update rule given in Eq. (4.36).

Note that Eq. (4.68) is also valid for a bipolar output function, i.e., when $s(m) = f(\mathbf{w}^T(m)\mathbf{a}(m)) = \pm 1$. Therefore Eq. (4.68) can be written as

$$\mathbf{w}(m + 1) = \mathbf{w}(m) + \eta\, e(m)\, \mathbf{a}(m) \tag{4.73}$$

where $e(m) = b(m) - s(m)$ is the error signal. If we use the instantaneous correlation (product) between the output error $e(m)$ and the activation value $x(m) = \mathbf{w}^T(m)\mathbf{a}(m)$ as a measure of performance $E(m)$, then

$$E(m) = -e(m)\, x(m) = -e(m)\, \mathbf{w}^T(m)\mathbf{a}(m) \tag{4.74}$$

The negative derivative of $E(m)$ with respect to the weight vector $\mathbf{w}(m)$ can be defined as the negative gradient of $E(m)$ and is given by

$$-\frac{\partial E(m)}{\partial \mathbf{w}(m)} = e(m)\, \mathbf{a}(m) \tag{4.75}$$

Thus the weight update $\eta\, e(m)\, \mathbf{a}(m)$ in the perceptron learning in Eq. (4.73) is proportional to the negative gradient of the performance measure $E(m)$.

**Perceptron representation problem:** Convergence in the perceptron learning takes place only if the pattern classes are linearly separable in the pattern space. Linear separability requires that the convex hulls of the pattern sets of the classes are disjoint. A convex hull of a pattern set $A_0$ is the smallest convex set in $\mathcal{R}^M$ that contains $A_0$. A convex set is a set of points in the M-dimensional space such that a line joining any two points in the set lies entirely in the region enclosed by the set. For linearly separable classes, the perceptron convergence theorem ensures that the final set of weights will be reached in a finite number of steps. These weights define a linear hyperplane separating the two classes. But in practice the number of linearly separable functions will decrease rapidly as the dimension of the input pattern space is increased [Cameron, 1960; Muroga, 19711. Table 4.3 shows the number of linearly separable functions for a two-class problem with binary input patterns for different dimensions of the input pattern space. For binary pattern classification problems ($M$ = 2), there are 14 functions which are linearly separable. The problem in **Figure 4.5a** is one of the linearly separable functions. There are two functions which are linearly inseparable, one of which is shown in Figure **4.5b.** These linearly inseparable problems do not lead to convergence of weights through the perceptron learning law, indicating that these problems are not ***representable*** by a single layer of perceptron discussed so far.

**Table 4.3** . Number of Linearly Separable Functions for a Two-class Problem

| Dimension of input data M | Number of possible functions $2^{2^M}$ | Number of linearly separable functions |
|---|---|---|
| *1* | *4* | *4* |
| *2* | *16* | *14* |
| *3* | *256* | *104* |
| *4* | *65536* | *1882* |
| *5* | $\sim 4.3 \times 10^9$ | *94572* |
| *6* | $\sim 1.8 \times 10^{19}$ | *15028134* |

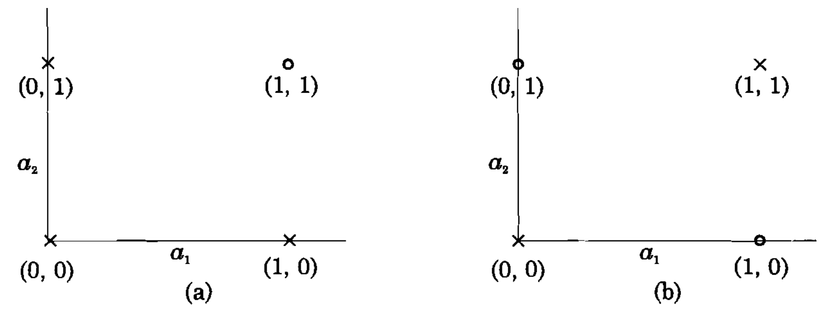


**Figure 4.5** Examples of (a) linearly separable and (b) linearly inseparable classification problems. The classes are indicated by 'x' and 'o'.

### 4.3.2 Linear Inseparability: Hard Problems

A two-layer feedforward network with hard-limiting threshold units in the output layer can solve linearly separable pattern classification problems. This is also called a single layer perceptron, as there is only one layer of nonlinear units. There are many problems which are not linearly separable, and hence are not representable by a single layer perceptron. These unrepresentable problems are called ***hard problems.*** Some of these problems are illustrated using the perceptron model consisting of sensory units, association units and the output layer as shown in Figure 4.6. The output unit of the perceptron computes a logical predicate, based on the information fed to it by the association units connected to it. The association units form a family of local predicates, computing a set of local properties or features. The family of local predicates are looking at a 'retina', which consists of points on a plane, which in the figure corresponds to the sensory input. In the simple 'linear' perceptron the output unit looks at the local predicates from the association units, takes their weighted sum, compares with a threshold, and then responds with a value for the overall logical predicate, which is either true or false. If it were

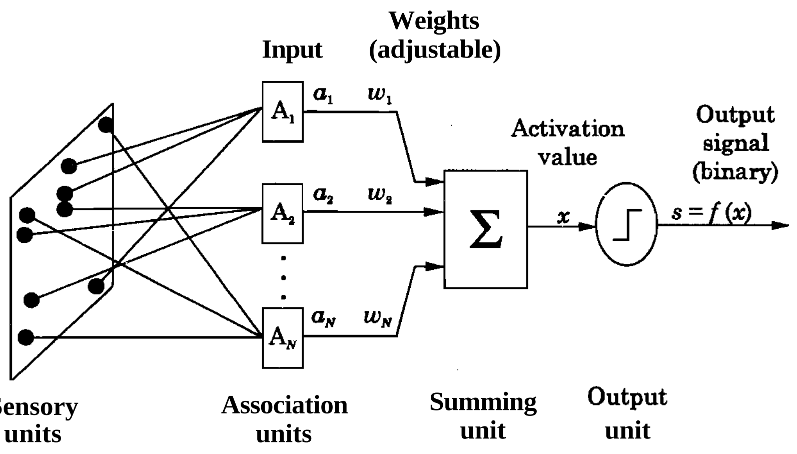


**Figure 4.6 Rosenblatt's perceptron model of a neuron.**

possible for the local predicates to look at every point in the entire retina, any possible logical predicate could be computed. This would be impractical, since the number of possible patterns on the retina grows exponentially with the size of the retina [Minsky and Papert, 19901. Two important limitations on the local predicates are: order-limited, where only a certain maximum order of retinal points could be connected to the local decision unit computing the local predicate, and diameter-limited, where only a geometrically restricted region of retina could be connected to the local predicate. The order-limited perceptron cannot compute the parity problem examples shown in Figure 4.7, where images with an even number of distinct

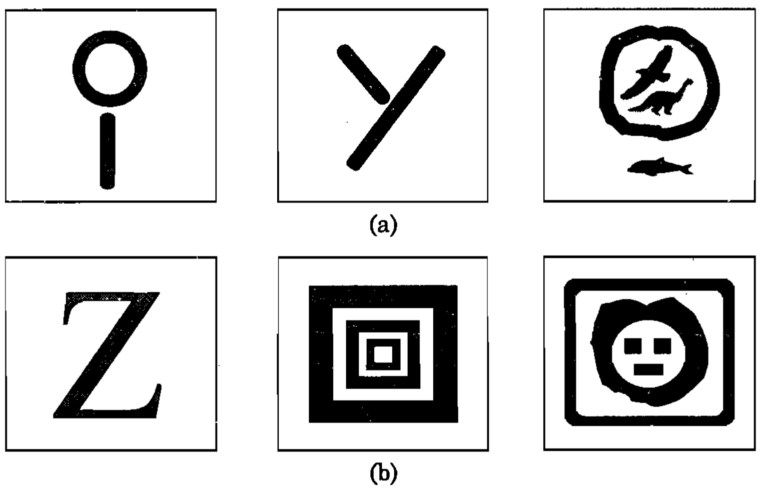


**Figure 4.7 A parity problem illustrating the order-limited perceptron: (a) Even parity and (b) Odd parity.**

unconnected patterns (Figure **4.7a**) should produce an output 1, otherwise the output for the odd number of patterns in Figure **4.7b** should be **0**. Likewise the diameter-limited perceptron cannot handle the connectedness problem examples shown in Figure 4.8, where to detect connectedness the output of the perceptron should be 1 if there is only one **connected** pattern as in Figure **4.8a**, otherwise it should be 0 for patterns shown in **Figure 4.8b [Aleksander** and Morton, 19901.

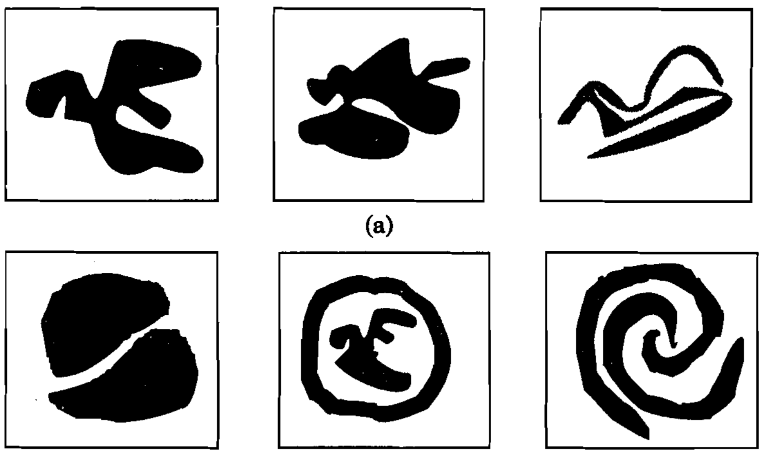
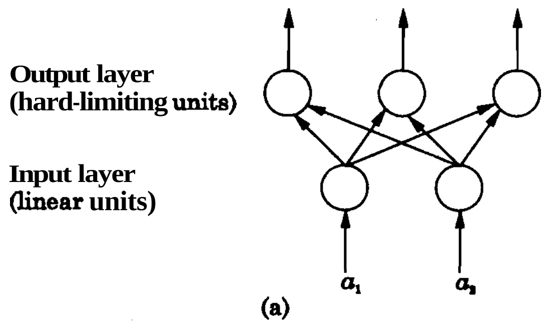


**Figure 4.8 A connectedness problem illustrating the diameter-limited perceptron: (a) Connected class and (b) Disconnected class.**

### 4.3.3 Geometrical Interpretation of Hard Problems: Multilayer Perceptron

In this section the problem of pattern classification and the performance of feedforward neural networks are discussed in geometric terms. A pattern classification problem can be viewed as determining the **hypersurfaces** separating the multidimensional patterns belonging to different classes. For convenience throughout this section we consider a **2-dimensional** pattern space. If the pattern classes are linearly separable then the hypersurfaces reduce to straight lines as shown in Figure 4.9. A two-layer network consisting of two input units and $N$ output units can produce $N$ distinct lines in the pattern space. These lines can be used to separate different classes, provided the regions formed by the pattern classification problem are linearly separable. As mentioned earlier (See Table **4.3**), linearly separable problems are in general far fewer among all possible problems, especially as the dimensionality of the input space increases. If the outputs of the second layer are combined by a set of units forming another layer, then it can be shown that any convex region can be formed by the separating surfaces **[Lippmann**, 1987; **Wasserman, 1989]**. A convex region is one in which a line joining any



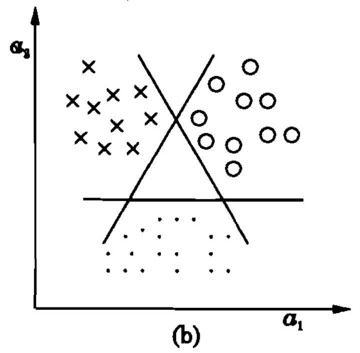


**Figure 4.9 An example of linearly separable classes: (a) Network and (b) Linearly separable classes.**

two points is entirely confined to the region itself. Figure **4.10b** illustrates the regions that can be created by a three layer network. In this case the number of units in the second layer determines the shape of the dividing surface. The number of units in the third layer decides the number of classes. It can be seen that the three-layer network (Fig. **4.10b**) is not general enough, as it is not guaranteed that the class regions in the pattern space form convex regions in all cases. In fad one could have a situation as shown for the classes with meshed regions, where the desired classes are enclosed by complicated nonconvex regions. Note that intersection of linear hyperplanes in the three layer network can only produce convex surfaces.

However, intersection of the convex regions may produce any nonconvex region also. Thus adding one more layer of units to combine the outputs of the third layer can yield surfaces which can separate even the nonconvex regions of the type shown in Figure **4.10c**. In fad it can be shown that a four-layer network with the input layer consisting of linear units, and the other three layers consisting of hard-limiting nonlinear units, can perform any complex pattern classification tasks. Thus all the hard problems mentioned earlier can be handled by a multilayer feedforward neural network, with nonlinear units. Such a network is also called a multilayer perceptron. Note that the two-layer network in Figure **4.10a** is a single-layer perceptron, and the three-layer and four-layer networks in the Figures **4.10b** and **4.10c** are two-layer and three-layer perceptrons, respectively.

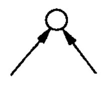
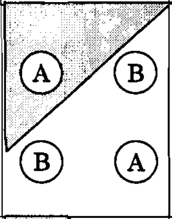
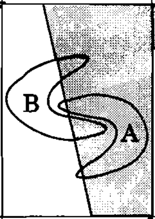
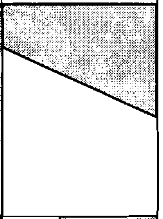
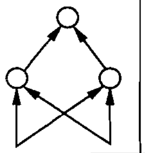
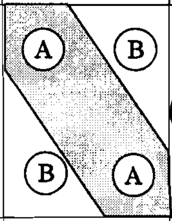
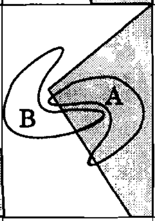
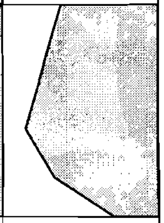
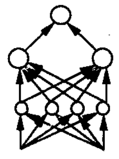
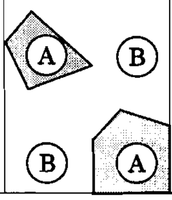
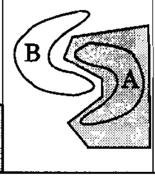
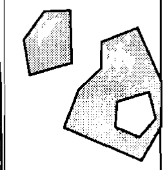
| Structure | Types of decision regions | Exclusive OR problem | Classes with meshed regions | Most general region shapes | |
|---|---|---|---|---|---|
| Two-layer network  | Half Plane (Bounded by hyperplane) |  |  |  | (a) |
| Three-layer network  | Convex (Open or closed regions) |  |  |  | (b) |
| Four-layer network  | Arbitrary (Complexity limited by number of neurons) |  |  |  | (c) |

**Figure 4.10 Geometrical interpretation of pattern classification. The figure shows decision regions for different layers of perceptron networks. [Adapted from Lippmann, 1987].**

The above discussion is focussed primarily on a multilayer perceptron network with units having hard-limiting nonlinear output functions. Similar behaviour is expected from a multilayer feedforward neural network when the output functions of the units are continuous nonlinear functions, such as sigmoid functions. In these cases the decision regions are typically bounded by smooth surfaces instead of linear hyperplanes, and hence geometrical visualization and interpretation is difficult.

