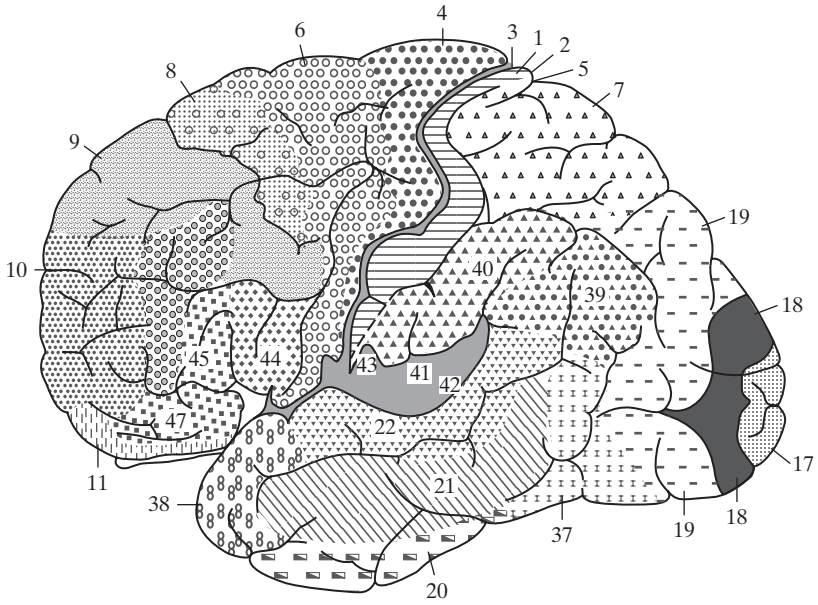


FIGURE 4 Cytoarchitectural map of the cerebral cortex. The different areas are identified by the thickness of their layers and types of cells within them. Some of the key sensory areas are as follows: Motor cortex: motor strip, area 4; premotor area, area 6; frontal eye fields, area 8. Somatosensory cortex: areas 3, 1, and 2. Visual cortex: areas 17, 18, and 19. Auditory cortex: areas 41 and 42. *(From A. Brodal, 1981; with permission of Oxford University Press.)*

3 MODELS OF A NEURON

A *neuron* is an information-processing unit that is fundamental to the operation of a neural network. The block diagram of Fig. 5 shows the *model* of a neuron, which forms the basis for designing a large family of neural networks studied in later chapters. Here, we identify three basic elements of the neural model:

1. A set of *synapses*, or *connecting links*, each of which is characterized by a *weight* or *strength* of its own. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}. It is important to make a note of the manner in which the subscripts of the synaptic weight w_{kj} are written. The first subscript in w_{kj} refers to the neuron in question, and the second subscript refers to the input end of the synapse to which the weight refers. Unlike the weight of a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.
2. An *adder* for summing the input signals, weighted by the respective synaptic strengths of the neuron; the operations described here constitute a *linear combiner*.
3. An *activation function* for limiting the amplitude of the output of a neuron. The activation function is also referred to as a *squashing function*, in that it squashes (limits) the permissible amplitude range of the output signal to some finite value.

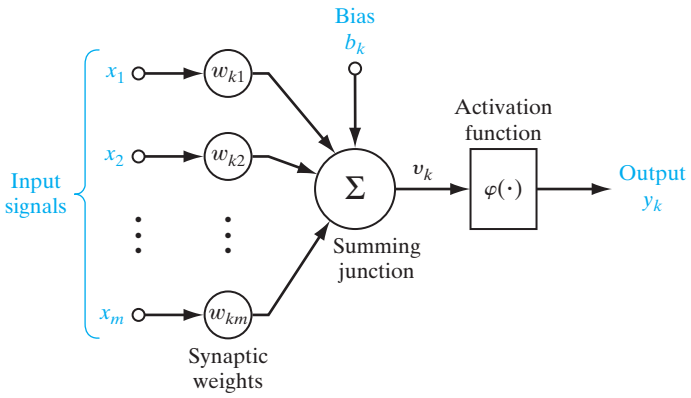


FIGURE 5 Nonlinear model of a neuron, labeled k.

Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0,1], or, alternatively, [−1,1].

The neural model of Fig. 5 also includes an externally applied *bias*, denoted by b_k. The bias b_k has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative, respectively.

In mathematical terms, we may describe the neuron k depicted in Fig. 5 by writing the pair of equations:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

and

$$y_k = \varphi(u_k + b_k) \tag{2}$$

where $x_1, x_2, \ldots, x_m$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{km}$ are the respective synaptic weights of neuron k; u_k (not shown in Fig. 5) is the *linear combiner output* due to the input signals; b_k is the bias; $\varphi(\cdot)$ is the *activation function*; and y_k is the output signal of the neuron. The use of bias b_k has the effect of applying an *affine transformation* to the output u_k of the linear combiner in the model of Fig. 5, as shown by

$$v_k = u_k + b_k \tag{3}$$

In particular, depending on whether the bias b_k is positive or negative, the relationship between the *induced local field*, or *activation potential*, v_k of neuron k and the linear combiner output u_k is modified in the manner illustrated in Fig. 6; hereafter, these two terms are used interchangeably. Note that as a result of this affine transformation, the graph of v_k versus u_k no longer passes through the origin.

The bias b_k is an external parameter of neuron k. We may account for its presence as in Eq. (2). Equivalently, we may formulate the combination of Eqs. (1) to (3) as follows:

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{4}$$

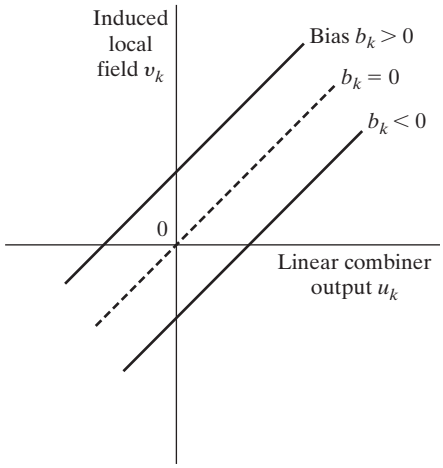


FIGURE 6 Affine transformation produced by the presence of a bias; note that $v_k = b_k$ at $u_k = 0$.

and

$$y_k = \varphi(v_k) \tag{5}$$

In Eq. (4), we have added a new synapse. Its input is

$$x_0 = +1 \tag{6}$$

and its weight is

$$w_{k0} = b_k \tag{7}$$

We may therefore reformulate the model of neuron k as shown in Fig. 7. In this figure, the effect of the bias is accounted for by doing two things: (1) adding a new input signal fixed at $+1$, and (2) adding a new synaptic weight equal to the bias b_k. Although the models of Figs. 5 and 7 are different in appearance, they are mathematically equivalent.

