which in turn applies a correction to the adjustable parameters of the model. It also supplies an output signal for transfer to the next level of neural processing for interpretation. By repeating this operation on a level-by-level basis, the information processed by the system tends to be of progressively higher quality (Mead, 1990).

Fill in the details of the level of signal processing next to that described in Fig. P2.21.

Statistical learning theory

2.22 Following a procedure similar to that described for deriving Eq. (2.62) from (2.61), derive the formula for the ensemble-averaged function $L_{av}(f(\mathbf{x}), F(\mathbf{x}, \mathcal{T}))$ defined in Eq. (2.66).

2.23 In this problem we wish to calculate the VC dimension of a rectangular region aligned with one of the axes in a plane. Show that the VC dimension of this concept is four. You may do this by considering the following:
(a) Four points in a plane and a dichotomy realized by an axis-aligned rectangle.
(b) Four points in a plane, for which there is no realizable dichotomy by an axis-aligned rectangle.
(c) Five points in a plane, for which there is also no realizable dichotomy by an axis-aligned rectangle.

2.24 Consider a linear binary pattern classifier whose input vector **x** has dimension m. The first element of the vector **x** is constant and set to unity so that the corresponding weight of the classifier introduces a bias. What is the VC dimension of the classifier with respect to the input space?

2.25 The inequality (2.97) defines a bound on the rate of uniform convergence, which is basic to the principle of empirical risk minimization.
(a) Justify the validity of Eq. (2.98), assuming that the inequality (2.97) holds.
(b) Derive Eq. (2.99) that defines the confidence interval ϵ_1.

2.26 Continuing with Example 2.3, show that the four uniformly spaced points of Fig. P2.26 cannot be shattered by the one parameter family of indicator functions $f(x,a)$, $a \in \mathbb{R}$.

2.27 Discuss the relationship between the bias-variance dilemma and structural risk minimization in the context of nonlinear regression.

2.28 (a) An algorithm used to train a multilayer feedforward network whose neurons use a sigmoid function is PAC learnable. Justify the validity of this statement.
(b) Can you make a similar statement for an arbitrary neural network whose neurons use a threshold activation function? Justify your answer.

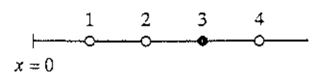

FIGURE P2.26

Single-Layer Perceptrons

3.1 INTRODUCTION

In the formative years of neural networks (1943–1958), several researchers stand out for their pioneering contributions:

- McCulloch and Pitts (1943) for introducing the idea of neural networks as computing machines.
- Hebb (1949) for postulating the first rule for self-organized learning.
- Rosenblatt (1958) for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).

The impact of the McCulloch–Pitts paper on neural networks was highlighted in Chapter 1. The idea of Hebbian learning was discussed at some length in Chapter 2. In this chapter we discuss Rosenblatt's *perceptron.*

The perceptron is the simplest form of a neural network used for the classification of patterns said to be *linearly separable* (i.e., patterns that lie on opposite sides of a hyperplane). Basically, it consists of a single neuron with adjustable synaptic weights and bias. The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt (1958, 1962) for his perceptron brain model.[1] Indeed, Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. The proof of convergence of the algorithm is known as the *perceptron convergence theorem.* The perceptron built around a *single neuron* is limited to performing pattern classification with only two classes (hypotheses). By expanding the output (computation) layer of the perceptron to include more than one neuron, we may correspondingly form classification with more than two classes. However, the classes have to be linearly separable for the perceptron to work properly. The important point is that insofar as the basic theory of the perceptron as a pattern classifier is concerned, we need consider only the case of a single neuron. The extension of the theory to the case of more than one neuron is trivial.

The single neuron also forms the basis of an *adaptive filter*, a functional block that is basic to the ever-expanding subject of *signal processing.* The development of adaptive filtering owes much to the classic paper of Widrow and Hoff (1960) for pioneering the so-called *least-mean-square (LMS) algorithm*, also known as the *delta rule*. The LMS algorithm is simple to implement yet highly effective in application. Indeed, it is the workhorse of *linear* adaptive filtering, linear in the sense that the neuron operates in its linear mode. Adaptive filters have been successfully applied in such diverse fields as antennas, communication systems, control systems, radar, sonar, seismology, and biomedical engineering (Widrow and Stearns, 1985; Haykin, 1996).

The LMS algorithm and the perceptron are naturally related. It is therefore proper for us to study them together in one chapter.

Organization of the Chapter

The chapter is organized in two parts. The first part, consisting of Sections 3.2 through 3.7 deals with linear adaptive filters and the LMS algorithm. The second part, consisting of Sections 3.8 through 3.10 deals with Rosenblatt's perceptron. From a presentation point of view, we find it more convenient to discuss linear adaptive filters first and then Rosenblatt's perceptron, reversing the historical order in which they appeared.

In Section 3.2 we address the adaptive filtering problem, followed by Section 3.3, a review of three unconstrained optimization techniques: the method of steepest descent, Newton's method, and Gauss-Newton method, which are particularly relevant to the study of adaptive filters. In Section 3.4 we discuss a linear least-squares filter, which asymptotically approaches the Wiener filter as the data length increases. The Wiener filter provides an ideal framework for the performance of linear adaptive filters operating in a stationary environment. In Section 3.5 we describe the LMS algorithm, including a discussion of its virtues and limitations. In Section 3.6 we explore the idea of learning curves commonly used to assess the performance of adaptive filters. This is followed by a discussion of annealing schedules for the LMS algorithm in Section 3.7.

Then moving on to Rosenblatt's perceptron, Section 3.8 presents some basic considerations involved in its operation. In Section 3.9 we describe the algorithm for adjusting the synaptic weight vector of the perceptron for pattern classification of linearly separable classes, and demonstrate convergence of the algorithm. In Section 3.10 we consider the relationship between the perceptron and the Bayes classifier for a Gaussian environment.

The chapter concludes with summary and discussion in Section 3.11.

3.2 ADAPTIVE FILTERING PROBLEM

Consider a *dynamical system*, the mathematical characterization of which is *unknown*. All that we have available on the system is a set of labeled input–output data generated by the system at discrete instants of time at some uniform rate. Specifically, when an m-dimensional stimulus $\mathbf{x}(i)$ is applied across m input nodes of

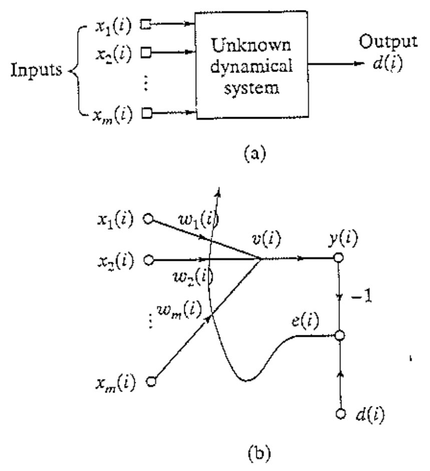

FIGURE 3.1 (a) Unknown dynamical system. (b) Signal-flow graph of adaptive model for the system.

the system, the system responds by producing a scalar output $d(i)$, where $i = 1, 2, \dots, n, \dots$ as depicted in Fig. 3.1a. Thus, the external behavior of the system is described by the data set

$$\mathcal{T}: \{\mathbf{x}(i), d(i); i = 1, 2, \dots, n, \dots\} \tag{3.1}$$

where

$$\mathbf{x}(i) = [x_1(i), x_2(i), \dots, x_m(i)]^T$$

The samples comprising $\mathcal{T}$ are identically distributed according to an unknown probability law. The dimension m pertaining to the input vector $\mathbf{x}(i)$ is referred to as the *dimensionality of the input space* or simply as *dimensionality*.

The stimulus $\mathbf{x}(i)$ can arise in one of two fundamentally different ways, one spatial and the other temporal:

- The m elements of $\mathbf{x}(i)$ originate at different points in space; in this case we speak of $\mathbf{x}(i)$ as a *snapshot* of data.
- The m elements of $\mathbf{x}(i)$ represent the set of present and $(m-1)$ past values of some excitation that are *uniformly spaced in time*.

The problem we address is how to design a multiple input-single output *model* of the unknown dynamical system by building it around a single linear neuron. The neuronal model operates under the influence of an algorithm that *controls* necessary adjustments to the synaptic weights of the neuron, with the following points in mind:

- The algorithm starts from an *arbitrary setting* of the neuron's synaptic weights.
- Adjustments to the synaptic weights, in response to statistical variations in the system's behavior, are made on a *continuous* basis (i.e., time is incorporated into the constitution of the algorithm).

- Computations of adjustments to the synaptic weights are completed inside a time interval that is one sampling period long.

The neuronal model described is referred to as an *adaptive filter*. Although the description is presented in the context of a task clearly recognized as one of *system identification*, the characterization of the adaptive filter is general enough to have wide application.