Table 4.4 gives a summary of the discussion on the perceptron network. The main difficulty with a multilayer perceptron network is that it is not straightforward to adjust the weights leading to the units in the intermediate layers, since the desired output values of the units in these layers are not known. The perceptron learning uses the knowledge of the error between the desired output and the actual output to adjust the weights. From the given data only the desired

Table 4.4 Summary of Issues in Perceptron Learning

1. **Perceptron** network

   Weighted sum of the input to a unit with hard-limiting output function
2. **Perceptron** classification problem

   For a two class (A, and A,) problem, determine the weights (w) and threshold (8) such that

   $$\mathbf{w}^T\mathbf{a} - \theta > 0, \text{ for } \mathbf{a} \in A_1 \text{ and } \mathrm{w}^T\mathrm{a} - 850, \text{ for } \mathbf{a} \in A_2.$$
3. **Perceptron** learning law

   The weights are determined in an iterative manner using the following learning law at the $(m + 1)^{\text{th}}$ iteration:

   $$\mathbf{w}(m+1) = \mathbf{w}(m) + \eta\, \mathbf{a}(m), \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) \le 8 \text{ and } \mathbf{a}(m) \in A_1$$
   $$= \mathbf{w}(m) - \eta\, \mathbf{a}(m), \text{ for } \mathbf{w}^T(m)\mathbf{a}(m) > 8 \text{ and } \mathbf{a}(m) \in A_2$$

   where $\eta$ is a (positive) learning rate parameter.
4. **Perceptron** learning as gradient descent

   The perceptron learning law can be rewritten as a single equation:

   $$\mathbf{w}(m + 1) = \mathbf{w}(m) + \eta\, e(m)\, \mathbf{a}(m), \text{ where } e(m) = b(m) - s(m).$$

   Denoting

   $$\Delta\mathbf{w}(m) = \eta\, e(m)\, \mathbf{a}(m),$$

   we have

   $$\Delta\mathbf{w}(m) = \eta \frac{-\partial E(m)}{\partial \mathbf{w}(m)}$$

   where $E(m) = -e(m)\, \mathbf{w}^T(m)\mathbf{a}(m$
5. **Perceptron** convergence theorem

   The perceptron learning law converges in a finite number of steps, provided that the given classification problem is representable.
6. Perceptron representation problem

   A classification problem is representable by a single layer perceptron if the classes are linearly separable, i.e., separable by linear hyperplanes in the input feature space. Classification problems that are not linearly separable are called hard problems.
7. **Multilayer** perceptron

   Any pattern classification prohlem, including the hard problems, can be represented by a multilayer perceptron network.

output values of the units in the final output layer are known. Thus, although a multilayer perceptron network can handle hard problems, the problem of learning or training such a network, called ***hard learning*** problem, remains. This problem is discussed in detail in the following section.

## 4.4 Analysis of Pattern Mapping Networks

### 4.4.1 Pattern Mapping Problem

If a set of input-output pattern pairs is given corresponding to an arbitrary function transforming a point in the M-dimensional input pattern space to a point in the N-dimensional output pattern space, then the problem of capturing the implied functional relationship is called a ***mapping*** problem. The network that accomplishes this task is called a mapping network. Since no restriction such as linear separability is placed on the set of input-output pattern pairs, the pattern mapping problem is a more general case of pattern classification problem.

Note that the objective in the pattern mapping problem is to capture the implied function, i.e., the generalization implied in the given input-output pattern pairs. This can also be viewed as an approximation of the function from a given data. For a complex system with multiple ($M$) inputs and multiple ($N$) outputs,'if several input-output pattern pairs are collected during experimentation, then the objective in the pattern mapping problem is to capture the system characteristics from the observed data. For a new input, the captured system is expected to produce an output close to the one that would have been obtained by the real system. In terms of function approximation, the approximate mapping system should give an output which is close to the values of the function for inputs close to the current input used during learning. Note that the approximate system does not produce strictly an interpolated output, as the function finally captured during learning may not fit any of the points given in the training set. This is illustrated in Figure 4.11.

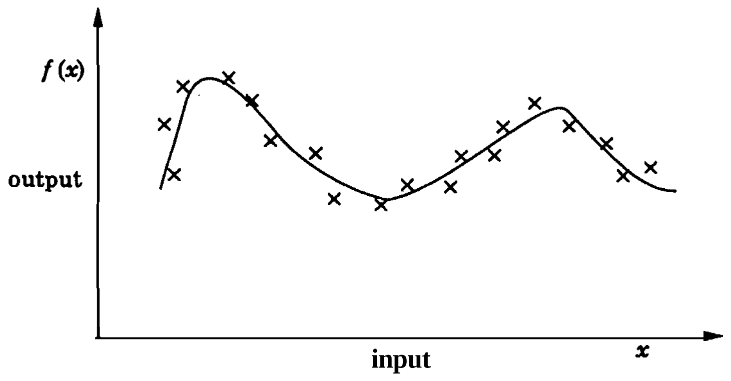


**Figure 4.11 Function approximation in pattern mapping problem.**

### 4.4.2 Pattern Mapping Network

Earlier we have seen that a multilayer feedforward neural network with at least two intermediate layers in addition to the input and output layers can perform any pattern classification task. Such a network can also perform a pattern mapping task. The additional layers are called hidden layers, and the number of units in the hidden layers depends on the nature of the mapping problem. For any arbitrary problem, generalization may be difficult. With a sufficiently large size of the network, it is possible to (virtually) store all the input-output pattern pairs given in the training set. Then the network will not be performing the desired mapping, because it will not be capturing the implied functional relationship between the given input-output pattern pairs.

Except in the input layer, the units in the other layers must be nonlinear in order to provide generalization capability for the network. In fact it can be shown that, if all the units are linear, then a multilayer network can be reduced to an equivalent two-layer network with a set of $N \times M$ weights.

Let $W_1$, $W_2$ and $W_3$ be the weight matrices of appropriate sizes between the input layer and the first hidden layer, the first hidden layer and the second hidden layer, and the second hidden layer and the output layer, respectively. Then if all the units are linear, the output and input patterns are related by the weight matrix containing $N \times M$ weight elements. That is,

$$W_{N \times M} = W_3 W_2 W_1 \tag{4.76}$$

As can be seen easily, such a network reduces to a linear associative network. But if the units in the output layer are nonlinear, then the network is limited by the linear separability constraint on the function relating the input-output pattern pairs. If the units in the hidden layers and in the output layer are nonlinear, then the number of unknown weights depend on the number of units in the hidden layers, besides the number of units in the input and output layers. The pattern mapping problem involves determining these weights, given a training set consisting of input-output pattern pairs. We need a systematic way of updating these weights when each input-output pattern pair is presented to the network. In order to do this updating of weights in a supervisory mode, it is necessary to know the desired output for each unit in the hidden and output layers. Once the desired output is known, the error, i.e., the difference between the desired and actual outputs from each unit may be used to guide the updating of the weights leading to the unit from the units in the previous layer. We know the desired output only for the units in the final output layer, and not for the units in the hidden layers. Therefore a straightforward application of a learning rule, that depends on the difference between the desired and the actual outputs, is not feasible in this case. The problem of updating the weights in this case is called a hard learning problem.

The hard learning problem is solved by using a differentiable nonlinear output function for each unit in the hidden and output layers. The corresponding learning law is based on propagating the error from the output layer to the hidden layers for updating the weights. This is an error correcting learning law, also called the generalized delta rule. It is based on the **principle** of gradient descent along the error surface.

Appendix C gives the background information needed for understanding the gradient descent methods. Table 4.5 gives a summary of the gradient search methods discussed in the Appendix C. In the following section we derive the generalized delta rule applicable for a **multilayer** feedforward network with nonlinear units.

**Table 4 5** Summary of Basic Gradient Search Methods

1. **Objective**

Determine the optimal set of weights for which the expected error $E(\mathbf{w})$ between the desired and actual outputs is minimum.

For a linear network the error surface is a quadratic function of the weights

$$E(\mathbf{w}) = \mathcal{E}[e^2] = E_{\min} + (\mathbf{w}-\mathbf{w}^*)^T R(\mathbf{w}-\mathbf{w}^*)$$

The **optimum weight** vector $\mathbf{w}^*$ is given by

$$\mathbf{w}^* = \mathrm{w} - \tfrac{1}{2}R^{-1}\nabla, \text{ where } \nabla = d E/d\mathbf{w}$$

and R is the autocorrelation matrix of the input data.

2. **Gradient Search Methods**

We can write the equation for adjustment of weights as

$$\mathbf{w}(m+1) = \mathbf{w}(m) - \tfrac{1}{2}R^{-1}\nabla_m$$

- If R and $\nabla_m$ are known exactly, then the above adjustment **gives** $\mathbf{w}^*$ in one step starting from any initial weights $\mathbf{w}(m)$.
- If R and $\nabla_m$ are known only approximately, then the optimum weight vector can be obtained in an iterative manner by writing

$$\mathbf{w}(m+1) = \mathbf{w}(m) - \eta R^{-1}\nabla_m,$$

where $\eta < 1/2$ for convergence. This is Newton's method. The error moves approximately along the path from $\mathbf{w}(m)$ to $\mathbf{w}^*$. Here $\eta$ is a dimensionless quantity.

- If the weights are adjusted in the direction of the negative gradient at each step, it becomes method of steepest descent.

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \mu(-\nabla_m),$$

where $\mu < 1/(2\lambda_{\max})$ for convergence and $\lambda_{\max}$ is the largest eigenvalue of R. The learning rate parameter $\mu$ has the dimensions of inverse of signal power. Here convergence is slower than in the Newton's method.

- In general, the gradient cannot be computed, but can only be estimated. Hence convergence of the gradient descent methods is not guaranteed. The estimate depends on our knowledge of the error surface.

Table 4.5 (Cont.)

**3. Nature of error surface**

- Error surface may not be quadratic if R is to be estimated from a small set of samples.
- Error surface is not quadratic for instantaneous measurement of error.
- Error surface is also not quadratic if the processing units are nonlinear.
- Error surface is not predictable for nonstationary input data, since R will be varying with time.

**4. Estimation of gradient**

- Derivative measurement: Uses general knowledge of the error surface.
- Instantaneous measurement (linear units): Uses specific knowledge of the error surface.

  ***LMS algorithm.*** Leads to convergence in the mean (stochastic gradient descent).
- Instantaneous measurement (nonlinear units): Uses specific knowledge of the error surface.

  ***Delta rule.*** No guarantee of convergence even in the mean as in the LMS algorithm.

### 4.4.3 Generalized Delta Rule: Backpropagation learning

The objective is to develop a learning algorithm for a multilayer feedforward neural network, so that the network can be trained to capture the mapping implicit in the given set of input-output pattern pairs. The approach to be followed is basically a gradient descent along the error surface to arrive at the optimum set of weights. The error is defined as the squared difference between the desired output (i.e., given output pattern) and the actual output obtained at the output layer of the network due to application of an input pattern from the given input-output pattern pair. The output is calculated using the current setting of the weights in all the layers. The optimum weights may be obtained if the weights are adjusted in such a way that the gradient descent is made along the total error surface. But the desirable characteristic of any learning law is to specify the incremental update of the weights of the network for each presentation of an input-output pattern pair. While this may result in a suboptimal solution, in most cases of practical significance the result is acceptable.

A learning law, called generalized delta rule or backpropagation law, is derived in this section [Werbos, 1974; Rumelhart et al, 1986a]. Let $(\mathbf{a}_l, \mathbf{b}_l), l = 1, 2, ..., L$ be the set of training pattern pairs. It is not necessary to have all the training data set at one time, nor the training data set to be a finite set. The objective is to determine the weight update for each presentation of an input-output pattern pair.

Since the given data may be used several times during training, let us use the index m to indicate the presentation step for the training pair at step m.

For training a multilayer feedforward neural network, we use the following estimate of the gradient descent along the error surface to determine the increment in the weight connecting the units j and i:

$$\Delta w_{ij}(m) = -\eta \frac{\partial E(m)}{\partial w_{ij}}, \tag{4.77}$$

where $\eta > 0$ is a learning rate parameter, which may also vary for each presentation of the training pair. The weight update is given by

$$w_{ij}(m+1) = w_{ij}(m) + \Delta w_{ij}(m) \tag{4.78}$$

The generalized delta rule to be derived below consists of deriving expressions for $\Delta w_{ij}$ for the connections at different layers. Let us consider the multilayer **feedforward** neural network given in **Figure 4.12.** The network consist of three layers of units, the first

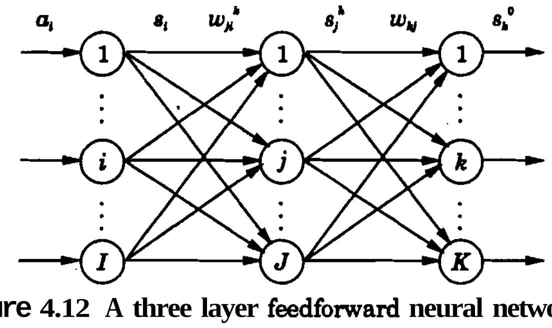


**Figure 4.12 A three layer feedforward neural network.**

layer has I linear input units indexed by i, the second layer has J nonlinear units indexed by j, and the third layer has $K$ nonlinear units indexed by $k$. For simplicity only one layer (the second layer) of hidden units is considered here. Extension of learning to a network consisting of several hidden layers is trivial.

Since the input vector $\mathbf{a}(m)$ is given at the input layer and the desired output $\mathbf{b}(m)$ is available only at the output layer, the error between the desired output vector $\mathbf{b}(m)$ and the actual output vector $\mathbf{b}'(m)$ is available only at the output layer. Using this error it is necessary to adjust the **weights** $(w_{ji}^h)$ from the input units to the hidden units, and the weights $(w_{kj})$ from the hidden units to the output units.

Let $(\mathbf{a}(m), \mathbf{b}(m))$ be the current sample of the function mapping the input space to the output space $\mathcal{R}^I \rightarrow \mathcal{R}^K$. Let $\mathbf{b}'(m)$ be the actual output of the network for the input $\mathbf{a}(m)$ at the step m. The mean squared error at the mth step is given by

$$E(m) = \frac{1}{2}\sum_{k=1}^{K}[b_k(m) - b_k'(m)]^2 = \frac{1}{2}\sum_{k=1}^{K}[b_k(m) - s_k^o]^2 \tag{4.79}$$

$$= \frac{1}{2}\sum_{k=1}^{K}[b_k(m) - f_k^o(x_k^o)]^2, \tag{4.80}$$

where

$$x_k^o = \sum_{j=1}^{J} w_{kj}\, s_j^h \tag{4.81}$$

$$s_j^h = f_j^h(x_j^h) \tag{4.82}$$

$$x_j^h = \sum_{i=1}^{I} w_{ji}^h\, s_i \tag{4.83}$$

$$s_i = x_i = a_i(m). \tag{4.84}$$

The superscript '*o*' refers to the output units quantities, the superscript '*h*' refers to the hidden units quantities, and $a_i$, $x_i$, and $s_i$ refer to the input, activation and output values for the unit i, respectively. For the weights leading to the units in the output layer:

$$\Delta w_{kj}(m) = -\eta\frac{\partial E(m)}{\partial w_{kj}} \tag{4.85}$$

$$\frac{\partial E(m)}{\partial w_{kj}} = \frac{1}{2}\frac{\partial}{\partial w_{kj}}\left[b_k - f_k^o\left(\sum_{j=1}^{J} w_{kj}\, s_j^h\right)\right]^2 \tag{4.86}$$

$$= -(b_k - f_k^o)\,\dot{f}_k^o\, s_j^h \tag{4.87}$$

$$= -\delta_k^o\, s_j^h, \tag{4.88}$$

where $\delta_k^o = (b_k - f_k^o)\,\dot{f}_k^o$. Here the iteration index is omitted in all the functions and variables on the right hand side for convenience. Therefore

$$\Delta w_{kj}(m) = \eta\delta_k^o s_j^h \tag{4.89}$$

and

$$w_{kj}(m+1) = w_{kj}(m) + \Delta w_{kj}(m) \tag{4.90}$$

$$= w_{kj}(m) + \eta\delta_k^o s_j^h. \tag{4.91}$$

For the weights leading to the units in the hidden layer:

$$\Delta w_{ji}^h(m) = -\eta\frac{\partial E(m)}{\partial w_{ji}^h} \tag{4.92}$$

$$\frac{\partial E(m)}{\partial w_{ji}^h} = -\sum_{k=1}^{K}(b_k - f_k^o)\,\frac{\partial f_k^o\left(\sum_{j=1}^{J} w_{kj}\, s_j^h\right)}{\partial w_{ji}^h} \qquad (4.93)$$

$$= -\sum_{k=1}^{K}(b_k - f_k^o)\,\dot{f}_k^o\, w_{kj}\,\frac{\partial s_j^h}{\partial w_{ji}^h}, \qquad (4.94)$$

Since $s_j^h = f_j^h(x_j^h)$, we get

$$\frac{\partial s_j^h}{\partial w_{ji}^h} = \dot{f}_j^h\,\frac{\partial x_j^h}{\partial w_{ji}^h}$$

Since $x_j^h = \sum_{i=1}^{I} w_{ji}^h\, s_i$, we get

$$\frac{\partial x_j^h}{\partial w_{ji}^h} = s_i$$

Therefore

$$\frac{\partial E(m)}{\partial w_{ji}^h} = -\sum_{k=1}^{K}(b_k - f_k^o)\,\dot{f}_k^o\, w_{kj}\,\dot{f}_j^h\, s_i \qquad (4.95)$$

$$= -\delta_j^h\, s_i \qquad (4.96)$$

where

$$\delta_j^h = \dot{f}_j^h \sum_{k=1}^{K} w_{kj}\,\delta_k^o \qquad (4.97)$$

Hence

$$\Delta w_{ji}^h(m) = \eta\,\delta_j^h\, s_i = \eta\,\delta_j^h\, a_i(m) \qquad (4.98)$$

since $s_i = x_i = a_i(m)$. Therefore

$$\begin{aligned} w_{ji}^h(m+1) &= w_{ji}^h(m) + \Delta w_{ji}^h(m) \\ &= w_{ji}^h(m) + \eta\,\delta_j^h\, a_i(m) \qquad (4.99) \end{aligned}$$

where $\delta_j^h = \dot{f}_j^h \sum_{k=1}^{K} w_{kj}\,\delta_k^o$ represents the error propagated back to the output of the hidden units from the next layer, hence the name *backpropagation* for this learning algorithm. Table 4.6 gives a summary of the backpropagation learning algorithm.