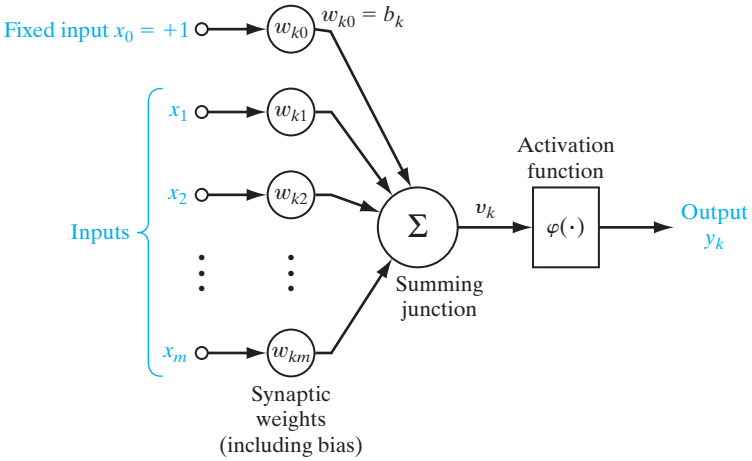


FIGURE 7 Another nonlinear model of a neuron; w_{k0} accounts for the bias b_k.

Types of Activation Function

The activation function, denoted by $\varphi(v)$, defines the output of a neuron in terms of the induced local field v. In what follows, we identify two basic types of activation functions:

1. *Threshold Function.* For this type of activation function, described in Fig. 8a, we have

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \tag{8}$$

In engineering, this form of a threshold function is commonly referred to as a *Heaviside function.* Correspondingly, the output of neuron k employing such a threshold function is expressed as

$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0 \\ 0 & \text{if } v_k < 0 \end{cases} \tag{9}$$

where v_k is the induced local field of the neuron; that is,

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{10}$$

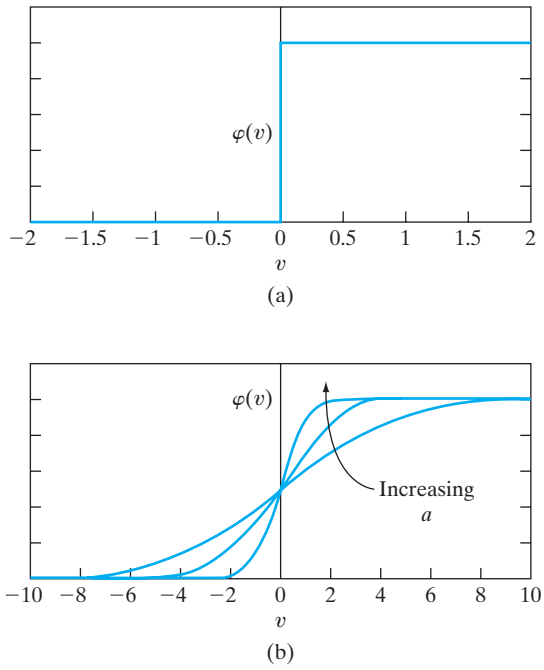


FIGURE 8 (a) Threshold function. (b) Sigmoid function for varying slope parameter a.

In neural computation, such a neuron is referred to as the *McCulloch–Pitts model*, in recognition of the pioneering work done by McCulloch and Pitts (1943). In this model, the output of a neuron takes on the value of 1 if the induced local field of that neuron is nonnegative, and 0 otherwise. This statement describes the *all-or-none property* of the McCulloch–Pitts model.

2. *Sigmoid Function.*[4] The sigmoid function, whose graph is "S"-shaped, is by far the most common form of activation function used in the construction of neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. An example of the sigmoid function is the *logistic function*,[5] defined by

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{11}$$

where a is the *slope parameter* of the sigmoid function. By varying the parameter a, we obtain sigmoid functions of different slopes, as illustrated in Fig. 8b. In fact, the slope at the origin equals $a/4$. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function. Whereas a threshold function assumes the value of 0 or 1, a sigmoid function assumes a continuous range of values from 0 to 1. Note also that the sigmoid function is differentiable, whereas the threshold function is not. (Differentiability is an important feature of neural network theory, as described in Chapter 4).

The activation functions defined in Eqs. (8) and (11) range from 0 to +1. It is sometimes desirable to have the activation function range from −1 to +1, in which case, the activation function is an odd function of the induced local field. Specifically, the threshold function of Eq. (8) is now defined as

$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{12}$$

which is commonly referred to as the *signum function.* For the corresponding form of a sigmoid function, we may use the *hyperbolic tangent function*, defined by

$$\varphi(v) = \tanh(v) \tag{13}$$

Allowing an activation function of the sigmoid type to assume negative values as prescribed by Eq. (13) may yield practical benefits over the logistic function of Eq. (11).

Stochastic Model of a Neuron

The neural model described in Fig. 7 is deterministic in that its input–output behavior is precisely defined for all inputs. For some applications of neural networks, it is desirable to base the analysis on a stochastic neural model. In an analytically tractable approach, the activation function of the McCulloch–Pitts model is given a probabilistic interpretation. Specifically, a neuron is permitted to reside in only one of two states: +1

or −1, say. The decision for a neuron to *fire* (i.e., switch its state from "off" to "on") is probabilistic. Let x denote the state of the neuron and $P(v)$ denote the *probability* of firing, where v is the induced local field of the neuron. We may then write

$$x = \begin{cases} +1 & \text{with probability } P(v) \\ -1 & \text{with probability } 1 - P(v) \end{cases} \tag{14}$$

A standard choice for $P(v)$ is the sigmoid-shaped function

$$P(v) = \frac{1}{1 + \exp(-v/T)} \tag{15}$$

where T is a *pseudotemperature* used to control the noise level and therefore the uncertainty in firing (Little, 1974). It is important to realize, however, that T is *not* the physical temperature of a neural network, be it a biological or an artificial neural network. Rather, as already stated, we should think of T merely as a parameter that controls the thermal fluctuations representing the effects of synaptic noise. Note that when $T \to 0$, the stochastic neuron described by Eqs. (14) and (15) reduces to a noiseless (i.e., deterministic) form, namely, the McCulloch–Pitts model.

