CHAPTER 2

Learning Processes

2.1 INTRODUCTION

The property that is of primary significance for a neural network is the ability of the network to *learn* from its environment, and to *improve* its performance through learning. The improvement in performance takes place over time in accordance with some prescribed measure. A neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels. Ideally, the network becomes more knowledgeable about its environment after each iteration of the learning process.

There are too many activities associated with the notion of "learning" to justify defining it in a precise manner. Moreover, the process of learning is a matter of viewpoint, which makes it all the more difficult to agree on a precise definition of the term. For example, learning as viewed by a psychologist is quite different from learning in a classroom sense. Recognizing that our particular interest is in neural networks, we use a definition of learning that is adapted from Mendel and McClaren (1970).

We define learning in the context of neural networks as:

> *Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.*

This definition of the learning process implies the following sequence of events:

1. The neural network is *stimulated* by an environment.
2. The neural network *undergoes changes* in its free parameters as a result of this stimulation.
3. The neural network *responds in a new way* to the environment because of the changes that have occurred in its internal structure.

A prescribed set of well-defined rules for the solution of a learning problem is called a *learning algorithm.*[1] As one would expect, there is no unique learning algorithm for the design of neural networks. Rather, we have a "kit of tools" represented by a diverse variety of learning algorithms, each of which offers advantages of its own. Basically, learning algorithms differ from each other in the way in which the adjust-

ment to a synaptic weight of a neuron is formulated. Another factor to be considered is the manner in which a neural network (learning machine), made up of a set of interconnected neurons, relates to its environment. In this latter context we speak of a *learning paradigm* that refers to a *model* of the environment in which the neural network operates.

Organization of the Chapter

The chapter is organized in four interrelated parts. In the first part, consisting of Sections 2.2 through 2.6, we discuss five basic learning rules: error-correction learning, memory-based learning, Hebbian learning, competitive learning, and Boltzmann learning. Error-correction learning is rooted in optimum filtering. Memory-based learning operates by memorizing the training data explicitly. Hebbian learning and competitive learning are both inspired by neurobiological considerations. Boltzmann learning is different because it is based on ideas borrowed from statistical mechanics.

The second part of the chapter explores learning paradigms. Section 2.7 discusses the credit-assignment problem, which is basic to the learning process. Sections 2.8 and 2.9 present overviews of the two fundamental learning paradigms: (1) learning *with* a teacher, and (2) learning *without* a teacher.

The third part of the chapter, consisting of Sections 2.10 through 2.12, examines the issues of learning tasks, memory, and adaptation.

The final part of the chapter, consisting of Sections 2.13 through 2.15, deals with probabilistic and statistical aspects of the learning process. Section 2.13 discusses the bias/variance dilemma. Section 2.14 discusses statistical learning theory, based on the notion of VC-dimension that provides a measure of machine capacity. Section 2.14 introduces another important concept: probably approximately correct (PAC) learning, which provides a conservative model for the learning process.

The chapter concludes with some final remarks in Section 2.16.

2.2 ERROR-CORRECTION LEARNING

To illustrate our first learning rule, consider the simple case of a neuron k constituting the only computational node in the output layer of a feedforward neural network, as depicted in Fig. 2.1a. Neuron k is driven by a *signal vector* $\mathbf{x}(n)$ produced by one or more layers of hidden neurons, which are themselves driven by an input vector (stimulus) applied to the source nodes (i.e., input layer) of the neural network. The argument n denotes discrete time, or more precisely, the time step of an iterative process involved in adjusting the synaptic weights of neuron k. The *output signal* of neuron k is denoted by $y_k(n)$. This output signal, representing the only output of the neural network, is compared to a *desired response* or *target output,* denoted by $d_k(n)$. Consequently, an *error signal,* denoted by $e_k(n)$, is produced. By definition, we thus have

$$e_k(n) = d_k(n) - y_k(n) \tag{2.1}$$

The error signal $e_k(n)$ actuates a *control mechanism,* the purpose of which is to apply a sequence of corrective adjustments to the synaptic weights of neuron k. The corrective adjustments are designed to make the output signal $y_k(n)$ come closer to the desired

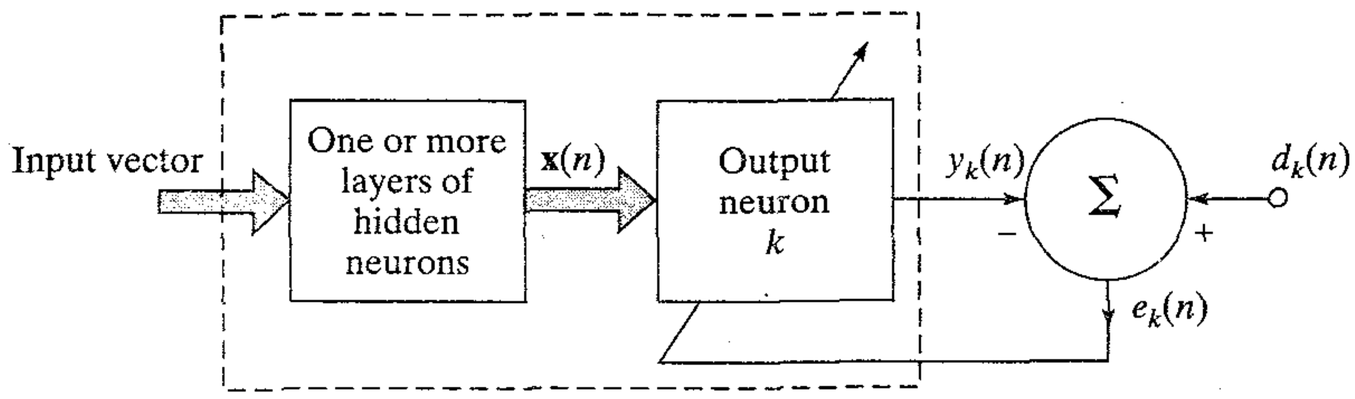

FIGURE 2.1 Illustrating error-correction learning.

response $d_k(n)$ in a step-by-step manner. This objective is achieved by minimizing a *cost function* or *index of performance*, $\mathscr{E}(n)$, defined in terms of the error signal $e_k(n)$ as:

$$\mathscr{E}(n) = \frac{1}{2} e_k^2(n) \tag{2.2}$$

That is, $\mathscr{E}(n)$ is the *instantaneous value of the error energy*. The step-by-step adjustments to the synaptic weights of neuron k are continued until the system reaches a *steady state* (i.e., the synaptic weights are essentially stabilized). At that point the learning process is terminated.