Figure 3.1b shows a signal-flow graph of the adaptive filter. Its operation consists of two continuous processes:

1. *Filtering process*, which involves the computation of two signals:
   - An output, denoted by $y(i)$, that is produced in response to the m elements of the stimulus vector $\mathbf{x}(i)$, namely, $x_1(i), x_2(i), \ldots, x_m(i)$.
   - An error signal, denoted by $e(i)$, that is obtained by comparing the output $y(i)$ to the corresponding output $d(i)$ produced by the unknown system. In effect, $d(i)$ acts as a *desired response* or *target signal.*
2. *Adaptive process*, which involves the automatic adjustment of the synaptic weights of the neuron in accordance with the error signal $e(i)$.

Thus, the combination of these two processes working together constitutes a *feedback loop* acting around the neuron.

Since the neuron is linear, the output $y(i)$ is exactly the same as the induced local field $v(i)$; that is,

$$y(i) = v(i) = \sum_{k=1}^{m} w_k(i)x_k(i) \tag{3.2}$$

where $w_1(i), w_2(i), \ldots, w_m(i)$ are the m synaptic weights of the neuron, measured at time i. In matrix form we may express $y(i)$ as an inner product of the vectors $\mathbf{x}(i)$ and $\mathbf{w}(i)$ as follows:

$$y(i) = \mathbf{x}^T(i)\mathbf{w}(i) \tag{3.3}$$

where

$$\mathbf{w}(i) = [w_1(i), w_2(i), \ldots, w_m(i)]^T$$

Note that the notation for a synaptic weight has been simplified here by *not* including an additional subscript to identify the neuron, since we only have a single neuron to deal with. This practice is followed throughout the chapter. The neuron's output $y(i)$ is compared to the corresponding output $d(i)$ received from the unknown system at time i. Typically, $y(i)$ is different from $d(i)$; hence, their comparison results in the error signal:

$$e(i) = d(i) - y(i) \tag{3.4}$$

The manner in which the error signal $e(i)$ is used to control the adjustments to the neuron's synaptic weights is determined by the cost function used to derive the adaptive filtering algorithm of interest. This issue is closely related to that of optimization. It is therefore appropriate to present a review of unconstrained optimization methods. The material is applicable not only to linear adaptive filters but also to neural networks in general.

3.3 UNCONSTRAINED OPTIMIZATION TECHNIQUES

Consider a cost function $\mathscr{E}(\mathbf{w})$ that is a *continuously differentiable* function of some unknown weight (parameter) vector $\mathbf{w}$. The function $\mathscr{E}(\mathbf{w})$ maps the elements of $\mathbf{w}$ into real numbers. It is a measure of how to choose the weight (parameter) vector $\mathbf{w}$ of an adaptive filtering algorithm so that it behaves in an optimum manner. We want to find an optimal solution $\mathbf{w}^*$ that satisfies the condition

$$\mathscr{E}(\mathbf{w}^*) \leq \mathscr{E}(\mathbf{w}) \tag{3.5}$$

That is, we need to solve an *unconstrained optimization problem*, stated as follows:

Minimize the cost function $\mathscr{E}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$ (3.6)

The necessary condition for optimality is

$$\nabla\mathscr{E}(\mathbf{w}^*) = \mathbf{0} \tag{3.7}$$

where ∇ is the *gradient operator*:

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T \tag{3.8}$$

and $\nabla\mathscr{E}(\mathbf{w})$ is the *gradient vector* of the cost function:

$$\nabla\mathscr{E}(\mathbf{w}) = \left[\frac{\partial\mathscr{E}}{\partial w_1}, \frac{\partial\mathscr{E}}{\partial w_2}, \ldots, \frac{\partial\mathscr{E}}{\partial w_m}\right]^T \tag{3.9}$$

A class of unconstrained optimization algorithms that is particularly well suited for the design of adaptive filters is based on the idea of local *iterative descent*:

Starting with an initial guess denoted by $\mathbf{w}(0)$, generate a sequence of weight vectors $\mathbf{w}(1)$, $\mathbf{w}(2)$, ..., such that the cost function $\mathscr{E}(\mathbf{w})$ is reduced at each iteration of the algorithm, as shown by

$$\mathscr{E}(\mathbf{w}(n+1)) < \mathscr{E}(\mathbf{w}(n)) \tag{3.10}$$

where $\mathbf{w}(n)$ is the old value of the weight vector and $\mathbf{w}(n+1)$ is its updated value.

We hope that the algorithm will eventually converge onto the optimal solution $\mathbf{w}^*$. We say "hope" because there is a distinct possibility that the algorithm will diverge (i.e., become unstable) unless special precautions are taken.

In this section we describe three unconstrained optimization methods that rely on the idea of iterative descent in one form or another (Bertsekas, 1995a).

Method of Steepest Descent

In the method of steepest descent, the successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the *gradient vector* $\nabla\mathscr{E}(\mathbf{w})$. For convenience of presentation we write

$$\mathbf{g} = \nabla\mathscr{E}(\mathbf{w}) \tag{3.11}$$

Accordingly, the steepest descent algorithm is formally described by

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta\mathbf{g}(n) \tag{3.12}$$

where η is a positive constant called the *stepsize* or *learning-rate parameter*, and $\mathbf{g}(n)$ is the gradient vector evaluated at the point $\mathbf{w}(n)$. In going from iteration n to $n + 1$ the algorithm applies the *correction*

$$\begin{aligned}\Delta\mathbf{w}(n) &= \mathbf{w}(n+1) - \mathbf{w}(n)\\ &= -\eta\mathbf{g}(n)\end{aligned} \tag{3.13}$$

Equation (3.13) is in fact a formal statement of the error-correction rule described in Chapter 2.

To show that the formulation of the steepest descent algorithm satisfies the condition of (3.10) for iterative descent, we use a first-order Taylor series expansion around $\mathbf{w}(n)$ to approximate $\mathscr{E}(\mathbf{w}(n+1))$ as

$$\mathscr{E}(\mathbf{w}(n+1)) \approx \mathscr{E}(\mathbf{w}(n)) + \mathbf{g}^T(n)\Delta\mathbf{w}(n)$$

the use of which is justified for small η. Substituting Eq. (3.13) in this approximate relation yields

$$\begin{aligned}\mathscr{E}(\mathbf{w}(n+1)) &\approx \mathscr{E}(\mathbf{w}(n)) - \eta\mathbf{g}^T(n)\mathbf{g}(n)\\ &= \mathscr{E}(\mathbf{w}(n)) - \eta\|\mathbf{g}(n)\|^2\end{aligned}$$

which shows that, for a positive learning-rate parameter η, the cost function is decreased as the algorithm progresses from one iteration to the next. The reasoning presented here is approximate in that this end result is only true for small enough learning rates.

The method of steepest descent converges to the optimal solution $\mathbf{w}^*$ slowly. Moreover, the learning-rate parameter η has a profound influence on its convergence behavior:

- When η is small, the transient response of the algorithm is *overdamped*, in that the trajectory traced by $\mathbf{w}(n)$ follows a smooth path in the W-plane, as illustrated in Fig. 3.2a.
- When η is large, the transient response of the algorithm is *underdamped*, in that the trajectory of $\mathbf{w}(n)$ follows a zigzagging (oscillatory) path, as illustrated in Fig. 3.2b.
- When η exceeds a certain critical value, the algorithm becomes unstable (i.e., it diverges).

Newton's Method

The basic idea of *Newton's method* is to minimize the quadratic approximation of the cost function $\mathscr{E}(\mathbf{w})$ around the current point $\mathbf{w}(n)$; this minimization is performed at each iteration of the algorithm. Specifically, using a *second-order* Taylor series expansion of the cost function around the point $\mathbf{w}(n)$, we may write

$$\begin{aligned}\Delta\mathscr{E}(\mathbf{w}(n)) &= \mathscr{E}(\mathbf{w}(n+1)) - \mathscr{E}(\mathbf{w}(n))\\ &\approx \mathbf{g}^T(n)\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n)\end{aligned} \tag{3.14}$$

