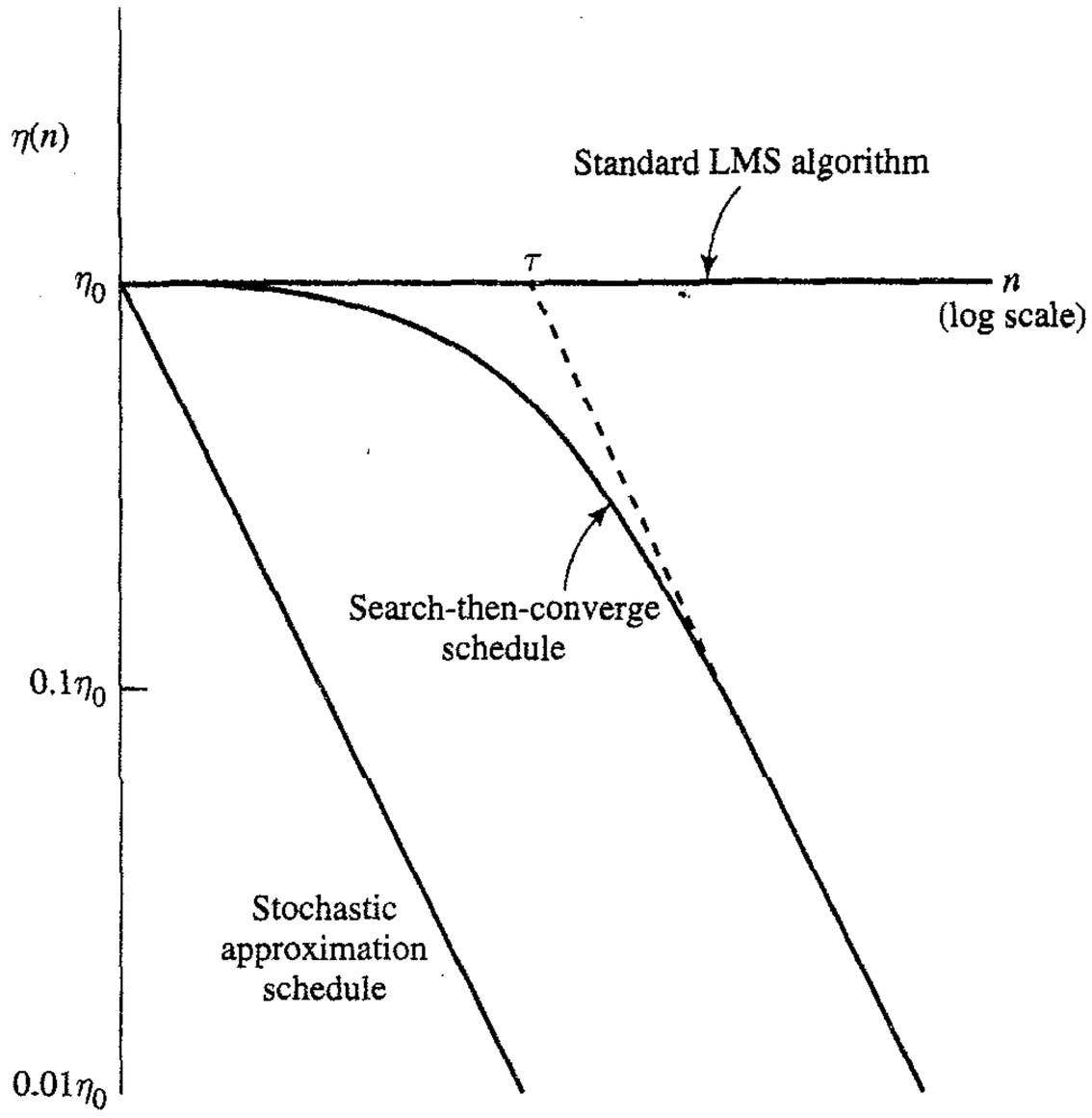

FIGURE 3.5 Learning-rate annealing schedules.

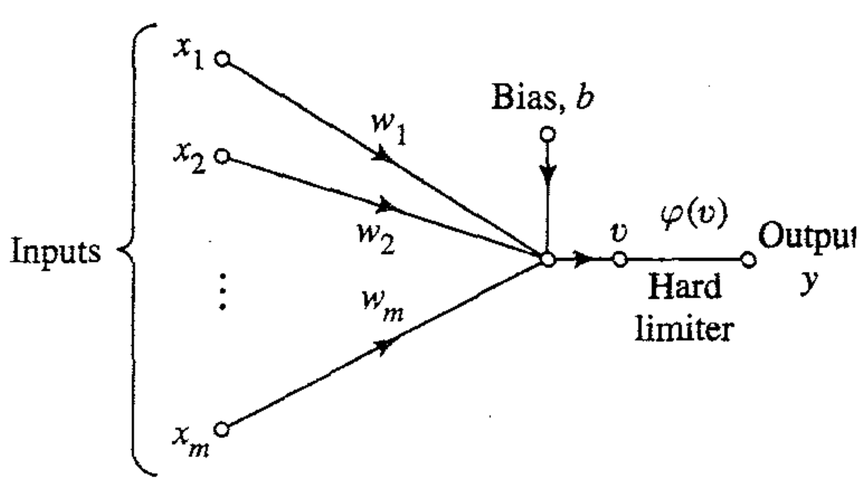

FIGURE 3.6 Signal-flow graph of the perceptron.