## 4.4.4 Discussion on Backpropagation Law

There are several issues which are important for understanding and implementing the backpropagation learning in practice [Haykin, 1994; Russo, 1991; Guyon, 1991; Hush and Horne, 1993; Werbos, 1994]. A summary of the issues is given in Table 4.7. A few of these issues will be discussed in this section.

Table 4.6 Backpropagation Algorithm (Generalized Delta Rule)

---

Given a set of input-output patterns $(\mathbf{a}_l, \mathbf{b}_l)$, $l = 1, 2, \ldots L$,
where the lth input vector $\mathbf{a}_l = (a_{l1}, a_{l2}, \ldots, a_{lI})^T$ and the **lth** output vector $\mathbf{b}_l = (b_{l1}, b_{l2}, \ldots, b_{lK})^T$.

Assume only one hidden layer and initial setting of weights to be arbitrary.

Assume input layer with only linear units.

Then the output signal is equal to the input activation value for each of these **units**. Let $\eta$ be the learning rate parameter.

Let $\quad a = \mathbf{a}(m) = \mathbf{a}_l$ and $b = \mathbf{b}(m) = \mathbf{b}_l$.

Activation of unit i in the input layer, $x_i = a_i(m)$

Activation of unit j in the hidden layer, $x_j^h = \sum_{i=1}^{I} w_{ji}^h x_i$

Output signal from the $j$**th** unit in the **hidden layer**, $s_j^h = f_j^h(x_j^h)$

Activation of unit k in the output layer, $x_k^o = \sum_{j=1}^{J} w_{kj} s_j^h$

Output signal from unit k in the output layer, $s_k^o = f_k^o(x_k^o)$

Error term for the kth output unit, 6; $= (b_k - s_k^o) \dot{f}_k^o$

Update weights on output layer, $w_{kj}(m+1) = w_{kj}(m) + \eta \delta_k^o s_j^h$

Error term for the jth hidden unit, $\delta_j^h = \dot{f}_j^h \sum_{k=1}^{K} \delta_k^o w_{kj}$

Update the weights on the hidden layer, $w_{ji}^h(m+1) = w_{ji}^h(m) + \eta \delta_j^h a_i$

Calculate the error for the lth pattern, $E_l = \frac{1}{2} \sum_{k=1}^{K} (b_{lk} - s_k^o)^2$

Total error for all patterns, $E = \sum_{l=1}^{L} E_l$

Apply the given patterns one by one, may be several times, in some random order and update the weights until the total error reduces to an acceptable value.

---

Table 4.7 Issues in Backpropagation Learning

---

Description **and** features of backpropagation

- Significance of error backpropagation
- Forward computation (inner product and nonlinear **function)**
- Backward operation (error calculation and derivative of output function)
- 'Nature of output function (semilinear)
- Stochastic gradient descent
- Local computations
- Stopping criterion

Performance of backpropagation learning

- Initialization of weights
- Presentation of training patterns: Pattern and batch **modes**

**Table 4.7** Issues **in** Backpropagation **Learning** (Cont.)

- Learning rate **parameter** $\eta$
  - Range and value of $\eta$ for stability and convergence
  - Learning rate adaptation for better convergence
- Momentum term for faster convergence
- Second order methods for better and faster convergence

**Refinement of backpropagation learning**

- Stochastic gradient descent, not an optimization method
- Nonlinear system identification: Extended Kalman-type algorithm
- Unconstrained optimization: Conjugate-gradient methods
- Asymptotic estimation of a ***posteriori*** class probabilities
- Fuzzy backpropagation learning

**Interpretation of results of learning**

- Ill-posed nature of solution
- Approximation of functions
- Good estimation of decision surfaces
- Nonlinear feature detector followed by linearly separable classification
- Estimation of a ***posteriori*** class probabilities

**Generalization**

- VC dimension
- Cross-validation
- Loading problem
- Size and efficiency of training set data
- Architectures of network
- Complexity of problem

**Tasks with backpropagation network**

- Logic function
- Pattern classification
- Pattern mapping
- Function approximation
- Probability estimation
- Prediction

**Limitations of backpropagation learning**

- Slow convergence (no proof of convergence)
- Local minima problem
- Scaling
- Need for teacher: Supervised learning

**Extensions to backpropagation**

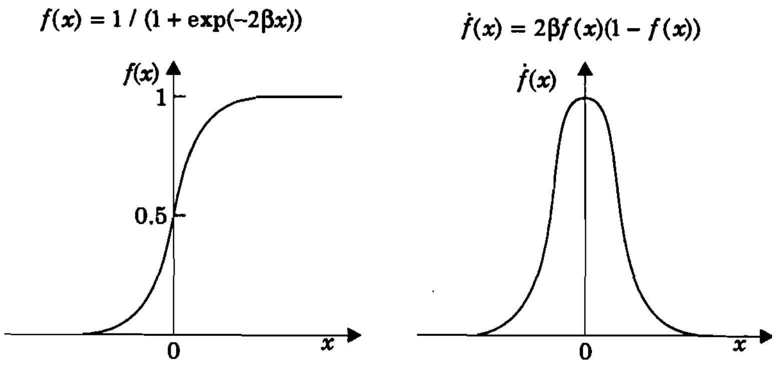
- Learning with critic
- Regularization
- Radial basis functions
- Probabilistic neural networks
- Fuzzy neural networks

**Description and features of backpropagation:** The training patterns are applied in some random order one by one, and the weights are adjusted using the backpropagation learning law. Each application of the training set patterns is called a cycle. The patterns may have to be applied for several training cycles to obtain the output error to an acceptable low value. Once the network is trained, it can be used to recall the appropriate pattern (in this case some interpolated output pattern) for a new input pattern. The computation for recall is straightforward, in the sense that the weights and the output functions of the units in different layers are used to compute the activation values and the output signals. The signals from the output layer correspond to the output.

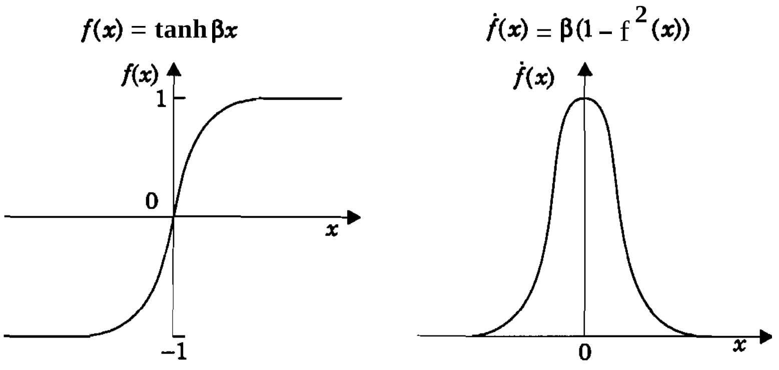
Backpropagation learning emerged as the most significant result in the field of artificial neural networks. In fact it is this learning law that led to the resurgence of interest in neural networks, nearly after 15 years period of lull due to exposition of limitations of the perceptron learning by Minsky and Papert (1969). In this section we will discuss various features including limitations of the backpropagation learning. We will also discuss the issues that determine the performance of the network resulting from the learning law. We will discuss these issues with reference to specific applications, and also with reference to some potential applications of the multilayer feedforward neural networks.

As noted earlier, the backpropagation learning involves propagation of 'the error backwards from the output layer to the hidden layers in order to determine the update for the weights leading to the units in a hidden layer. The error at the output layer itself is computed using the difference between the desired output and the actual output at each of the output units. The actual output for a given input training pattern is determined by computing the outputs of units for each hidden layer in the forward pass of the input data. Note that the error in the output is propagated backwards only to determine the weight updates. There is no feedback of the signal itself at any stage, as it is a feedforward neural network.

Since the backpropagation learning is exactly the same as the delta learning (see Section 1.6.3) at the output layer and is similar to the delta learning with the propagated error at the hidden layers, it is also called ***generalized delta rule.*** The term 'generalized' is used because the delta learning could be extended to the hidden layer units. Backpropagation of error is possible only if the output functions of the nonlinear processing units are differentiable. Note that if these output functions are linear, then we cannot realize the advantage of a multilayer network to generate complex decision boundaries for a nonlinearly separable (hard) classification problems. In fact a multilayer feedforward network with linear processing units is equivalent to a linear associative network, as discussed in Eq. (4.76), which, in turn, is limited to solving simple pattern association problems. On the other hand, hard-limiting output function as in a multilayer perceptron cannot be used for learning the weights. A common differentiable output function used in the backpropagation learning is one which possesses a sigmoid nonlinearity. Two examples of **sigmoidal** nonlinear function are the logistic function and hyperbolic tangent function (See Figure 4.13):



**(a) Logistic function and its derivative**



**(b) Hyperbolic tangent function and its derivative**

**Figure 4.13 Logistic and hyperbolic tangent functions and their derivatives for $\beta = 0.5$.**

Logistic function

$$f(x) = \frac{1}{1+e^{-x}}, \quad -\infty < x < \infty \tag{4.100}$$

Hyperbolic function

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad -\infty < x < \infty. \tag{4.101}$$

For the logistic function the limits are $0 \leq f(x)$ I 1, and for the hyperbolic tangent function the limits are $-1$ I $f(x)$ I 1.

Let us consider the derivative of the logistic function

$$\dot{f}(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)\,[1 - f(x)] \tag{4.102}$$

It can be seen from Eq. (4.102) that $\dot{f}(x)$ has the maximum value of 0.25 when $f(x) = 0.5$, and has the minimum value of 0 when $f(x) = 0$ or 1. Since the amount of change in the weight value leading to any unit $i$ in the network is proportional to $\dot{f}_i(x)$, the change is maximum in the midrange of the activation value. This feature of the learning law contributes to its stability [Rumelhart et al, 1986a].

Note that the hyperbolic tangent function can be viewed as a biased and scaled version of the logistic function. That is

$$a \tanh(bx) = \frac{2a}{1+e^{-bx}} - a. \tag{4.103}$$

The asymmetry of the hyperbolic tangent function seems to make the learning faster by reducing the number of iterations required for training [Guyon, 19911.

The backpropagation learning is based on the gradient descent along the error surface. That is, the weight adjustment is proportional to the negative gradient of the error with respect to the weight. The error is the instantaneous error between the desired and the actual values of the output of the network. This instantaneous error is due to a given training pattern, which can be assumed to be a sample function of a random process. Thus the error can be assumed to be a random variable. Therefore this gradient descent method is a stochastic gradient learning method. Due to this stochastic nature, the path to the minimum of the error surface will be zigzag. The error surface itself will be an approximation to the true error surface determined by the entire training set of patterns. Moreover, even the true error surface is not a smooth quadratic surface as in the case of the Adaline. In fact the error surface may contain several local minima besides the global minimum. Hence the stochastic approximation of the gradient descent used in the backpropagation learning need not converge. There is no proof of convergence even in the mean as in the case of the LMS algorithm. The issues in the convergence of gradient descent methods are summarized in Table 4.8.

Since there is no proof of convergence, some heuristic criteria are used to stop the process of learning. They are based on the values of the gradient and the error in successive iterations and also on the total number of iterations. The average gradient value over each training cycle (presentation of all the training patterns once) is observed, and if this average value is below a preset threshold value for successive cycles, then the training process may be stopped. Likewise, the training process may be stopped using a threshold for the average error and observing the average error in successive cycles.

**Table** 4.8 Gradient Descent and Convergence

1. Let the input-output vector pair (**a, b**) be the sample function of a random process.
   **True** ensemble average of the error
   $$E(\mathbf{w}) = \mathcal{E}[e^2(m)]$$
   where $e(m)$ is the instantaneous error for a given sample function. For linear units the error surface $E(\mathbf{w})$ is a smooth bowl-shaped in the weight space and hence the gradient descent $\partial E/\partial w_{ij}$ converges to the optimal weight vector $\mathbf{w}^*$.
2. Estimation of the error from a finite set of input-output pairs:
   $$E(\mathbf{w}) = \sum_{m=1}^{M} e^2(m)$$
   For linear units, this error surface is an approximation to the bowl-shape in the weight space and hence convergence of the gradient descent is only approximate.
3. Instantaneous error (Linear units):
   $$E(\mathbf{w}) = e^2(m)$$
   For linear units, the gradient descent converges only in the mean (stochastic convergence)
4. Instantaneous error (Nonlinear units):
   $$E(\mathbf{w}) = e^2(m)$$
   For nonlinear units, there is no proof of convergence even in the stochastic sense.

Sometimes both the average gradient as well as the average error may be used in the stopping criterion. But the main objective is to capture the implicit pattern behaviour in the training set data so that adequate generalization takes place in the network. The generalization feature is verified by testing the performance of the network for several new (test) patterns.

**Performance of the backpropagation learning law:** The performance of the backpropagation learning law depends on the initial setting of the weights, learning rate parameter, output functions of the units, presentation of the training data, besides the specific pattern recognition task (like classification, mapping, etc.) or specific application (like function approximation, probability estimation, prediction, logic function, etc.). It is important to initialize the weight values properly before applying the learning law for a given training set [Hush et al, 1991; Lee et al, 19911. Initial weights correspond to a priori knowledge. If we have the knowledge and also if we know how to present the knowledge in the form of initial weights, then the overall performance of the resulting trained network in terms of speed of learning and generalization would improve significantly. In general it is not known how to collect the relevant knowledge a priori. The more difficult part is to know how to include it in the form of weighta. Therefore all the weights in the network are initialized to random numbers that are **uniformly** distributed in a small range of values. The range is typically $[-a/\sqrt{N_i}, +a/\sqrt{N_i}]$ where $N_i$ is the number of inputs to the ith unit. Thus the range can be different for each unit. The value of $a$ is typically in the range (1 to 3) [Wessels and Barnard, 19921. Initial weighta that are in very small range will result in long learning times. On the other hand, large initial weight values may result in the network output values in the saturation region of the output function. In the saturation region the gradient value is small. If the saturation is at the incorrect level, it may result in slow learning due to small changes made in the weighta in each iteration. Incorrect saturation rarely occurs if the unit operates in the linear range of the output function.