The learning process described herein is obviously referred to as *error-correction learning*. In particular, minimization of the cost function $\mathscr{E}(n)$ leads to a learning rule commonly referred to as the *delta rule* or *Widrow–Hoff rule*, named in honor of its originators (Widrow and Hoff, 1960). Let $w_{kj}(n)$ denote the value of synaptic weight w_{kj} of neuron k excited by element $x_j(n)$ of the signal vector $\mathbf{x}(n)$ at time step n. According to the delta rule, the adjustment $\Delta w_{kj}(n)$ applied to the synaptic weight w_{kj} at time step n is defined by

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \tag{2.3}$$

where η is a positive constant that determines the *rate of learning* as we proceed from one step in the learning process to another. It is therefore natural that we refer to η as the *learning-rate parameter*. In other words, the delta rule may be stated as:

The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse in question.

Keep in mind that the delta rule, as stated herein, presumes that the error signal is *directly measurable*. For this measurement to be feasible we clearly need a supply of desired response from some external source, which is directly accessible to neuron k. In other words, neuron k is *visible* to the outside world, as depicted in Fig. 2.1a. From this figure we also observe that error-correction learning is in fact *local* in nature. This is merely saying that the synaptic adjustments made by the delta rule are localized around neuron k.

Having computed the synaptic adjustment $\Delta w_{kj}(n)$, the updated value of synaptic weight w_{kj} is determined by

$$w_{kj}(n + 1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{2.4}$$

In effect, $w_{kj}(n)$ and $w_{kj}(n + 1)$ may be viewed as the *old* and *new* values of synaptic weight w_{kj}, respectively. In computational terms we may also write

$$w_{kj}(n) = z^{-1}[w_{kj}(n + 1)] \tag{2.5}$$

where z^{-1} is the *unit-delay operator*. That is, z^{-1} represents a *storage element*.

Figure 2.1b shows a signal-flow graph representation of the error-correction learning process, focusing on the activity surrounding neuron k. The input signal x_j and induced local field v_k of neuron k are referred to as the *presynaptic* and *postsynaptic signals* of the jth synapse of neuron k, respectively. From Fig. 2.1b we see that error-correction learning is an example of a *closed-loop feedback system*. From control theory we know that the stability of such a system is determined by those parameters that constitute the feedback loops of the system. In our case we only have a single feedback loop, and one of those parameters of particular interest is the learning-rate parameter η. It is therefore important that η is carefully selected to ensure that the stability or convergence of the iterative learning process is achieved. The choice of η also has a profound influence on the accuracy and other aspects of the learning process. In short, the learning-rate parameter η plays a key role in determining the performance of error-correction learning in practice.

Error-correction learning is discussed in much greater detail in Chapter 3, which discusses single-layer feedforward networks and in Chapter 4, which details multilayer feedforward networks.

2.3 MEMORY-BASED LEARNING

In *memory-based learning*, all (or most) of the past experiences are explicitly stored in a large memory of correctly classified input–output examples: $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ denotes an input vector and d_i denotes the corresponding desired response. Without loss of generality, we have restricted the desired response to be a scalar. For example, in a binary pattern classification problem there are two classes/hypotheses, denoted by $\mathscr{C}_1$ and $\mathscr{C}_2$, to be considered. In this example, the desired response d_i takes the value 0 (or -1) for class $\mathscr{C}_1$ and the value 1 for class $\mathscr{C}_2$. When classification of a test vector $\mathbf{x}_{\text{test}}$ (not seen before) is required, the algorithm responds by retrieving and analyzing the training data in a "local neighborhood" of $\mathbf{x}_{\text{test}}$.

All memory-based learning algorithms involve two essential ingredients:

- Criterion used for defining the local neighborhood of the test vector $\mathbf{x}_{test}$.
- Learning rule applied to the training examples in the local neighborhood of $\mathbf{x}_{test}$.

The algorithms differ from each other in the way in which these two ingredients are defined.

In a simple yet effective type of memory-based learning known as the *nearest neighbor rule,*[2] the local neighborhood is defined as the training example that lies in the immediate neighborhood of the test vector $\mathbf{x}_{test}$. In particular, the vector

$$\mathbf{x}'_N \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \tag{2.6}$$

is said to be the nearest neighbor of $\mathbf{x}_{test}$ if

$$\min_i d(\mathbf{x}_i, \mathbf{x}_{test}) = d(\mathbf{x}'_N, \mathbf{x}_{test}) \tag{2.7}$$

where $d(\mathbf{x}_i, \mathbf{x}_{test})$ is the Euclidean distance between the vectors $\mathbf{x}_i$ and $\mathbf{x}_{test}$. The class associated with the minimum distance, that is, vector $\mathbf{x}'_N$, is reported as the classification of $\mathbf{x}_{test}$. This rule is independent of the underlying distribution responsible for generating the training examples.

Cover and Hart (1967) have formally studied the nearest neighbor rule as a tool for pattern classification. The analysis presented therein is based on two assumptions:

- The classified examples $(\mathbf{x}_i, d_i)$ are *independently and identically distributed (iid),* according to the joint probability distribution of the example $(\mathbf{x}, d)$.
- The sample size N is infinitely large.

Under these two assumptions, it is shown that the probability of classification error incurred by the nearest neighbor rule is bounded above by twice the *Bayes probability of error,* that is, the minimum probability of error over all decision rules. Bayes probability of error is discussed in Chapter 3. In this sense, it may be said that half the classification information in a training set of infinite size is contained in the nearest neighbor, which is a surprising result.

A variant of the nearest neighbor classifier is the *k-nearest neighbor classifier,* which proceeds as follows:

- Identify the k classified patterns that lie nearest to the test vector $\mathbf{x}_{test}$ for some integer k.
- Assign $\mathbf{x}_{test}$ to the class (hypothesis) that is most frequently represented in the k nearest neighbors to $\mathbf{x}_{test}$ (i.e., use a majority vote to make the classification).

Thus the k-nearest neighbor classifier acts like an averaging device. In particular, it discriminates against a single outlier, as illustrated in Fig. 2.2 for $k = 3$. An *outlier* is an observation that is improbably large for a nominal model of interest.

In Chapter 5 we discuss another important type of memory-based classifier known as the radial-basis function network.