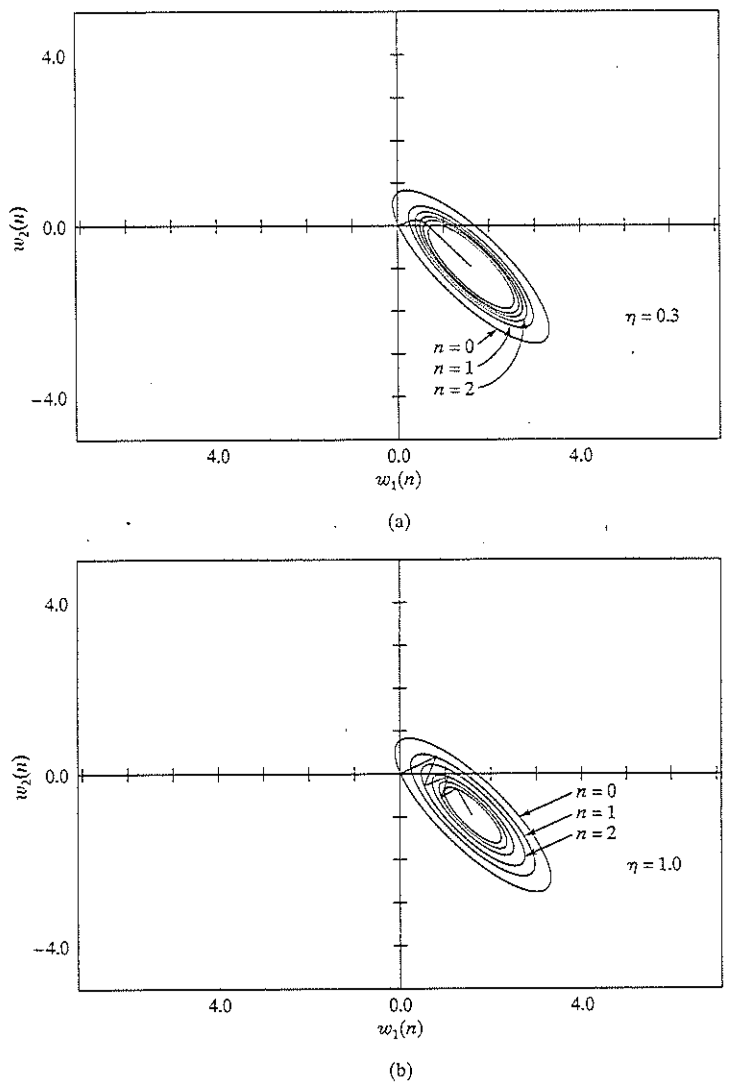

FIGURE 3.2 Trajectory of the method of steepest descent in a two-dimensional space for two different values of learning-rate parameter: (a) $\eta = 0.3$, (b) $\eta = 1.0$. The coordinates w_1 and w_2 are elements of the weight vector $\mathbf{w}$.

As before, $\mathbf{g}(n)$ is the m-by-1 gradient vector of the cost function $\mathscr{E}(\mathbf{w})$ evaluated at the point $\mathbf{w}(n)$. The matrix $\mathbf{H}(n)$ is the m-by-m *Hessian matrix* of $\mathscr{E}(\mathbf{w})$, also evaluated at $\mathbf{w}(n)$. The Hessian of $\mathscr{E}(\mathbf{w})$ is defined by

$$\mathbf{H} = \nabla^2\mathscr{E}(\mathbf{w})$$

$$= \begin{bmatrix} \frac{\partial^2\mathscr{E}}{\partial w_1^2} & \frac{\partial^2\mathscr{E}}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2\mathscr{E}}{\partial w_1 \partial w_m} \\ \frac{\partial^2\mathscr{E}}{\partial w_2 \partial w_1} & \frac{\partial^2\mathscr{E}}{\partial w_2^2} & \cdots & \frac{\partial^2\mathscr{E}}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2\mathscr{E}}{\partial w_m \partial w_1} & \frac{\partial^2\mathscr{E}}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2\mathscr{E}}{\partial w_m^2} \end{bmatrix} \qquad (3.15)$$

Equation (3.15) requires the cost function $\mathscr{E}(\mathbf{w})$ to be twice continuously differentiable with respect to the elements of $\mathbf{w}$. Differentiating[2] Eq. (3.14) with respect to $\Delta\mathbf{w}$, the change $\Delta\mathscr{E}(\mathbf{w})$ is minimized when

$$\mathbf{g}(n) + \mathbf{H}(n)\Delta\mathbf{w}(n) = \mathbf{0}$$

Solving this equation for $\Delta\mathbf{w}(n)$ yields

$$\Delta\mathbf{w}(n) = -\mathbf{H}^{-1}(n)\mathbf{g}(n)$$

That is,

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \Delta\mathbf{w}(n) \\ &= \mathbf{w}(n) - \mathbf{H}^{-1}(n)\mathbf{g}(n) \end{aligned} \qquad (3.16)$$

where $\mathbf{H}^{-1}(n)$ is the inverse of the Hessian of $\mathscr{E}(\mathbf{w})$.

Generally speaking, Newton's method converges quickly asymptotically and does *not* exhibit the zigzagging behavior that sometimes characterizes the method of steepest descent. However, for Newton's method to work, the Hessian $\mathbf{H}(n)$ has to be a *positive definite matrix*[3] for all n. Unfortunately, in general there is no guarantee that $\mathbf{H}(n)$ is positive definite at every iteration of the algorithm. If the Hessian $\mathbf{H}(n)$ is not positive definite, modification of Newton's method is necessary (Powell, 1987; Bertsekas, 1995a).

Gauss–Newton Method

The *Gauss–Newton method* is applicable to a cost function that is expressed as the sum of error squares. Let

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^2(i) \qquad (3.17)$$

where the scaling factor 1/2 is included to simplify matters in subsequent analysis. All the error terms in this formula are calculated on the basis of a weight vector $\mathbf{w}$ that is fixed over the entire observation interval $1 \le i \le n$.

The error signal $e(i)$ is a function of the adjustable weight vector $\mathbf{w}$. Given an operating point $\mathbf{w}(n)$, we linearize the dependence of $e(i)$ on $\mathbf{w}$ by writing

$$e'(i, \mathbf{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \mathbf{w}}\right]^T_{\mathbf{w}=\mathbf{w}(n)} (\mathbf{w} - \mathbf{w}(n)), \quad i = 1, 2, \ldots, n \qquad (3.18)$$

Equivalently, by using matrix notation we may write

$$\mathbf{e}'(n,\mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n)\,(\mathbf{w} - \mathbf{w}(n)) \qquad (3.19)$$

where $\mathbf{e}(n)$ is the error vector

$$\mathbf{e}(n) = [e(1), e(2), \ldots, e(n)]^T$$

and $\mathbf{J}(n)$ is the n-by-m *Jacobian matrix* of $\mathbf{e}(n)$:

$$\mathbf{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{bmatrix}_{\mathbf{w}=\mathbf{w}(n)} \qquad (3.20)$$

The Jacobian $\mathbf{J}(n)$ is the transpose of the m-by-n gradient matrix $\nabla\mathbf{e}(n)$, where

$$\nabla\mathbf{e}(n) = [\nabla e(1), \nabla e(2), \ldots, \nabla e(n)]$$

The updated weight vector $\mathbf{w}(n+1)$ is then defined by

$$\mathbf{w}(n+1) = \arg\min_{\mathbf{w}}\left\{\frac{1}{2}\|\mathbf{e}'(n,\mathbf{w})\|^2\right\} \qquad (3.21)$$

Using Eq. (3.19) to evaluate the squared Euclidean norm of $\mathbf{e}'(n,\mathbf{w})$, we get

$$\frac{1}{2}\|\mathbf{e}'(n,\mathbf{w})\|^2 = \frac{1}{2}\|\mathbf{e}(n)\|^2 + \mathbf{e}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$$
$$+ \frac{1}{2}(\mathbf{w} - \mathbf{w}(n))^T\mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$$

Hence, differentiating this expression with respect to $\mathbf{w}$ and setting the result equal to zero, we obtain

$$\mathbf{J}^T(n)\mathbf{e}(n) + \mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)) = \mathbf{0}$$

Solving this equation for $\mathbf{w}$, we may thus write in light of Eq. (3.21):

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n))^{-1}\mathbf{J}^T(n)\mathbf{e}(n) \tag{3.22}$$

which describes the pure form of the Gauss–Newton method.

Unlike Newton's method that requires knowledge of the Hessian matrix of the cost function $\mathscr{E}(n)$, the Gauss–Newton method only requires the Jacobian matrix of the error vector $\mathbf{e}(n)$. However, for the Gauss–Newton iteration to be computable, the matrix product $\mathbf{J}^T(n)\mathbf{J}(n)$ must be nonsingular.

With regard to the latter point, we recognize that $\mathbf{J}^T(n)\mathbf{J}(n)$ is always nonnegative definite. To ensure that it is nonsingular, the Jacobian $\mathbf{J}(n)$ must have row *rank n*; that is, the n rows of $\mathbf{J}(n)$ in Eq. (3.20) must be linearly independent. Unfortunately, there is no guarantee that this condition will always hold. To guard against the possibility that $\mathbf{J}(n)$ is rank deficient, the customary practice is to add the diagonal matrix $\delta\mathbf{I}$ to the matrix $\mathbf{J}^T(n)\mathbf{J}(n)$. The parameter δ is a small positive constant chosen to ensure that

$$\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I} : \text{positive definite for all } n$$

On this basis, the Gauss–Newton method is implemented in the slightly modified form:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \delta\mathbf{I})^{-1}\mathbf{J}^T(n)\mathbf{e}(n) \tag{3.23}$$

The effect of this modification is progressively reduced as the number of iterations, n, is increased. Note also that the recursive equation (3.23) is the solution of the *modified* cost function:

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2}\left\{\delta\|\mathbf{w} - \mathbf{w}(n)\|^2 + \sum_{i=1}^{n} e^2(i)\right\} \tag{3.24}$$

where $\mathbf{w}(n)$ is the *current value* of the weight vector $\mathbf{w}(i)$.

We are now equipped with the optimization tools we need to address the specific issues involved in linear adaptive filtering.