Adjustment of the weights using backpropagation learning law is done by presenting the given set of training patterns several times. Randomizing the presentation of these patterns tends to make the search in the weight space stochastic, and thus reduces the possibility of limit cycles' in the trajectory in the weight space during learning [Haykin, 1994, p. 151]. Presentation of the training data pattern by pattern for adjustment of the weights makes it possible to have the learning online. This pattern mode also reduces the problem of local minima. But to speed up the learning process it is preferable to update the weights in a batch mode, in which the gradient of the error, computed over all the training patterns, is used. The batch. mode gives a better estimation of the gradient of the overall error surface.

Learning rate parameter $\eta$ plays a crucial role in the backpropagation learning. The order of values for $\eta$ depends on the variance of the input data. For the case of Adaline, the learning rate parameter $\eta < 1/(2\lambda_{max})$, where $\lambda_{max}$ is the largest eigenvalue of the autocorrelation matrix of the input data. This gives an indication for the choice of $\eta$, since the derivation in the backpropagation does not suggest any clue for this choice. Since it is a stochastic gradient descent learning, too small an $\eta$ will result in a smooth trajectory in the weight space, but takes long time to converge. On the other hand, too large an $\eta$ may increase the speed of learning, but will result in large random fluctuations in the weight space, which in turn may lead to an unstable situation in the sense that the network weights may not converge.

It is desirable to adjust the weights in such a way that all the units learn nearly at the same rate. That is, the net change in all the weights leading to a unit should be nearly the same. To accomplish this, the learning rate parameters should be different for different weights. The weights leading to a unit with many inputs should have smaller $\eta$ compared to the $\eta$ for the weights leading to a unit with fewer inputs. Also, the gradient of the error with respect to the weights leading to the output layer will be larger than the gradient of the error with respect to the weights leading to the hidden layers. Therefore the learning rate parameters $\eta$ should be typically smaller for the weights at the output layer and larger for the weights leading to the units in the hidden layers. This will ensure that the net change in the weights remains nearly the same for all layers.

Better convergence in learning can be achieved by adapting the learning rate parameter $\eta$ suitably for each iteration. For this the change in $\eta$ is made proportional to the negative gradient of the instantapeous error with respect to $\eta$ [Haykin, 1994, p. 1951. That is

$$\Delta\eta_{ji}(m+1) = -\gamma\frac{\partial e^2(m)}{\partial\eta_{ji}(m)} \tag{4.104}$$

where $\gamma$ is a proportionality constant.

It was shown in [Haykin, 19941 that

$$\frac{\partial e^2(m)}{\partial\eta_{ji}(m)} = \frac{\partial e^2(m)}{\partial w_{ji}(m)}\frac{\partial e^2(m-1)}{\partial w_{ji}(m-1)} \tag{4.105}$$

This is called delta-delta learning rule [Jacobs, 19881. The change in the learning rate parameter depends on the instantaneous gradients at the previous two iterations. In this learning it is difficult to choose suitable values for the proportionality constant $\gamma$ if the magnitudes of the two gradients in the product are either too small or too large. To overcome this limitation a modification of the above learning rule, namely, delta-bar-delta learning rule was proposed [Jacobs, 1988; Minai and Williams, 19901.

The adaptation of the learning rate parameter using the delta-delta learning rule or the delta-bar-delta learning rule slows down the backpropagation learning significantly due to additional complexity in computation at each iteration. It is possible to reduce this complexity by using the idea of the gradient reuse method, in which the gradient estimate is obtained by averaging the gradient values corresponding to several training patterns. Thus

$$w_{ji}(m+1) = w_{ji}(m) + \eta_{ji}(m)\sum_{l=1}^{L}\delta_j^l(m)\, s_i^l(m) \tag{4.106}$$

where $l$ is the index for training pattern and $\delta_j^l(m)$ is the propagated error. The learning rate parameter $\eta_{ji}(m)$ is also computed using the averaged gradient for several training patterns.

The values of the learning rate parameters computed using any of the above methods are very low, thus resulting in slow learning.

One way to increase the rate of learning is by using a momentum term in the weight change as follows [Plaut et al, 1986; Fahlman, 1989; Rumelhart et al, 1986a]:

$$\Delta w_{ji}(m) = \alpha \, \Delta w_{ji}(m-1) + \eta \, \delta_j(m) \, s_i(m) \qquad (4.107)$$

where $0 \le \alpha < 1$ is the momentum constant. The use of the momentum term accelerates the descent to the minimum of the error surface. It will also help in reducing the effects of local minima of the error surface.

The expression for the updated weight which includes momentum term as well as the learning rate adaptation is given by

$$w_{ji}(m+1) = w_{ji}(m) + \alpha \, \Delta w_{ji}(m-1) + \eta_{ji}(m) \sum_{l=1}^{L} \delta_j^l(m) \, s_i^l(m) \qquad (4.108)$$

Normally the backpropagation learning uses the weight change proportional to the negative gradient of the instantaneous error. Thus it uses only the first derivative of the instantaneous error with respect to the weight. If the weight change is made using the information in the second derivative of the error, then a better estimate of the optimum weight change towards the minimum may be obtained. The momentum method is one such method where both the weight change at the previous step and the gradient at the current step are used to determine the weight change for the current step.

More effective methods [Battiti, 1992] can be derived starting with the following Taylor series expression of the error as a function of the weight vector

$$E(\mathbf{w} + \Delta\mathbf{w}) = E(\mathbf{w}) + \mathbf{g}^T \Delta\mathbf{w} + \frac{1}{2} \Delta\mathbf{w}^T H \Delta\mathbf{w} + \ldots \qquad (4.109)$$

where $\mathbf{g} = \frac{\partial E}{\partial \mathbf{w}}$ is the gradient vector, and $H = \frac{\partial^2 E}{\partial \mathbf{w}^2}$ the Hessian matrix. For small $\Delta\mathbf{w}$, the higher order terms can be neglected, so that we get

$$\Delta E = E(\mathbf{w} + \Delta\mathbf{w}) - E(\mathbf{w}) \qquad (4.110)$$

$$= \mathbf{g}^T \Delta\mathbf{w} + \frac{1}{2} \Delta\mathbf{w}^T H \Delta\mathbf{w} \qquad (4.111)$$

Taking the derivative of E with respect to w gives the gradient. That is

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{g} \qquad (4.112)$$

On the other hand, taking the derivative of $\Delta E$ with respect to $\Delta\mathbf{w}$ gives

$$\frac{\partial \Delta E}{\partial \Delta\mathbf{w}} = \mathbf{g} + H \Delta\mathbf{w} \qquad (4.113)$$

Setting this to zero gives an optimum value of Aw, taking **upto** the second order term into account. Therefore

$$\Delta\mathbf{w}^* = -H^{-1}\mathbf{g} \tag{4.114}$$

**Thus** the new weight **vector** taking the optimal value of Aw is given by

$$\mathbf{w}(m+1) = \mathbf{w}(m) - H^{-1}\mathbf{g} \tag{4.115}$$

This is the Newton's method. Note that this is similar to the expression (**C**.17) in Appendix-C.

For the quadratic error function $E(\mathbf{w})$, the optimal step Aw* will lead to the final weight value w* starting from any initial weight vector **w**(0). That is

$$\mathbf{w}^* = \mathbf{w}(0) - H^{-1}\mathbf{g} \tag{4.116}$$

provided $H^{-1}\mathbf{g}$ is known at w = **w**(0). For a nonquadratic error surface, as in the network with nonlinear units, the Newton's method gives the optimal weight change if the variation of the error is considered only **upto** the second derivative. Note that the Newton's method is different from the gradient descent. Since the Newton's method uses more information of the error surface than the gradient descent, it is expected to converge faster. But there is no guarantee that this choice of the weight change will converge.

Implementation of Newton's method is cumbersome due to the need for computation of the Hessian matrix. Methods were proposed which will avoid the need for the computation of the Hessian matrix. The conjugate gradient method is one such method, where the increment in the weight at the mth step is given by

$$\Delta\mathbf{w} = \mathbf{w}(m+1) - \mathbf{w}(m) = \eta(m)\,\mathbf{d}(m) \tag{4.117}$$

where the direction of the increment $\mathbf{d}(m)$ in the weight is a linear combination of the current gradient vector and the previous direction of the increment in the weight. That is

$$\mathbf{d}(m) = -\,\mathbf{g}(m) + \alpha(m-1)\,\mathbf{d}(m-1) \tag{4.118}$$

where the value of $\alpha(m)$ is obtained in terms of the gradient by one of the following formulae [**Fletcher** and Reeves, 1964; Polak and Ribiere, 19691.

$$\alpha(m) = \frac{\mathbf{g}^T(m+1)\mathbf{g}(m+1)}{\mathbf{g}^T(m)\mathbf{g}(m)} \tag{4.119}$$

**or**

$$\alpha(m) = \frac{\mathbf{g}^T(m+1)[\mathbf{g}(m+1) - \mathbf{g}(m)]}{\mathbf{g}^T(m)\mathbf{g}(m)} \tag{4.120}$$

Computation of the learning rate parameters $\eta(m)$ in Eq. (4.117) requires line minimization for each iteration [Johansson et al, 19901.

The objective is to determine the value of $\eta$ for which the error $E[\mathbf{w}(m) + \eta\, \mathbf{d}(m)]$ is minimized for given values of $\mathbf{w}(m)$ and $\mathbf{d}(m)$. Performance of the conjugate-gradient method depends critically on the choice of $\eta(m)$ and hence on the line minimization. But generally the conjugate-gradient method converges much faster than the standard backpropagation learning, although there is no proof of convergence in this case also due to the nonquadratic nature of the error surface [**Kramer** and Sangiovanni-Vincentelli, 19891.

**Refinements of the backpropagation learning:** The backpropagation learning is based on the steepest descent along the surface of the instantaneous error in the weight space. It is only a first order approximation of the descent as the weight change is assumed to be proportional to the negative gradient. The instantaneous error is a result of a single training pattern, which can be viewed as a sample function of a random process. The search for the global minimum of the error surface is stochastic in nature as it uses only the instantaneous error at each step. The stochastic nature of the gradient descent results in a zig-zag path of the trajectory in the weight space in our search for the global minimum of the error surface. Note that the zig-zag path is also due to the nonquadratic nature of the error surface, which in **turn** is due to the nonlinear output functions of the units. Note also that the backpropagation learning is based only on the gradient descent and not on any optimization criterion.

A better learning in terms of convergence towards the global minimum may be achieved if the information from the given training patterns are used more effectively. One such approach is based on posing the supervised learning problem as a nonlinear system identification problem [**Haykin,** 19911. The resulting learning algorithm is called an extended Kalman-type learning [Singhal and Wu, 19891 which uses piecewise linear approximation to the nonlinear optimal filtering problem.

Better learning can also be achieved if the supervised learning is posed as an unconstrained optimization problem, where the cost function is the error function $E(\mathbf{w})$ [Battiti, 19921. In this case the optimal value of the increment in the weight is obtained by considering only **upto** second order derivatives of the error function. The resulting expression for the optimal Aw requires computation of the second derivatives of $E(\mathbf{w})$ with respect to all the weights, namely, the Hessian matrix. The convergence will be faster than the gradient descent, but there is no guarantee for convergence in this case also.

A multilayer **feedforward** neural network with backpropagation learning on a finite set of independent and identically distributed samples leads to an asymptotic approximation of the underlying a posteriori class probabilities provided that the size of the training set data is large, and the learning algorithm does not get struck in a local minima [Hampshire and Pearlmutter, 19901.

If the a posteriori conditional probabilities are used as the desired response in a learning algorithm based on an information theoretic measure for the cost function [Kullback, 1968; **Haykin,** 1994, **Sec.** 6.201, then the network captures these conditional probability distributions. In particular, the output of the network can be interpreted as estimates of the a posteriori conditional probabilities for the underlying distributions in the given training data.

Yet another way of formulating the learning problem for a multilayer neural network is by using the fuzzy representation for input or output or for both. This results in a fuzzy backpropagation learning law [Ishibuchi et al, 19931. The convergence of the fuzzy backpropagation learning is significantly faster, and the resulting minimum mean squared **error** is also significantly lower than the usual backpropagation learning.

**Interpretation of the result of learning: A** trained multilayer feedforward neural network is expected to capture the functional relationship between the input-output pattern pairs in the given training data. It is implicitly assumed that the mapping function corresponding to the data is a smooth one. But due to limited number of training samples, the problem becomes an ill-posed problem, in the sense that there will be many solutions satisfying the given data, but none of them may be the **desired/correct** one **[Tikhonov** and **Arsenin,** 1977; Wieland and Leighton, 19871. Figure 4.14 illustrates the basic

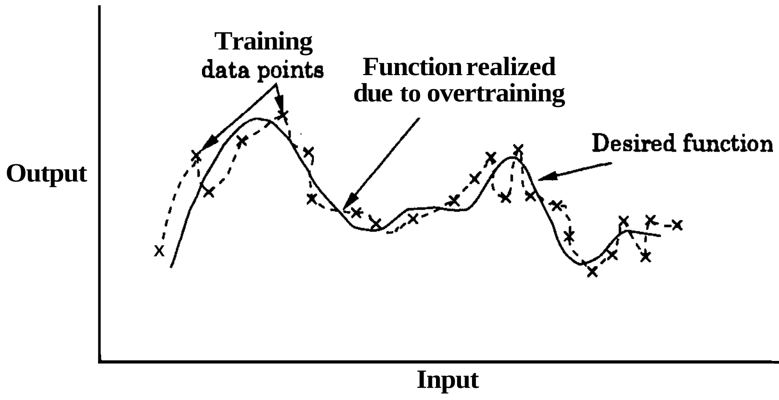


**Figure 4.14 Illustration of an ill-posed problem for a function of one variable.**

idea of an ill-posed problem for a function of one variable. Given the samples marked 'x', the objective is to capture the function represented by the solid curve. But depending on the size of the network, several solutions are possible, including the overtraining situations (shown by dotted curve) in which for all the training data the error is zero. In fact there could be several functions passing through the given set of points, none of which is the desired one. This happens if the number of free parameters (weights) of the network is very large. Such a situation results in a large error when some other (test) samples are given to validate the network model for the function. This is called 'poor generalization' by the network. On the other hand, fewer number of the free parameters may result in a large error even for the training data, and hence a poor approximation to the desired function. The function approximation interpretation of a multilayer feedforward neural network enables us to view different hidden layers of the network performing different functions. For example, the first hidden layer can be interpreted as capturing some local features in the input space. The second hidden layer can be interpreted as capturing some global features. This two-stage approximation has been shown to realize any continuous vector-valued function [Sontag, 1992b]. The universal approximation theorem of Cybenko seems to suggest that even a single layer of nonlinear units would suffice to realize any continuous function [Cybenko, 19891. But this result assumes that a hidden layer of unlimited size is available, and that the continuous function to be approximated is also available. Thus Cybenko's theorem gives only an existence proof, but it is not useful to realize the function by training a single hidden layer network.

A trained multilayer neural network can be interpreted as a classifier, with complex decision surfaces separating the classes. These decision surfaces are due to multiple layers of nonlinear units. In the limiting case of hard-limiting nonlinear units, the geometrical arguments for the creation of the complex decision surfaces in a multilayer perceptron discussed in Section 4.3.3 are applicable.

It is also possible to view that the hidden layers perform a nonlinear feature extraction to map the input data into linearly separable classes in the feature space. At the output layer the unit with the largest output is considered as the class to which the input belongs.

As mentioned earlier, the output of a trained multilayer neural network can also be considered as an approximation to the a posteriori class probabilities.

**Generalization:** A backpropagation learning network is expected to generalize from the training set data, so that the network can be used to determine the output for a new test input. As mentioned earlier, 'generalization' is different from 'interpolation', since in generalization the network is expected to model the unknown system or function from which the training set data has been obtained. The problem of determination of weights from the training set data is called the loading' problem [Judd, 1990; Blum and Rivest, 19921. The generalization performance depends on the size and efficiency of the training set, besides the architecture of the network and the complexity of the problem [Hush and Horne, 1993]. Testing the performance of the network with new data is called cross-validation. If the performance for the test data is as good as for the training data, then the network is said to have generalized from the training data. Further discussion on generalization is given later in Section 7.3 and in Appendix D.