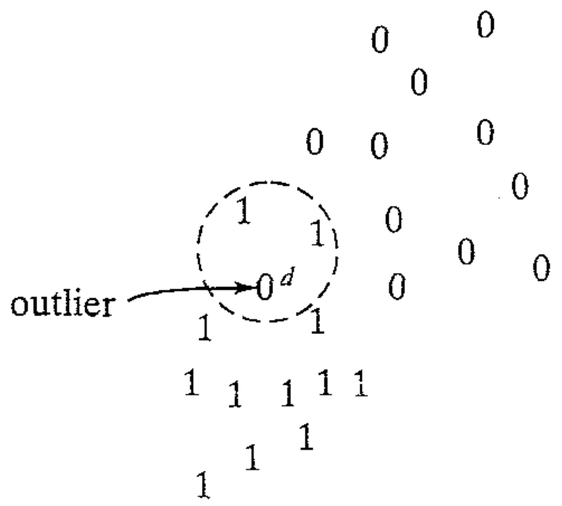

FIGURE 2.2 The area lying inside the dashed circle includes two points pertaining to class 1 and an outlier from class 0. The point *d* corresponds to the test vector $\mathbf{x}_{test}$. With $k = 3$, the *k-nearest* neighbor *classifier* assigns class 1 to point *d* even though it lies closest to the *outlier.*

2.4 HEBBIAN LEARNING

Hebb's postulate of learning is the oldest and most famous of all learning rules; it is named in honor of the neuropsychologist Hebb (1949). Quoting from Hebb's book, *The Organization of Behavior* (1949, p.62):

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

Hebb proposed this change as a basis of associative learning (at the cellular level), which would result in an enduring modification in the activity pattern of a spatially distributed "assembly of nerve cells."

This statement is made in a neurobiological context. We may expand and rephrase it as a two-part rule (Stent, 1973; Changeux and Danchin, 1976):

1. *If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.*
2. *If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.*

Such a synapse is called a *Hebbian synapse.*[3] (The original Hebb rule did not contain part 2.) More precisely, we define a Hebbian synapse as a synapse that uses a *time-dependent, highly local, and strongly interactive mechanism to increase synaptic efficiency as a function of the correlation between the presynaptic and postsynaptic activities.* From this definition we may deduce the following four key mechanisms (properties) that characterize a Hebbian synapse (Brown et al., 1990):

1. *Time-dependent mechanism.* This mechanism refers to the fact that the modifications in a Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals.

2. *Local mechanism.* By its very nature, a synapse is the transmission site where information-bearing signals (representing ongoing activity in the presynaptic and postsynaptic units) are in *spatiotemporal* contiguity. This locally available information is used by a Hebbian synapse to produce a local synaptic modification that is input specific.

3. *Interactive mechanism.* The occurrence of a change in a Hebbian synapse depends on signals on both sides of the synapse. That is, a Hebbian form of learning depends on a "true interaction" between presynaptic and postsynaptic signals in the sense that we cannot make a prediction from either one of these two activities by itself. Note also that this dependence or interaction may be deterministic or statistical in nature.

4. *Conjunctional* or *correlational mechanism.* One interpretation of Hebb's postulate of learning is that the condition for a change in synaptic efficiency is the conjunction of presynaptic and postsynaptic signals. Thus, according to this interpretation, the co-occurrence of presynaptic and postsynaptic signals (within a short interval of time) is sufficient to produce the synaptic modification. It is for this reason that a Hebbian synapse is sometimes referred to as a *conjunctional synapse.* For another interpretation of Hebb's postulate of learning, we may think of the interactive mechanism characterizing a Hebbian synapse in statistical terms. In particular, the correlation over time between presynaptic and postsynaptic signals is viewed as being responsible for a synaptic change. Accordingly, a Hebbian synapse is also referred to as a *correlational synapse.* Correlation is indeed the basis of learning (Eggermont, 1990).

Synaptic Enhancement and Depression

The definition of a Hebbian synapse presented here does not include additional processes that may result in weakening of a synapse connecting a pair of neurons. Indeed, we may generalize the concept of a Hebbian modification by recognizing that positively correlated activity produces synaptic strengthening, and that either uncorrelated or negatively correlated activity produces synaptic weakening (Stent, 1973). Synaptic depression may also be of a noninteractive type. Specifically, the interactive condition for synaptic weakening may simply be noncoincident presynaptic or postsynaptic activity.

We may go one step further by classifying synaptic modifications as *Hebbian, anti-Hebbian, and non-Hebbian* (Palm, 1982). According to this scheme, a Hebbian synapse increases its strength with positively correlated presynaptic and postsynaptic signals, and decreases its strength when these signals are either uncorrelated or negatively correlated. Conversely, an anti-Hebbian synapse weakens positively correlated presynaptic and postsynaptic signals, and strengthens negatively correlated signals. In both Hebbian and anti-Hebbian synapses, however, the modification of synaptic efficiency relies on a mechanism that is time-dependent, highly local, and strongly interactive in nature. In that sense, an anti-Hebbian synapse is still Hebbian in nature, though not in function. A non-Hebbian synapse, on the other hand, does not involve a Hebbian mechanism of either kind.

Mathematical Models of Hebbian Modifications

To formulate Hebbian learning in mathematical terms, consider a synaptic weight w_{kj} of neuron k with presynaptic and postsynaptic signals denoted by x_j and y_k, respectively. The adjustment applied to the synaptic weight w_{kj} at time step n is expressed in the general form

$$\Delta w_{kj}(n) = F(y_k(n), x_j(n)) \tag{2.8}$$

where $F(\cdot,\cdot)$ is a function of both postsynaptic and presynaptic signals. The signals $x_j(n)$ and $y_k(n)$ are often treated as dimensionless. The formula of Eq. (2.8) admits many forms, all of which qualify as Hebbian. In what follows, we consider two such forms.

Hebb's hypothesis. The simplest form of Hebbian learning is described by

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n) \tag{2.9}$$

where η is a positive constant that determines the *rate of learning*. Equation (2.9) clearly emphasizes the correlational nature of a Hebbian synapse. It is sometimes referred to as the *activity product rule*. The top curve of Fig. 2.3 shows a graphical representation of Eq. (2.9) with the change Δw_{kj} plotted versus the output signal (postsynaptic activity) y_k. From this representation we see that the repeated application of the input signal (presynaptic activity) x_j leads to an increase in y_k and therefore *exponential growth* that finally drives the synaptic connection into saturation. At that point no information will be stored in the synapse and selectivity is lost.