3.4 LINEAR LEAST-SQUARES FILTER

As the name implies, a *linear least-squares filter* has two distinctive characteristics. First, the single neuron around which it is built is linear, as shown in the model of Fig. 3.1b. Second, the cost function $\mathscr{E}(\mathbf{w})$ used to design the filter consists of the sum of error squares, as defined in Eq. (3.17). On this basis, using Eqs. (3.3) and (3.4), we may express the error vector $\mathbf{e}(n)$ as follows:

$$\begin{aligned}\mathbf{e}(n) &= \mathbf{d}(n) - [\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n)]^T\mathbf{w}(n)\\ &= \mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n)\end{aligned} \tag{3.25}$$

where $\mathbf{d}(n)$ is the n-by-1 *desired response vector*:

$$\mathbf{d}(n) = [d(1), d(2), \ldots, d(n)]^T$$

and $\mathbf{X}(n)$ is the n-by-m *data matrix*:

$$\mathbf{X}(n) = [\mathbf{x}(n), \mathbf{x}(2), \ldots, \mathbf{x}(n)]^T$$

Differentiating Eq. (3.25) with respect to $\mathbf{w}(n)$ yields the gradient matrix

$$\nabla\mathbf{e}(n) = -\mathbf{X}^T(n)$$

Correspondingly, the Jacobian of $\mathbf{e}(n)$ is

$$\mathbf{J}(n) = -\mathbf{X}(n) \tag{3.26}$$

Since the error equation (3.19) is already linear in the weight vector $\mathbf{w}(n)$, the Gauss–Newton method converges in a single iteration, as shown here. Substituting Eqs. (3.25) and (3.26) in (3.22) yields

$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) + (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)(\mathbf{d}(n) - \mathbf{X}(n)\mathbf{w}(n))\\ &= (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n)\end{aligned} \tag{3.27}$$

The term $(\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)$ is recognized as the *pseudoinverse* of the data matrix $\mathbf{X}(n)$ as shown in Golub and Van Loan (1996), and Haykin (1996); that is,

$$\mathbf{X}^+(n) = (\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n) \tag{3.28}$$

Hence, we may rewrite Eq. (3.27) in the compact form

$$\mathbf{w}(n+1) = \mathbf{X}^+(n)\mathbf{d}(n) \tag{3.29}$$

This formula represents a convenient way of saying: "The weight vector $\mathbf{w}(n+1)$ solves the linear least-squares problem defined over an observation interval of duration n."

Wiener Filter: Limiting form of the Linear Least-Squares Filter for an Ergodic Environment

A case of particular interest is when the input vector $\mathbf{x}(i)$ and desired response $d(i)$ are drawn from an *ergodic* environment that is also stationary. We may then substitute long-term sample averages or time-averages for expectations or ensemble averages (Gray and Davisson, 1986). Such an environment is partially described by the second-order statistics:

- *Correlation matrix* of the input vector $\mathbf{x}(i)$; it is denoted by $\mathbf{R}_x$
- *Cross-correlation vector* between the input vector $\mathbf{x}(i)$ and desired response $d(i)$; it is denoted by $\mathbf{r}_{xd}$.

These two quantities are defined as follows, respectively:

$$\begin{aligned}\mathbf{R}_x &= E[\mathbf{x}(i)\mathbf{x}^T(i)]\\ &= \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}(n)\mathbf{x}^T(i)\\ &= \lim_{n\to\infty}\frac{1}{n}\mathbf{X}^T(n)\mathbf{X}(n)\end{aligned} \tag{3.30}$$

$$\begin{aligned}\mathbf{r}_{xd} &= E[\mathbf{x}(i)d(i)]\\ &= \lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}(i)d(i) \qquad (3.31)\\ &= \lim_{n\to\infty}\frac{1}{n}\mathbf{X}^T(n)\mathbf{d}(n)\end{aligned}$$

where E denotes the statistical expectation operator. Accordingly, we may reformulate the linear least-squares solution of Eq. (3.27) as follows:

$$\begin{aligned}\mathbf{w}_o &= \lim_{n\to\infty}\mathbf{w}(n+1)\\ &= \lim_{n\to\infty}(\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\mathbf{X}^T(n)\mathbf{d}(n)\\ &= \lim_{n\to\infty}\frac{1}{n}(\mathbf{X}^T(n)\mathbf{X}(n))^{-1}\lim_{n\to\infty}\frac{1}{n}\mathbf{X}^T(n)\mathbf{d}(n) \qquad (3.32)\\ &= \mathbf{R}_x^{-1}\mathbf{r}_{xd}\end{aligned}$$

where $\mathbf{R}_x^{-1}$ is the inverse of the correlation matrix $\mathbf{R}_x$. The weight vector $\mathbf{w}_o$ is called the *Wiener solution* to the linear optimum filtering problem in recognition of the contributions of Norbert Wiener to this problem (Widrow and Stearns, 1985; Haykin, 1996). Accordingly, we may make the following statement:

For an ergodic process, the linear least-squares filter asymptotically approaches the Wiener filter as the number of observations approaches infinity.

Designing the Wiener filter requires knowledge of the second-order statistics: the correlation matrix $\mathbf{R}_x$ of the input vector $\mathbf{x}(n)$ and the cross-correlation vector $\mathbf{r}_{xd}$ between $\mathbf{x}(n)$ and the desired response $d(n)$. However, this information is not available in many important situations encountered in practice. We may deal with an unknown environment by using a *linear adaptive filter*, adaptive in the sense that the filter is able to adjust its free parameters in response to statistical variations in the environment. A highly popular algorithm for doing this kind of adjustment on a continuing basis is the least-mean-square algorithm, which is intimately related to the Wiener filter.

3.5 LEAST-MEAN-SQUARE ALGORITHM

The *least-mean-square (LMS) algorithm* is based on the use of *instantaneous values* for the cost function, namely,

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2}e^2(n) \qquad (3.33)$$

where $e(n)$ is the error signal measured at time n. Differentiating $\mathscr{E}(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$ yields

$$\frac{\partial\mathscr{E}(\mathbf{w})}{\partial\mathbf{w}} = e(n)\frac{\partial e(n)}{\partial\mathbf{w}} \qquad (3.34)$$

As with the linear least-squares filter, the LMS algorithm operates with a linear neuron so we may express the error signal as

$$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n) \qquad (3.35)$$

Hence,

$$\frac{\partial e(n)}{\partial\mathbf{w}(n)} = -\mathbf{x}(n)$$

and

$$\frac{\partial\mathscr{E}(\mathbf{w})}{\partial\mathbf{w}(n)} = -\mathbf{x}(n)e(n)$$

Using this latter result as an *estimate* for the gradient vector, we may write

$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n) \qquad (3.36)$$

Finally, using Eq. (3.36) for the gradient vector in Eq. (3.12) for the method of steepest descent, we may formulate the LMS algorithm as follows:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n) \qquad (3.37)$$

where η is the learning-rate parameter. The feedback loop around the weight vector $\hat{\mathbf{w}}(n)$ in the LMS algorithm behaves like a *low-pass filter*, passing the low frequency components of the error signal and attenuating its high frequency components (Haykin, 1996). The average time constant of this filtering action is inversely proportional to the learning-rate parameter η. Hence, by assigning a small value to η, the adaptive process will progress slowly. More of the past data are then remembered by the LMS algorithm, resulting in a more accurate filtering action. In other words, the inverse of the learning-rate parameter η is a measure of the *memory* of the LMS algorithm.

In Eq. (3.37) we have used $\hat{\mathbf{w}}(n)$ in place of $\mathbf{w}(n)$ to emphasize the fact that the LMS algorithm produces an *estimate* of the weight vector that would result from the use of the method of steepest descent. As a consequence, in using the LMS algorithm we sacrifice a distinctive feature of the steepest descent algorithm. In the steepest descent algorithm the weight vector $\mathbf{w}(n)$ follows a well-defined trajectory in weight space for a prescribed η. In contrast, in the LMS algorithm the weight vector $\hat{\mathbf{w}}(n)$ traces a random trajectory. For this reason, the LMS algorithm is sometimes referred to as a "stochastic gradient algorithm." As the number of iterations in the LMS algorithm approaches infinity, $\hat{\mathbf{w}}(n)$ performs a random walk (Brownian motion) about the Wiener solution $\mathbf{w}_o$. The important point is the fact that, unlike the method of steepest descent, the LMS algorithm does *not* require knowledge of the statistics of the environment.

A summary of the LMS algorithm is presented in Table 3.1, which clearly illustrates the simplicity of the algorithm. As indicated in this table, for the *initialization* of the algorithm, it is customary to set the initial value of the weight vector in the algorithm equal to zero.

TABLE 3.1 Summary of the LMS Algorithm

Training Sample:	Input signal vector = $\mathbf{x}(n)$
	Desired response = $d(n)$

User-selected parameter: η
Initialization. Set $\hat{\mathbf{w}}(0) = \mathbf{0}$.
Computation. For $n = 1, 2, \ldots,$ compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$
$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\,\mathbf{x}(n)e(n)$$

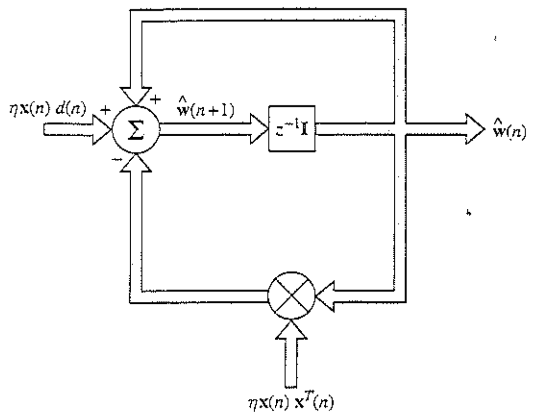

FIGURE 3.3 Signal-flow graph representation of the LMS algorithm.