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

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

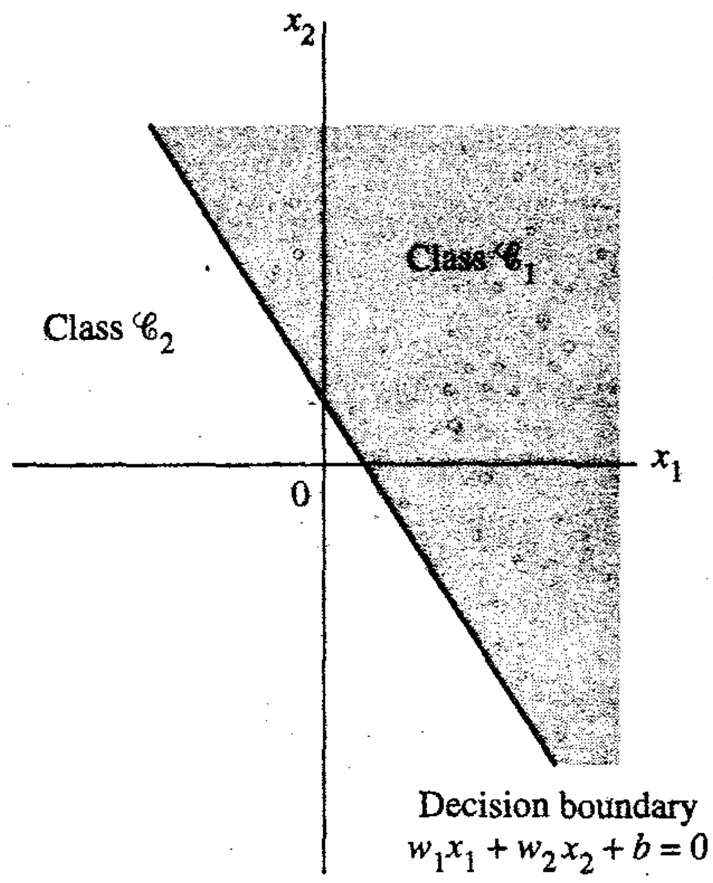

FIGURE 3.7 Illustration of the hyperplane (in this example, a straight line) as decision boundary for a two-dimensional, two-class pattern-classification problem.

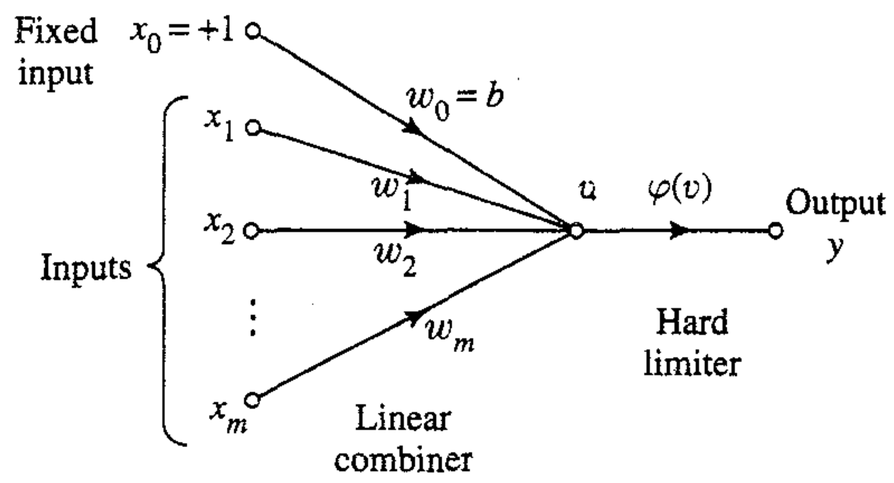

FIGURE 3.8 Equivalent signal-flow graph of the perceptron; dependence on time has been omitted for clarity.

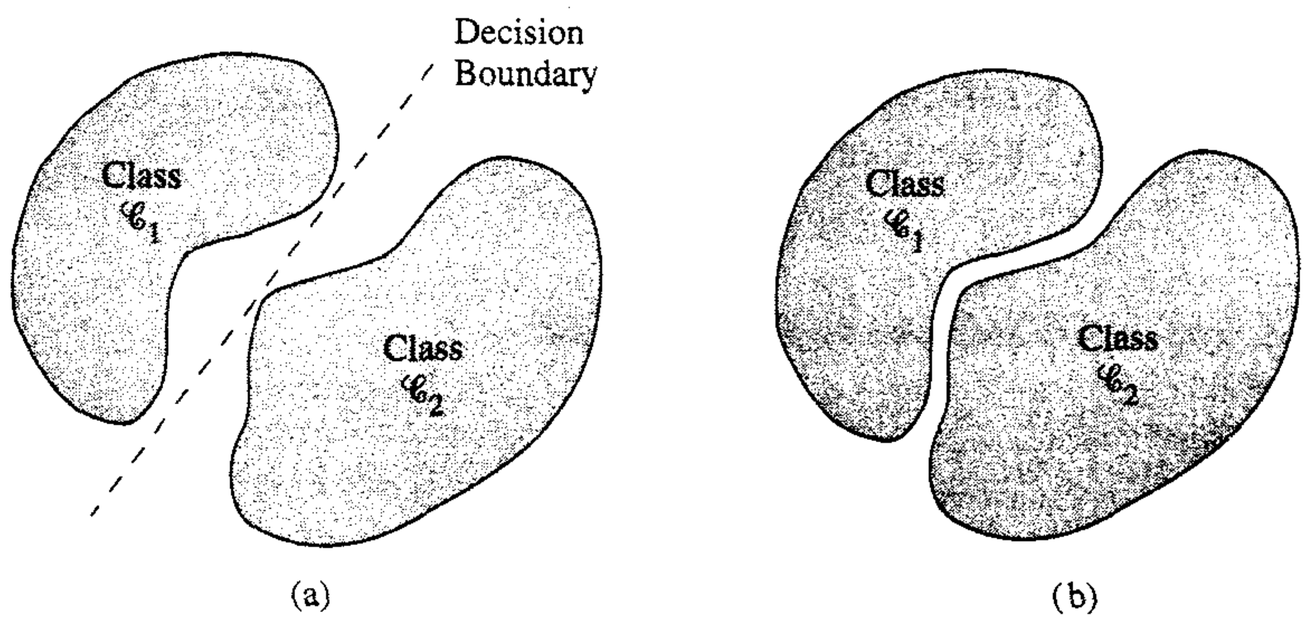

FIGURE 3.9 (a) A pair of linearly separable patterns. (b) A pair of nonlinearly separable patterns.