Covariance hypothesis. One way of overcoming the limitation of Hebb's hypothesis is to use the *covariance hypothesis* introduced in Sejnowski (1977a, b). In this hypothesis, the presynaptic and postsynaptic signals in Eq. (2.9) are replaced by the departure of presynaptic and postsynaptic signals from their respective average values over a certain time interval. Let $\bar{x}$ and $\bar{y}$ denote the *time-averaged values* of the presynaptic signal x_j and postsynaptic signal y_k, respectively. According to the covariance hypothesis, the adjustment applied to the synaptic weight w_{kj} is defined by

$$\Delta w_{kj} = \eta(x_j - \bar{x})(y_k - \bar{y}) \tag{2.10}$$

where η is the learning-rate parameter. The average values x and y constitute presynaptic and postsynaptic thresholds, which determine the sign of synaptic modification. In particular, the covariance hypothesis allows for the following:

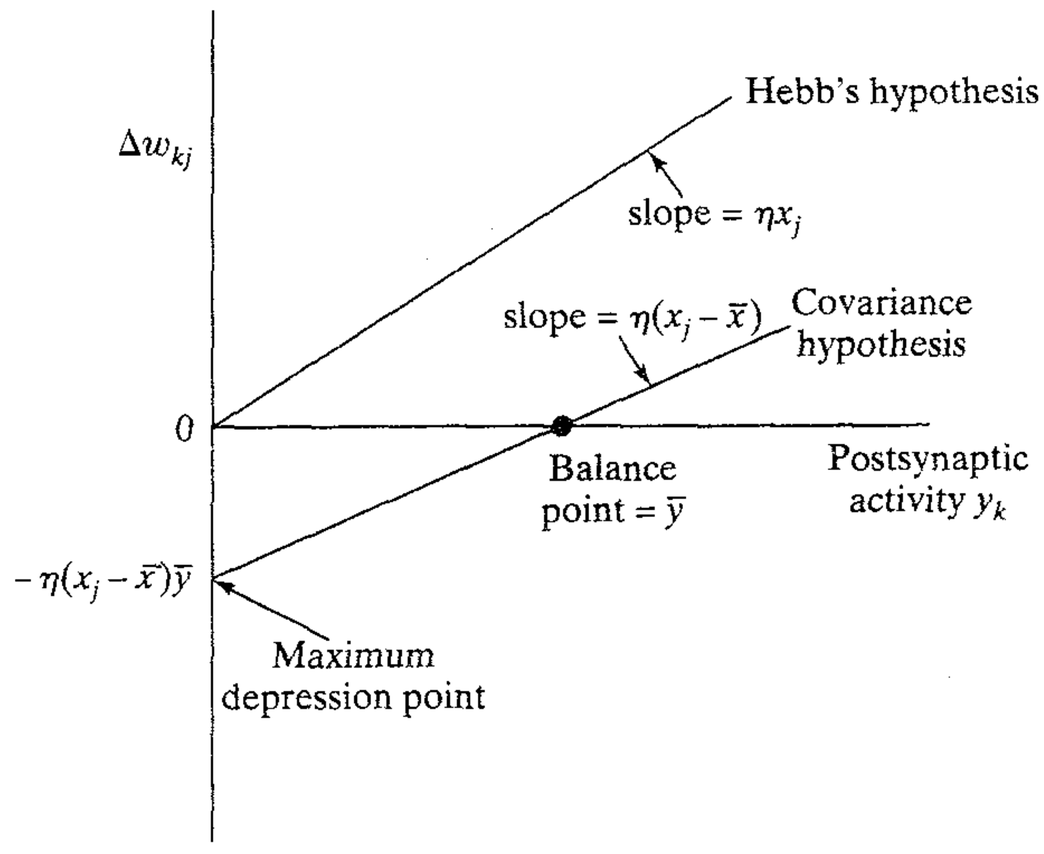

FIGURE 2.3 Illustration of Hebb's hypothesis and the covariance hypothesis.

- Convergence to a nontrivial state, which is reached when $x_k = \bar{x}$ or $y_j = \bar{y}$.
- Prediction of both synaptic *potentiation* (i.e., increase in synaptic strength) and synaptic *depression* (i.e., decrease in synaptic strength).

Figure 2.3 illustrates the difference between Hebb's hypothesis and the covariance hypothesis. In both cases the dependence of Δw_{kj} on y_k is linear; however, the intercept with the y_k-axis in Hebb's hypothesis is at the origin, whereas in the covariance hypothesis it is at $y_k = \bar{y}$.

We make the following important observations from Eq. (2.10):

1. Synaptic weight w_{kj} is enhanced if there are sufficient levels of presynaptic and postsynaptic activities, that is, the conditions $x_j > \bar{x}$ and $y_k > \bar{y}$ are both satisfied.
2. Synaptic weight w_{kj} is depressed if there is either
   - a presynaptic activation (i.e., $x_j > \bar{x}$) in the absence of sufficient postsynaptic activation (i.e., $y_k < \bar{y}$), or
   - a postsynaptic activation (i.e., $y_k > \bar{y}$) in the absence of sufficient presynaptic activation (i.e., $x_j < \bar{x}$).

This behavior may be regarded as a form of temporal competition between the incoming patterns.

There is strong physiological evidence[4] for Hebbian learning in the area of the brain called the *hippocampus*. The hippocampus plays an important role in certain aspects of learning or memory. This physiological evidence makes Hebbian learning all the more appealing.

2.5 COMPETITIVE LEARNING

In *competitive learning,*[5] as the name implies, the output neurons of a neural network compete among themselves to become active (fired). Whereas in a neural network based on Hebbian learning several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at any one time. It is this feature that makes competitive learning highly suited to discover statistically salient features that may be used to classify a set of input patterns.

There are three basic elements to a competitive learning rule (Rumelhart and Zipser, 1985):

- A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore *respond differently* to a given set of input patterns.
- A *limit* imposed on the "strength" of each neuron.
- A mechanism that permits the neurons to *compete* for the right to respond to a given subset of inputs, such that only *one* output neuron, or only one neuron per group, is active (i.e., "on") at a time. The neuron that wins the competition is called a *winner-takes-all neuron.*

Accordingly the individual neurons of the network learn to specialize on ensembles of similar patterns; in so doing they become *feature detectors* for different classes of input patterns.