**Tasks with backpropagation network:** A backpropagation network can be used for several applications such as realization of logic functions, pattern classification, pattern mapping, function approximation, estimation of probability distribution and prediction [Hush and Horne, 1993]. These tasks were demonstrated in several real world applications such as in speech, character recognition, system identification, passive sonar detection/classification, speech synthesis, etc. [Sejnowski and Rosenberg, 1987; Cohen et al, 1993; LeCun et al, 1990; Narendra and Parthasarathy, 1990; Casselman et al, 1991].

**Limitations of backpropagation:** The major limitation of the backpropagation learning is its slow convergence. Moreover, there is no proof of convergence, although it seems to perform well in practice. Due to stochastic gradient descent on a nonlinear error surface, it is likely that most of the time the result may converge to some local minimum on the error surface [Gori and Tesi, 1992]. There is no easy way to eliminate this effect completely, although stochastic learning algorithms were proposed to reduce the effects of local minima [Wasserman, 1988]. Another major problem is the problem of scaling. When the complexity of the problem is increased, there is no guarantee that a given network would converge, and even if it converges, there is no guarantee that good generalization would result. The complexity of a problem can be defined in terms of its size or its predicate order [Minsky and Papert, 1990; Hush and Horne, 1993]. Effects of scaling can be handled by using the prior information of the problem, if possible. Also, modular architectures can also reduce the effects of the scaling problem [Ballard, 1990; Jacobs et al, 1991; Haykin, 1994].

For many applications, the desired output may not be known precisely. In such a case the backpropagation learning cannot be used directly. Other learning laws have been developed based on the information whether the response is correct or wrong. This mode of learning is called reinforcement learning or learning with critic [Sutton et al, 1991; Barto, 1992] as discussed in Section 2.4.6.

**Extensions of backpropagation:** Principles analogous to the ones used in the backpropagation network have been applied to extend the scope of the network in several directions as in the case of probabilistic neural networks, **fuzzy** backpropagation networks, regularization networks and radial basis function networks [**Wasserman, 19931.**

## 4.5 Summary and Discussion

We have presented a detailed analysis of feedforward networks in this chapter with emphasis on the pattern recognition tasks that can be realized using these networks. A network with linear units (Adaline units) performs a pattern association task provided the input patterns are linearly independent. Linear independence of input patterns also limits the number of patterns to the dimensionality of the input pattern space. We have seen that this limitation is overcome by using hard-limiting threshold units (perceptron units) in the feedforward network. Since threshold units in the output layer results in a discrete set of states, the resulting network performs pattern classification task. The hard-limiting threshold units provide a set of inequalities to be satisfied by the network. Thus the weights of the network are not unique any more and hence they are determined by means of the perceptron learning law.

A single layer perceptron is limited to linearly separable classes only. For an arbitrary pattern classification problem, a multilayer perceptron (**MLP**) is needed. But due to absence of desired output at the units in the intermediate layers of units, the MLP network cannot be trained by the simple perceptron learning law. This hard learning problem can be solved by using nonlinear units with differentiable output functions. Since the output functions are now continuous, the multilayer feedforward neural network can perform pattern mapping task. The output error **backpropagation** is used in the learning algorithm for these multilayer networks.

Since the backpropagation learning is based on stochastic gradient descent along a rough error surface, there is no guarantee that the learning law converges towards the desired solution for a given pattern mapping task. Several variations of the backpropagation learning have been suggested to improve the convergence as well as the result of convergence. Although there is no proof of convergence, the backpropagation learning algorithm seems to perform effectively for many tasks such as pattern classification, function approximation, time series prediction, etc.

How well a trained feedforward network performs a given task is discussed both theoretically and experimentally in the literature on generalization. The issue of generalization is an important topic, but it is not discussed in this book. There are excellent treatments of this topic in [**Vidyasagar, 1997;** Valiant, **19941.** Appendix D gives an overview of generalization in neural networks.

Some of the limitations of backpropagation such as convergence

Chapter 5

# Feedback Neural Networks

## 5.1 Introduction

This chapter presents a detailed analysis of the pattern recognition tasks that can be performed by feedback artificial neural networks. In its most general form a feedback network consists of a set of processing units, the **output** of each unit is fed as input to all other units including the same unit. With each link connecting any two units, a weight is associated which determines the amount of output a unit feeds as input to the other unit. A general feedback network does not have any structure, and hence is not likely to be useful for solving any pattern recognition task.

However, by appropriate choice of the parameters of a feedback network, it is possible to perform several pattern recognition tasks. The simplest one is an autoassociation task, which can be performed by a feedback network consisting of linear processing units. A detailed analysis of the linear autoassociative network shows that the network is severely limited in its capabilities. In particular, a linear autoassociative network merely gives out what is given to it as input. That is, if the input is noisy, it comes out as noisy output, thus giving an error in recall even with optimal setting of weights. Therefore a linear autoassociative network does not have any practical use. By using a nonlinear output function for each processing unit, a feedback network can be used for pattern storage. The function of a feedback network with nonlinear units can be described in terms of the **trajectory** of the state of the network with time. By associating an energy with each state, the trajectory describes a traversal along the energy landscape. The minima of the energy landscape correspond to stable states, which can be used,to store the given input patterns. The number of patterns that can be stored in a given network depends on the number of units and the strengths of the connecting links. It is quite possible that the number of available energy minima is less than the number of patterns to be stored. In such a case the given pattern storage problem becomes a hard problem for the network. If on the other hand, the number of energy minima in the energy landscape of a network is greater than the required number of patterns to be stored, then there is likely to be an error in the recall of the stored patterns due to the additional false minima. The hard problem can be solved by providing additional (hidden) units in a feedback network, and the errors in recall of the stored patterns due to false minima can be reduced using probabilistic update for the output function of a unit. A feedback network with hidden units and probabilistic update is called a Boltzmann machine. It can be used to store a pattern environment, described by a set of patterns to be stored, together with the probability of occurrence of each of these patterns.

Table **5.1** shows the organization of the topics to be discussed in this chapter. A detailed analysis of linear autoassociative feedforward networks is considered first in Section **5.2.** The pattern storage problem is analyzed in detail in Section **5.3.** In particular, the **Hopfield** energy analysis, and the issues of hard problem and false minima are discussed in this section. The Boltzmann machine is introduced in Section **5.4.** This section also deals with the details of the pattern environment storage problem and the Boltzmann learning law. Some practical issues in the implementation of learning laws for feedback networks including simulated annealing are discussed in Section **5.5.**

**Table 5.1** Pattern Recognition Tasks by Feedback Neural Networks

**Autoassociation**

- ***Architecture:*** Single **layer** with feedback, linear processing units

  ***Learning:*** Not important
- Recall: Activation dynamics until stable states are reached
- ***Limitution:*** No accretive behaviour
- ***To overcome:*** Nonlinear processing units, leads to a pattern storage problem

**Pattern Storage**

- ***Architecture:*** Feedback neural network, nonlinear processing units, states, **Hopfield** energy analysis
- ***Learning:*** Not important
- ***Recall:*** Activation dynamics until stable states are reached
- ***Limitation:*** Hard problems, limited number of patterns, false minima
- ***To overcome:*** Stochastic update, hidden units

**Pattern Environment Storage**

- ***Architecture:*** Boltzmann machine, nonlinear processing units, hidden units, stochastic update
- ***Learning:*** Boltzmann learning law, simulated annealing
- ***Recall:*** Activation dynamics, simulated annealing
- ***Limitation:*** Slow learning
- ***To Overcome:*** Different architecture

## 5.2 Analysis of Linear Autoassociative FF Networks

**First** we cansider the realization of an autoassociative task with a feedforward **network** as shown in Figure 5.1. Analogous to the hetero-

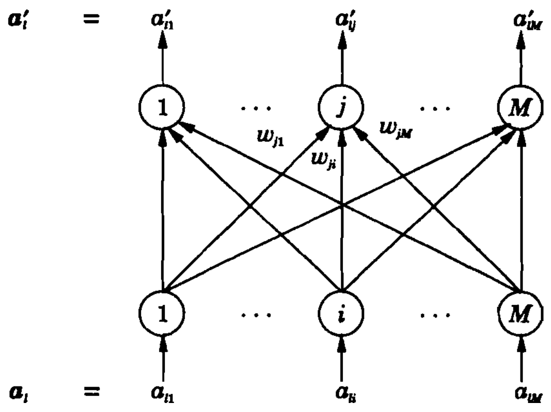


**Figure** 5.1 Linear autoassociative **feedforward** network.

association, in autoassociation the objective is to associate a given pattern with itself during training, and then to recall the associated pattern when an **approximate/noisy** version of the same pattern is given during testing. In other words, in **autoassociation** the associated output pattern $\mathbf{b}_l$ is same as the input pattern $\mathbf{a}_l$ for the ***l*th** pattern. That is, with reference to the pattern association task, $\mathbf{b}_l = \mathbf{a}_l$, 1 = 1, 2, ..., L in the case of autoassociation. In recall it is desired to obtain $\mathbf{b}_l$ as output for an approximate input $\mathbf{a}_l + \varepsilon$. The weight matrix W = $[w_{ij}]$ of a linear autoassociative network (Figure 5.1) can be determined as in the case of the linear heteroassociator, for a fixed set of input pattern vectors $\{\mathbf{a}_l\}$. Since we want WA= A, the optimal weights are given by (see Section **4.2.2**)

$$W = AA^+ \tag{5.1}$$

where A+ is the pseudoinverse of the M **x** L matrix A consisting of the input vectors $\{\mathbf{a}_l\}$. The pseudoinverse is given in **terms** of the components of singular value decomposition of the matrix A as follows:

$$A = \sum_{i=1}^{L} \lambda_i^{1/2}\, \mathbf{p}_i \mathbf{q}_i^T \tag{5.2}$$

where $\lambda_i$ are the eigenvalues, and $\mathbf{p}_i$ and $\mathbf{q}_i$ are the eigenvectors of the matrices AA$^T$ and $A^TA$, respectively. That is,

$$AA^T\mathbf{p}_i = \lambda_i\, \mathbf{p}_i \tag{5.3}$$

and

$$A^TA\mathbf{q}_i = \lambda_i\, \mathbf{q}_i \tag{5.4}$$

The sets of **eigenvectors** $\{\mathbf{p}_i\}$ and $\{\mathbf{q}_i\}$ are orthonormal sets. The **eigenvalues** $\lambda_i$ are real and nonnegative, since the matrices $AA^T$ and $A^TA$ are symmetric. The eigenvalues $\lambda_i$ are ordered, **i.e.**, $\lambda_i \geq \lambda_{i+1}$. If the **rank** of the matrix A is $r$ ($\leq$ L), then the eigenvalues $\lambda_i$, $i > r$ will be zero. Therefore

$$A = \sum_{i=1}^{r} \lambda_i^{1/2} \mathbf{p}_i \mathbf{q}_i^T \tag{5.5}$$

and the **pseudoinverse**

$$A^+ = \sum_{i=1}^{r} \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T \tag{5.6}$$

The minimum error for the choice of the optimum weight matrix $W = AA^+$ is given from Eq. (4.16) as

$$\begin{aligned} E_{\min} &= \frac{1}{L} \sum_{i=r+1}^{L} \| A\mathbf{q}_i \|^2 \\ &= \frac{1}{L} \sum_{i=r+1}^{L} \left\| \sum_j \lambda_j^{1/2} \mathbf{p}_j \mathbf{q}_j^T \mathbf{q}_i \right\|^2 \\ &= \frac{1}{L} \sum_{i=r+1}^{L} \lambda_i \quad \text{since } \mathbf{q}_j^T \mathbf{q}_i = 0, \quad j \neq i \\ &\qquad\qquad\qquad\qquad = 1, j = i \end{aligned} \tag{5.7}$$

But since $\lambda_i = 0$ for $i > r$, $E_{\min} = 0$. Thus in the case of linear autoassociative network there is no error in the recall due to linear dependency of the input patterns, unlike in the case of linear heteroassociative network. In other words, in this case the input comes out as output without any error.

When noise is added to the input vector, the noisy input vectors are given by

$$\mathbf{c}_l = \mathbf{a}_l + \varepsilon, \quad l = 1, 2, \ldots, L \tag{5.8}$$

where the noise **vector** $\varepsilon$ is **uncorrelated** with the input vector $\mathbf{a}_l$, and has the average power or variance $\sigma^2$. For the choice of $W = AA^+$, the error in recall is given **from** Eq. **(4.19)** as [Murakami and Aibara, **1987**]

$$\begin{aligned} E(W) &= \frac{1}{L} \left[ \sum_{i=r+1}^{L} \| A\mathbf{q}_i \|^2 + L\sigma^2 \sum_{i=1}^{r} \lambda_i^{-1} \| A\mathbf{q}_i \|^2 \right] \\ &= \sigma^2 r \end{aligned} \tag{5.9}$$

**Thus** the error in the recall is mainly due to noise, as the linear dependence component of the error is zero in the case of **auto**-association. Note that this is because a noisy input vector comes out as a noisy output and hence its difference from the true vector in the recall is only due to noise.

The error in recall due to noise can be reduced by the choice of $W = A\hat{A}^+$, where

$$\hat{A}^+ = \sum_{i=1}^{s} \lambda_i^{-1/2} \mathbf{q}_i \mathbf{p}_i^T \tag{5.10}$$

where $s$ is given by

$$\frac{L\sigma^2}{\lambda_s} < 1 < \frac{L\sigma^2}{\lambda_{s+1}} \tag{5.11}$$

That is, for a given noise power $\sigma^2$, the error can be reduced by moving the error into the linear dependency term, which is realized by an appropriate choice of the number of terms in the expression for the pseudoinverse. The resulting error for the optimal choice of $W = A\hat{A}^+$ is

$$E_{\min} = \frac{1}{L}\left[\sum_{i=s+1}^{r} \lambda_i + L\sigma^2 s\right] \tag{5.12}$$

The linear autoassociation task can also be realized by a single layer feedback network with linear processing units shown in Figure 5.2. The

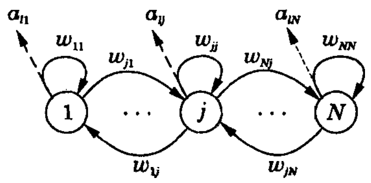


Figure 5.2 Linear autoassociation by a feedback network.

condition for autoassociation, namely, $W\mathbf{a}_l = \mathbf{a}_l$, is satisified if $W = \mathbf{I}$, an identity matrix. This trivial choice of the weight matrix is realized if the input vectors are linearly independent, so that $W = AA^{-1} = \mathbf{I}$. For this choice of W, the output for a noisy input $\mathbf{a}_l + \varepsilon$ is given by $W(\mathbf{a}_l + \varepsilon) = \mathbf{a}_l + \varepsilon$, which is the noisy input itself. This is due to lack of accretive behaviour during recall, and such a feedback network is not useful for storing information. It is possible to make a feedback network useful, especially for pattern storage, if the linear processing units are replaced with processing units having nonlinear output functions. We discuss this case in the next section and give a detailed analysis of pattern storage networks.

## 5.3 Analysis of Pattern Storage Networks

### 5.3.1 Pattern Storage Networks

The objective in a pattern storage task is to store a given set of patterns, so that any of them can be recalled exactly when an approximate version of the corresponding pattern is presented to the network. For this purpose, the features and their spatial relations in the patterns need to be stored. The pattern recall should take place even when the features and their spatial relations are slightly disturbed due to noise and distortion or due to natural variation of the pattern generating process. The approximation of a pattern refers to the closeness of the features and their spatial relations to the original stored pattern.

Sometimes the data itself is actually stored through the weights, as in the case of binary patterns. In this case the approximation can be measured in terms of some distance, like Hamming distance, between the patterns. The distance is automatically captured by the threshold feature of the output functions of the processing units in a feedback network Freeman and Skapura, **19911.**

Pattern storage is generally accomplished by a feedback network consisting of processing units with nonlinear output functions. The outputs of the processing units at any instant of time define the output state of the network at that instant. Likewise, the activation values of the units at any instant determine the activation state of the network at that instant.

The state of the network at successive instants of time, **i.e.**, the trajectory of the state, is determined by the activation dynamics model used for the network. Recall of a stored pattern involves starting at some initial state of the network depending on the input, and applying the activation dynamics **until** the trajectory reaches an equilibrium state. The final equilibrium state is the stored pattern resulting **from** the network for the given input.

Associated with each output state is an energy (to be defined later) which depends on the network parameters like the weights and bias, besides the state of the network. The energy as a function of the state of the network corresponds to something like an ***energy landscape.*** The shape of the energy landscape is determined by the network parameters and states. The feedback among the units and the nonlinear processing in the units may create basins of attraction in the energy landscape, when the weights satisfy certain constraints. Figure 5.3 shows energy landscapes as a function of the output state for the two cases of with and without the basins of attraction. In the latter case the energy fluctuates quickly and randomly from one state to another as shown in Figure **5.3b.** But in the energy landscape with basins of attraction as in Figure **5.3a,** the states around the stable state correspond to small deviations from the stable state. The deviation can be measured in some suitable distance measure, such as Hamming distance for binary patterns. The Hamming distance between two binary patterns each of length $N$ is defined ***as*** the number of bit positions in which the patterns differ. Thus the states

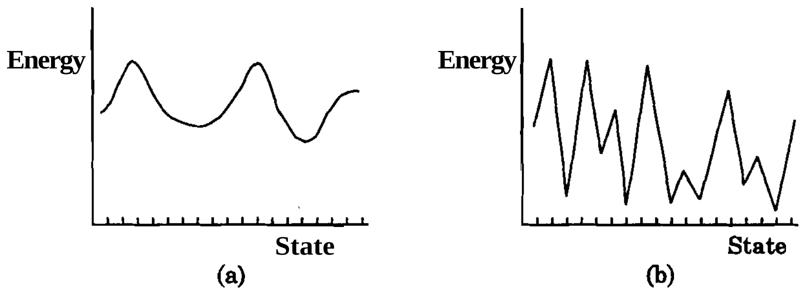


**Figure 5.3** **Energy landscapes (a) with basins of attraction and (b) without basins of attraction.**

closer to the stable states correspond to patterns with smaller Hamming distance.

The basins of attraction in the energy landscape tend to be the regions of stable equilibrium states [Cohen and Grossberg, 19831. If there is a fixed state in each of the basins where the energy is minimum, then that state corresponds to a fixed point of equilibrium. The basins could also be periodic (oscillatory) regions or chaotic regions of equilibria. For an oscillatory region, the state of the network changes continuously in a periodic manner. For a chaotic region, the state of the network is not predictable, but it is confined to the equilibrium region. Throughout the subsequent discussion we consider only the fixed points of equilibrium in the energy landscape.

It is the existence of the basins of attraction or regions of equilibrium states that is exploited for the pattern storage task. The fixed points in these regions correspond to the states of the energy minima, and they are used to store the desired patterns. These stored patterns can be recalled even with approximate patterns as inputs. An erroneous pattern is more likely to be closer to the corresponding true pattern than to the other stored patterns according to some distance measure. Each input pattern results in an initial state of the network, which may be closer to the desired true state in the sense that it may lie near the basin of attraction corresponding to the true state. An arbitrary state may not correspond to an equilibrium or a stable state. As the dynamics of the network evolves, the network may eventually settle at a stable state, from which the pattern may be read or derived.

Given a network specified by the number of processing units, their connection strengths and the activation dynamics, it is not normally possible to determine exactly the number of basins of attraction in the energy landscape as well as their relative spacings and depths in the state space of the network. The spacing between two states can be measured .by a suitable distance measure, such as the Hamming distance for binary patterns. The number of patterns that can be stored is called the capacity of the network. It is possible to estimate the capacity of the network and also the average probability of error in recall. The probability of error in recall can be reduced by adjusting the weights in such a way that the resulting energy landscape is matched to the probability distribution of the desired patterns.

Typically the capacity of a fully connected network is of the order of N, the number of processing units. Although there are $2^N$ different states for a network with binary state units, the network can be used to store only of the order of N binary patterns, as there will be only that many fixed points or energy minima in the energy landscape.

In general, the number of desired patterns is independent of the number of basins of attractions. The latter depends only on the network units and their interconnections. If the number of patterns is more than the number of basins of attraction, then the pattern storage problem becomes a hard problem, in the sense that the patterns cannot be stored in the given network. On the other hand, if the number of patterns is less than the number of basins of attraction, then there will be the so called false wells or minima due to the additional basins of attraction. During recall, it is likely that the state of the network, as it evolves from the initial state corresponding to the input pattern, may settle in a false well. The recalled pattern corresponding to the false well may not be the desired pattern, thus resulting in an error in the recall.

In the next subsection we will consider the Hopfield model of a feedback network for the pattern storage and discuss the working of a discrete Hopfield model. The Hopfield model is a fully connected feedback network with symmetric weights. In the discrete Hopfield network the state update is asynchronous and the units have binary/bipolar output functions. In the continuous Hopfield model the state update is dictated by the activation dynamics, and the units have continuous nonlinear output functions.

### 5.3.2 The Hopfield Model

Consider the McCulloch-Pitts neuron model for the units of a feedback network, where the output of each unit is fed to all the other units with weights $w_{ij}$, for all i and j. Let the output function of each of the units be bipolar ($\pm 1$) so that

$$s_i = f(x_i) = \text{sgn}(x_i), \tag{5.13}$$

and

$$x_i = \sum_{j=1}^{N} w_{ij} s_j - \theta_i \tag{5.14}$$

where $\theta_i$ is the threshold for the unit i. We will assume $\theta_i = 0$ for convenience. The state of each unit is either +1 or –1 at any given instant of time. Due to feedback, the state of a unit depends on the states of the other units. The updating of the state of a unit can be done synchronously or asynchronously. In the synchronous update all the units are simultaneously updated at each time instant, assuming that the state of the network is frozen until update is made for **all** the units. In the asynchronous update a unit is **selected** at random and its new state is computed. Another unit is selected at random and its state is updated using the current state of the network. The **updating using** the random choice of a unit is continued until no further change in the state takes place for all the units. That is, the state at time $(t + 1)$ is the same as the state at time $t$ for all the units. That is

$$s_i(t+1) = s_i(t), \quad \text{for all } i \tag{5.15}$$

In this situation we can say that the network activation dynamics reached a stable state. We assume **asynchronous** update throughout the following discussion. Note that the asynchronous update ensures that the next state is at most unit Hamming distance from the current state.

If the network is to store a pattern $a = (a_1, a_2, ..., a_N)^T$, then in a stable state we must have the updated state value to be the same as the current state value. That is

$$\text{sgn}\left(\sum_{j=1}^{N} w_{ij}\, a_j\right) = a_i, \quad \text{for all } i \tag{5.16}$$

This can happen if $w_{ij} = (1/N)\, a_i\, a_j$, because

$$\sum_{j=1}^{N} w_{ij}\, a_j = \frac{1}{N}\sum_{j=1}^{N} a_i\, a_j\, a_j = \frac{a_i}{N}\sum_{j=1}^{N} a_j^2 = a_i \tag{5.17}$$

where $a_j^2 = 1$ for bipolar $(\pm 1)$ states.

For storing L patterns, we could choose a general Hebbian rule given by the **summation** of the Hebbian terms for each pattern. That is,

$$w_{ij} = \frac{1}{N}\sum_{l=1}^{L} a_{li}\, a_{lj} \tag{5.18}$$

Then the state $\mathbf{a}_k$ will be stable if

$$\text{sgn}\left(\frac{1}{N}\sum_{j=1}^{N}\sum_{l=1}^{L} a_{li}\, a_{lj}\, a_{kj}\right) = a_{ki}, \quad \text{for all } i \tag{5.19}$$

Taking out the $l = k$ *term* in the summation and simplifying it using $a_{kj}^2 = 1$, we get

$$\text{sgn}\left(a_{ki} + \frac{1}{N}\sum_{j=1}^{N}\sum_{l \neq k} a_{li}\, a_{lj}\, a_{kj}\right) = a_{ki}, \quad \text{for all } i \tag{5.20}$$

Since $a_{ki} = \pm 1$, the above is true for all $a_{ki}$, provided the **crossterm** in Eq. (5.20) does not change the sign of $a_{ki}$ plus the crossterm. Table 5.2 gives an algorithm for storing and recall of patterns in a **Hopfield** network.

Table 6.2 **Hopfield** Network Algorithm to Store and Recall a Set of Bipolar Patterns

Let the network **consist** of N fully connected **units** with each unit having hard-limiting bipolar threshold output function. Let $\mathbf{a}_l$, $l = 1, 2, ..., L$ be the vectors to be stored. The vectors $\{\mathbf{a}_l\}$ are assumed to have bipolar components, i.e., $a_{li} = \pm 1$, $i = 1, 2, ..., N$.

1. **Assign** the connection weights

$$w_{ij} = \frac{1}{N} \sum_{l=1}^{L} a_{li} a_{lj}, \quad \text{for } i \neq j$$
$$= 0, \text{ for } i = j, \ 1 \leq i, j \leq N$$

2. Initialize the network output with the given **unknown** input pattern a

$$s_i(0) = a_i, \text{ for } i = 1, 2, ..., N$$

where $s_i(0)$ **is** the output of the unit i at time t = 0

3. Iterate until convergence

$$s_i(t+1) = \text{sgn}\left[\sum_{j=1}^{N} w_{ij} s_j(t)\right], \quad \text{for } i = 1, 2, ..., N$$

The process is repeated until the outputs **remain** unchanged with further iteration. The steady outputa of the units represent the stored pattern that best matches the given input.

In general, the **crossterm** in Eq. (5.20) is negligible if LIN << 1. Eq. (5.20) is satisfied if the number of patterns L is limited to the storage capacity of the network, **i.e.**, the maximum number of patterns that can be stored in the network.

### 5.3.3 Capacity of Hopfield Model

We consider the discrete **Hopfield** model to derive the capacity of the network. Let us consider the following quantity [Hertz et al, 19911

$$c_i^k = -a_{ki} \frac{1}{N} \sum_{j=1}^{N} \sum_{l \neq k} a_{li} a_{lj} a_{kj} \tag{5.21}$$

If $c_i^k$ is negative then the cross term and $a_{ki}$ have the same sign in Eq. (5.20) and hence the pattern $\mathbf{a}_k$ is stable. On the other hand, if $c_i^k$ is positive and greater than 1, then the sign of the cross term changes the sign of $a_{ki}$ plus the cross term in Eq. (5.20). The result is that the pattern $\mathbf{a}_k$ turns out to be unstable, and hence the desired pattern cannot be stored.

Therefore the probability of **error** is given by

$$P_e = \text{Prob}\,(c_i^k > 1) \tag{5.22}$$

To compute this probability, let us assume that the probability of $a_{li}$ equal to +1 or –1 is 0.5. For random **patterns**, the cross term corresponds to 1/$N$ times the sum of about $NL$ independent random numbers, each of which is +1 or –1. Thus $c_i^k$ is a sum of random variables having a binomial distribution with zero mean and **variance** $\sigma^2 = L/N$. *If* $NL$ is assumed large, then the distribution of $c_i^k$ can be approximated by a Gaussian distribution with zero mean and $\sigma^2$ variance [**Papoulis**, 19911. Therefore,

$$\begin{aligned} P_e &= \frac{1}{\sqrt{2\pi}\,\sigma} \int_1^\infty e^{-x^2/(2\sigma^2)}\,dx \\ &= \frac{1}{2}\left[1 - \text{erf}\left(\sqrt{\frac{N}{2L}}\right)\right] \end{aligned} \tag{5.23}$$

where **erf**($x$) is error function given by

$$\text{erf}\,(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2}\,dx \tag{5.24}$$

This gives a value of $L_{\max}/N = 0.105$ for $P_e = 0.001$. Thus the maximum number of patterns that can be stored for a probability of error of 0.001 is $L_{\max} = 0.105\,N$.

A more sophisticated calculation [**Amit** et al, 1987; **Amit, 1989**] using probabilistic update leads to a capacity of $L_{\max} = 0.138\,N$.

### 5.3.4 Energy Analysis of Hoptield Network

**Discrete Hoptleld model:** Associated with each state of the network, **Hopfield** proposed an energy function whose value always either reduces or remains the same as the state of the network changes. Assuming the threshold value of the unit i to be $\theta_i$, the energy function is given by [Hopfield, 19821

$$V(\mathbf{s}) = V = -\frac{1}{2} \sum_i \sum_j w_{ij}\, s_i\, s_j + \sum_i \theta_i\, s_i \tag{5.25}$$

The energy $V(\mathbf{s})$ as a function of the state s of the network describes the energy landscape in the state space. The energy landscape is determined only by the network architecture, **i.e.**, the number of units, their output functions, threshold values, connections between units and the strengths of the connections. **Hopfield** has shown that for symmetric weights with no self-feedback, **i.e.**, $w_{ij} = w_{ji}$ , and with bipolar I–1, +1} or binary {0, 1) output functions, the dynamics of the network using the asynchronous update always leads towards energy minima at equilibrium. The states corresponding to these energy minima turn out to be stable states, which means that small perturbations around it lead to unstable states. Hence the dynamics of the network takes the network back to a stable state again. It is the existence of these stable states that enables us to store patterns, one at each of these states.

To show that AV $\leq$ 0, let us consider the change of state due to update of one unit, say k, at some instant. All other units remain unchanged. We can write the expressions for energy before and after the change **as follows** [Freeman and **Skapura**, 19911:

$$V^{\text{old}} = -\frac{1}{2} \sum_i \sum_j w_{ij}\, s_i^{\text{old}}\, s_j^{\text{old}} + \sum_i \theta_i\, s_i^{\text{old}}$$

$$V^{\text{new}} = -\frac{1}{2} \sum_i \sum_j w_{ij}\, s_i^{\text{new}}\, s_j^{\text{new}} + \sum_i \theta_i\, s_i^{\text{new}} \qquad (5.26)$$

The change in energy due to update of the kth unit is given by

$$\Delta V = V^{\text{new}} - V^{\text{old}}$$

$$= -\frac{1}{2} \sum_{i \neq k} \sum_{j \neq k} w_{ij}\, (s_i^{\text{new}}\, s_j^{\text{new}} - s_i^{\text{old}}\, s_j^{\text{old}}) + \sum_{i \neq k} \theta_i\, (s_i^{\text{new}} - s_i^{\text{old}})$$

$$-\frac{1}{2} \sum_i w_{ik}\, s_i^{\text{new}}\, s_k^{\text{new}} - \frac{1}{2} \sum_j w_{kj}\, s_k^{\text{new}}\, s_j^{\text{new}} + \theta_k\, s_k^{\text{new}}$$

$$+\frac{1}{2} \sum_i w_{ik}\, s_i^{\text{old}}\, s_k^{\text{old}} + \frac{1}{2} \sum_j w_{kj}\, s_k^{\text{old}}\, s_j^{\text{old}} - \theta_k\, s_k^{\text{old}} \qquad (5.27)$$

Since $s_i^{\text{new}} = s_i^{\text{old}}$, for i $\neq$ k, the first two **terms** on the right hand side of Eq. (5.27) will be zero. Hence,

$$\Delta V = s_k^{\text{new}} \left[ -\frac{1}{2} \sum_i w_{ik}\, s_i^{\text{new}} - \frac{1}{2} \sum_j w_{kj}\, s_j^{\text{new}} + \theta_k \right]$$

$$+ s_k^{\text{old}} \left[ +\frac{1}{2} \sum_i w_{ik}\, s_i^{\text{old}} + \frac{1}{2} \sum_j w_{kj}\, s_j^{\text{old}} - \theta_k \right] \qquad (5.28)$$

If the weights are assumed symmetric, **i.e.**, $w_{ij} = w_{ji}$, then we get

$$\Delta V = -s_k^{\text{new}} \left[ \sum_i w_{ki}\, s_i^{\text{new}} - \theta_k \right] + s_k^{\text{old}} \left[ \sum_i w_{ki}\, s_i^{\text{old}} - \theta_k \right] \qquad (5.29)$$

If, in addition, $w_{kk} = 0$, then since $s_i^{\text{new}} = s_i^{\text{old}}$ for i $\neq$ k, the terms in both the parentheses are equal. Therefore,

$$\Delta V = (s_k^{\text{old}} - s_k^{\text{new}}) \left[ \sum_i w_{ki}\, s_i^{\text{old}} - \theta_k \right] \qquad (5.30)$$

The update rule for each unit $k$ is as follows:

Case A: If $\sum_i w_{ki} s_i^{\text{old}} - \theta_k > 0$, then $s_k^{\text{new}} = +1$

Case B: If $\sum_i w_{ki} s_i^{\text{old}} - \theta_k < 0$, then $s_k^{\text{new}} = -1$

Case C: If $\sum_i w_{ki} s_i^{\text{old}} - \theta_k = 0$, then $s_k^{\text{new}} = s_k^{\text{old}}$

For case A, if $s_k^{\text{old}} = +1$, then AV$=0$, and if $s_k^{\text{old}} = -1$, then AV$\leq 0$.
For case B, if $s_k^{\text{old}} = +1$, then $\Delta V < 0$, and if $s_k^{\text{old}} = -1$, then AV$=0$.
For case C, irrespective of the value of $s_k^{\text{old}}$, AV $=$ 0.