4 NEURAL NETWORKS VIEWED AS DIRECTED GRAPHS

The *block diagram* of Fig. 5 or that of Fig. 7 provides a functional description of the various elements that constitute the model of an artificial neuron. We may simplify the appearance of the model by using the idea of signal-flow graphs without sacrificing any of the functional details of the model. Signal-flow graphs, with a well-defined set of rules, were originally developed by Mason (1953, 1956) for linear networks. The presence of nonlinearity in the model of a neuron limits the scope of their application to neural networks. Nevertheless, signal-flow graphs do provide a neat method for the portrayal of the flow of signals in a neural network, which we pursue in this section.

A *signal-flow graph* is a network of directed *links* (*branches*) that are interconnected at certain points called *nodes*. A typical node j has an associated *node signal* x_j. A typical directed link originates at node j and terminates on node k; it has an associated *transfer function*, or *transmittance*, that specifies the manner in which the signal y_k at node k depends on the signal x_j at node j. The flow of signals in the various parts of the graph is dictated by three basic rules:

Rule 1. A signal flows along a link only in the direction defined by the arrow on the link.

Two different types of links may be distinguished:

- *Synaptic links*, whose behavior is governed by a *linear* input–output relation. Specifically, the node signal x_j is multiplied by the synaptic weight w_{kj} to produce the node signal y_k, as illustrated in Fig. 9a.
- *Activation links*, whose behavior is governed in general by a *nonlinear* input–output relation. This form of relationship is illustrated in Fig. 9b, where $\varphi(\cdot)$ is the nonlinear activation function.

CHAPTER 1

Rosenblatt's Perceptron

ORGANIZATION OF THE CHAPTER

The perceptron occupies a special place in the historical development of neural networks: It was the first algorithmically described neural network. Its invention by Rosenblatt, a psychologist, inspired engineers, physicists, and mathematicians alike to devote their research effort to different aspects of neural networks in the 1960s and the 1970s. Moreover, it is truly remarkable to find that the perceptron (in its basic form as described in this chapter) is as valid today as it was in 1958 when Rosenblatt's paper on the perceptron was first published.

The chapter is organized as follows:

1. Section 1.1 expands on the formative years of neural networks, going back to the pioneering work of McCulloch and Pitts in 1943.
2. Section 1.2 describes Rosenblatt's perceptron in its most basic form. It is followed by Section 1.3 on the perceptron convergence theorem. This theorem proves convergence of the perceptron as a linearly separable pattern classifier in a finite number time-steps.
3. Section 1.4 establishes the relationship between the perceptron and the Bayes classifier for a Gaussian environment.
4. The experiment presented in Section 1.5 demonstrates the pattern-classification capability of the perceptron.
5. Section 1.6 generalizes the discussion by introducing the perceptron cost function, paving the way for deriving the batch version of the perceptron convergence algorithm.

Section 1.7 provides a summary and discussion that conclude the chapter.

1.1 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 idea of Hebbian learning will be discussed at some length in Chapter 8. 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 perform 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.

1.2 PERCEPTRON

Rosenblatt's perceptron is built around a nonlinear neuron, namely, the *McCulloch–Pitts model* of a neuron. From the introductory chapter we recall that such a neural modeling consists of a linear combiner followed by a hard limiter (performing the signum function), as depicted in Fig. 1.1. The summing node of the neural model computes a linear combination of the inputs applied to its synapses, as well as incorporates an externally applied bias. The resulting sum, that is, the induced local field, is applied to a hard

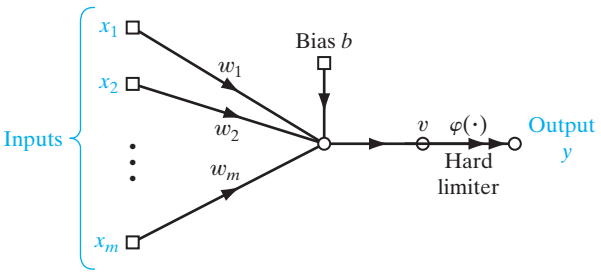


FIGURE 1.1 Signal-flow graph of the perceptron.

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. 1.1, the synaptic weights of the perceptron are denoted by $w_1, w_2, \ldots, w_m$. Correspondingly, the inputs applied to 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

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{1.1}$$

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 $x_1, x_2, \ldots, x_m$. In the simplest form of the perceptron, there are two decision regions separated by a *hyperplane*, which is defined by

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{1.2}$$

This is illustrated in Fig. 1.2 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, discussed next.

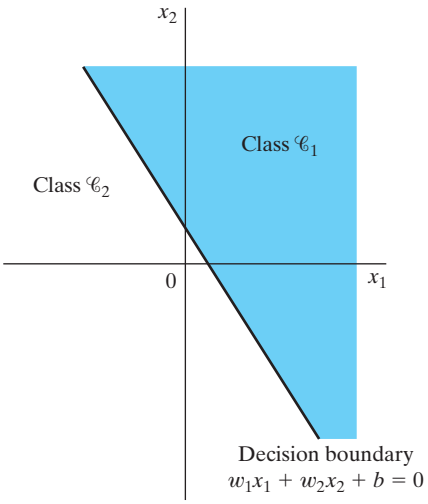


FIGURE 1.2 Illustration of the hyperplane (in this example, a straight line) as decision boundary for a two-dimensional, two-class pattern-classification problem.

1.3 THE 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. 1.3. In this second model, which is equivalent to that of Fig. 1.1, 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

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

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

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

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

$$\begin{aligned} v(n) &= \sum_{i=0}^{m} w_i(n)x_i(n) \\ &= \mathbf{w}^T(n)\mathbf{x}(n) \end{aligned} \tag{1.3}$$

where, in the first line, $w_0(n)$, corresponding to $i = 0$, represents the bias b. For fixed n, the equation $\mathbf{w}^T\mathbf{x} = 0$, plotted in an m-dimensional space (and 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. 1.4 for the case of a two-dimensional perceptron. In Fig. 1.4a, 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 striaght 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. 1.4b, 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 $\mathscr{H}_1$ be the subspace of training vectors $\mathbf{x}_1(1), \mathbf{x}_1(2), \ldots$ that belong to class $\mathscr{C}_1$, and let $\mathscr{H}_2$ be the subspace of training vectors $\mathbf{x}_2(1), \mathbf{x}_2(2), \ldots$ that belong to class $\mathscr{C}_2$. The union of $\mathscr{H}_1$ and $\mathscr{H}_2$ is the complete space denoted by $\mathscr{H}$. Given the sets

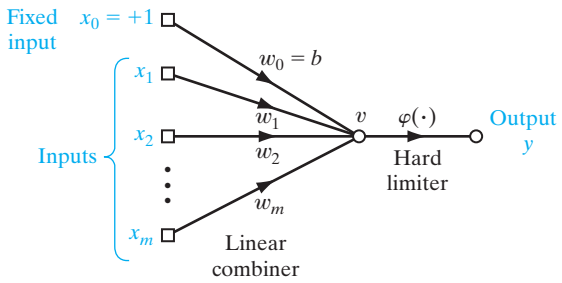


FIGURE 1.3 Equivalent signal-flow graph of the perceptron; dependence on time has been omitted for clarity.

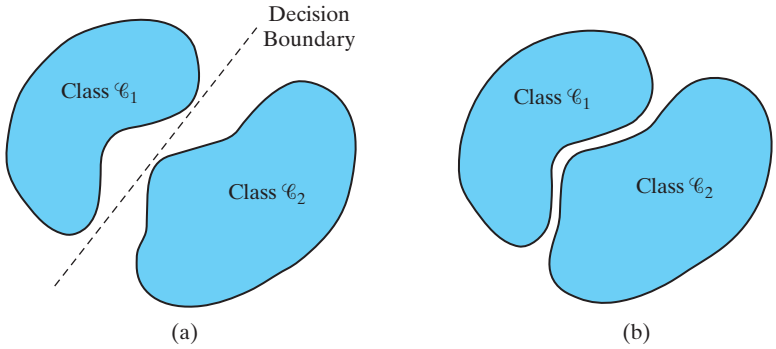


FIGURE 1.4 (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable patterns.

of vectors $\mathscr{H}_1$ and $\mathscr{H}_2$ 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

$$\begin{aligned} \mathbf{w}^T\mathbf{x} &> 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } \mathscr{C}_1 \\ \mathbf{w}^T\mathbf{x} &\leq 0 \text{ for every input vector } \mathbf{x} \text{ belonging to class } \mathscr{C}_2 \end{aligned} \tag{1.4}$$

In the second line of Eq. (1.4), 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 $\mathscr{H}_1$ and $\mathscr{H}_2$, the training problem for the perceptron is then to find a weight vector $\mathbf{w}$ such that the two inequalities of Eq. (1.4) 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:

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{aligned} \tag{1.5}$$

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

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \end{aligned} \tag{1.6}$$

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, then 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.

Proof of the perceptron convergence algorithm[2] is presented for the initial condition $\mathbf{w}(0) = \mathbf{0}$. Suppose that $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ for $n = 1, 2, ...$, and the input vector $\mathbf{x}(n)$ belongs to the subset $\mathcal{H}_1$. That is, the perceptron incorrectly classifies the vectors $\mathbf{x}(1)$, $\mathbf{x}(2)$, ..., since the first condition of Eq. (1.4) is violated. Then, with the constant $\eta(n) = 1$, we may use the second line of Eq. (1.6) to write

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \mathbf{x}(n) \qquad \text{for } \mathbf{x}(n) \text{ belonging to class } \mathcal{C}_1 \tag{1.7}$$

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

$$\mathbf{w}(n + 1) = \mathbf{x}(1) + \mathbf{x}(2) + \cdots + \mathbf{x}(n) \tag{1.8}$$

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

$$\alpha = \min_{\mathbf{x}(n) \in \mathcal{H}_1} \mathbf{w}_o^T \mathbf{x}(n) \tag{1.9}$$

Hence, multiplying both sides of Eq. (1.8) by the row vector $\mathbf{w}_o^T$, we get

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

Accordingly, in light of the definition given in Eq. (1.9), we have

$$\mathbf{w}_o^T\mathbf{w}(n + 1) \geq n\alpha \tag{1.10}$$

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

$$\|\mathbf{w}_o\|^2 \|\mathbf{w}(n + 1)\|^2 \geq [\mathbf{w}_o^T\mathbf{w}(n + 1)]^2 \tag{1.11}$$

where $\|\cdot\|$ denotes the Euclidean norm of the enclosed argument vector, and the inner product $\mathbf{w}_o^T\mathbf{w}(n + 1)$ is a scalar quantity. We now note from Eq. (1.10) that $[\mathbf{w}_o^T\mathbf{w}(n + 1)]^2$ is equal to or greater than $n^2\alpha^2$. From Eq. (1.11) we note that $\|\mathbf{w}_o\|^2\|\mathbf{w}(n + 1)\|^2$ is equal to or greater than $[\mathbf{w}_o^T\mathbf{w}(n + 1)]^2$. It follows therefore that

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

or, equivalently,

$$\|\mathbf{w}(n + 1)\|^2 \geq \frac{n^2\alpha^2}{\|\mathbf{w}_o\|^2} \tag{1.12}$$

We next follow another development route. In particular, we rewrite Eq. (1.7) in the form

$$\mathbf{w}(k + 1) = \mathbf{w}(k) + \mathbf{x}(k) \qquad \text{for } k = 1, ..., n \quad \text{and} \quad \mathbf{x}(k) \in \mathcal{H}_1 \tag{1.13}$$

By taking the squared Euclidean norm of both sides of Eq. (1.13), we obtain

$$\|\mathbf{w}(k + 1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \tag{1.14}$$

But, $\mathbf{w}^T(k)\mathbf{x}(k) \leq 0$. We therefore deduce from Eq. (1.14) that

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

or, equivalently,

$$\|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \leq \|\mathbf{x}(k)\|^2, \qquad k = 1, ..., n \tag{1.15}$$

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

$$\begin{aligned}\|\mathbf{w}(n+1)\|^2 &\leq \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2 \\ &\leq n\beta\end{aligned} \tag{1.16}$$

where β is a positive number defined by

$$\beta = \max_{\mathbf{x}(k) \in \mathcal{H}_1} \|\mathbf{x}(k)\|^2 \tag{1.17}$$

Equation (1.16) 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. (1.16) is clearly in conflict with the earlier result of Eq. (1.12) for sufficiently large values of n. Indeed, we can state that n cannot be larger than some value $n_{\max}$ for which Eqs. (1.12) and (1.16) are both satisfied with the equality sign. That is, $n_{\max}$ is the solution of the equation

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

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

$$n_{\max} = \frac{\beta \|\mathbf{w}_o\|^2}{\alpha^2} \tag{1.18}$$

We have thus proved that for $\eta(n) = 1$ for all n and $\mathbf{w}(0) = \mathbf{0}$, and given that a solution vector $\mathbf{w}_o$ exists, the rule for adapting the synaptic weights of the perceptron must terminate after at most $n_{\max}$ interations. Surprisingly, this statement, proved for hypothesis $\mathcal{H}_1$, also holds for huypothesis $\mathcal{H}_2$. Note however,

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

> *Let the subsets of training vectors $\mathcal{H}_1$ and $\mathcal{H}_2$ be linearly separable. Let the inputs presented to the perceptron originate from these two subsets. The perceptron converges after some n_o iterations, in the sense that*
>
> $$\mathbf{w}(n_o) = \mathbf{w}(n_o + 1) = \mathbf{w}(n_o + 2) = \cdots$$
>
> *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 the condition

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

holds. 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, at iteration n, 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 $\mathbf{w}_o$. Regardless of the value assigned to $\mathbf{w}(0)$, the perceptron is assured of convergence.

In Table 1.1, we present a summary of the *perceptron convergence algorithm* (Lippmann, 1987). The symbol $\text{sgn}(\cdot)$, used in step 3 of the table for computing the actual response of the perceptron, stands for the *signum function*:

$$\text{sgn}(v) = \begin{cases} +1 & \text{if } v > 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{1.19}$$

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

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)] \tag{1.20}$$

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

TABLE 1.1 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
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = 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 as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where $\text{sgn}(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
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.

$(m+1)$-by-1 vector whose first element equals the bias b. One other important point to note in Table 1.1 is that we have introduced a *quantized desired response* $d(n)$, defined by

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases} \tag{1.21}$$

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

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n) \tag{1.22}$$

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.

1.4 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) as

$$\begin{aligned} \mathscr{R} = {} & c_{11}p_1 \int_{\mathscr{H}_1} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathscr{H}_2} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathscr{H}_2} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} + c_{12}p_2 \int_{\mathscr{H}_1} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} \end{aligned} \tag{1.23}$$

where the various terms are defined as follows:

p_i = *prior probability* that the observation vector $\mathbf{x}$ (representing a realization of the random vector $\mathbf{X}$) corresponds to an object in class $\mathcal{C}_1$, 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 $\mathscr{H}_i$ when class $\mathscr{C}_j$ is true (i.e., observation vector $\mathbf{x}$ corresponds to an object in class $\mathcal{C}_1$), with $i, j = 1, 2$

$p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_i)$ = conditional probability density function of the random vector $\mathbf{X}$, given that the observation vector $\mathbf{x}$ corresponds to an object in class $\mathcal{C}_1$, with $i = 1, 2$.

The first two terms on the right-hand side of Eq. (1.23) 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., *prior* 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,

$$\mathscr{X} = \mathscr{X}_1 + \mathscr{X}_2 \tag{1.24}$$

Accordingly, we may rewrite Eq. (1.23) in the equivalent form

$$\begin{aligned}\mathscr{R} = {} & c_{11}p_1 \int_{\mathscr{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} + c_{22}p_2 \int_{\mathscr{X}-\mathscr{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} \\ & + c_{21}p_1 \int_{\mathscr{X}-\mathscr{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)\mathbf{x} + c_{12}p_2 \int_{\mathscr{X}_1} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x}\end{aligned} \tag{1.25}$$

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

$$\int_{\mathscr{X}} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)d\mathbf{x} = \int_{\mathscr{X}} p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)d\mathbf{x} = 1 \tag{1.26}$$

Hence, Eq. (1.25) reduces to

$$\begin{aligned}\mathscr{R} = {} & c_{21}p_1 + c_{22}p_2 \\ & + \int_{\mathscr{X}_1} [p_2(c_{12} - c_{22})\, p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2) - p_1(c_{21} - c_{11})\, p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)]d\mathbf{x}\end{aligned} \tag{1.27}$$

The first two terms on the right-hand side of Eq. (1.27) represent a fixed cost. Since the requirement is to minimize the average risk $\mathscr{R}$, we may therefore deduce the following strategy from Eq.(1.27) 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 subset $\mathscr{X}_1$ (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 subset $\mathscr{X}_1$ (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 assingned arbitrarily. We shall assume that these points are assigned to subset $\mathscr{X}_2$ (i.e., class $\mathscr{C}_2$).

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

If the condition

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

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

To simplify matters, define

$$\Lambda(\mathbf{x}) = \frac{p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_1)}{p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_2)} \tag{1.28}$$

and

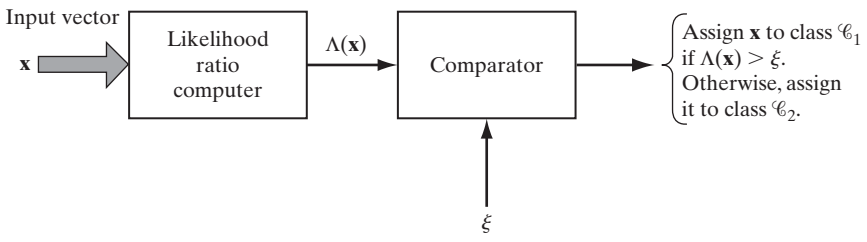
$$\xi = \frac{p_2(c_{12} - c_{22})}{p_1(c_{21} - c_{11})} \tag{1.29}$$

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 the following

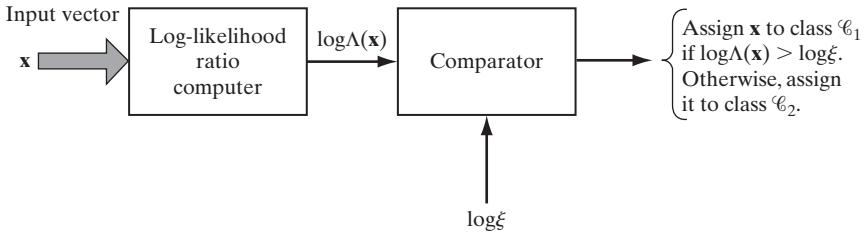
If, for an observation vector $\mathbf{x}$*, the likelihood ratio* $\Lambda(\mathbf{x})$ *is greater than the threshold* ξ*, assign* $\mathbf{x}$ *to class* $\mathscr{C}_1$*. Otherwise, assign it to class* $\mathscr{C}_2$*.*

Figure 1.5a depicts a block-diagram representation of the Bayes classifier. The important points in this block diagram are twofold:

1. The data processing involved in designing the Bayes classifier is confined entirely to the computation of the likelihood ratio $\Lambda(\mathbf{x})$.



(a)



(b)

FIGURE 1.5 Two equivalent implementations of the Bayes classifier: (a) Likelihood ratio test, (b) Log-likelihood ratio test.

2. This computation is completely invariant to the values assigned to the prior 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. 1.5b. 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,

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

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 the multivariate Gaussian distribution

$$p_{\mathbf{X}}(\mathbf{x}|\mathscr{C}_i) = \frac{1}{(2\pi)^{m/2}(\det(\mathbf{C}))^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, 2 \tag{1.30}$$

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

We further assume the following:

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

$$p_1 = p_2 = \frac{1}{2} \tag{1.31}$$

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

$$c_{21} = c_{12} \quad \text{and} \quad c_{11} = c_{22} = 0 \tag{1.32}$$

We now have the information we need to design the Bayes classifier for the two-class problem. Specifically, by substituting Eq. (1.30) into (1.28) and taking the natural logarithm, we get (after simplifications)

$$\begin{aligned}\log\Lambda(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) \\ &= (\boldsymbol{\mu}_1-\boldsymbol{\mu}_2)^T\mathbf{C}^{-1}\mathbf{x} + \frac{1}{2}(\boldsymbol{\mu}_2^T\mathbf{C}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T\mathbf{C}^{-1}\boldsymbol{\mu}_1)\end{aligned} \tag{1.33}$$

By substituting Eqs. (1.31) and (1.32) into Eq. (1.29) and taking the natural logarithm, we get

$$\log\xi = 0 \tag{1.34}$$

Equations (1.33) and (1.34) state that the Bayes classifier for the problem at hand is a *linear classifier*, as described by the relation

$$y = \mathbf{w}^T\mathbf{x} + b \tag{1.35}$$

where

$$\mathbf{y} = \log\Lambda(\mathbf{x}) \tag{1.36}$$

$$\mathbf{w} = \mathbf{C}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{1.37}$$

$$b = \frac{1}{2}(\boldsymbol{\mu}_2^T\mathbf{C}^{-1}\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T\mathbf{C}^{-1}\boldsymbol{\mu}_1) \tag{1.38}$$

More specifically, the classifier consists of a linear combiner with weight vector $\mathbf{w}$ and bias b, as shown in Fig. 1.6.

On the basis of Eq. (1.35), 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. (1.1) and (1.35). 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 distributions of the two patterns assumed in the derivation of the Bayes classifier certainly do *overlap* each other and are therefore *not* separable. The extent of the overlap is determined by the mean vectors

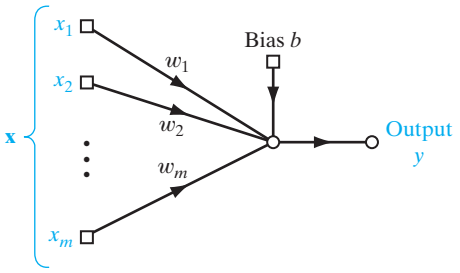


FIGURE 1.6 Signal-flow graph of Gaussian classifier.

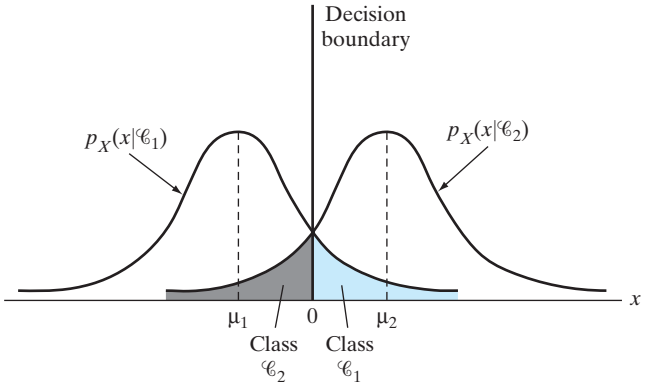


FIGURE 1.7 Two overlapping, one-dimensional Gaussian distributions.

$\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ and the covariance matrix $\mathbf{C}$. The nature of this overlap is illustrated in Fig. 1.7 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. 1.7, 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 be 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.

1.5 COMPUTER EXPERIMENT: PATTERN CLASSIFICATION

The objective of this computer experiment is twofold:

(i) to lay down the specifications of a *double-moon classification problem* that will serve as the basis of a prototype for the part of the book that deals with pattern-classification experiments;

(ii) to demonstrate the capability of Rosenblatt's perceptron algorithm to correctly classify linearly separable patterns and to show its breakdown when the condition of linear separability is violated.

Specifications of the Classification Problem

Figure 1.8 shows a pair of "moons" facing each other in an *asymmetrically* arranged manner. The moon labeled "Region A" is positioned symmetrically with respect to the y-axis, whereas the moon labeled "Region B" is displaced to the right of the y-axis by an amount equal to the radius r and below the x-axis by the distance d. The two moons have identical parameters:

$$\text{radius of each moon, } r = 10$$
$$\text{width of each moon, } w = 6$$

The vertical distance d separating the two moons is adjustable; it is measured with respect to the x-axis, as indicated in Fig. 1.8:

- Increasingly positive values of d signify increased separation between the two moons;
- increasingly negative values of d signify the two moons' coming closer to each other.

The training sample $\mathcal{T}$ consists of 1,000 pairs of data points, with each pair consisting of one point picked from region A and another point picked from region B, both randomly. The test sample consists of 2,000 pairs of data points, again picked in a random manner.