Signal-Flow Graph Representation of the LMS Algorithm

By combining Eqs. (3.35) and (3.37) we may express the evolution of the weight vector in the LMS algorithm as follows:

$$\begin{aligned}\hat{\mathbf{w}}(n+1) &= \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)] \\ &= [\mathbf{I} - \eta\mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)d(n)\end{aligned} \tag{3.38}$$

where $\mathbf{I}$ is the identity matrix. In using the LMS algorithm, we recognize that

$$\hat{\mathbf{w}}(n) = z^{-1}[\hat{\mathbf{w}}(n+1)] \tag{3.39}$$

where z^{-1} is the *unit-delay operator*, implying storage. Using Eqs. (3.38) and (3.39), we may thus represent the LMS algorithm by the signal-flow graph depicted in Fig. 3.3. This signal-flow graph reveals that the LMS algorithm is an example of a *stochastic feedback system*. The presence of feedback has a profound impact on the convergence behavior of the LMS algorithm.

Convergence Considerations of the LMS Algorithm

From control theory we know that the stability of a feedback system is determined by the parameters that constitute its feedback loop. From Fig. 3.3 we see that it is the lower feedback loop that adds variability to the behavior of the LMS algorithm. In particular, there are two distinct quantities, the learning-rate parameter η and the input vector $\mathbf{x}(n)$, that determine the transmittance of this feedback loop. We therefore deduce that the convergence behavior (i.e., stability) of the LMS algorithm is influenced by the statistical characteristics of the input vector $\mathbf{x}(n)$ and the value assigned to the learning-rate parameter η. Casting this observation in a different way, we may state that for a specified environment that supplies the input vector $\mathbf{x}(n)$, we have to exercise care in the selection of the learning-rate parameter η for the LMS algorithm to be convergent.

The first criterion for convergence of the LMS algorithm is *convergence of the mean*, described by

$$E[\hat{\mathbf{w}}(n)] \to \mathbf{w}_o \qquad \text{as } n \to \infty \tag{3.40}$$

where $\mathbf{w}_o$ is the Wiener solution. Unfortunately, such a convergence criterion is of little practical value, since a sequence of zero-mean, but otherwise arbitrary, random vectors converges in this sense.

From a practical point of view, the convergence issue that really matters is *convergence in the mean square*, described by

$$E[e^2(n)] \to \text{constant as } n \to \infty \tag{3.41}$$

Unfortunately, a detailed convergence analysis of the LMS algorithm in the mean square is rather complicated. To make the analysis mathematically tractable, the following assumptions are usually made:

1. The successive input vectors $\mathbf{x}(1), \mathbf{x}(2), \ldots$ are statistically independent of each other.
2. At time step n, the input vector $\mathbf{x}(n)$ is statistically independent of all previous samples of the desired response, namely $d(1), d(2), \ldots, d(n-1)$.
3. At time step n, the desired response $d(n)$ is dependent on $\mathbf{x}(n)$, but statistically independent of all previous values of the desired response.
4. The input vector $\mathbf{x}(n)$ and desired response $d(n)$ are drawn from Gaussian-distributed populations.

A statistical analysis of the LMS algorithm so based is called the *independence theory* (Widrow et al., 1976).

By invoking the elements of independence theory and assuming that the learning-rate parameter η is sufficiently small, it is shown in Haykin (1996) that the LMS is convergent in the mean square provided that η satisfies the condition

$$0 < \eta < \frac{2}{\lambda_{\max}} \tag{3.42}$$

where $\lambda_{\max}$ is the *largest eigenvalue* of the correlation matrix $\mathbf{R}_x$. In typical applications of the LMS algorithm, however, knowledge of $\lambda_{\max}$ is not available. To overcome this difficulty, the *trace* of $\mathbf{R}_x$ may be taken as a conservative estimate for $\lambda_{\max}$, in which case the condition of Eq. (3.42) may be reformulated as

$$0 < \eta < \frac{2}{\operatorname{tr}[\mathbf{R}_x]} \tag{3.43}$$

where tr$[\mathbf{R}_x]$ denotes the trace of matrix $\mathbf{R}_x$. By definition, the trace of a square matrix is equal to the sum of its diagonal elements. Since each diagonal element of the correlation matrix $\mathbf{R}_x$ equals the mean-square value of the corresponding sensor input, we may restate the condition for convergence of the LMS algorithm in the mean square as follows:

$$0 < \eta < \frac{2}{\text{sum of mean-square values of the sensor inputs}} \tag{3.44}$$

Provided the learning-rate parameter satisfies this condition, the LMS algorithm is also assured of convergence of the mean. That is, convergence in the mean square implies convergence of the mean, but the converse is not necessarily true.

Virtues and Limitations of the LMS Algorithm

An important virtue of the LMS algorithm is its simplicity, as exemplified by the summary of the algorithm presented in Table 3.1. Moreover, the LMS algorithm is model independent and therefore *robust*, which means that small model uncertainty and small disturbances (i.e., disturbances with small energy) can only result in small estimation errors (error signals). In precise mathematical terms, the LMS algorithm is optimal in accordance with the H^∞ *(or minimax) criterion* (Hassibi et al., 1993, 1996). The basic philosophy of optimality in the H^∞ sense is to cater to the worst-case scenario[4]:

If you do not know what you are up against, plan for the worst and optimize.

For a long time the LMS algorithm was regarded as an instantaneous approximation to the gradient-descent algorithm. However, the H^∞ optimality of LMS provides this widely used algorithm with a rigorous footing. In particular, it explains its ability to work satisfactorily in a stationary as well as in a nonstationary environment. By a "nonstationary" environment we mean one where the statistics vary with time. In such an environment, the optimum Wiener solution takes on a time-varying form, and the LMS algorithm now has the additional task of *tracking* variations in the parameters of the Wiener filter.

The primary limitations of the LMS algorithm are its slow rate of convergence and sensitivity to variations in the eigenstructure of the input (Haykin, 1996). The LMS algorithm typically requires a number of iterations equal to about 10 times the dimensionality of the input space for it to reach a steady-state condition. The slow rate of convergence becomes particularly serious when the dimensionality of the input space becomes high. As for sensitivity to changes in environmental conditions, the LMS algorithm is particularly sensitive to variations in the *condition number* or *eigenvalue spread* of the correlation matrix $\mathbf{R}_x$ of the input vector $\mathbf{x}$. The condition number of $\mathbf{R}_x$, denoted by $\chi(\mathbf{R}_x)$, is defined by

$$\chi(\mathbf{R}_x) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{3.45}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of the matrix $\mathbf{R}_x$, respectively. The sensitivity of the LMS algorithm to variations in the condition number $\chi(\mathbf{R}_x)$ becomes particularly acute when the training sample to which the input vector $\mathbf{x}(n)$ belongs is *ill conditioned*, that is, when the condition number $\chi(\mathbf{R}_x)$ is high.[5]

Note that in the LMS algorithm the *Hessian matrix*, defined as the second derivative of the cost function $\mathscr{E}(\mathbf{w})$ with respect to $\mathbf{w}$, is equal to the correlation matrix $\mathbf{R}_x$; see Problem 3.8. Thus, in the discussion presented here, we could have just as well spoken in terms of the Hessian as the correlation matrix $\mathbf{R}_x$.

3.6 LEARNING CURVES

An informative way of examining the convergence behavior of the LMS algorithm, or an adaptive filter in general, is to plot the *learning curve* of the filter under varying environmental conditions. The learning curve is a *plot of the mean-square value of the estimation error, $\mathscr{E}_{av}(n)$, versus the number of iterations, n.*

Imagine an experiment involving an *ensemble* of adaptive filters, with each filter operating under the control of a specific algorithm. It is assumed that the details of the algorithm, including initialization, are the same for all the filters. The differences between the filters arise from the *random* manner in which the input vector $\mathbf{x}(n)$ and the desired response $d(n)$ are drawn from the available training sample. For each filter we plot the squared value of the estimation error (i.e., the difference between the desired response and the actual filter output) versus the number of iterations. A *sample* learning curve so obtained consists of *noisy* exponentials, the noise being due to the inherently stochastic nature of the adaptive filter. To compute the *ensemble-averaged learning curve* (i.e., plot of $\mathscr{E}_{av}(n)$ versus n), we take the average of these sample learning curves over the ensemble of adaptive filters used in the experiment, thereby smoothing out the effects of noise.