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

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

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

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

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:

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

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

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

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

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

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:*

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

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

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

TABLE 3.2 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

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

$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
= $[b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$

$b(n)$ = bias

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

$d(n)$ = desired response

η = 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 $n = 1, 2, \ldots$.

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:

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

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):

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

where the various terms are defined as follows:

p_i = *a priori probability* that the observation vector $\mathbf{x}$ (representing a realization of the random vector $\mathbf{X}$) is drawn from subspace $\mathscr{X}_i$, with $i = 1, 2$. and $p_1 + p_2 = 1$.

c_{ij} = cost of deciding in favor of class $\mathscr{C}_i$ represented by subspace $\mathscr{X}_i$ when class $\mathscr{C}_j$ is true (i.e., observation vector $\mathbf{x}$ is drawn from subspace $\mathscr{X}_j$). with $(i, j) = 1, 2$.

$f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_i)$ = conditional probability density function of the random vector $\mathbf{X}$, given that the observation vector $\mathbf{x}$ is drawn from subspace $\mathscr{X}_i$, with $i = 1, 2$.

The first two terms on the right-hand side of Eq. (3.72) represent *correct* decisions (i.e., correct classifications), whereas the last two terms represent *incorrect* decisions (i.e., misclassifications). Each decision is weighted by the product of two factors: the cost involved in making the decision, and the relative frequency (i.e., *a priori* probability) with which it occurs.

The intention is to determine a strategy for the *minimum average risk*. Because we require that a decision be made, each observation vector $\mathbf{x}$ must be assigned in the overall observation space $\mathscr{X}$ to either $\mathscr{X}_1$ or $\mathscr{X}_2$. Thus,

$$\mathscr{X} = \mathscr{X}_1 + \mathscr{X}_2 \tag{3.73}$$

Accordingly, we may rewrite Eq. (3.72) in the equivalent form

$$\begin{aligned}\mathscr{R} = {} & c_{11}p_1 \int_{\mathscr{X}_1} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathscr{X}-\mathscr{X}_1} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathscr{X}-\mathscr{X}_1} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)\mathbf{x} + c_{12}p_2 \int_{\mathscr{X}_1} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x}\end{aligned} \tag{3.74}$$

where $c_{11} < c_{21}$ and $c_{22} < c_{12}$. We now observe the fact that

$$\int_{\mathscr{X}} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} = \int_{\mathscr{X}} f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} = 1 \tag{3.75}$$

Hence, Eq. (3.74) reduces to

$$\begin{aligned}\mathscr{R} = {} & c_{21}p_1 + c_{22}p_2 \\ & + \int_{\mathscr{X}_1} [p_2(c_{12} - c_{22}) f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2) - p_1(c_{21} - c_{11}) f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)]d\mathbf{x}\end{aligned} \tag{3.76}$$

The first two terms on the right-hand side of Eq. (3.76) represent a fixed cost. Since the requirement is to minimize the average risk $\mathscr{R}$, we may therefore deduce the following strategy from Eq. (3.76) for optimum classification:

1. All values of the observation vector $\mathbf{x}$ for which the integrand (i.e., the expression inside the square brackets) is negative should be assigned to subspace $\mathscr{X}_1$ (i.e., class $\mathscr{C}_1$) for the integral would then make a negative contribution to the risk $\mathscr{R}$.
2. All values of the observation vector $\mathbf{x}$ for which the integrand is positive should be excluded from subspace $\mathscr{X}_1$ (i.e., assigned to class $\mathscr{C}_2$) for the integral would then make a positive contribution to the risk $\mathscr{R}$.

3. Values of $\mathbf{x}$ for which the integrand is zero have no effect on the average risk $\mathscr{R}$ and may be assigned arbitrarily. We shall assume that these points are assigned to subspace $\mathscr{X}_2$ (i.e., class $\mathscr{C}_2$).

On this basis, we may formulate the Bayes classifier as follows:

If the condition

$$p_1(c_{21} - c_{11}) f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1) > p_2(c_{12} - c_{22}) f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)$$

holds, assign the observation vector $\mathbf{x}$ *to subspace* $\mathscr{X}_1$ *(i.e., class* $\mathscr{C}_1$*). Otherwise assign* $\mathbf{x}$ *to* $\mathscr{X}_2$ *(i.e., class* $\mathscr{C}_2$*).*

To simplify matters, define

$$\Lambda(\mathbf{x}) = \frac{f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)}{f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)} \tag{3.77}$$

and

$$\xi = \frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})} \tag{3.78}$$

The quantity $\Lambda(\mathbf{x})$, the ratio of two conditional probability density functions, is called the *likelihood ratio*. The quantity ξ is called the *threshold* of the test. Note that both $\Lambda(\mathbf{x})$ and ξ are always positive. In terms of these two quantities, we may now reformulate the Bayes classifier by stating:

If, for an observation vector $\mathbf{x}$*, the likelihood ratio* $\Lambda(\mathbf{x})$ *is greater than the threshold* ξ*, assign* $\mathbf{x}$ *to class* $\mathscr{C}_1$*. Otherwise, assign it to class* $\mathscr{C}_2$*.*

Figure 3.10a depicts a block-diagram representation of the Bayes classifier. The important points in this block diagram are twofold:

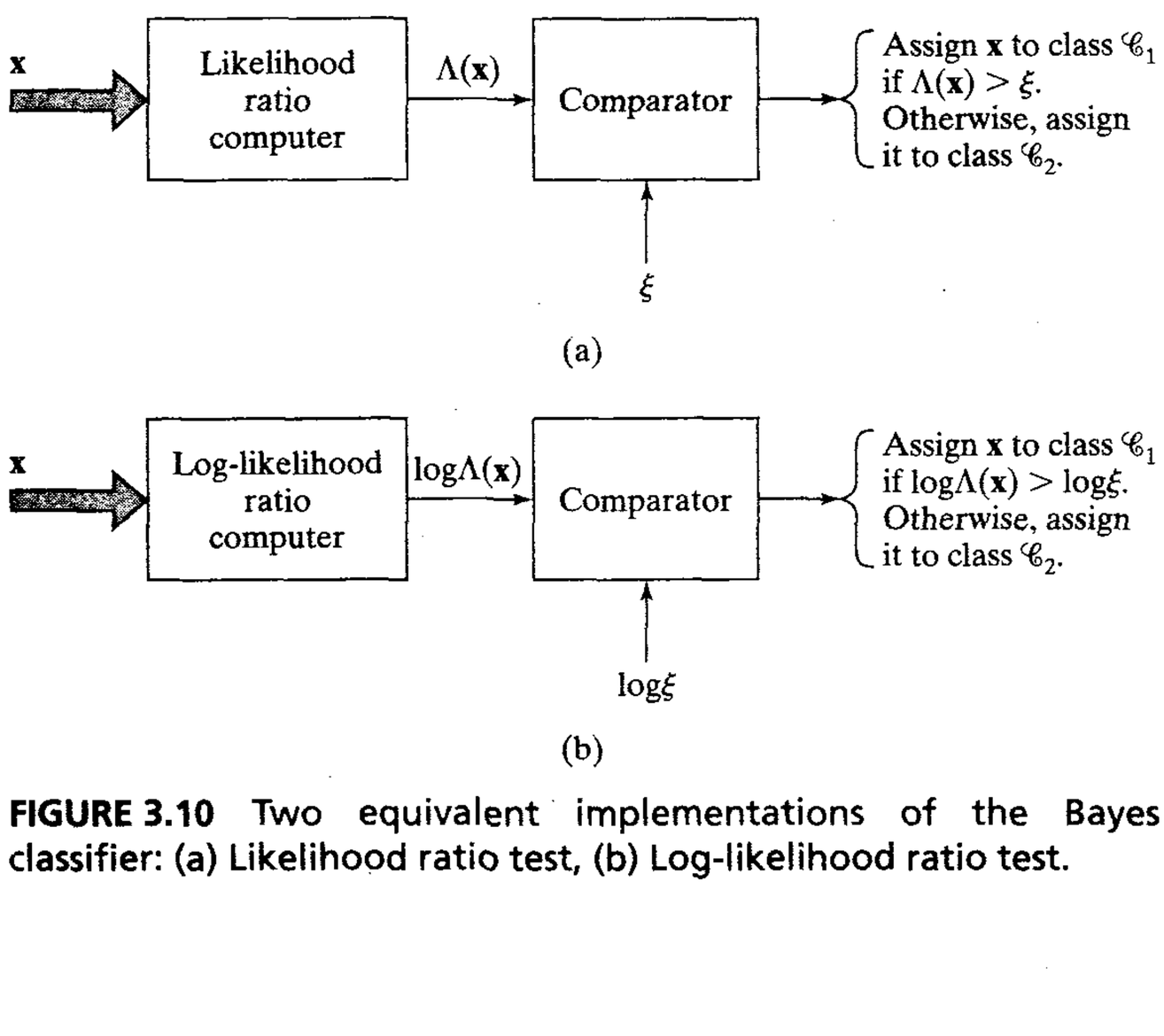

FIGURE 3.10 Two equivalent implementations of the Bayes classifier: (a) Likelihood ratio test, (b) Log-likelihood ratio test.

1. The data processing involved in designing the Bayes classifier is confined entirely to the computation of the likelihood ratio $\Lambda(\mathbf{x})$.
2. This computation is completely invariant to the values assigned to the *a priori* probabilities and costs involved in the decision-making process. These quantities merely affect the value of the threshold ξ.

From a computational point of view, we find it more convenient to work with the logarithm of the likelihood ratio rather than the likelihood ratio itself. We are permitted to do this for two reasons. First, the logarithm is a monotonic function. Second, the likelihood ratio $\Lambda(\mathbf{x})$ and threshold ξ are both positive. Therefore, the Bayes classifier may be implemented in the equivalent form shown in Fig. 3.10b. For obvious reasons, the test embodied in this latter figure is called the *log-likelihood ratio test*.

Bayes Classifier for a Gaussian Distribution

Consider now the special case of a two-class problem, for which the underlying distribution is Gaussian. The random vector $\mathbf{X}$ has a mean value that depends on whether it belongs to class $\mathscr{C}_1$ or class $\mathscr{C}_2$, but the covariance matrix of $\mathbf{X}$ is the same for both classes. That is to say:

$$\begin{aligned} \text{Class } \mathscr{C}_1: \quad & E[\mathbf{X}] = \boldsymbol{\mu}_1 \\ & E[(\mathbf{X} - \boldsymbol{\mu}_1)(\mathbf{X} - \boldsymbol{\mu}_1)^T] = \mathbf{C} \\ \text{Class } \mathscr{C}_2: \quad & E[\mathbf{X}] = \boldsymbol{\mu}_2 \\ & E[(\mathbf{X} - \boldsymbol{\mu}_2)(\mathbf{X} - \boldsymbol{\mu}_2)^T] = \mathbf{C} \end{aligned}$$

The covariance matrix $\mathbf{C}$ is nondiagonal, which means that the samples drawn from classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are *correlated*. It is assumed that $\mathbf{C}$ is nonsingular, so that its inverse matrix $\mathbf{C}^{-1}$ exists.

With this background we may express the conditional probability density function of $\mathbf{X}$ as follows:

$$f_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_i) = \frac{1}{(2\pi)^{m/2}(\det(\mathbf{C}))^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, 2 \tag{3.79}$$

where m is the dimensionality of the observation vector $\mathbf{x}$.

It is further assumed that

1. The two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are equiprobable:

$$p_1 = p_2 = \frac{1}{2} \tag{3.80}$$

2. Misclassifications carry the same cost, and no cost is incurred on correct classifications:

$$c_{21} = c_{12} \qquad \text{and} \qquad c_{11} = c_{22} = 0 \tag{3.81}$$

We now have the information we need to design the Bayes classifier for the two-class problem. Specifically, by substituting Eq. (3.79) in (3.77) and taking the natural logarithm, we get (after simplifications):

$$\begin{aligned}\log \Lambda(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T \mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T \mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) \\ &= (\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^T \mathbf{C}^{-1}\mathbf{x} + \frac{1}{2}(\boldsymbol{\mu}_2^T \mathbf{C}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T \mathbf{C}^{-1}\boldsymbol{\mu}_1)\end{aligned} \tag{3.82}$$

By substituting Eqs. (3.80) and (3.81) in Eq. (3.78) and taking the natural logarithm, we get

$$\log \xi = 0 \tag{3.83}$$

Equations (3.82) and (3.83) state that the Bayes classifier for the problem at hand is a *linear classifier*, as described by the relation

$$y = \mathbf{w}^T\mathbf{x} + b \tag{3.84}$$

where

$$y = \log \Lambda(\mathbf{x}) \tag{3.85}$$

$$\mathbf{w} = \mathbf{C}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{3.86}$$

$$b = \frac{1}{2}(\boldsymbol{\mu}_2^T \mathbf{C}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T \mathbf{C}^{-1}\boldsymbol{\mu}_1) \tag{3.87}$$

More specifically, the classifier consists of a linear combiner with weight vector $\mathbf{w}$ and bias b, as shown in Fig. 3.11.

On the basis of Eq. (3.84), we may now describe the log-likelihood ratio test for our two-class problem as follows:

> *If the output y of the linear combiner (including the bias b) is positive, assign the observation vector $\mathbf{x}$ to class $\mathscr{C}_1$. Otherwise, assign it to class $\mathscr{C}_2$.*

The operation of the Bayes classifier for the Gaussian environment described herein is analogous to that of the perceptron in that they are both linear classifiers; see Eqs. (3.71) and (3.84). There are, however, some subtle and important differences between them, which should be carefully examined (Lippmann, 1987):

- The perceptron operates on the premise that the patterns to be classified are *linearly separable*. The Gaussian distribution of the two patterns assumed in the derivation of the Bayes classifier certainly do *overlap* each other and are therefore

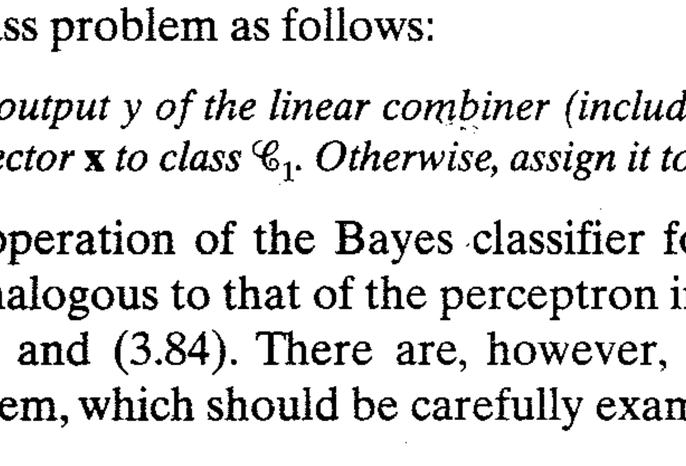

FIGURE 3.11 Signal-flow graph of Gaussian classifier.

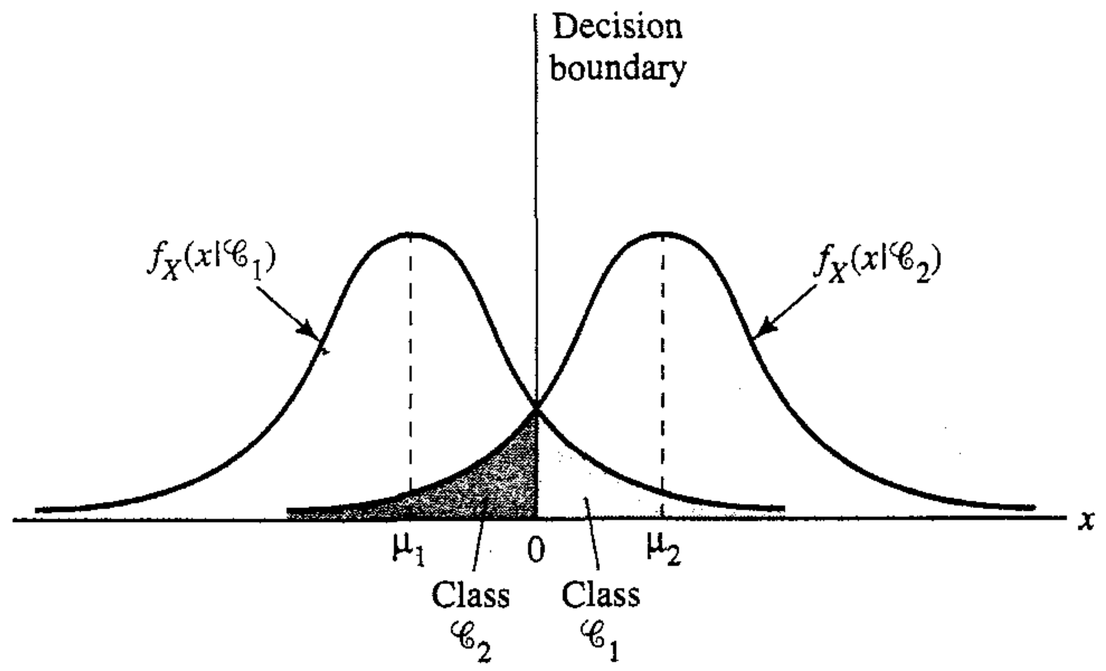

FIGURE 3.12 Two overlapping, one-dimensional Gaussian distributions.

not separable. The extent of the overlap is determined by the mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, and the covariance matrix $\mathbf{C}$. The nature of this overlap is illustrated in Fig. 3.12 for the special case of a scalar random variable (i.e., dimensionality $m = 1$). When the inputs are nonseparable and their distributions overlap as illustrated, the perceptron convergence algorithm develops a problem because decision boundaries between the different classes may oscillate continuously.

- The Bayes classifier minimizes the probability of classification error. This minimization is independent of the overlap between the underlying Gaussian distributions of the two classes. For example, in the special case illustrated in Fig. 3.12, the Bayes classifier always positions the decision boundary at the point where the Gaussian distributions for the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ cross each other.
- The perceptron convergence algorithm is *nonparametric* in the sense that it makes no assumptions concerning the form of the underlying distributions. It operates by concentrating on errors that occur where the distributions overlap. It may therefore work well when the inputs are generated by nonlinear physical mechanisms and when their distributions are heavily skewed and non-Gaussian. In contrast, the Bayes classifier is *parametric*; its derivation is contingent on the assumption that the underlying distributions are Gaussian, which may limit its area of application.
- The perceptron convergence algorithm is both adaptive and simple to implement; its storage requirement is confined to the set of synaptic weights and bias. On the other hand, the design of the Bayes classifier is fixed; it can be made adaptive, but at the expense of increased storage requirements and more complex computations.

3.11 SUMMARY AND DISCUSSION

The perceptron and an adaptive filter using the LMS algorithm are naturally related, as evidenced by their weight updates. Indeed, they represent different implementations of a *single-layer perceptron based on error-correction-learning*. The term "single-layer" is used here to signify that in both cases the computation layer consists of a single neuron—hence the title of the chapter. However, the perceptron and LMS algorithm differ from each other in some fundamental respects:

- The LMS algorithm uses a linear neuron, whereas the perceptron uses the McCulloch–Pitts formal model of a neuron.
- The learning process in the perceptron is performed for a finite number of iterations and then stops. In contrast, *continuous learning* takes place in the LMS algorithm in the sense that learning happens while signal processing is being performed in a manner that never stops.

A hard limiter constitutes the nonlinear element of the McCulloch–Pitts neuron. It is tempting to raise the question: Would the perceptron perform better if it used a sigmoidal nonlinearity in place of the hard limiter? It turns out that the steady-state, decision-making characteristics of the perceptron are basically the same, regardless of whether we use hard-limiting or soft-limiting as the source of nonlinearity in the neuronal model (Shynk, 1990; Shynk and Bershad, 1991). We may therefore state formally that so long as we limit ourselves to the model of a neuron that consists of a linear combiner followed by a nonlinear element, then regardless of the form of nonlinearity used, a single-layer perceptron can perform pattern classification only on linearly separable patterns.

We close this discussion of single-layer perceptrons on a historical note. The perceptron and the LMS algorithm emerged roughly about the same time, during the late 1950s. The LMS algorithm has truly survived the test of time. Indeed, it has established itself as the workhorse of adaptive signal processing because of its simplicity in implementation and its effectiveness in application. The importance of Rosenblatt's perceptron is largely historical.

The first real critique of Rosenblatt's perceptron was presented by Minsky and Selfridge (1961). Minsky and Selfridge pointed out that the perceptron as defined by Rosenblatt could not even generalize toward the notion of binary parity, let alone make general abstractions. The computational limitations of Rosenblatt's perceptron were subsequently put on a solid mathematical foundation in the famous book, *Perceptrons,* by Minsky and Papert (1969, 1988). After the presentation of some brilliant and highly detailed mathematical analyses of the perceptron, Minsky and Papert proved that the perceptron as defined by Rosenblatt is inherently incapable of making some global generalizations on the basis of locally learned examples. In the last chapter of their book, Minsky and Papert make the conjecture that the limitations they had discovered for Rosenblatt's perceptron would also hold true for its variants, more specifically, multilayer neural networks. Quoting from Section 13.2 of their book (1969):

> The perceptron has shown itself worthy of study despite (and even because of !) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension to multilayer systems is sterile.

This conclusion was largely responsible for casting serious doubt on the computational capabilities of not only the perceptron but neural networks in general up to the mid-1980s.

History has shown, however, that the conjecture made by Minsky and Papert seems to be unjustified in that we now have several advanced forms of neural networks that are computationally more powerful than Rosenblatt's perceptron. For

example, multilayer perceptrons trained with the back-propagation algorithm discussed in Chapter 4, the radial basis-function networks discussed in Chapter 5, and support vector machines discussed in Chapter 6, overcome the computational limitations of the single-layer perceptron in their own individual ways.

NOTES AND REFERENCES

1. The network organization of the original version of the perceptron as envisioned by Rosenblatt (1962) has three types of units: sensory units, association units, and response units. The connections from the sensory units to the association units have fixed weights, and the connections from the association units to the response units have variable weights. The association units act as preprocessors designed to extract a pattern from the environmental input. Insofar as the variable weights are concerned, the operation of Rosenblatt's original perceptron is essentially the same as that for the case of a single response unit (i.e., single neuron).

2. **Differentiation with respect to a vector**
Let $f(\mathbf{w})$ denote a real-valued function of parameter vector $\mathbf{w}$. The derivative of $f(\mathbf{w})$ with respect to $\mathbf{w}$ is defined by the vector:

$$\frac{\partial f}{\partial \mathbf{w}} = \left[\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_m} \right]^T$$

where m is the dimension of vector $\mathbf{w}$. The following two cases are of special interest:

CASE 1 The function $f(\mathbf{w})$ is defined by the inner product:

$$f(\mathbf{w}) = \mathbf{x}^T\mathbf{w} = \sum_{i=1}^{m} x_i w_i$$

Hence,

$$\frac{\partial f}{\partial w_i} = x_i, \qquad i = 1, 2, \ldots, m$$

or equivalently, in matrix form:

$$\frac{\partial f}{\partial \mathbf{w}} = \mathbf{x} \tag{1}$$

CASE 2 The function $f(\mathbf{w})$ is defined by the quadratic form:

$$f(\mathbf{w}) = \mathbf{w}^T\mathbf{R}\mathbf{w} = \sum_{i=1}^{m}\sum_{j=1}^{m} w_i r_{ij} w_j$$

where r_{ij} is the ij-th element of the m-by-m matrix $\mathbf{R}$. Hence,

$$\frac{\partial f}{\partial w_i} = \sum_{j=1}^{m} r_{ij} w_j, \qquad i = 1, 2, \ldots, m$$

or equivalently, in matrix form:

$$\frac{\partial f}{\partial \mathbf{w}} = \mathbf{R}\mathbf{w} \tag{2}$$

Equations (1) and (2) provide two useful rules for the differentiation of a real-valued function with respect to a vector.