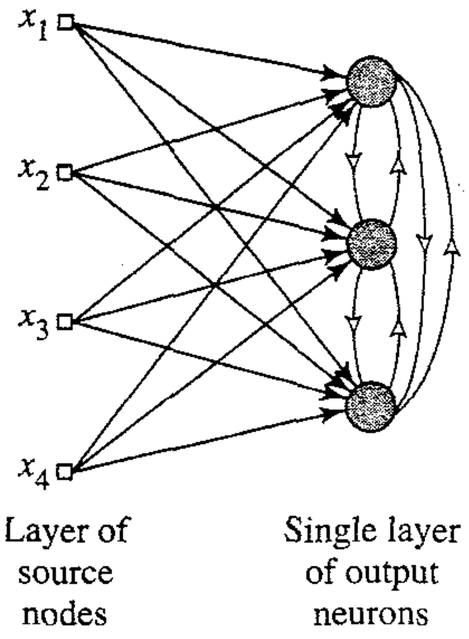

FIGURE 2.4 Architectural graph of a simple competitive learning network with feedforward (excitatory) connections from the source nodes to the neurons, and lateral (inhibitory) connections among the neurons; the lateral connections are signified by open arrows.

In the simplest form of competitive learning, the neural network has a single layer of output neurons, each of which is fully connected to the input nodes. The network may include feedback connections among the neurons, as indicated in Fig. 2.4. In the network architecture described herein, the feedback connections perform *lateral inhibition,*[6] with each neuron tending to inhibit the neuron to which it is laterally connected. In contrast, the feedforward synaptic connections in the network of Fig. 2.4 are all *excitatory*.

For a neuron k to be the winning neuron, its induced local field v_k for a specified input pattern $\mathbf{x}$ must be the largest among all the neurons in the network. The output signal y_k of winning neuron k is set equal to one; the output signals of all the neurons that lose the competition are set equal to zero. We thus write

$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all } j, j \neq k \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

where the induced local field v_k represents the combined action of all the forward and feedback inputs to neuron k.

Let w_{kj} denote the synaptic weight connecting input node j to neuron k. Suppose that each neuron is allotted a *fixed* amount of synaptic weight (i.e., all synaptic weights are positive), which is distributed among its input nodes; that is,

$$\sum_j w_{kj} = 1 \qquad \text{for all } k \tag{2.12}$$

A neuron then learns by shifting synaptic weights from its inactive to active input nodes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron. If a particular neuron wins the competition, each input node of that neuron relinquishes some proportion of its synaptic weight, and the weight relinquished is then distributed equally among the active input nodes. According to the standard *competitive learning rule,* the change Δw_{kj} applied to synaptic weight w_{kj} is defined by

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron } k \text{ wins the competition} \\ 0 & \text{if neuron } k \text{ loses the competition} \end{cases} \tag{2.13}$$

where η is the learning-rate parameter. This rule has the overall effect of moving the synaptic weight vector $\mathbf{w}_k$ of winning neuron k toward the input pattern $\mathbf{x}$.

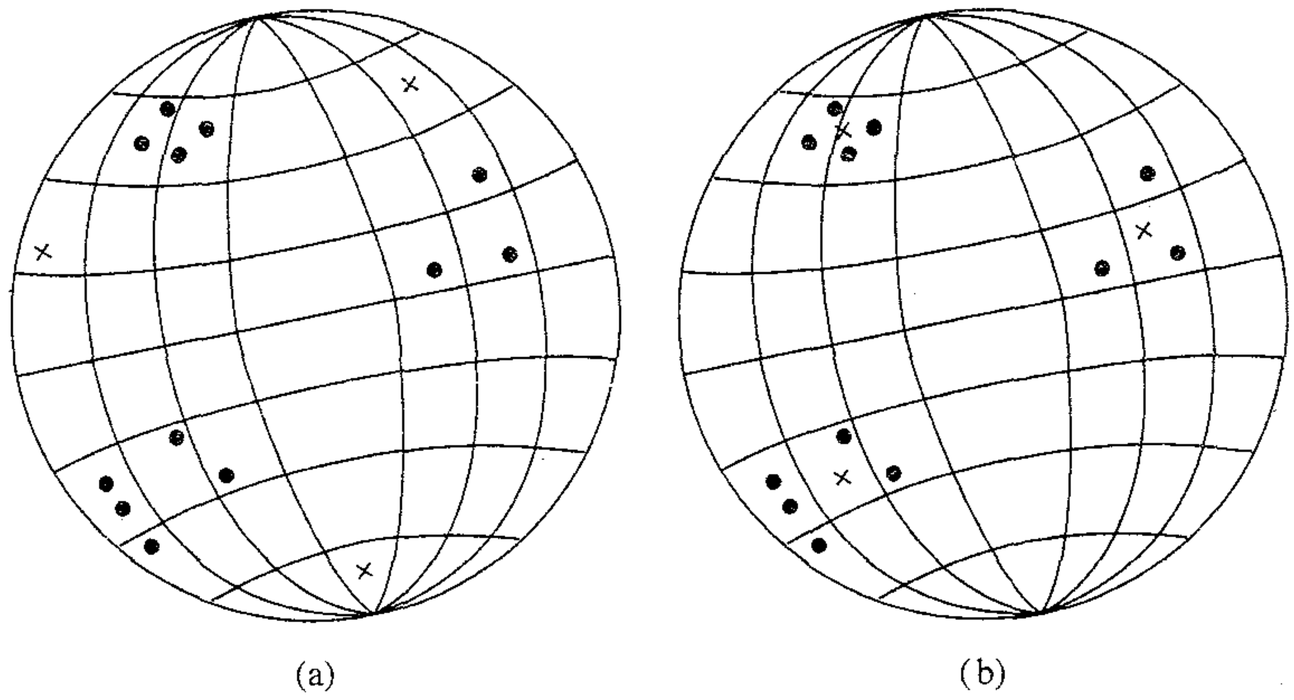

FIGURE 2.5 Geometric interpretation of the competitive learning process. The dots represent the input vectors, and the crosses represent the synaptic weight vectors of three output neurons. (a) Initial state of the network. (b) Final state of the network.