Thus we have $\Delta V \leq 0$. Therefore the energy decreases or remains the same when a unit, selected at random, is updated, provided the weights are symmetric, and the self-feedback is zero. This is the energy analysis for discrete **Hopfield** model.

That the expression for **V** in Eq. (5.25) does indeed represent some form of energy can be seen from the following arguments based on **Hebb's** law:

If a given pattern vector $\mathbf{a}_l$ is to be stored in the network state vector **s**, then the match will be perfect if both the vectors coincide. That is, the magnitude of their inner product is maximum. Alternatively, the negative of the magnitude of their inner product is minimum. Thus we can choose a quantity [**Hertz** et al, 19911

$$V = -\frac{1}{2N}\left(\sum_{i=1}^{N} s_i a_{li}\right)^2 \tag{5.31}$$

to be minimized for storing a pattern vector $\mathbf{a}_l$ in the network. For storing L pattern vectors we can write the resulting **V** as a summation of the contributions due to each pattern vector. That is

$$\begin{aligned} V &= -\frac{1}{2N}\sum_{l=1}^{L}\left(\sum_{i=1}^{N} s_i a_{li}\right)^2 \\ &= -\frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{N} s_i s_j \sum_{l=1}^{L} a_{li} a_{lj} \end{aligned} \tag{5.32}$$

If we identify the weights $w_{ij}$ with the term $(1/N) \sum_{l=1}^{L} a_{li} a_{lj}$, then we get

$$V = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} s_i s_j \tag{5.33}$$

which is same as the first term in the **Hopfield** energy expression given in Eq. (5.25).

This method of identifying $w_{ij}$ from an energy function is useful, especially to solve several optimization problems. Given an energy function or a cost function or an objective function for a problem in terms of its variables and constraints, if we can identify the coefficients associated with $s_i s_j$, $s_i$ and constant terms in the function, then a feedback network can be built with weights corresponding to these coefficients. Then using an activation dynamics for the network, the equilibrium state or states can be found. These states correspond to the minima or maxima of the energy function. Higher order terms consisting of product of three $(s_i s_j s_k)$ or more variables cannot be handled by the feedback model with **pairwise** connections.

**Continuous Hopfield model.** In this subsection we will consider the energy analysis for a continuous **Hopfield** model [Hopfield, **1984**; Hertz et al, **1991**; Freeman and **Skapura, 1991]**. A continuous model is a fully connected feedback network with a continuous nonlinear output function in each unit. The output function is typically a sigmoid function $f(\lambda x) = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$ which is shown in Figure 5.4 for different

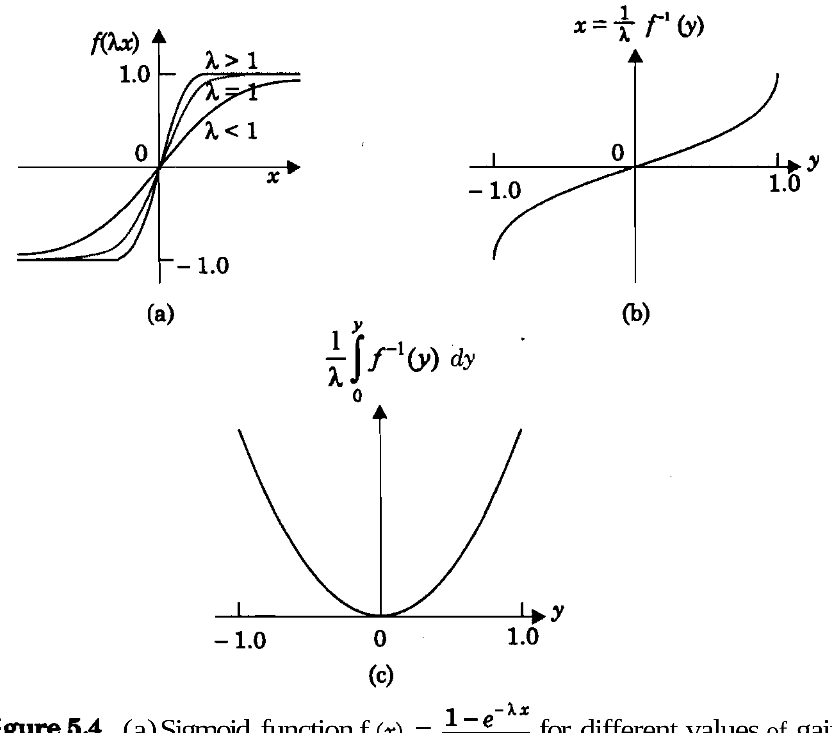


**Figure 5.4** (a) Sigmoid function $f(x) = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$ for different values of gain parameter $\lambda$. (b) The inverse function. (c) Contribution of $f(.)$ to the energy function.

values of the gain parameter $\lambda$. In the continuous model all the units will change their output signals ($s_i$) continuously and **simultaneously** towards values determined by the output function. The activation values ($x_i$) will also change continuously according to $x_i = \sum_j w_{ij} s_j$. This is reflected in the following equation for the activation dynamics:

$$\tau_i \dot{x}_i = -x_i + \sum_j w_{ij} s_j \tag{5.34}$$

where $\tau_i$ is the time constant and $s_j = f(x_j)$.

Consider the following energy function [Hopfield, 19841:

$$V = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i \int_0^{s_i} f^{-1}(s)\, ds \tag{5.35}$$

We can show that in this case ($dV/dt$) I 0.

$$\frac{dV}{dt} = -\frac{1}{2} \sum_i \sum_j w_{ij} \frac{ds_i}{dt} s_j - \frac{1}{2} \sum \sum w_{ij} s_i \frac{ds_j}{dt} + \sum f^{-1}(s_i) \frac{ds_i}{dt} \tag{5.36}$$

Assuming symmetry of weights, i.e., $w_{ij} = w_{ji}$, we get

$$\frac{dV}{dt} = -\sum_i \frac{ds_i}{dt} \left[ \sum_j w_{ij} s_j - f^{-1}(s_i) \right]$$

$$= -\sum_i \frac{ds_i}{dt} \left[ \sum_j w_{ij} s_j - x_i \right] \tag{5.37}$$

Using the relation in Eq. (5.34), we get

$$\frac{dV}{dt} = -\sum_i \tau_i \frac{ds_i}{dt} \frac{dx_i}{dt}$$

$$= -\sum_i \tau_i \dot{f}(x_i) \left( \frac{dx_i}{dt} \right)^2 \tag{5.38}$$

Since $f(x)$ is a monotonically increasing function, $\dot{f}(x) > 0$. Hence $dV/dt \leq 0$.

Note that $dV/dt = 0$ when $dx_i/dt = 0$, for **all** i. This shows that the activation dynamics eventually leads to a state where the energy function has a local minimum value, i.e., $dV/dt = 0$. This happens when the activation state reaches an equilibrium steady state at which there is no further change in the activation values, i.e., $dx_i/dt = 0$. The above result, namely, $dV/dt$ I 0, shows that the energy always decreases as the state of the network changes.

Let us examine the differences between the continuous model and the discrete model. In the discrete model only one unit is considered at a time for update. The choice of the unit for update is random and the dynamics is that of the steady activation values ($\dot{x}_i = 0$), since the transients are assumed to have died down at each update of the state of the unit. Hence in the discrete case $\dot{x}_i = 0$ and $\dot{V}(\mathbf{x}) = 0$ are different conditions. In the continuous case the states of all the units and hence the state of the network change continuously, as dictated by the differential equations for the activation dynamics. Hence, in this case $\dot{x}_i = 0$ for all i implies that $\dot{V} = 0$. The energy function V is also called the **Lyapunov** function of the dynamical system.

The **difference** in the energy functions for the discrete and continuous case is due to the extra term $\sum_i \int_0^{s_i} f^{-1}(s)\, ds$ in Eq. (5.35). This expression is for a gain value $\lambda = 1$. For a general gain value this term is given by $(1/\lambda) \sum_i \int_0^{s_i} f^{-1}(s)\, ds$. The integral term is 0 for $s_i = 0$ and becomes very large as $s_i$ approaches $\pm 1$ (see Figure **5.4c**). But for high gain values (h >> **1**), this term in the energy function becomes negligibly small, and hence the energy function approaches that of the discrete case. In fact when $\lambda \rightarrow \infty$, the output function becomes a bipolar function, and hence is equal to the discrete case. In the discrete case the energy minima are at some of the corners of the hypercube in the N-dimensional space, since all the states are at the corners of the hypercube. On the other hand, for moderate or small values of h, the integral term contributes to large positive values near the surfaces, edges and corners of the hypercube, and it contributes small values interior to the hypercube. This is because the value of $s_i$ is 1 at the surfaces, edges and corners. Thus the energy minima will be **displaced** to the interior of the hypercube. As $\lambda \rightarrow 0$, the minima of the energy function disappear one by one, since all the states will tend to have the same energy value.

The energy analysis so far shows that, for symmetric weights on the connections, there exist basins of attraction with a fixed point or a stable point for each basin corresponding to an energy minimum. If the connections are not symmetric, then the basins of attraction may correspond to oscillatory or chaotic states regions. In the case of purely random connections, with mean 0 and variance $\sigma^2$, there will be a transition from stable to chaotic behaviour as $\sigma^2$ is **increased** [**Sompolinsky** et al, 1988; Hertz, 1995; Hertz et al, 19911.

We can summarize the behaviour of feedback networks in relation to the complexity of the network as follows: To make a network useful for pattern storage, the output functions of the units are made **hard**-limiting nonlinear units. For analysis in terms of storage capacity, as well as for the recall of information from the stable states, we have imposed **Hopfield** conditions of symmetry of weights and asynchronous update. A more natural situation will be to use continuous output functions, so that any type of pattern can be stored. But the analysis of the performance of the network will be more difficult. In addition, if we relax the conditions on the symmetry of weights, we may still get stable regions, but it is not possible to analyse the network in terms of its storage capacity and retrieval of information. If we further relax the constraints to make the feedback system more closer to the natural biological system, then we may be able to get better functionality, but it is almost impossible to analyse such complex networks. For example it is not possible to predict the global pattern behaviour of a feedback network with random weights. Thus, although the networks may get more and more powerful by relaxing the constraints on the network, they become less useful, if we cannot predict and control the pattern storage and recall of the desired information.

### 5.3.5 State Transition Diagram

**Derivation of state transition diagram:** The energy analysis of the Hopfield network in the previous section shows that the energy of the network at each state either decreases or remains the same as the network dynamics evolves. In other words, the network either remains in the same state or moves to a state having a lower energy. This can also be demonstrated by means of a state transition diagram which gives the states of the network and their energies, together with the probability of transition from one state to another. In this section we will illustrate the state transition diagram (adapted from [Aleksander and Morton, 19901) for a 3-unit feedback network with symmetric weights $w_{ij} = w_{ji}$. The units have a threshold value of $\theta_i$, i = 1, 2, 3 and a binary {0, 1) output function. A binary output function is assumed for convenience, although the conclusions are equally valid for the bipolar (–1, +1} case.

Figure 5.5 shows a 3-unit feedback network. The state update for the unit i is governed by the following equation:

$$\begin{aligned} s_i(t+1) &= 1, \quad \text{if } \sum_j w_{ij}\, s_j(t) > \theta_i \\ &= 0, \quad \text{if } \sum_j w_{ij}\, s_j(t) \le \theta_i \end{aligned} \tag{5.39}$$

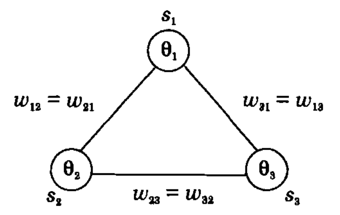


**Figure 5.5** A 3-unit feedback network with symmetric weights $w_{ij}$, threshold values $\theta_i$ and the output states $s_i$, i = 1, 2, 3.

The energy at any state $s_1 s_2 s_3$ of the network is given by

$$V(s_1 s_2 s_3) = -\frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i s_i \theta_i \tag{5.40}$$

There are eight different states for the 3-unit network, as each of the $s_i$ may assume a value either 0 or 1. Thus the states are: 000, 001, 010, 100, 011, 101, 110 and 111. Assuming the values

$$w_{12} = w_{21} = -0.5, \quad w_{23} = w_{32} = 0.4, \quad w_{31} = w_{13} = 0.5$$

$$\theta_1 = -0.1, \quad \theta_2 = -0.2, \quad \text{and} \quad \theta_3 = 0.7,$$

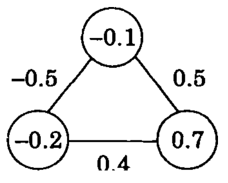
we get the following energy **values,for** each state.

$$V(000) = 0.0, \quad V(001) = 0.7, \quad V(010) = -0.2, \quad V(100) = -0.1,$$
$$V(011) = 0.1, \quad V(101) = 0.1, \quad V(110) = 0.2, \quad \text{and } V(111) = 0.0.$$

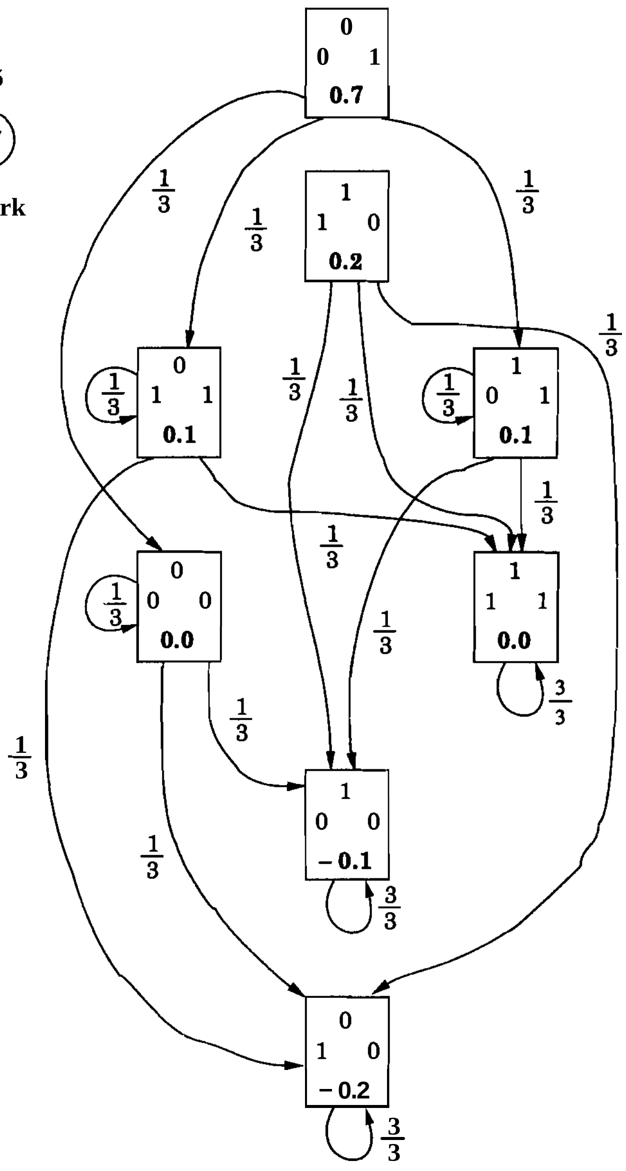
The transition from any state to the next state can be computed using the state update Eq. (5.39). For example, if the current state is 000, by selecting any one unit, say unit 2, at random, we can find its next state by computing the activation value $x_2 = \sum_{j=1}^{3} w_{2j} s_j$ and comparing it with the threshold $\theta_2$. Since $x_2$ (= 0) > $\theta_2$ (= − 0.2) the state of the unit 2 changes from 0 to 1. Thus if we select this unit, there will be a transition from the state 000 to 010. Since we can select any one of the three units with equal probability, i.e., U3, the probability of making a transition from 000 to 010 is thus U3. Likewise by selecting the unit 1 for update, the network makes a transition from 000 to 100 with a probability U3. Selecting the unit 3 for update results in a transition from 000 to itself, since the activation $x_3$ (= 0) < $\theta_3$ (= 0.7). By computing the transition probabilities for all the states, we get the state transition diagram shown in Figure 5.6. Note that while computing the transitions, only asynchronous update of each unit selected at random was used. Table 5.3 shows the computation of the state transitions by comparing the weighted inputs with the threshold value for each unit. The entries in the parenthesis are $\left(\sum_j w_{ij} s_j <> \theta_i\right)$.