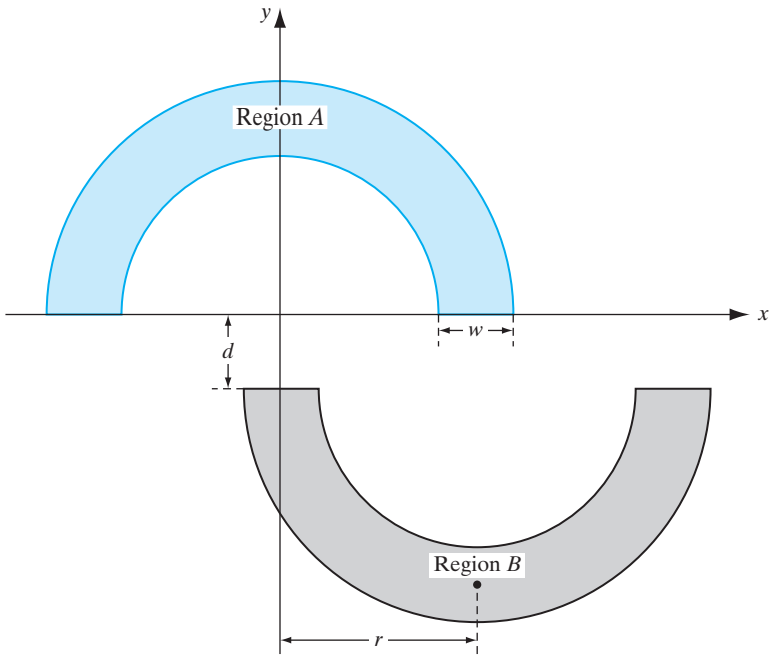


FIGURE 1.8 The double-moon classification problem.

The Experiment

The perceptron parameters picked for the experiment were as follows:

$$\text{size of the input layer} = 2$$
$$\beta = 50; \text{ see Eq. (1.17)}$$

The learning-rate parameter η was varied linearly from 10^{-1} down to 10^{-5}.

The weights were initially all set at zero.

Figure 1.9 presents the results of the experiment for $d = 1$, which corresponds to perfect linear separability. Part (a) of the figure presents the *learning curve*, where the mean-square error (MSE) is plotted versus the number of epochs; the figure shows convergence of the algorithm in three iterations. Part (b) of the figure shows the decision boundary computed through training of the perceptron algorithm, demonstrating perfect separability of all 2,000 test points.

In Fig. 1.10, the separation between the two moons was set at $d = -4$, a condition that violates linear separability. Part (a) of the figure displays the learning curve where the perceptron algorithm is now found to fluctuate continuously, indicating breakdown of the algorithm. This result is confirmed in part (b) of the figure, where the decision boundary (computed through training) intersects both moons, with a classification error rate of $(186/2000) \times 100\% = 9.3\%$.

1.6 THE BATCH PERCEPTRON ALGORITHM

The derivation of the perceptron convergence algorithm summarized in Table 1.1 was presented without reference to a cost function. Moreover, the derivation focused on a single-sample correction. In this section, we will do two things:

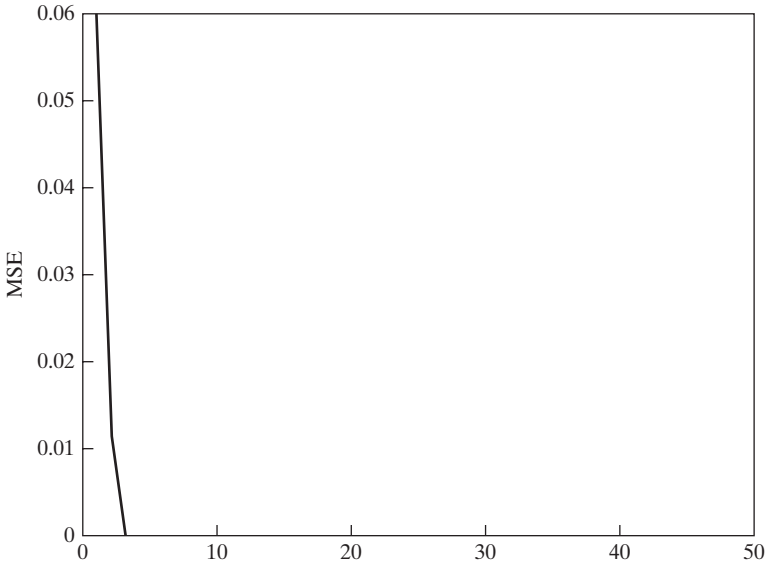
1. introduce the generalized form of a perceptron cost function;
2. use the cost function to formulate a batch version of the perceptron convergence algorithm.

The cost function we have in mind is a function that permits the application of a gradient search. Specifically, we define the *perceptron cost function* as

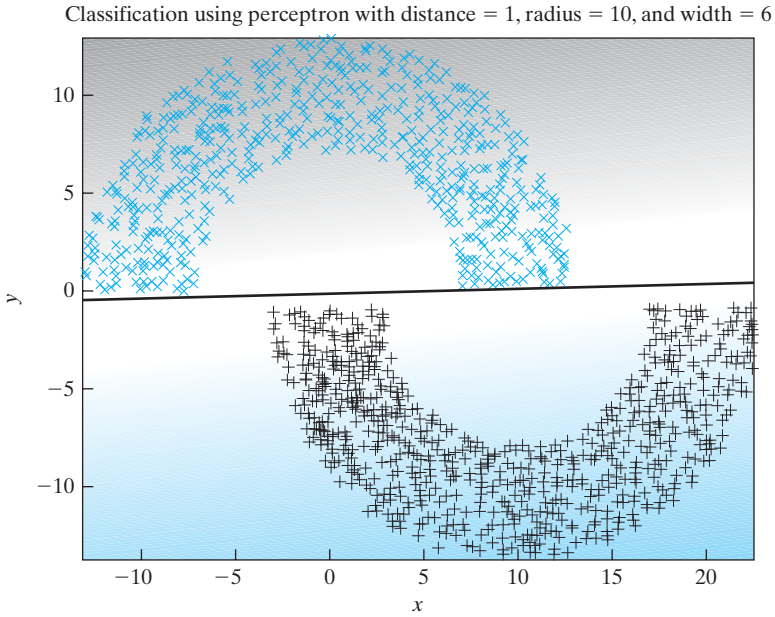
$$J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathcal{X}} (-\mathbf{w}^T \mathbf{x}(n) d(n)) \tag{1.39}$$

where $\mathcal{X}$ is the set of samples $\mathbf{x}$ *misclassified* by a perceptron using $\mathbf{w}$ as its weight vector (Duda et al., 2001). If all the samples are classified correctly, then the set $\mathcal{X}$ is empty, in which case the cost function $J(\mathbf{w})$ is zero. In any event, the nice feature of the cost function $J(\mathbf{w})$ is that it is *differentiable* with respect to the weight vector $\mathbf{w}$. Thus, differentiating $J(\mathbf{w})$ with respect to $\mathbf{w}$ yields the *gradient vector*