We may use the geometric analogy depicted in Fig. 2.5 to illustrate the essence of competitive learning (Rumelhart and Zipser, 1985). It is assumed that each input pattern (vector) $\mathbf{x}$ has some constant Euclidean length so that we may view it as a point on an N-dimensional unit sphere where N is the number of input nodes. N also represents the dimension of each synaptic weight vector $\mathbf{w}_k$. It is further assumed that all neurons in the network are constrained to have the same Euclidean length (norm), as shown by

$$\sum_j w_{kj}^2 = 1 \qquad \text{for all } k \tag{2.14}$$

When the synaptic weights are properly scaled they form a set of vectors that fall on the same N-dimensional unit sphere. In Fig. 2.5a we show three natural groupings (clusters) of the stimulus patterns represented by dots. This figure also includes a possible initial state of the network (represented by crosses) that may exist before learning. Figure 2.5b shows a typical final state of the network that results from the use of competitive learning. In particular, each output neuron has discovered a cluster of input patterns by moving its synaptic weight vector to the center of gravity of the discovered cluster (Rumelhart and Zipser, 1985; Hertz et al., 1991). This figure illustrates the ability of a neural network to perform *clustering* through competitive learning. However, for this function to be performed in a "stable" fashion the input patterns must fall into sufficiently distinct groupings to begin with. Otherwise the network may be unstable because it will no longer respond to a given input pattern with the same output neuron.

2.6 BOLTZMANN LEARNING

The Boltzmann learning rule, named in honor of Ludwig Boltzmann, is a stochastic learning algorithm derived from ideas rooted in statistical mechanics.[7] A neural net-

work designed on the basis of the Boltzmann learning rule is called a *Boltzmann machine* (Ackley et al., 1985; Hinton and Sejnowski, 1986).

In a Boltzmann machine the neurons constitute a recurrent structure, and they operate in a binary manner since, for example, they are either in an "on" state denoted by $+1$ or in an "off" state denoted by -1. The machine is characterized by an *energy function, E*, the value of which is determined by the particular states occupied by the individual neurons of the machine, as shown by

$$E = -\frac{1}{2}\sum_{j}\sum_{\substack{k \\ j \neq k}} w_{kj} x_k x_j \tag{2.15}$$

where x_j is the state of neuron j, and w_{kj} is the synaptic weight connecting neuron j to neuron k. The fact that $j \neq k$ means simply that none of the neurons in the machine has self-feedback. The machine operates by choosing a neuron at random—for example, neuron k—at some step of the learning process, then flipping the state of neuron k from state x_k to state $-x_k$ at some temperature T with probability

$$P(x_k \rightarrow -x_k) = \frac{1}{1 + \exp(-\Delta E_k / T)} \tag{2.16}$$

where ΔE_k is the *energy change* (i.e., the change in the energy function of the machine) resulting from such a flip. Notice that T is not a physical temperature, but rather a *pseudotemperature,* as explained in Chapter 1. If this rule is applied repeatedly, the machine will reach *thermal equilibrium*.

The neurons of a Boltzmann machine partition into two functional groups: *visible* and *hidden*. The visible neurons provide an interface between the network and the environment in which it operates, whereas the hidden neurons always operate freely. There are two modes of operation to be considered:

- *Clamped condition,* in which the visible neurons are all clamped onto specific states determined by the environment.
- *Free-running condition,* in which all the neurons (visible and hidden) are allowed to operate freely.

Let ρ_{kj}^{+} denote the *correlation* between the states of neurons j and k, with the network in its clamped condition. Let ρ_{kj}^{-} denote the *correlation* between the states of neurons j and k with the network in its free-running condition. Both correlations are averaged over all possible states of the machine when it is in thermal equilibrium. Then, according to the *Boltzmann learning rule,* the change Δw_{kj} applied to the synaptic weight w_{kj} from neuron j to neuron k is defined by (Hinton and Sejnowski, 1986)

$$\Delta w_{kj} = \eta(\rho_{kj}^{+} - \rho_{kj}^{-}), \qquad j \neq k \tag{2.17}$$

where η is a learning-rate parameter. Note that both ρ_{kj}^{+} and ρ_{kj}^{-} range in value from -1 to $+1$.

A brief review of statistical mechanics is presented in Chapter 11; in that chapter we also present a detailed treatment of the Boltzmann machine and other stochastic machines.

2.7 CREDIT-ASSIGNMENT PROBLEM

When studying learning algorithms for distributed systems, it is useful to consider the notion of *credit assignment* (Minsky, 1961). Basically, the credit-assignment problem is the problem of assigning *credit* or *blame* for overall outcomes to each of the internal decisions made by a learning machine and which contributed to those outcomes. (The credit assignment problem is also referred to as the *loading problem,* the problem of "loading" a given set of training data into the free parameters of the network.)

In many cases the dependence of outcomes on internal decisions is mediated by a sequence of actions taken by the learning machine. In other words, internal decisions affect which particular actions are taken, and then the actions, not the internal decisions, directly influence overall outcomes. In these situations, we may decompose the credit-assignment problem into two subproblems (Sutton, 1984):

1. The assignment of credit for outcomes to actions. This is called the *temporal credit-assignment problem* in that it involves the instants of time *when* the actions that deserve credit were actually taken.
2. The assignment of credit for actions to internal decisions. This is called the *structural credit-assignment problem* in that it involves assigning credit to the *internal structures* of actions generated by the system.

The structural credit-assignment problem is relevant in the context of a multicomponent learning machine when we must determine precisely which particular component of the system should have its behavior altered and by how much in order to improve overall system performance. On the other hand, the temporal credit-assignment problem is relevant when there are many actions taken by a learning machine that result in certain outcomes, and we must determine which of these actions were responsible for the outcomes. The combined temporal and structural credit-assignment problem faces any distributed learning machine that attempts to improve its performance in situations involving temporally extended behavior (Williams, 1988).

The credit-assignment problem, for example, arises when error-correction learning is applied to a multilayer feedforward neural network. The operation of each hidden neuron, as well as that of each output neuron in such a network is important to its correct overall operation on a learning task of interest. That is, in order to solve the prescribed task the network must assign certain forms of behavior to all of its neurons through the specification of error-correction learning. With this background in mind, consider the situation described in Fig. 2.1a. Since the output neuron k is visible to the outside world, it is possible to supply a desired response to this neuron. As far as the output neuron is concerned, it is a straightforward matter to adjust the synaptic weights of the output neuron in accordance with error-correction learning, as outlined in Section 2.2. But how do we assign credit or blame for the action of the hidden neurons when the error-correction learning process is used to adjust the respective synaptic weights of these neurons? The answer to this fundamental question requires more detailed attention; it is presented in Chapter 4, where algorithmic details of the design of multilayer feedforward neural networks are described.