Assuming that the adaptive filter is stable, we find that the ensemble-averaged learning curve starts from a large value $\mathscr{E}_{av}(0)$ determined by the initial conditions, then decreases at some rate depending on the type of filter used, and finally converges to a steady-state value $\mathscr{E}_{av}(\infty)$, as illustrated in Fig. 3.4. On the basis of this learning curve we may define the *rate of convergence* of the adaptive filter as the number of iterations, n, required for $\mathscr{E}_{av}(n)$ to decrease to some arbitrarily chosen value, such as 10 percent of the initial value $\mathscr{E}_{av}(0)$.

Another useful characteristic of an adaptive filter that is deduced from the ensemble-averaged learning curve is the *misadjustment*, denoted by $\mathscr{M}$. Let $\mathscr{E}_{\min}$ denote the minimum mean-square error produced by the Wiener filter, designed on the basis of known values of the correlation matrix $\mathbf{R}_x$ and cross-correlation vector $\mathbf{r}_{xd}$. We may define the *misadjustment* for the adaptive filter as follows (Widrow and Stearns, 1985; Haykin, 1996):

$$\begin{aligned} \mathscr{M} &= \frac{\mathscr{E}(\infty) - \mathscr{E}_{\min}}{\mathscr{E}_{\min}} \\ &= \frac{\mathscr{E}(\infty)}{\mathscr{E}_{\min}} - 1 \end{aligned} \tag{3.46}$$

The misadjustment $\mathscr{M}$ is a dimensionless quantity, providing a measure of how close the adaptive filter is to optimality in the mean-square error sense. The smaller $\mathscr{M}$ is compared to unity, the more *accurate* is the adaptive filtering action of the algorithm. It is customary to express the misadjustment $\mathscr{M}$ as a percentage. Thus, for example, a misadjustment of 10 percent means that the adaptive filter produces a mean-square error

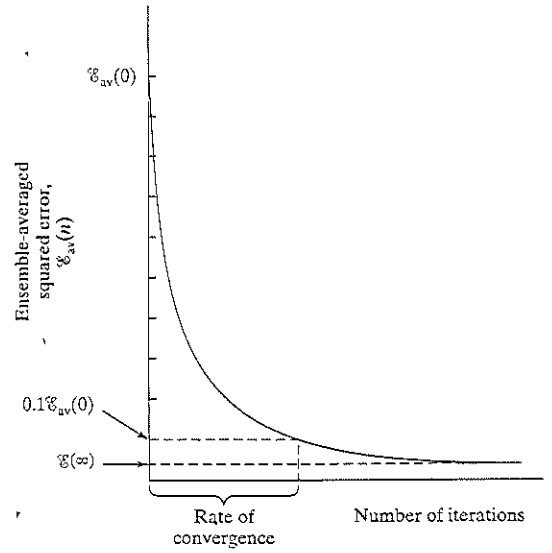

FIGURE 3.4 Idealized learning curve of the LMS algorithm.

(after adaptation is completed) that is 10 percent greater than the minimum mean-square error $\mathscr{E}_{\min}$ produced by the corresponding Wiener filter. Such performance is ordinarily considered to be satisfactory in practice.

Another important characteristic of the LMS algorithm is the *settling time*. However, there is no unique definition for the settling time. We may, for example, approximate the learning curve by a single exponential with *average time constant* τ_{av}, and so use τ_{av} as a rough measure of the settling time. The smaller the value of τ_{av} is, the faster the settling time will be (i.e., the faster the LMS algorithm will converge to a "steady-state" condition).

To a good degree of approximation, the misadjustment $\mathcal{M}$ of the LMS algorithm is directly proportional to the learning-rate parameter η, whereas the average time constant τ_{av} is inversely proportional to the learning-rate parameter η (Widrow and Stearns, 1985; Haykin, 1996). We therefore have conflicting results in the sense that if the learning-rate parameter is reduced so as to reduce the misadjustment, then the settling time of the LMS algorithm is increased. Conversely, if the learning-rate parameter is increased so as to accelerate the learning process, then the misadjustment is increased. Careful attention must be given to the choice of the learning parameter η in the design of the LMS algorithm in order to produce a satisfactory overall performance.

3.7 LEARNING-RATE ANNEALING SCHEDULES

The difficulties encountered with the LMS algorithm may be attributed to the fact that the learning-rate parameter is maintained constant throughout the computation, as shown by

$$\eta(n) = \eta_0 \qquad \text{for all } n \tag{3.47}$$

This is the simplest possible form the learning-rate parameter can assume. In contrast, in *stochastic approximation*, which goes back to the classic paper by Robbins and Monro (1951), the learning-rate parameter is time-varying. The particular time-varying form most commonly used in stochastic approximation literature is described by

$$\eta(n) = \frac{c}{n} \tag{3.48}$$

where c is a constant. Such a choice is indeed sufficient to guarantee convergence of the stochastic approximation algorithm (Ljung, 1977; Kushner and Clark, 1978). However, when the constant c is large, there is a danger of parameter blowup for small n.

As an alternative to Eqs. (3.47) and (3.48), we may use the *search-then-converge schedule*, defined by Darken and Moody (1992)

$$\eta(n) = \frac{\eta_0}{1 + (n/\tau)} \tag{3.49}$$

where η_0 and τ are user-selected constants. In the early stages of adaptation involving a number of iterations n, small compared to the *search time constant* τ, the learning-rate parameter $\eta(n)$ is approximately equal to η_0, and the algorithm operates essentially as the "standard" LMS algorithm, as indicated in Fig. 3.5. Hence, by choosing a high value for η_0 within the permissible range, we hope that the adjustable weights of the filter will find and hover about a "good" set of values. Then, for a number of iterations n large compared to the search time constant τ, the learning-rate parameter $\eta(n)$ approximates as c/n, where $c = \tau\eta_0$, as illustrated in Fig. 3.5. The algorithm now operates as a traditional stochastic approximation algorithm, and the weights converge to their optimum values. Thus the search-then-converge schedule has the potential to combine the desirable features of the standard LMS with traditional stochastic approximation theory.

3.8 PERCEPTRON

We now come to the second part of the chapter that deals with Rosenblatt's perceptron, henceforth referred to simply as the *perceptron*. Whereas the LMS algorithm described in the preceding sections is built around a linear neuron, the perceptron is built around a nonlinear neuron, namely, the *McCulloch–Pitts model* of a neuron. From Chapter 1 we recall that such a neuronal model consists of a linear combiner followed by a hard limiter (performing the signum function), as depicted in Fig. 3.6. The summing node of the neuronal model computes a linear combination of the inputs applied to its synapses, and also incorporates an externally applied bias. The resulting sum, that is, the induced local field, is applied to a hard limiter. Accordingly, the neuron produces an output equal to +1 if the hard limiter input is positive, and −1 if it is negative.

In the signal-flow graph model of Fig. 3.6, the synaptic weights of the perceptron are denoted by $w_1, w_2, \ldots, w_m$. Correspondingly, the inputs applied to the perceptron

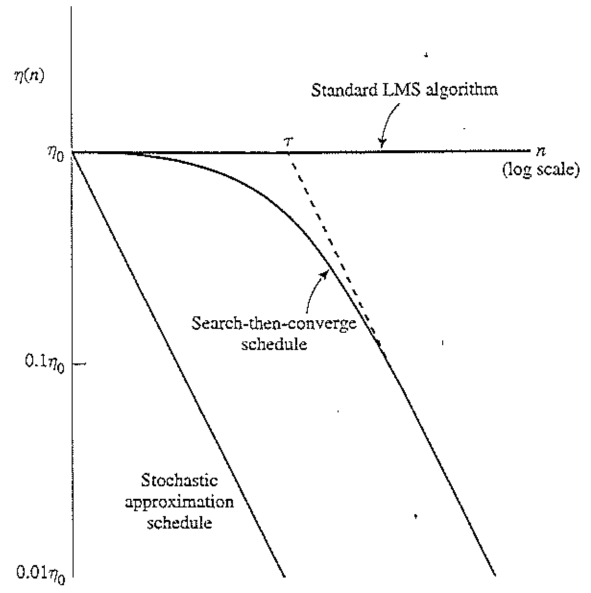

FIGURE 3.5 Learning-rate annealing schedules.

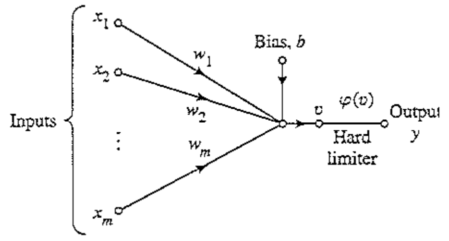

FIGURE 3.6 Signal-flow graph of the perceptron.

are denoted by $x_1, x_2, \ldots, x_m$. The externally applied bias is denoted by b. From the model we find that the hard limiter input or induced local field of the neuron is

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{3.50}$$

The goal of the perceptron is to correctly classify the set of externally applied stimuli $x_1, x_2, \ldots, x_m$ into one of two classes, $\mathscr{C}_1$ or $\mathscr{C}_2$. The decision rule for the classification is to assign the point represented by the inputs $x_1, x_2, \ldots, x_m$ to class $\mathscr{C}_1$ if the perceptron output y is $+1$ and to class $\mathscr{C}_2$ if it is -1.

To develop insight into the behavior of a pattern classifier, it is customary to plot a map of the decision regions in the m-dimensional signal space spanned by the m

input variables x_1, x_2, x_m. In the simplest form of the perceptron there are two decision regions separated by a *hyperplane* defined by

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{3.51}$$

This is illustrated in Fig. 3.7 for the case of two input variables x_1 and x_2, for which the decision boundary takes the form of a straight line. A point (x_1, x_2) that lies above the boundary line is assigned to class $\mathscr{C}_1$ and a point (x_1, x_2) that lies below the boundary line is assigned to class $\mathscr{C}_2$. Note also that the effect of the bias b is merely to shift the decision boundary away from the origin.

The synaptic weights $w_1, w_2, \ldots, w_m$ of the perceptron can be adapted on an iteration-by-iteration basis. For the adaptation we may use an error-correction rule known as the perceptron convergence algorithm.

3.9 PERCEPTRON CONVERGENCE THEOREM

To derive the error-correction learning algorithm for the perceptron, we find it more convenient to work with the modified signal-flow graph model in Fig. 3.8. In this second model, which is equivalent to that of Fig. 3.6, the bias $b(n)$ is treated as a synaptic weight driven by a fixed input equal to $+1$. We may thus define the $(m + 1)$-by-1 input vector

$$\mathbf{x}(n) = [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$$

where n denotes the iteration step in applying the algorithm. Correspondingly we define the $(m + 1)$-by-1 weight vector as

$$\mathbf{w}(n) = [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$$

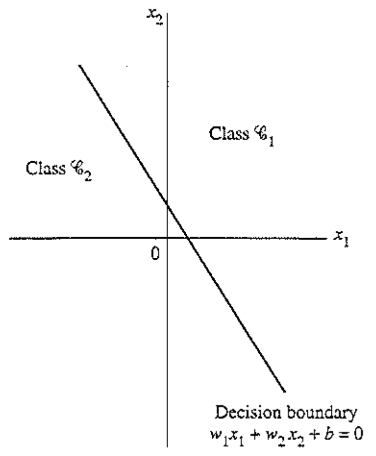

FIGURE 3.7 Illustration of the hyperplane (in this example, a straight line) as decision boundary for a two-dimensional, two-class pattern-classification problem.

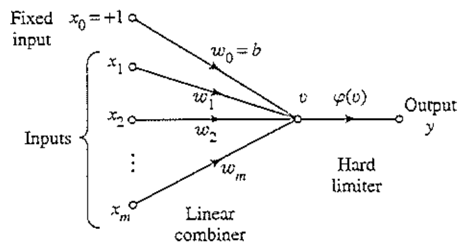

FIGURE 3.8 Equivalent signal-flow graph of the perceptron; dependence on time has been omitted for clarity.

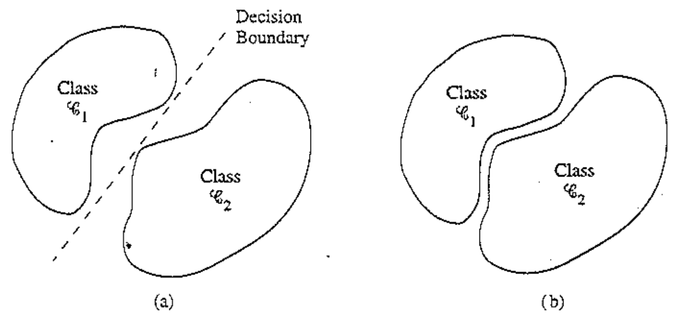

FIGURE 3.9 (a) A pair of linearly separable patterns. (b) A pair of nonlinearly separable patterns.

Accordingly, the linear combiner output is written in the compact form

$$v(n) = \sum_{i=0}^{m} w_i(n)x_i(n) = \mathbf{w}^T(n)\mathbf{x}(n) \tag{3.52}$$

where $w_0(n)$ represents the bias $b(n)$. For fixed n, the equation $\mathbf{w}^T\mathbf{x} = 0$, plotted in an m-dimensional space (plotted for some prescribed bias) with coordinates $x_1, x_2, \ldots, x_m$, defines a hyperplane as the decision surface between two different classes of inputs.

For the perceptron to function properly, the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ must be *linearly separable*. This, in turn, means that the patterns to be classified must be sufficiently separated from each other to ensure that the decision surface consists of a hyperplane. This requirement is illustrated in Fig. 3.9 for the case of a two-dimensional perceptron. In Fig. 3.9a, the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are sufficiently separated from each other for us to draw a hyperplane (in this case a straight line) as the decision boundary. If, however, the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are allowed to move too close to each other, as in Fig. 3.9b, they become nonlinearly separable, a situation that is beyond the computing capability of the perceptron.

Suppose then that the input variables of the perceptron originate from two linearly separable classes. Let $\mathscr{X}_1$ be the subset of training vectors $\mathbf{x}_1(1), \mathbf{x}_1(2), \ldots$ that belong to class $\mathscr{C}_1$, and let $\mathscr{X}_2$ be the subset of training vectors $\mathbf{x}_2(1), \mathbf{x}_2(2), \ldots$ that belong to class $\mathscr{C}_2$. The union of $\mathscr{X}_1$ and $\mathscr{X}_2$ is the complete training set $\mathscr{X}$. Given the sets of vectors $\mathscr{X}_1$ and $\mathscr{X}_2$ to train the classifier, the training process involves the adjustment of the weight vector $\mathbf{w}$ in such a way that the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are linearly separable. That is, there exists a weight vector $\mathbf{w}$ such that we may state

$$\begin{aligned} &\mathbf{w}^T\mathbf{x} > 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } \mathscr{C}_1 \\ &\mathbf{w}^T\mathbf{x} \le 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } \mathscr{C}_2 \end{aligned} \tag{3.53}$$

In the second line of Eq. (3.53) we have arbitrarily chosen to say that the input vector $\mathbf{x}$ belongs to class $\mathscr{C}_2$ if $\mathbf{w}^T\mathbf{x} = 0$. Given the subsets of training vectors $\mathscr{X}_1$ and $\mathscr{X}_2$, the training problem for the elementary perceptron is then to find a weight vector $\mathbf{w}$ such that the two inequalities of Eq. (3.53) are satisfied.

The algorithm for adapting the weight vector of the elementary perceptron may now be formulated as follows:

1. If the nth member of the training set, $\mathbf{x}(n)$, is correctly classified by the weight vector $\mathbf{w}(n)$ computed at the nth iteration of the algorithm, no correction is made to the weight vector of the perceptron in accordance with the rule:

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) \quad \text{if } \mathbf{w}^T\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) \quad \text{if } \mathbf{w}^T\mathbf{x}(n) \le 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{aligned} \tag{3.54}$$

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \le 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \end{aligned} \tag{3.55}$$

where the *learning-rate parameter* $\eta(n)$ controls the adjustment applied to the weight vector at iteration n.

If $\eta(n) = \eta > 0$, where η is a constant independent of the iteration number n, we have a *fixed increment adaptation rule* for the perceptron.

In the sequel we first prove the convergence of a fixed increment adaptation rule for which $\eta = 1$. Clearly the value of η is unimportant, so long as it is positive. A value of $\eta \neq 1$ merely scales the pattern vectors without affecting their separability. The case of a variable $\eta(n)$ is considered later.

The proof is presented for the initial condition $\mathbf{w}(0) = \mathbf{0}$. Suppose that $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ for $n = 1, 2, \ldots$, and the input vector $\mathbf{x}(n)$ belongs to the subset $\mathscr{X}_1$. That is, the perceptron incorrectly classifies the vectors $\mathbf{x}(1), \mathbf{x}(2), \ldots$, since the second condition of Eq. (3.53) is violated. Then, with the constant $\eta(n) = 1$, we may use the second line of Eq. (3.55) to write

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{x}(n) \quad \text{for } \mathbf{x}(n) \text{ belonging to class } \mathscr{C}_1 \tag{3.56}$$

Given the initial condition $\mathbf{w}(0) = \mathbf{0}$, we may iteratively solve this equation for $\mathbf{w}(n+1)$, obtaining the result

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \cdots + \mathbf{x}(n) \tag{3.57}$$

Since the classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are assumed to be linearly separable, there exists a solution $\mathbf{w}_0$ for which $\mathbf{w}^T\mathbf{x}(n) > 0$ for the vectors $\mathbf{x}(1), \ldots, \mathbf{x}(n)$ belonging to the subset $\mathscr{X}_1$. For a fixed solution $\mathbf{w}_0$, we may then define a positive number α as

$$\alpha = \min_{\mathbf{x}(n) \in \mathscr{X}_1} \mathbf{w}_0^T \mathbf{x}(n) \tag{3.58}$$

Hence, multiplying both sides of Eq. (3.57) by the row vector $\mathbf{w}_0^T$, we get

$$\mathbf{w}_0^T\mathbf{w}(n+1) = \mathbf{w}_0^T\mathbf{x}(1) + \mathbf{w}_0^T\mathbf{x}(2) + \cdots + \mathbf{w}_0^T\mathbf{x}(n)$$

Accordingly, in light of the definition given in Eq. (3.58), we have

$$\mathbf{w}_0^T\mathbf{w}(n+1) \geq n\alpha \tag{3.59}$$

Next we make use of an inequality known as the Cauchy–Schwarz inequality. Given two vectors $\mathbf{w}_0$ and $\mathbf{w}(n+1)$, the *Cauchy–Schwarz inequality* states that

$$\|\mathbf{w}_0\|^2 \|\mathbf{w}(n+1)\|^2 \geq [\mathbf{w}_0^T\mathbf{w}(n+1)]^2 \tag{3.60}$$

where $\|\cdot\|$ denotes the Euclidean norm of the enclosed argument vector, and the inner product $\mathbf{w}_0^T\mathbf{w}(n+1)$ is a scalar quantity. We now note from Eq. (3.59) that $[\mathbf{w}_0^T\mathbf{w}(n+1)]^2$ is equal to or greater than $n^2\alpha^2$. From Eq. (3.60) we note that $\|\mathbf{w}_0\|^2 \|\mathbf{w}(n+1)\|^2$ is equal to or greater than $[\mathbf{w}_0^T\mathbf{w}(n+1)]^2$. It follows therefore that

$$\|\mathbf{w}_0\|^2 \|\mathbf{w}(n+1)\|^2 \geq n^2\alpha^2$$

or equivalently,

$$\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|\mathbf{w}_0\|^2} \tag{3.61}$$

We next follow another development route. In particular, we rewrite Eq. (3.56) in the form

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \qquad \text{for } k = 1, \ldots, n \quad \text{and} \quad \mathbf{x}(k) \in \mathscr{X}_1 \tag{3.62}$$

By taking the squared Euclidean norm of both sides of Eq. (3.62), we obtain

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \tag{3.63}$$

But, under the assumption that the perceptron incorrectly classifies an input vector $\mathbf{x}(k)$ belonging to the subset $\mathscr{X}_1$, we have $\mathbf{w}^T(k)\mathbf{x}(k) < 0$. We therefore deduce from Eq. (3.63) that

$$\|\mathbf{w}(k+1)\|^2 \leq \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2$$

or equivalently,

$$\|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \leq \|\mathbf{x}(k)\|^2, \qquad k = 1, \ldots, n \tag{3.64}$$

Adding these inequalities for $k = 1, \ldots, n$, and invoking the assumed initial condition $\mathbf{w}(0) = \mathbf{0}$, we get the following inequality:

$$\begin{aligned} \|\mathbf{w}(n+1)\|^2 &\leq \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2 \\ &\leq n\beta \end{aligned} \tag{3.65}$$

where β is a positive number defined by

$$\beta = \max_{\mathbf{x}(k) \in \mathscr{X}_1} \|\mathbf{x}(k)\|^2 \tag{3.66}$$

Equation (3.65) states that the squared Euclidean norm of the weight vector $\mathbf{w}(n+1)$ grows at most linearly with the number of iterations n.

The second result of Eq. (3.65) is clearly in conflict with the earlier result of Eq. (3.61) for sufficiently large values of n. Indeed, we can state that n cannot be larger than some value $n_{\max}$ for which Eqs. (3.61) and (3.65) are both satisfied with the equality sign. That is, $n_{\max}$ is the solution of the equation

$$\frac{n_{\max}^2\alpha^2}{\|\mathbf{w}_0\|^2} = n_{\max}\beta$$

Solving for $n_{\max}$, given a solution vector $\mathbf{w}_0$, we find that

$$n_{\max} = \frac{\beta\|\mathbf{w}_0\|^2}{\alpha^2} \tag{3.67}$$

We have thus proved that for $\eta(n) = 1$ for all n, and $\mathbf{w}(0) = \mathbf{0}$, and given that a solution vector $\mathbf{w}_0$ exists, the rule for adapting the synaptic weights of the perceptron must terminate after at most $n_{\max}$ iterations. Note also from Eqs. (3.58), (3.66), and (3.67) that there is *no* unique solution for $\mathbf{w}_0$ or $n_{\max}$.

We may now state the *fixed-increment covergence theorem* for the perceptron as follows (Rosenblatt, 1962):

> Let the subsets of training vectors $\mathscr{X}_1$ and $\mathscr{X}_2$ be linearly separable. Let the inputs presented to the perceptron originate from these two subsets. The perceptron converges after some n_0 iterations, in the sense that
>
> $$\mathbf{w}(n_0) = \mathbf{w}(n_0+1) = \mathbf{w}(n_0+2) = \cdots$$
>
> is a solution vector for $n_0 \leq n_{\max}$.

Consider next the *absolute error-correction procedure* for the adaptation of a single-layer perceptron, for which $\eta(n)$ is variable. In particular, let $\eta(n)$ be the smallest integer for which

$$\eta(n)\mathbf{x}^T(n)\mathbf{x}(n) > |\mathbf{w}^T(n)\mathbf{x}(n)|$$

With this procedure we find that if the inner product $\mathbf{w}^T(n)\mathbf{x}(n)$ at iteration n has an incorrect sign, then $\mathbf{w}^T(n+1)\mathbf{x}(n)$ at iteration $n+1$ would have the correct sign. This suggests that if $\mathbf{w}^T(n)\mathbf{x}(n)$ has an incorrect sign, we may modify the training sequence at iteration $n+1$ by setting $\mathbf{x}(n+1) = \mathbf{x}(n)$. In other words, each pattern is presented repeatedly to the perceptron until that pattern is classified correctly.

Note also that the use of an initial value $\mathbf{w}(0)$ different from the null condition merely results in a decrease or increase in the number of iterations required to converge.

depending on how $\mathbf{w}(0)$ relates to the solution $\mathbf{w}_0$. Regardless of the value assigned to $\mathbf{w}(0)$, the perceptron is assured of convergence.

In Table 3.2 we present a summary of the *perceptron convergence algorithm* (Lippmann, 1987). The symbol sgn(·), used in step 3 of the table for computing the actual response of the perceptron, stands for the *signum function:*

$$\operatorname{sgn}(v) = \begin{cases} +1 & \text{if } v > 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{3.68}$$

We may thus express the *quantized response* $y(n)$ of the perceptron in the compact form

$$y(n) = \operatorname{sgn}(\mathbf{w}^T(n)\mathbf{x}(n)) \tag{3.69}$$

TABLE 3.2 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n) = (m+1)$-by-1 input vector

$= [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$

$\mathbf{w}(n) = (m + 1)$-by-1 weight vector

$= [b(n), w_1(n), w_2(n), \dots, w_m(n)]^T$

$b(n) =$ bias

$y(n) =$ actual response (quantized)

$d(n) =$ desired response

$\eta =$ learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time step $n = 1, 2, \dots$.

2. *Activation.* At time step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.

3. *Computation of Actual Response.* Compute the actual response of the perceptron:

$$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where sgn(·) is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron:

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

Notice that the input vector $\mathbf{x}(n)$ is an $(m + 1)$-by-1 vector whose first element is fixed at +1 throughout the computation. Correspondingly, the weight vector $\mathbf{w}(n)$ is an $(m + 1)$-by-1 vector whose first element equals the bias $b(n)$. One other important point in Table 3.2: We have introduced a *quantized desired response* $d(n)$, defined by

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases} \tag{3.70}$$

Thus, the adaptation of the weight vector $\mathbf{w}(n)$ is summed up nicely in the form of the *error-correction learning rule:*

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n) \tag{3.71}$$

where η is the *learning-rate parameter*, and the difference $d(n) - y(n)$ plays the role of an *error signal.* The learning-rate parameter is a positive constant limited to the range $0 < \eta \leq 1$. When assigning a value to it inside this range, we must keep in mind two conflicting requirements (Lippmann, 1987):

- *Averaging* of past inputs to provide stable weight estimates, which requires a small η
- *Fast adaptation* with respect to real changes in the underlying distributions of the process responsible for the generation of the input vector $\mathbf{x}$, which requires a large η

3.10 RELATION BETWEEN THE PERCEPTRON AND BAYES CLASSIFIER FOR A GAUSSIAN ENVIRONMENT

The perceptron bears a certain relationship to a classical pattern classifier known as the Bayes classifier. When the environment is Gaussian, the Bayes classifier reduces to a linear classifier. This is the same form taken by the perceptron. However, the linear nature of the perceptron is *not* contingent on the assumption of Gaussianity. In this section we study this relationship, and thereby develop further insight into the operation of the perceptron. We begin the discussion with a brief review of the Bayes classifier.

Bayes Classifier

In the *Bayes classifier* or *Bayes hypothesis testing procedure*, we minimize the *average risk*, denoted by $\mathscr{R}$. For a two-class problem, represented by classes $\mathscr{C}_1$ and $\mathscr{C}_2$, the average risk is defined by Van Trees (1968):

$$\begin{aligned}\mathscr{R} = {} & c_{11}p_1 \int_{\mathscr{X}_1} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathscr{X}_2} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathscr{X}_2} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} + c_{12}p_2 \int_{\mathscr{X}_1} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x}\end{aligned} \tag{3.72}$$