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathcal{X}} (-\mathbf{x}(n) d(n)) \tag{1.40}$$

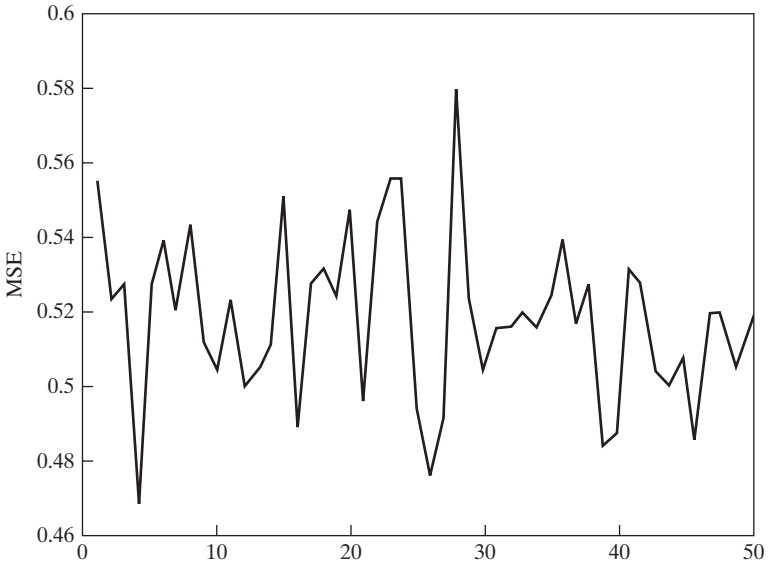


(a) learning curve

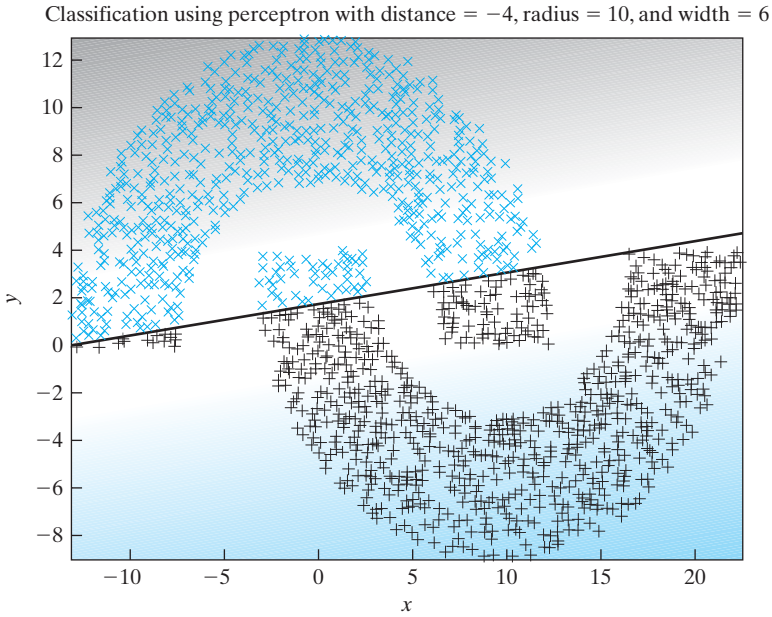


(b) testing result

FIGURE 1.9 Perceptron with the double-moon set at distance $d = 1$.



(a) learning curve



(b) testing result

FIGURE 1.10 Perceptron with the double-moon set at distance $d = -4$.

where the *gradient operator*

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_m}\right]^T \tag{1.41}$$

In the *method of steepest descent,* the adjustment to the weight vector $\mathbf{w}$ at each time-step of the algorithm is applied in a direction *opposite* to the gradient vector $\nabla J(\mathbf{w})$. Accordingly, the algorithm takes the form

$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\nabla J(\mathbf{w}) \\ &= \mathbf{w}(n) + \eta(n)\sum_{\mathbf{x}(n)\in\mathcal{X}} \mathbf{x}(n)d(n)\end{aligned} \tag{1.42}$$

which includes the single-sample correction version of the perceptron convergence algorithm as a special case. Moreover, Eq. (1.42) embodies the *batch perceptron algorithm* for computing the weight vector, given the sample set $\mathbf{x}(1), \mathbf{x}(2), ...$ In particular, the adjustment applied to the weight vector at time-step $n+1$ is defined by the sum of all the samples misclassified by the weight vector $\mathbf{w}(n)$, with the sum being scaled by the learning-rate parameter $\eta(n)$. The algorithm is said to be of the "batch" kind because at each time-step of the algorithm, a batch of misclassified samples is used to compute the adjustment.

1.7 SUMMARY AND DISCUSSION

The perceptron is a single-layer neural network, the operation of which is based on error-correlation learning. The term "single layer" is used here to signify the fact that the computation layer of the network consists of a single neuron for the case of binary classification. The learning process for pattern classification occupies a finite number of iterations and then stops. For the classification to be successful, however, the patterns would have to be linearly separable.

The perceptron uses the McCulloch–Pitts model of a neuron. In this context, 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 neural 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.

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. Section 13.2 of their book (1969) says the following:

> *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 also 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 and learning machines 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 the support vector machines discussed in Chapter 6 overcome the computational limitations of the single-layer perceptron in their own individual ways.

In closing the discussion, we may say that the perceptron is an elegant neural network designed for the classification of linearly separable patterns. Its importance is not only historical but also of practical value in the classification of linearly separable patters.

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. Proof of the perceptron convergence algorithm presented in Section 1.3 follows the classic book of Nilsson (1965).

PROBLEMS

1.1 Verify that Eqs. (1.19)–(1.22), summarizing the perceptron convergence algorithm, are consistent with Eqs. (1.5) and (1.6).

1.2 Suppose that in the signal-flow graph of the perceptron shown in Fig. 1.1, 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* $y > \xi$, *where* ξ *is a threshold; otherwise,* $\mathbf{x}$ *belongs to class* $\mathscr{C}_2$.

Show that the decision boundary so constructed is a hyperplane.

1.3 **(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.

1.4 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.

1.5 Equations (1.37) and (1.38) 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 and $\mathbf{I}$ is the identity matrix.

Computer Experiment

1.6 Repeat the computer experiment of Section 1.5, this time, however, positioning the two moons of Figure 1.8 to be on the edge of separability, that is, $d = 0$. Determine the classification error rate produced by the algorithm over 2,000 test data points.