3. **Positive definite matrix**
   An m-by-m matrix $\mathbf{R}$ is said to be nonnegative definite if it satisfies the condition

$$\mathbf{a}^T\mathbf{R}\mathbf{a} \geq 0 \qquad \text{for any vector } \mathbf{a} \in \mathbb{R}^m$$

   If this condition is satisfied with the inequality sign, the matrix $\mathbf{R}$ is said to be positive definite.

   An important property of a positive definite matrix $\mathbf{R}$ is that it is *nonsingular,* that is, the inverse matrix $\mathbf{R}^{-1}$ exists.

   Another important property of a positive definite matrix $\mathbf{R}$ is that its eigenvalues, or roots or the characteristic equation

$$\det(\mathbf{R}) = 0$$

   are all positive.

4. **Robustness**
   The H^∞ criterion is due to Zames (1981), and it is developed in Zames and Francis (1983). The criterion is discussed in Doyle et al. (1989), Green and Limebeer (1995), and Hassibi et al. (1998).

5. To overcome the limitations of the LMS algorithm, namely, slow rate of convergence and sensitivity to variations in the condition number of the correlation matrix $\mathbf{R}_x$, we may use the *recursive least-squares (RLS) algorithm*, which follows from a recursive implementation of the linear least-squares filter described in Section 3.4. The RLS algorithm is a special case of the Kalman filter, which is known to be the optimum linear filter for a nonstationary environment. Most importantly, the Kalman filter exploits all past data extending up to and including the time instant at which the computations are made. For more details about the RLS algorithm and its relationship to the Kalman filter, see Haykin (1996). The Kalman filter is discussed in Chapter 15.

PROBLEMS

Unconstrained optimization

3.1 Explore the method of steepest descent involving a single weight w by considering the following cost function:

$$\mathscr{E}(w) = \frac{1}{2}\sigma^2 - r_{xd}w + \frac{1}{2}r_x w^2$$

where σ^2, r_{xd}, and r_x are constants.

3.2 Consider the cost function

$$\mathscr{E}(\mathbf{w}) = \frac{1}{2}\sigma^2 - \mathbf{r}_{xd}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R}_x\mathbf{w}$$

where σ^2 is some constant, and

$$\mathbf{r}_{xd} = \begin{bmatrix} 0.8182 \\ 0.354 \end{bmatrix}$$

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0.8182 \\ 0.8182 & 1 \end{bmatrix}$$

(a) Find the optimum value $\mathbf{w}^*$ for which $\mathscr{E}(\mathbf{w})$ reaches its minimum value.

(b) Use the method of steepest descent to compute $\mathbf{w}^*$ for the following two values of learning-rate parameter:

(i) $\eta = 0.3$

(ii) $\eta = 1.0$

For each case, plot the trajectory traced by the evolution of the weight vector $\mathbf{w}(n)$ in the W-plane.

Note: The trajectories obtained for cases (i) and (ii) of part (b) should correspond to the pictures displayed in Fig. 3.2.

3.3 Consider the cost function of Eq. (3.24) that represents a modified form of the sum of error squares defined in Eq. (3.17). Show that the application of the Gauss–Newton method to Eq. (3.24) yields the weight-update described in Eq. (3.23).

LMS Algorithm

3.4 The correlation matrix $\mathbf{R}_x$ of the input vector $\mathbf{x}(n)$ in the LMS algorithm is defined by

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Define the range of values for the learning-rate parameter η of the LMS algorithm for it to be convergent in the mean square.

3.5 The *normalized LMS algorithm* is described by the following recursion for the weight vector:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\eta}{\|\mathbf{x}(n)\|^2} e(n)\mathbf{x}(n)$$

where η is a positive constant and $\|\mathbf{x}(n)\|$ is the Euclidean norm of the input vector $\mathbf{x}(n)$. The error signal $e(n)$ is defined by

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

where $d(n)$ is the desired response. For the normalized LMS algorithm to be convergent in the mean square, show that

$$0 < \eta < 2$$

3.6 The LMS algorithm is used to implement the generalized sidelobe canceler shown in Fig. 2.16. Set up the equations that define the operation of this system, assuming the use of a single neuron for the neural network.

3.7 Consider a linear predictor with its input vector made up of the samples $x(n-1)$, $x(n-2)$, ..., $x(n-m)$, where m is the prediction order. The requirement is to use the LMS algorithm to make a prediction $\hat{x}(n)$ of the input sample $x(n)$. Set up the recursions that may be used to compute the tap weight $w_1, w_2, \ldots, w_m$ of the predictor.

3.8 The ensemble-averaged counterpart to the sum of error squares viewed as a cost function is the mean-square value of the error signal:

$$J(\mathbf{w}) = \frac{1}{2} E[e^2(n)]$$

$$= \frac{1}{2} E[(d(n) - \mathbf{x}^T(n)\mathbf{w})^2]$$

(a) Assuming that the input vector $\mathbf{x}(n)$ and desired response $d(n)$ are drawn from a stationary environment, show that

$$J(\mathbf{w}) = \frac{1}{2}\sigma_d^2 - \mathbf{r}_{xd}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{R}_x\mathbf{w}$$

where

$$\sigma_d^2 = E[d^2(n)]$$

$$\mathbf{r}_{xd} = E[\mathbf{x}(n)\,d(n)]$$

$$\mathbf{R}_x = E[\mathbf{x}(n)\mathbf{x}^T(n)]$$

(b) For this cost function, show that the gradient vector and Hessian matrix of $J(\mathbf{w})$ are as follows, respectively:

$$\mathbf{g} = -\mathbf{r}_{xd} + \mathbf{R}_x\mathbf{w}$$

$$\mathbf{H} = \mathbf{R}_x$$

(c) In the *LMS/Newton algorithm*, the gradient vector $\mathbf{g}$ is replaced by its instantaneous value (Widrow and Stearns, 1985). Show that this algorithm, incorporating a learning-rate parameter η, is described by

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

The inverse of the correlation matrix $\mathbf{R}_x$, assumed to be positive definite, is calculated ahead of time.