From the state transition diagram we observe the following points: The diagram is drawn in such a way that the higher energy states are shown above the lower energy states. The transition is always from a higher energy state to a state with equal or lower energy. Thus the Hopfield result AV ≤ 0 is satisfied. There are some states 010, 100 and 111 which have a self-transition probability of 1. That means, once these states are reached, the network remains in these states, which is equivalent to saying that the activation dynamics equation is such that

$$f\left(\sum_{j=1}^{3} w_{ij} s_j - \theta_i\right) = s_i, \quad i = 1, 2, 3 \tag{5.41}$$



**(a) A 3-unit network**



**(b) State transition diagram**

**Figure 5.6 A 3-unit network and the corresponding state transition diagram. (Adapted from [Aleksander and Morton, 1990]).**

where $f(.)$ is the binary output function, i.e., $f(x) = 0$, for $x \leq 0$ and $f(x) = 1$, for $x > 0$. Since there is no transition from these states to other states, these are stable states. Note that only three out of the total eight are stable states. As per the approximate capacity calculations made in Section 5.3.3 for a Hopfield network, the number

**Table 6.3 Computation of State Transitions for Figure 5.6**

| | Unit 1 | Unit 2 | Unit 3 |
|---|---|---|---|
| 000 | (0 > − 0.1) 100 | (0 > − 0.2) 010 | (0 < 0.7) 000 |
| 001 | (0.5 > − 0.1) 101 | (0.4 > − 0.2) 011 | (0 < 0.7) 000 |
| 010 | (− 0.5 < − 0.1) 010 | (0 > − 0.2) 010 | (0.5 < 0.7) 010 |
| 100 | (0 > − 0.1) 100 | (− 0.5 < − 0.2) 100 | (0.5 < 0.7) 100 |
| 011 | (0 > − 0.1) 111 | (0.4 > − 0.2) 011 | (0.4 < 0.7) 010 |
| 101 | (0.5 > − 0.1) 101 | (− 0.1 > − 0.2) 111 | (0.5 < 0.7) 100 |
| 110 | (− 0.5 < − 0.1) 010 | (− 0.5 < − 0.2) 100 | (0.9 > 0.7) 111 |
| 111 | (0 > − 0.1) 111 | (− 0.1 > − 0.2) 111 | (0.9 > 0.7) 111 |

of stable states will be much fewer than the number of possible states, and in fact the number of stable states are of the order $N$. The stable states are always at the energy minima, so that the transition to any of these states is always from a state with a higher energy value than the energy value of the stable state.

**Computation of weights for pattern storage:** So far we have considered the analysis of a given feedback network and studied its characteristics. But patterns can be stored at the stable states by design. That is, it is possible to determine the weights of a network by calculation in order to store a given set of patterns in the network. Let 010 and 111 be the two patterns to be stored in a 3-unit binary network. Then at each of these states the following activation dynamics equations must be satisfied:

$$f\left(\sum_j w_{ij} s_j - \theta_i\right) = s_i, \qquad i = 1, 2, 3 \tag{5.42}$$

This will result in the following inequalities for each of the states:
For the state $s_1 s_2 s_3 = 010$

$$w_{12} - \theta_1 \leq 0, \; w_{22} - \theta_2 > 0, \; w_{32} - \theta_3 \leq 0,$$

and for the state $s_1 s_2 s_3 = 111$

$$w_{11} + w_{12} + w_{13} - \theta_1 > 0, \quad w_{21} + w_{22} + w_{23} - \theta_2 > 0,$$

$$w_{31} + w_{32} + w_{33} - \theta_3 > 0$$

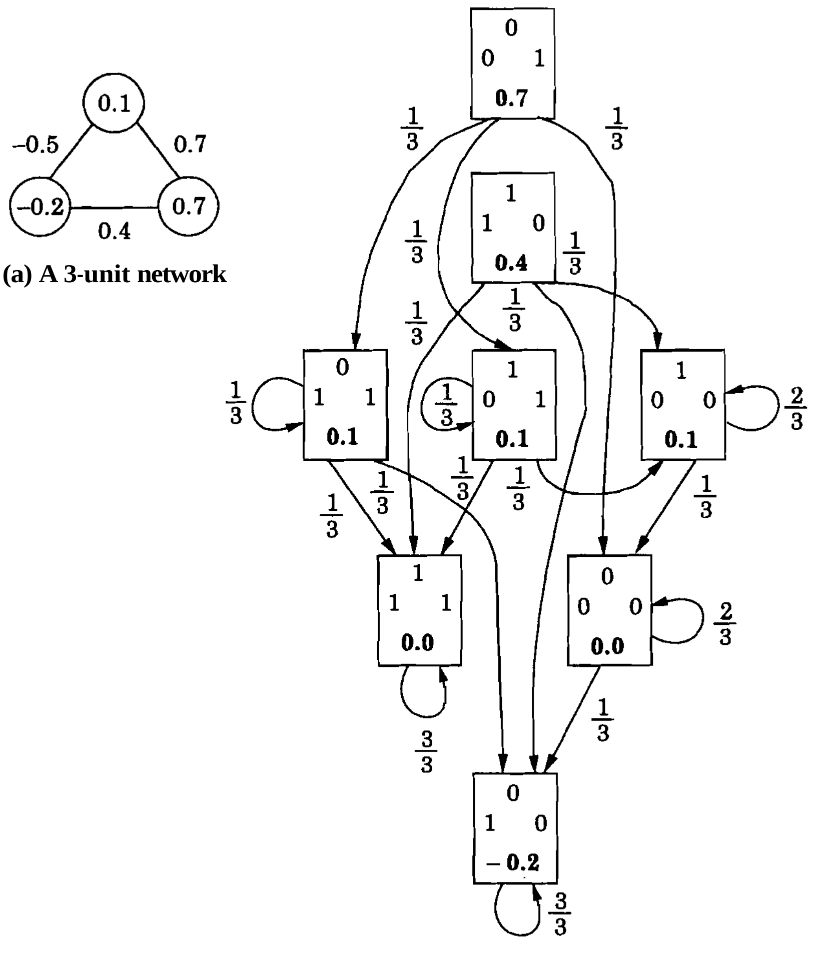
Since we assume symmetry of the weights ($w_{ij} = w_{ji}$) and $w_{ii} = 0$, the above inequalities reduce to

$$w_{12} \leq \theta_1,\ \theta_2 < 0,\ w_{23} \leq \theta_3,\ w_{12} + w_{31} > \theta_1,\ w_{12} + w_{23} > \theta_2,\ w_{31} + w_{23} > \theta_3$$

The following choice of the thresholds and weights, namely,

$$\theta_1 = 0.1,\ \theta_2 = -0.2,\ \theta_3 = 0.7,\ w_{12} = -0.5,\ w_{23} = 0.4,\ w_{31} = 0.7$$

satisfies the above inequalities and hence the resulting network given in Figure 5.7a stores the given two patterns. These two patterns correspond to the stable states 010 and 111 in the network as can be seen from the state transition diagram in Figure 5.7b. The energy values for different states are as follows:



(a) A 3-unit network

(b) State transition diagram

Figure 5.7 A 3-unit network and the corresponding state transition diagram. (Adapted from [Aleksander and Morton, 19901).

The energies of different states are:

$V(000) = 0.0$, $V(001) = 0.7$, $V(010) = -0.2$, $V(100) = 0.1$
$V(011) = 0.1$, $V(101) = 0.1$, $V(110) = 0.4$, $V(111) = 0.0$

Table 5.4 shows the computation of the state transitions by comparing the weighted inputs with the threshold values for each unit in each state. The entries in the parenthesis are $\left(\sum_j w_{ij} s_j <=> \theta_i\right)$.

**Table** 6.4 Computation of State Transitions for Figure 5.7

| | unit 1 | Unit 2 | Unit 3 |
|---|---|---|---|
| 000 | (0 < 0.1)<br>000 | (0 > – 0.2)<br>010 | (0 < 0.7)<br>000 |
| 001 | (0.7 > 0.1)<br>101 | (0.4 > – 0.2)<br>011 | (0 < 0.7)<br>000 |
| 010 | (– 0.5 < 0.1)<br>010 | (0 > – 0.2)<br>010 | (0.4 < 0.7)<br>010 |
| 100 | (0 < 0.1)<br>000 | (– 0.5 < – 0.2)<br>100 | (0.7 = 0.7)<br>100 |
| 011 | (0.2 > 0.1)<br>111 | (0.4 > – 0.2)<br>011 | (0.4 < 0.7)<br>010 |
| 101 | (0.7 > 0.1)<br>101 | (– 0.1 > – 0.2)<br>111 | (0.7 = 0.7)<br>100 |
| 110 | (– 0.5 < 0.1)<br>010 | (– 0.5 < – 0.2)<br>100 | (1.1 > 0.7)<br>111 |
| 111 | (0.2 > 0.1)<br>111 | (– 0.1 > – 0.2)<br>111 | (1.1 > 0.7)<br>111 |

### 5.3.6 Pattern Storage by Computation of Weights-Problem of False Energy Minima

For another choice of the values of $\theta_i$ and $w_{ij}$ which satisfies all the inequalities for the above problem of storage of the patterns 010 and 111 in a 3-unit network, there may be more than two energy minima or stable states in the network. Two of them correspond to the desired patterns and the other extra states correspond to false minima. This is illustrated for the choice of the thresholds and weights shown in **Figure** 5.6a. The corresponding state **transition** diagram is given in the Figure 5.6b. Here there are three energy minima corresponding to the three stable states 010, 100, 111.

The presence of the extra stable state may result in recalling a pattern not in the set of the desired patterns to be stored. If an approximate input is given to the units in the network, so that the network is forced into the state, say $s_1 s_2 s_3 = 000$ initially, then since this state is unstable, the dynamics of the network will eventually lead to either the state 010 (the desired pattern) or to the state 100

(See the state transition diagram in **Figure 5.6b**). Both these states are stable states and have equal probability of transition from the initial state 000. While the state 010 is the desirable pattern to be recalled, there is an equal chance that the pattern 100 may be recalled. Likewise, if the initial state is 110, then there is an equal chance that any one of the stable states 010, 100 and 111 may be recalled. The recall of the pattern 111 results in an undetectable error, as the desired pattern is 010 for the approximate input 110. The recall of the pattern 100 at any time will give as output a pattern which was not stored in the network intentionally, since in our pattern storage task we have specified only 010 and 111 as the desired patterns to be stored in the network. The stable state 100 in this case corresponds to a false (undesirable) energy minimum.

Errors in recall due to false minima can be reduced in two ways:

1. By designing the energy minima for the given patterns in an optimal way, so that the given patterns correspond to the lowest energy minima in the network.
2. By using a stochastic update of the state for each unit, instead of the deterministic update dictated by the activation values and the output function.

The issue of stochastic update will be discussed in Section 5.4, and the issue of designing energy wells by learning in Section 5.5.

**Pattern storage—Hard problems:** In the previous subsection we have discussed the effect of having more minima in the energy landscape than the number required to store the given patterns. In this section we consider the case of the so called *hard* problems of pattern storage. Let us consider the problem of storing the patterns say 000, 011, 101 and 110. By using the condition $f\left(\sum_j w_{ij} s_j - \theta_i\right) = s_i$ for each unit i, the inequalities to be satisfied to make these states stable in a 3-unit feedback network can be derived. In this case no choice of thresholds and weights can satisfy all the constraints in the inequalities. The reason is that the number of desired patterns is more than the capacity of the network, and hence they cannot be represented/stored in a feedback network with 3 units. In some cases, even if the number of desired patterns is within the capacity limit of a network, the specific patterns may not be representable in a given type (binary) of a feedback network For example, for storing the patterns 000 and 100, the following inequalities have to be satisfied by the type of network we have been considering so far.

$$\theta_1 \geq 0, \theta_2 \geq 0, \theta_3 \geq 0, \text{ and } \theta_1 < 0, w_{21} \leq \theta_2, w_{13} \leq \theta_3 \qquad (5.43)$$

The conditions on $\theta_1 < 0$ and $\theta_1 \geq 0$ cannot obviously be satisfied simultaneously by any choice of $\theta_1$. In fact any pair of patterns within a Hamming distance of 1 cannot be stored in a 3-unit network.

Pattern storage problems which cannot be represented by a feedback network of a given size, can be called hard problems. This is analogous to the hard problems in the pattern classification task for a single layer perceptron network. Hard problems in the pattern storage task are handled by introducing additional units in the feedback network. These units are called hidden units. But with hidden units it is difficult to write a set of inequalities as before to make the given patterns correspond to stable states in the feedback network. Thus the design of a network with hidden units becomes difficult due to lack of a straightforward approach for determining the weights of the network. In other words, this may be viewed as hard learning problems. We will see in Section 5.5 how this problem is addressed in Boltzmann learning law. To store a given number of patterns, a network with sufficiently large number of units may have to be considered. But in general it is difficult to know the required number of units exactly for a given number of patterns to be stored.

## 5.4 Stochastic Networks and Simulated Annealing

### 5.4.1 Stochastic Update

Error in pattern recall due to false minima can be reduced significantly if initially the desired patterns are stored (by careful training) at the lowest energy minima of a network. The error can be reduced further by using suitable activation dynamics. Let us assume that by training we have achieved a set of weights which will enable the desired patterns to be stored at the lowest energy minima. The activation dynamics is modified so that the network can also move to a state of higher energy value initially, and then to the nearest deep energy minimum. This way errors in recall due to false minima can be reduced.

It is possible to realize a transition to a higher energy state from a lower energy state by using a stochastic update in each unit instead of the deterministic update of the output function as in the Hopfield model. In a stochastic update the activation value of a unit does not decide the next output state of the unit by directly using the output function $f(x)$ as shown in Figure 5.8a. Instead, the update is expressed in probabilistic terms, like the probability of firing by the unit being greater than 0.5 if the activation value exceeds a threshold, and less than 0.5 if the activation value is less than the threshold. Note that the output function $f(x)$ is still a nonlinear function, either a hard-limiting threshold logic function or a semilinear sigmoidal function, but the function itself is applied in a stochastic manner.

Figure 5.8b shows a typical probability function that can be used for stochastic update of units. The output function itself is the binary logic function $f(x)$ shown in Figure 5.8a.