2.8 LEARNING WITH A TEACHER

We now turn our attention to learning paradigms. We begin by considering *learning with a teacher,* which is also referred to as *supervised learning*. Figure 2.6 shows a block diagram that illustrates this form of learning. In conceptual terms, we may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of *input–output examples*. The environment is, however, *unknown* to the neural network of interest. Suppose now that the teacher and the neural network are both exposed to a training vector (i.e., example) drawn from the environment. By virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Indeed, the desired response represents the optimum action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. The *error signal* is defined as the difference between the desired response and the actual response of the network. This adjustment is carried out iteratively in a step-by-step fashion with the aim of eventually making the neural network *emulate* the teacher; the emulation is presumed to be optimum in some statistical sense. In this way knowledge of the environment available to the teacher is transferred to the neural network through training as fully as possible. When this condition is reached, we may then dispense with the teacher and let the neural network deal with the environment completely by itself.

The form of supervised learning we have just described is the error-correction learning discussed previously in Section 2.2. It is a closed-loop feedback system, but the unknown environment is not in the loop. As a performance measure for the system we may think in terms of the mean-square error or the sum of squared errors over the training sample, defined as a function of the free parameters of the system. This function may be visualized as a multidimensional *error-performance surface* or simply *error surface,* with the free parameters as coordinates. The true error surface is *averaged* over all possible input–output examples. Any given operation of the system under the

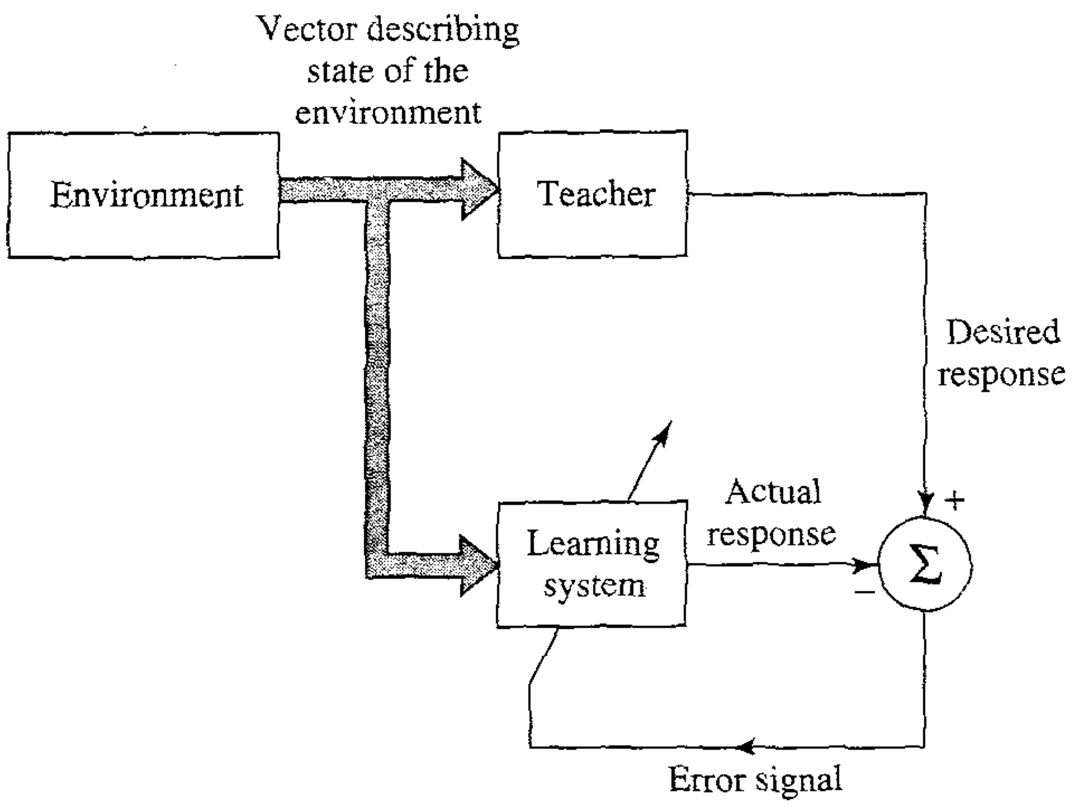

FIGURE 2.6 Block diagram of learning with a teacher.

teacher's supervision is represented as a point on the error surface. For the system to improve performance over time and therefore learn from the teacher, the operating point has to move down successively toward a minimum point of the error surface; the minimum point may be a local minimum or a global minimum. A supervised learning system is able to do this with the useful information it has about the *gradient* of the error surface corresponding to the current behavior of the system. The gradient of an error surface at any point is a vector that points in the direction of *steepest descent.* In fact, in the case of supervised learning from examples, the system may use an *instantaneous estimate* of the gradient vector, with the example indices presumed to be those of time. The use of such an estimate results in a motion of the operating point on the error surface that is typically in the form of a "random walk." Nevertheless, given an algorithm designed to minimize the cost function, an adequate set of input–output examples, and enough time permitted to do the training, a supervised learning system is usually able to perform such tasks as pattern classification and function approximation.

2.9 LEARNING WITHOUT A TEACHER

In supervised learning, the learning process takes place under the tutelage of a teacher. However, in the paradigm known as *learning without a teacher,* as the name implies, there is *no* teacher to oversee the learning process. That is to say, there are no labeled examples of the function to be learned by the network. Under this second paradigm, two subdivisions are identified:

1. Reinforcement learning/Neurodynamic programming

In *reinforcement learning,*[8] the learning of an input–output mapping is performed through continued interaction with the environment in order to minimize a scalar index of performance. Figure 2.7 shows the block diagram of one form of a reinforcement learning system built around a *critic* that converts a *primary reinforcement signal* received from the environment into a higher quality reinforcement signal called the *heuristic reinforcement signal,* both of which are scalar inputs (Barto et al., 1983). The