3.9 In this problem we revisit the correlation matrix memory discussed in Section 2.11. A shortcoming of this memory is that when a key pattern $\mathbf{x}_j$ is presented to it, the actual response $\mathbf{y}$ produced by the memory may not be close enough (in a Euclidean sense) to the desired response (memorized pattern) $\mathbf{y}_j$ for the memory to associate perfectly. This shortcoming is inherited from the use of Hebbian learning that has no provision for feedback from the output to the input. As a remedy for this shortcoming, we may incorporate an error-correction mechanism into the design of the memory, forcing it to associate properly (Anderson, 1983).

Let $\hat{\mathbf{M}}(n)$ denote the memory matrix learned at iteration n of the error-correction learning process. The memory matrix $\hat{\mathbf{M}}(n)$ learns the information represented by the associations:

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

(a) Adapting the LMS algorithm for the problem at hand, show that the updated value of the memory matrix is defined by

$$\hat{\mathbf{M}}(n+1) = \hat{\mathbf{M}}(n) + \eta[\mathbf{y}_k - \hat{\mathbf{M}}(n)\mathbf{x}_k]\mathbf{x}_k^T$$

where η is the learning-rate parameter.

(b) For autoassociation, $\mathbf{y}_k = \mathbf{x}_k$. For this special case, show that as the number of iterations, n, approaches infinity, the memory autoassociates perfectly, as shown by

$$\mathbf{M}(\infty)\mathbf{x}_k = \mathbf{x}_k, \qquad k = 1, 2, \ldots, q$$

(c) The result described in part (b) may be viewed as an *eigenvalue problem*. In that context, $\mathbf{x}_k$ represents an eigenvector of $\mathbf{M}(\infty)$. What are the eigenvalues of $\mathbf{M}(\infty)$?

3.10 In this problem we investigate the effect of bias on the condition number of a correlation matrix, and therefore the performance of the LMS algorithm.

Consider a random vector $\mathbf{X}$ with covariance matrix

$$\mathbf{C} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

and mean vector

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

(a) Calculate the condition number of the covariance matrix $\mathbf{C}$.

(b) Calculate the condition number of the correlation matrix $\mathbf{R}$.

Comment on the effect of the bias $\boldsymbol{\mu}$ on the performance of the LMS algorithm.

Rosenblatt's Perceptron

3.11 In this problem, we consider another method for deriving the update equation for Rosenblatt's perceptron. Define the *perceptron criterion function* (Duda and Hart, 1973):

$$J_p(\mathbf{w}) = \sum_{\mathbf{x} \in \mathcal{X}(\mathbf{w})} (-\mathbf{w}^T \mathbf{x})$$

where $\mathcal{X}(\mathbf{w})$ is the set of samples misclassified by the choice of weight vector $\mathbf{w}$. Note that $J_p(\mathbf{w})$ is defined as zero if there are no misclassified samples, and the output is misclassified if $\mathbf{w}_{\mathbf{x}}^T \leq 0$.

(a) Demonstrate geometrically that $J_p(\mathbf{w})$ is proportional to the sum of Euclidean distances from the misclassified samples to the decision boundary.

(b) Determine the gradient of $J_p(\mathbf{w})$ with respect to the weight vector $\mathbf{w}$.

(c) Using the result obtained in part (b), show that the weight-update for the perceptron is:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n) \sum_{\mathbf{x} \in \mathcal{X}(\mathbf{w}(n))} \mathbf{x}$$

where $\mathcal{X}(\mathbf{w}(n))$ is the set of samples misclassified by the use of weight vector $\mathbf{w}(n)$, and $\eta(n)$ is the learning-rate parameter. Show that this result, for the case of a single-sample correction, is basically the same as that described by Eqs. (3.54) and (3.55).

3.12 Verify that Eqs. (3.68)–(3.71), summarizing the perceptron convergence algorithm, are consistent with Eqs. (3.54) and (3.55).

3.13 Consider two one-dimensional, Gaussian-distributed classes $\mathscr{C}_1$ and $\mathscr{C}_2$ that have a common variance equal to 1. Their mean values are

$$\mu_1 = -10$$

$$\mu_2 = +10$$

These two classes are essentially linearly separable. Design a classifier that separates these two classes.

3.14 Suppose that in the signal-flow graph of the perceptron shown in Fig. 3.6 the hard limiter is replaced by the sigmoidal nonlinearity:

$$\varphi(v) = \tanh\left(\frac{v}{2}\right)$$

where v is the induced local field. The classification decisions made by the perceptron are defined as follows:

> *Observation vector* $\mathbf{x}$ *belongs to class* $\mathscr{C}_1$ *if the output* $y > \theta$ *where* θ *is a threshold; otherwise,* $\mathbf{x}$ *belongs to class* $\mathscr{C}_2$.

Show that the decision boundary so constructed is a hyperplane.

3.15 **(a)** The perceptron may be used to perform numerous logic functions. Demonstrate the implementation of the binary logic functions AND, OR, and COMPLEMENT.

(b) A basic limitation of the perceptron is that it cannot implement the EXCLUSIVE OR function. Explain the reason for this limitation.

3.16 Equations (3.86) and (3.87) define the weight vector and bias of the Bayes classifier for a Gaussian environment. Determine the composition of this classifier for the case when the covariance matrix $\mathbf{C}$ is defined by

$$\mathbf{C} = \sigma^2\mathbf{I}$$

where σ^2 is a constant.