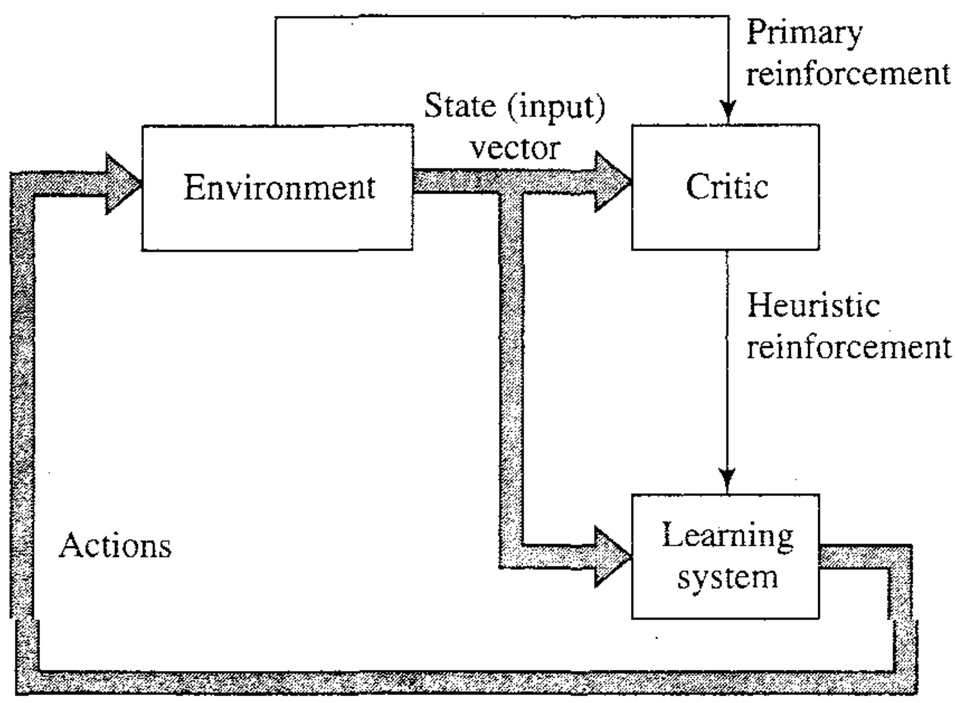

FIGURE 2.7 Block diagram of reinforcement learning.

system is designed to learn under *delayed reinforcement,* which means that the system observes a temporal sequence of stimuli (i.e., state vectors) also received from the environment, which eventually result in the generation of the heuristic reinforcement signal. The goal of learning is to minimize a *cost-to-go function,* defined as the expectation of the cumulative cost of *actions* taken over a sequence of steps instead of simply the immediate cost. It may turn out that certain actions taken earlier in that sequence of time steps are in fact the best determinants of overall system behavior. The function of the *learning machine,* which constitutes the second component of the system, is to *discover* these actions and to feed them back to the environment.

Delayed-reinforcement learning is difficult to perform for two basic reasons:

- There is no teacher to provide a desired response at each step of the learning process.
- The delay incurred in the generation of the primary reinforcement signal implies that the learning machine must solve a *temporal credit assignment problem.* By this we mean that the learning machine must be able to assign credit and blame individually to each action in the sequence of time steps that led to the final outcome, while the primary reinforcement may only evaluate the outcome.

Notwithstanding these difficulties, delayed-reinforcement learning is very appealing. It provides the basis for the system to interact with its environment, thereby developing the ability to learn to perform a prescribed task solely on the basis of the outcomes of its experience that result from the interaction.

Reinforcement learning is closely related to *dynamic programming,* which was developed by Bellman (1957) in the context of optimal control theory. Dynamic programming provides the mathematical formalism for sequential decision making. By casting reinforcement learning within the framework of dynamic programming, the subject matter becomes all the richer for it, as demonstrated in Bertsekas and Tsitsiklis (1996). An introductory treatment of dynamic programming and its relationship to reinforcement learning is presented in Chapter 12.

2. Unsupervised Learning

In *unsupervised* or *self-organized* learning there is no external teacher or critic to oversee the learning process, as indicated in Fig. 2.8. Rather, provision is made for a *task-independent measure* of the quality of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representations for encoding features of the input and thereby to create new classes automatically (Becker, 1991).

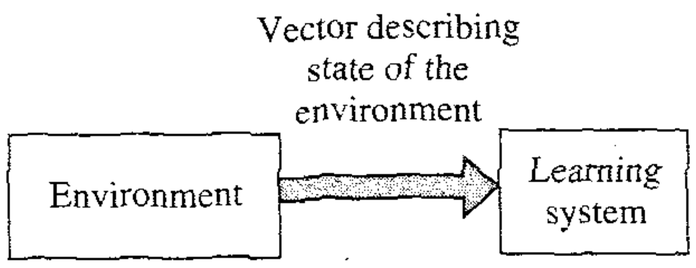

FIGURE 2.8 Block diagram of unsupervised learning.

To perform unsupervised learning we may use a competitive learning rule. For example, we may use a neural network that consists of two layers—an input layer and a competitive layer. The input layer receives the available data. The competitive layer consists of neurons that compete with each other (in accordance with a learning rule) for the "opportunity" to respond to features contained in the input data. In its simplest form, the network operates in accordance with a "winner-takes-all" strategy. As described in Section 2.5, in such a strategy the neuron with the greatest total input "wins" the competition and turns on; all the other neurons then switch off.

Different algorithms for unsupervised learning are described in Chapters 8 through 11.

2.10 LEARNING TASKS

In previous sections of this chapter we have discussed different learning algorithms and learning paradigms. In this section, we describe some basic learning tasks. The choice of a particular learning algorithm is influenced by the learning task that a neural network is required to perform. In this context we identify six learning tasks that apply to the use of neural networks in one form or another.

Pattern Association

An *associative memory* is a brainlike distributed memory that learns by *association*. Association has been known to be a prominent feature of human memory since Aristotle, and all models of cognition use association in one form or another as the basic operation (Anderson, 1995).

Association takes one of two forms: *autoassociation* or *heteroassociation*. In autoassociation, a neural network is required to *store* a set of patterns (vectors) by repeatedly presenting them to the network. The network is subsequently presented a partial description or distorted (noisy) version of an original pattern stored in it, and the task is to *retrieve* (*recall*) that particular pattern. Heteroassociation differs from autoassociation in that an arbitrary set of input patterns is *paired* with another arbitrary set of output patterns. Autoassociation involves the use of unsupervised learning, whereas the type of learning involved in heteroassociation is supervised.

Let $\mathbf{x}_k$ denote a *key pattern* (vector) applied to an associative memory and $\mathbf{y}_k$ denote a *memorized pattern* (vector). The pattern association performed by the network is described by

$$\mathbf{x}_k \rightarrow \mathbf{y}_k, \qquad k = 1, 2, \ldots, q \tag{2.18}$$

where q is the number of patterns stored in the network. The key pattern $\mathbf{x}_k$ acts as a stimulus that not only determines the storage location of memorized pattern $\mathbf{y}_k$, but also holds the key for its retrieval.

In an autoassociative memory, $\mathbf{y}_k = \mathbf{x}_k$, so the input and output (data) spaces of the network have the same dimensionality. In a heteroassociative memory, $\mathbf{y}_k \neq \mathbf{x}_k$; hence, the dimensionality of the output space in this second case may or may not equal the dimensionality of the input space.

There are two phases involved in the operation of an associative memory: