12 Object Recognition

One of the most interesting aspects of the world is that it can be considered to be made up of patterns.
A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made, rather than by the intrinsic nature of these elements.

Norbert Wiener

Preview

We conclude our coverage of digital image processing with an introduction to techniques for object recognition. As noted in Section 1.1, we have defined the scope covered by our treatment of digital image processing to include recognition of *individual* image regions, which in this chapter we call *objects* or *patterns*.

The approaches to pattern recognition developed in this chapter are divided into two principal areas: decision-theoretic and structural. The first category deals with patterns described using quantitative descriptors, such as length, area, and texture. The second category deals with patterns best described by qualitative descriptors, such as the relational descriptors discussed in Section 11.5.

Central to the theme of recognition is the concept of "learning" from sample patterns. Learning techniques for both decision-theoretic and structural approaches are developed and illustrated in the material that follows.

12.1 Patterns and Pattern Classes

A *pattern* is an *arrangement of descriptors*, such as those discussed in Chapter 11. The name *feature* is used often in the pattern recognition literature to denote a descriptor. A *pattern class* is a family of patterns that share some common properties. Pattern classes are denoted $\omega_1, \omega_2, \ldots, \omega_W$, where W is the number of classes. Pattern recognition by machine involves

techniques for assigning patterns to their respective classes—automatically and with as little human intervention as possible.

Three common pattern arrangements used in practice are vectors (for quantitative descriptions) and strings and trees (for structural descriptions). Pattern vectors are represented by bold lowercase letters, such as **x**, **y**, and **z**, and take the form

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{12.1-1}$$

Consult the book Web site for a brief review of vectors and matrices.

where each component, x_i, represents the ith descriptor and n is the total number of such descriptors associated with the pattern. Pattern vectors are represented as columns (that is, $n \times 1$ matrices). Hence a pattern vector can be expressed in the form shown in Eq. (12.1-1) or in the equivalent form $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, where T indicates transposition. You will recognize this notation from Section 11.4.

The nature of the components of a pattern vector **x** depends on the approach used to describe the physical pattern itself. Let us illustrate with an example that is both simple and gives a sense of history in the area of classification of measurements. In a classic paper, Fisher [1936] reported the use of what then was a new technique called *discriminant analysis* (discussed in Section 12.2) to recognize three types of iris flowers (*Iris setosa*, *virginica*, and *versicolor*) by measuring the widths and lengths of their petals (Fig. 12.1).

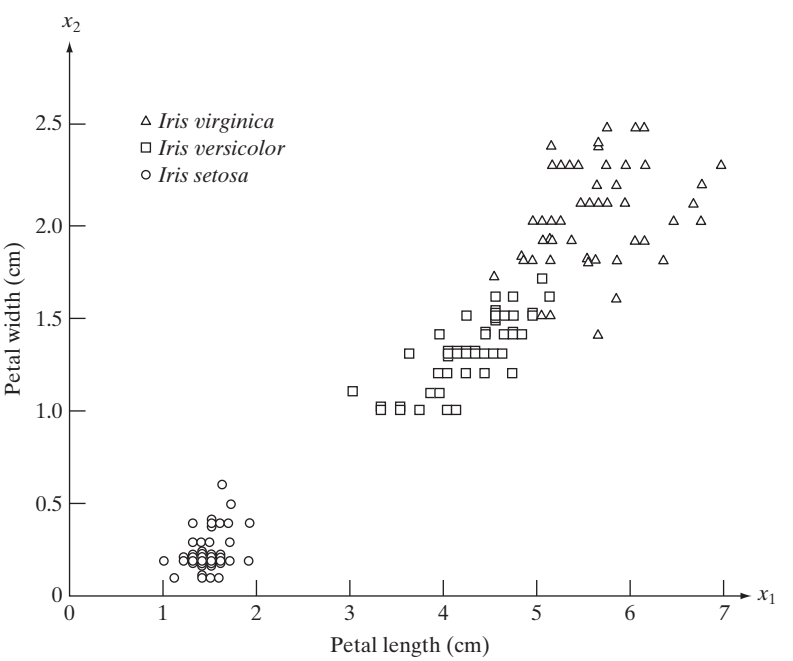


FIGURE 12.1 Three types of iris flowers described by two measurements.

In our present terminology, each flower is *described* by two measurements, which leads to a 2-D pattern vector of the form

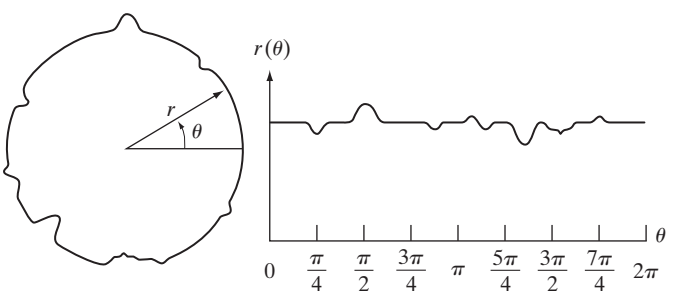
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{12.1-2}$$

where x_1 and x_2 correspond to petal length and width, respectively. The three pattern classes in this case, denoted ω_1, ω_2, and ω_3, correspond to the varieties *setosa, virginica*, and *versicolor*, respectively.

Because the petals of flowers vary in width and length, the pattern vectors describing these flowers also will vary, not only between different classes, but also within a class. Figure 12.1 shows length and width measurements for several samples of each type of iris. After a set of measurements has been selected (two in this case), the components of a pattern vector become the entire description of each physical sample. Thus each flower in this case becomes a point in 2-D Euclidean space. We note also that measurements of petal width and length in this case adequately separated the class of *Iris setosa* from the other two but did not separate as successfully the *virginica* and *versicolor* types from each other. This result illustrates the classic *feature selection* problem, in which the degree of class separability depends strongly on the choice of descriptors selected for an application. We say considerably more about this issue in Sections 12.2 and 12.3.

Figure 12.2 shows another example of pattern vector generation. In this case, we are interested in different types of noisy shapes, a sample of which is shown in Fig. 12.2(a). If we elect to represent each object by its signature (see Section 11.1.5), we would obtain 1-D signals of the form shown in Fig. 12.2(b). Suppose that we elect to describe each signature simply by its sampled amplitude values; that is, we sample the signatures at some specified interval values of θ, denoted $\theta_1, \theta_2, \ldots, \theta_n$. Then we can form pattern vectors by letting $x_1 = r(\theta_1), x_2 = r(\theta_2), \ldots, x_n = r(\theta_n)$. These vectors become points in n-dimensional Euclidean space, and pattern classes can be imagined to be "clouds" in n dimensions.

Instead of using signature amplitudes directly, we could compute, say, the first n statistical moments of a given signature (Section 11.2.4) and use these descriptors as components of each pattern vector. In fact, as may be evident by now, pattern vectors can be generated in numerous other ways.



a b

FIGURE 12.2 A noisy object and its corresponding signature.

We present some of them throughout this chapter. For the moment, the key concept to keep in mind is that selecting the descriptors on which to base each component of a pattern vector has a profound influence on the eventual performance of object recognition based on the pattern vector approach.

The techniques just described for generating pattern vectors yield pattern classes characterized by quantitative information. In some applications, pattern characteristics are best described by structural relationships. For example, fingerprint recognition is based on the interrelationships of print features called *minutiae*. Together with their relative sizes and locations, these features are primitive components that describe fingerprint ridge properties, such as abrupt endings, branching, merging, and disconnected segments. Recognition problems of this type, in which not only quantitative measures about each feature but also the spatial relationships between the features determine class membership, generally are best solved by structural approaches. This subject was introduced in Section 11.5. We revisit it briefly here in the context of pattern descriptors.

Figure 12.3(a) shows a simple staircase pattern. This pattern could be sampled and expressed in terms of a pattern vector, similar to the approach used in Fig. 12.2. However, the basic structure, consisting of repetitions of two simple primitive elements, would be lost in this method of description. A more meaningful description would be to define the elements a and b and let the pattern be the string of symbols $w = \ldots abababab \ldots$, as shown in Fig. 12.3(b). The structure of this particular class of patterns is captured in this description by requiring that connectivity be defined in a head-to-tail manner, and by allowing only alternating symbols. This structural construct is applicable to staircases of any length but excludes other types of structures that could be generated by other combinations of the primitives a and b.

String descriptions adequately generate patterns of objects and other entities whose structure is based on relatively simple connectivity of primitives, usually associated with boundary shape. A more powerful approach for many applications is the use of tree descriptions, as defined in Section 11.5. Basically, most hierarchical ordering schemes lead to tree structures. For example, Fig. 12.4 is a satellite image of a heavily built downtown area and surrounding residential areas. Let us define the entire image area by the symbol \$. The (upside down) tree representation shown in Fig. 12.5 was obtained by

a b

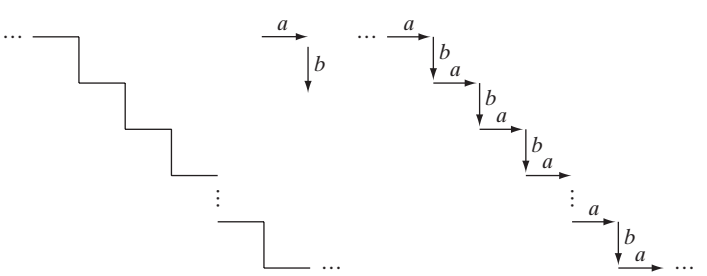


FIGURE 12.3
(a) Staircase structure.
(b) Structure coded in terms of the primitives a and b to yield the string description $\ldots ababab \ldots$.

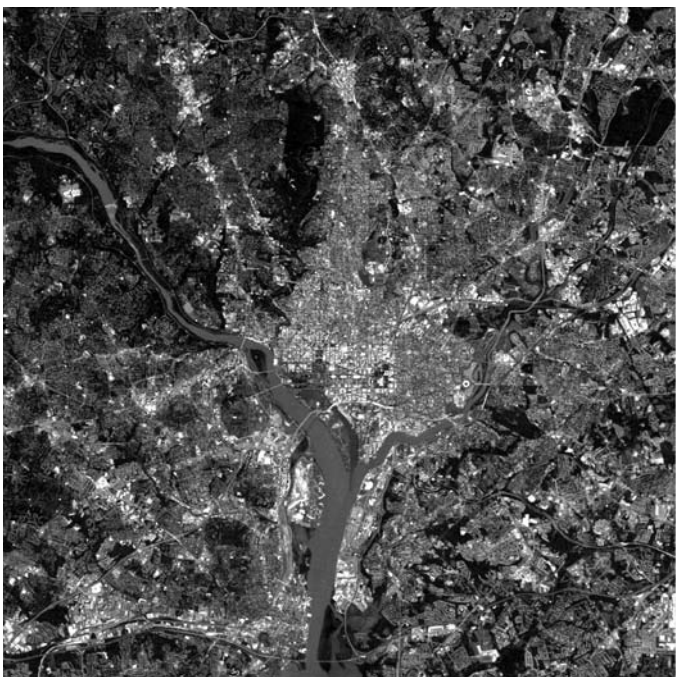


FIGURE 12.4 Satellite image of a heavily built downtown area (Washington, D.C.) and surrounding residential areas. (Courtesy of NASA.)

using the structural relationship "composed of." Thus the root of the tree represents the entire image. The next level indicates that the image is composed of a downtown and residential area. The residential area, in turn is composed of housing, highways, and shopping malls. The next level down further describes the housing and highways. We can continue this type of subdivision until we reach the limit of our ability to resolve different regions in the image.

We develop in the following sections recognition approaches for objects described by the techniques discussed in the preceding paragraphs.

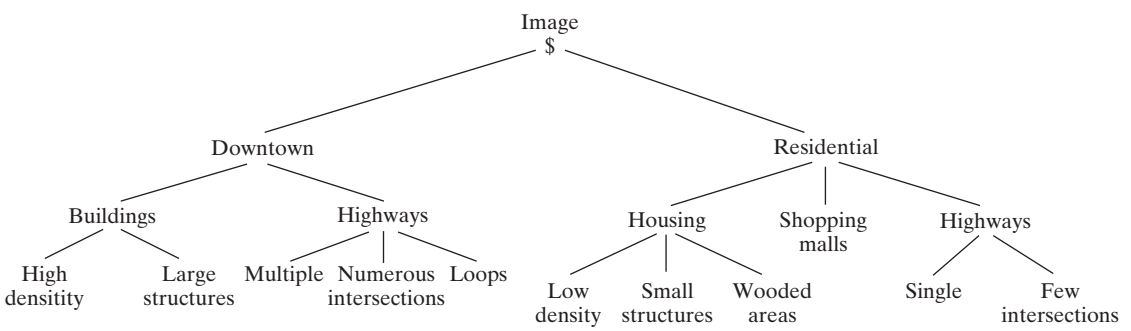


FIGURE 12.5 A tree description of the image in Fig. 12.4.

12.2 Recognition Based on Decision-Theoretic Methods

Decision-theoretic approaches to recognition are based on the use of *decision* (or *discriminant*) *functions*. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ represent an n-dimensional pattern vector, as discussed in Section 12.1. For W pattern classes $\omega_1, \omega_2, \ldots, \omega_W$, the basic problem in decision-theoretic pattern recognition is to find W decision functions $d_1(\mathbf{x}), d_2(\mathbf{x}), \ldots, d_W(\mathbf{x})$ with the property that, if a pattern $\mathbf{x}$ belongs to class ω_i, then

$$d_i(\mathbf{x}) > d_j(\mathbf{x}) \qquad j = 1, 2, \ldots, W; j \neq i \tag{12.2-1}$$

In other words, an unknown pattern $\mathbf{x}$ is said to belong to the ith pattern class if, upon substitution of $\mathbf{x}$ into all decision functions, $d_i(\mathbf{x})$ yields the largest numerical value. Ties are resolved arbitrarily.

The *decision boundary* separating class ω_i from ω_j is given by values of $\mathbf{x}$ for which $d_i(\mathbf{x}) = d_j(\mathbf{x})$ or, equivalently, by values of $\mathbf{x}$ for which

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0 \tag{12.2-2}$$

Common practice is to identify the decision boundary between two classes by the single function $d_{ij}(\mathbf{x}) = d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0$. Thus $d_{ij}(\mathbf{x}) > 0$ for patterns of class ω_i and $d_{ij}(\mathbf{x}) < 0$ for patterns of class ω_j. The principal objective of the discussion in this section is to develop various approaches for finding decision functions that satisfy Eq. (12.2-1).

12.2.1 Matching

Recognition techniques based on matching represent each class by a prototype pattern vector. An unknown pattern is assigned to the class to which it is closest in terms of a predefined metric. The simplest approach is the minimum distance classifier, which, as its name implies, computes the (Euclidean) distance between the unknown and each of the prototype vectors. It chooses the smallest distance to make a decision. We also discuss an approach based on correlation, which can be formulated directly in terms of images and is quite intuitive.

Minimum distance classifier

Suppose that we define the prototype of each pattern class to be the mean vector of the patterns of that class:

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x}_j \qquad j = 1, 2, \ldots, W \tag{12.2-3}$$

where N_j is the number of pattern vectors from class ω_j and the summation is taken over these vectors. As before, W is the number of pattern classes. One way to determine the class membership of an unknown pattern vector $\mathbf{x}$ is to assign it to the class of its closest prototype, as noted previously. Using the Euclidean distance to determine closeness reduces the problem to computing the distance measures:

$$D_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\| \qquad j = 1, 2, \ldots, W \tag{12.2-4}$$

where $\|\mathbf{a}\| = (\mathbf{a}^T\mathbf{a})^{1/2}$ is the Euclidean norm. We then assign $\mathbf{x}$ to class ω_i if $D_i(\mathbf{x})$ is the smallest distance. That is, the smallest distance implies the best match in this formulation. It is not difficult to show (Problem 12.2) that selecting the smallest distance is equivalent to evaluating the functions

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{m}_j \qquad j = 1, 2, \ldots, W \tag{12.2-5}$$

and assigning $\mathbf{x}$ to class ω_i if $d_i(\mathbf{x})$ yields the largest numerical value. This formulation agrees with the concept of a decision function, as defined in Eq. (12.2-1).

From Eqs. (12.2-2) and (12.2-5), the decision boundary between classes ω_i and ω_j for a minimum distance classifier is

$$\begin{aligned} d_{ij}(\mathbf{x}) &= d_i(\mathbf{x}) - d_j(\mathbf{x}) \\ &= \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i - \mathbf{m}_j)^T(\mathbf{m}_i + \mathbf{m}_j) = 0 \end{aligned} \tag{12.2-6}$$

The surface given by Eq. (12.2-6) is the perpendicular bisector of the line segment joining $\mathbf{m}_i$ and $\mathbf{m}_j$ (see Problem 12.3). For $n = 2$, the perpendicular bisector is a line, for $n = 3$ it is a plane, and for $n > 3$ it is called a *hyperplane*.

EXAMPLE 12.1: Illustration of the minimum distance classifier.

■ Figure 12.6 shows two pattern classes extracted from the iris samples in Fig. 12.1. The two classes, *Iris versicolor* and *Iris setosa*, denoted ω_1 and ω_2, respectively, have sample mean vectors $\mathbf{m}_1 = (4.3, 1.3)^T$ and $\mathbf{m}_2 = (1.5, 0.3)^T$. From Eq. (12.2-5), the decision functions are

$$\begin{aligned} d_1(\mathbf{x}) &= \mathbf{x}^T\mathbf{m}_1 - \frac{1}{2}\mathbf{m}_1^T\mathbf{m}_1 \\ &= 4.3x_1 + 1.3x_2 - 10.1 \end{aligned}$$

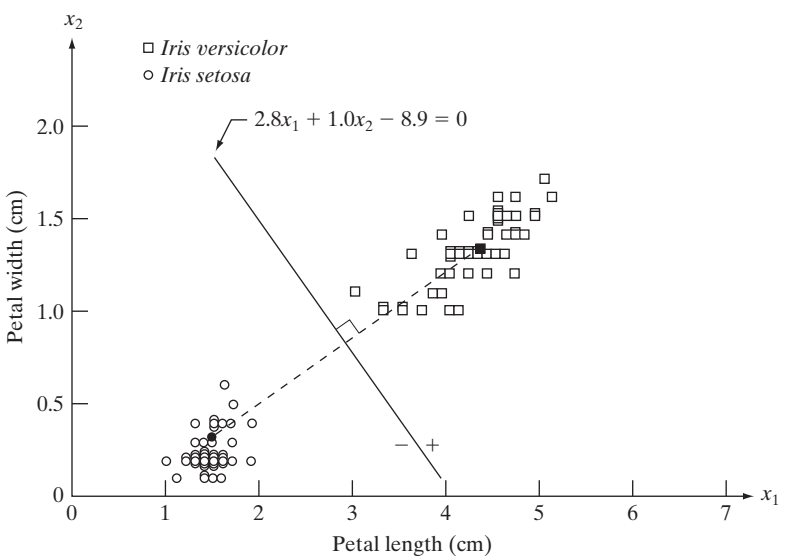


FIGURE 12.6 Decision boundary of minimum distance classifier for the classes of *Iris versicolor* and *Iris setosa*. The dark dot and square are the means.

and

$$d_2(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_2 - \frac{1}{2}\mathbf{m}_2^T\mathbf{m}_2$$
$$= 1.5x_1 + 0.3x_2 - 1.17$$

From Eq. (12.2-6), the equation of the boundary is

$$d_{12}(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x})$$
$$= 2.8x_1 + 1.0x_2 - 8.9 = 0$$

Figure 12.6 shows a plot of this boundary (note that the axes are not to the same scale). Substitution of any pattern vector from class ω_1 would yield $d_{12}(\mathbf{x}) > 0$. Conversely, any pattern from class ω_2 would yield $d_{12}(\mathbf{x}) < 0$. In other words, given an unknown pattern belonging to one of these two classes, the sign of $d_{12}(\mathbf{x})$ would be sufficient to determine the pattern's class membership. ■

In practice, the minimum distance classifier works well when the distance between means is large compared to the spread or randomness of each class with respect to its mean. In Section 12.2.2 we show that the minimum distance classifier yields optimum performance (in terms of minimizing the average loss of misclassification) when the distribution of each class about its mean is in the form of a spherical "hypercloud" in n-dimensional pattern space.

The simultaneous occurrence of large mean separations and relatively small class spread occur seldomly in practice unless the system designer controls the nature of the input. An excellent example is provided by systems designed to read stylized character fonts, such as the familiar American Banker's Association E-13B font character set. As Fig. 12.7 shows, this particular font set consists of 14 characters that were purposely designed on a 9×7 grid in order to facilitate their reading. The characters usually are printed in ink that contains finely ground magnetic material. Prior to being read, the ink is subjected to a magnetic field, which accentuates each character to simplify detection. In other words, the segmentation problem is solved by artificially highlighting the key characteristics of each character.

The characters typically are scanned in a horizontal direction with a single-slit reading head that is narrower but taller than the characters. As the head moves across a character, it produces a 1-D electrical signal (a signature) that is conditioned to be proportional to the rate of increase or decrease of the character area under the head. For example, consider the waveform associated with the number 0 in Fig. 12.7. As the reading head moves from left to right, the area seen by the head begins to increase, producing a positive derivative (a positive rate of change). As the head begins to leave the left leg of the 0, the area under the head begins to decrease, producing a negative derivative. When the head is in the middle zone of the character, the area remains nearly constant, producing a zero derivative. This pattern repeats itself as the head enters the right leg of the character. The design of the font ensures that the waveform of each character is distinct from that of all others. It also

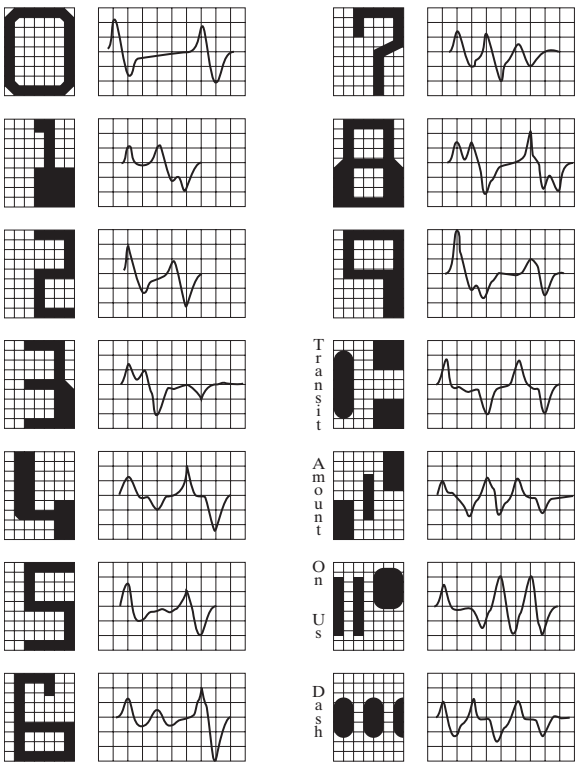


FIGURE 12.7 American Bankers Association E-13B font character set and corresponding waveforms.

ensures that the peaks and zeros of each waveform occur approximately on the vertical lines of the background grid on which these waveforms are displayed, as shown in Fig. 12.7. The E-13B font has the property that sampling the waveforms only at these points yields enough information for their proper classification. The use of magnetized ink aids in providing clean waveforms, thus minimizing scatter.

Designing a minimum distance classifier for this application is straightforward. We simply store the sample values of each waveform and let each set of samples be represented as a prototype vector $\mathbf{m}_j$, $j = 1, 2, \ldots, 14$. When an unknown character is to be classified, the approach is to scan it in the manner just described, express the grid samples of the waveform as a vector, $\mathbf{x}$, and identify its class by selecting the class of the prototype vector that yields the highest value in Eq. (12.2-5). High classification speeds can be achieved with analog circuits composed of resistor banks (see Problem 12.4).

Matching by correlation

We introduced the basic idea of spatial correlation in Section 3.4.2 and used it extensively for spatial filtering in that section. We also mentioned the correlation theorem briefly in Section 4.6.7 and Table 4.3. From Eq. (3.4-1), we know that correlation of a mask $w(x, y)$ of size $m \times n$, with an image $f(x, y)$ may be expressed in the form

To be formal, we should refer to correlation as *crosscorrelation* when the functions are different and as *autocorrelation* when they are same. However, it is customary to use the generic term *correlation* when it is clear whether the two functions in a given application are equal or different.

$$c(x, y) = \sum_s \sum_t w(s, t) f(x + s, y + t) \tag{12.2-7a}$$

where the limits of summation are taken over the region shared by w and f. This equation is evaluated for all values of the displacement variables x and y so that all elements of w visit every pixel of f, where f is assumed to be larger than w. Just as spatial convolution is related to the Fourier transform of the functions via the convolution theorem, spatial correlation is related to the transforms of the functions via the correlation theorem:

$$f(x, y) \star w(x, y) \Leftrightarrow F^*(u, v) W(u, v) \tag{12.2-7b}$$

where "$\star$" indicates spatial convolution and F^* is the complex conjugate of F. The other half of the correlation theorem stated in Table 4.3 is of no interest in the present discussion. Equation (12.2-7b) is a Fourier transform pair whose interpretation is identical to the discussion of Eq. (4.6-24), except that we use the complex conjugate of one of the functions. The inverse Fourier transform of Eq. (12.2-7b) yields a two-dimensional circular correlation analogous to Eq. (4.6-23), and the padding issues discussed in Section 4.6.6 regarding convolution are applicable also to correlation.

We do not dwell on either of the preceding equations because they are both sensitive to scale changes in f and w. Instead, we use the following *normalized correlation coefficient*

$$\gamma(x, y) = \frac{\sum_s \sum_t \left[w(s, t) - \overline{w} \right] \sum_s \sum_t \left[f(x + s, y + t) - \overline{f}(x + s, y + t) \right]}{\left\{ \sum_s \sum_t \left[w(s, t) - \overline{w} \right]^2 \sum_s \sum_t \left[f(x + s, y + t) - \overline{f}(x + s, y + t) \right]^2 \right\}^{\frac{1}{2}}} \tag{12.2-8}$$

You will find it helpful to review Section 3.4.2 regarding the mechanics of spatial correlation.

where the limits of summation are taken over the region shared by w and f, $\overline{w}$ is the average value of the mask (computed only once), and $\overline{f}(x + s, y + t)$ is the average value of f in the region coincident with w. Often, w is referred to as a *template* and correlation is referred to as *template matching*. It can be shown (Problem 12.7) that $\gamma(x, y)$ has values in the range $[-1, 1]$ and is thus normalized to changes in the amplitudes of w and f. The maximum value of $\gamma(x, y)$ occurs when the normalized w and the corresponding normalized region in f are identical. This indicates *maximum correlation* (i.e., the best possible match). The minimum occurs with the two normalized functions exhibit the least similarity in the sense of Eq. (12.2-8). The correlation coefficient cannot be computed using the Fourier transform because of the nonlinear terms in the equation (division and squares).

Figure 12.8 illustrates the mechanics of the procedure just described. The border around f is the padding necessary to provide for the situation when the center of w is on the border of f, as explained in Section 3.4.2. (In template matching, values of correlation when the center of the template is past the border of the image generally are of no interest, so the padding is limited to half the mask width.) As usual, we limit attention to templates of odd size for notational convenience.

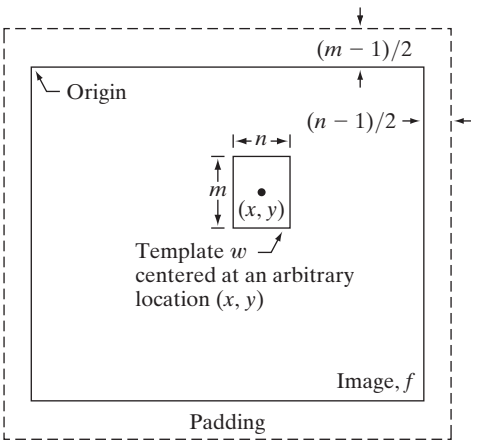


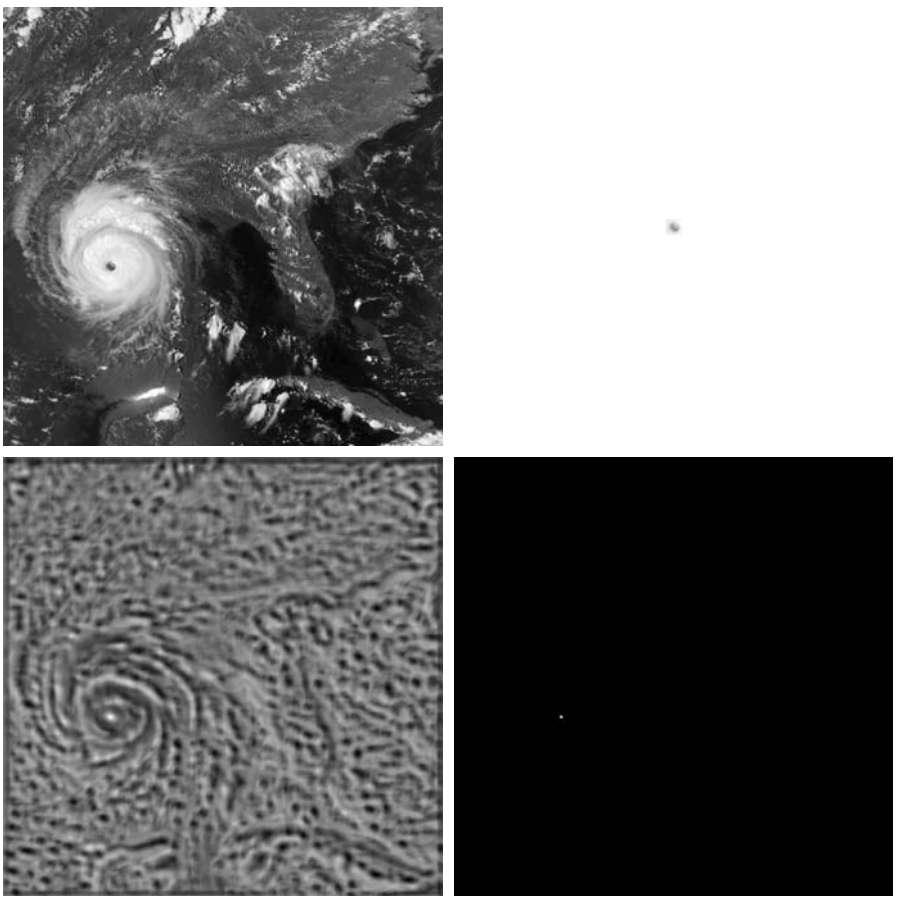
FIGURE 12.8 The mechanics of template matching.

Figure 12.8 shows a template of size $m \times n$ whose center is at an arbitrary location (x, y). The correlation at this point is obtained by applying Eq. (12.2-8). Then the center of the template is incremented to an adjacent location and the procedure is repeated. The complete correlation coefficient $\gamma(x, y)$ is obtained by moving the center of the template (i.e., by incrementing x and y) so that the *center* of w visits every pixel in f. At the end of the procedure, we look for the maximum in $\gamma(x, y)$ to find where the best match occurred. It is possible to have multiple locations in $\gamma(x, y)$ with the same maximum value, indicating several matches between w and f.

EXAMPLE 12.2: Matching by correlation.

■ Figure 12.9(a) shows a 913×913 satellite image of Hurricane Andrew, in which the eye of the storm is clearly visible. As an example of correlation we wish to find the location of the best match in (a) of the template in Fig. 12.9(b), which is a small (31×31) subimage of the eye of the storm. Figure 12.9(c) shows the result of computing the correlation coefficient in Eq. (12.2-8). The original size of this image was 943×943 pixels due to padding (see Fig. 12.8), but we cropped it to the size of the original image for display purposes. Intensity in this image is proportional to correlation value, and all negative correlations were clipped at 0 (black) to simplify the visual analysis of the image. The brightest point of the correlation image is clearly visible near the eye of the storm. Figure 12.9(d) shows as a white dot the location of the maximum correlation (in this case there was a unique match whose maximum value was 1), which we see corresponds closely with the location of the eye in Fig. 12.9(a). ■

The preceding discussion shows that it is possible to normalize correlation for changes in intensity values of the functions being processed. Normalizing for size and rotation is a more complicated problem. Normalizing for size involves spatial scaling, which, as explained in Sections 2.6.5 and 4.5.4, is image resampling. In order for resampling to make sense, the size to which an image should be rescaled must be known. In some situations, this can become a difficult issue unless spatial cues are available. For example, in a remote sensing application, if the viewing geometry of the imaging sensors



a b
c d

FIGURE 12.9
(a) Satellite image of Hurricane Andrew, taken on August 24, 1992. (b) Template of the eye of the storm. (c) Correlation coefficient shown as an image (note the brightest point). (d) Location of the best match. This point is a single pixel, but its size was enlarged to make it easier to see. (Original image courtesy of NOAA.)

is known (which typically is the case), then knowing the altitude of the sensor with respect to the area being imaged may be sufficient to be able to normalize image size, assuming a fixed viewing angle. Normalizing for rotation similarly requires that the angle to which images should be rotated be known. This again requires spatial cues. In the remote sensing example just given, the direction of flight may be sufficient to be able to rotate the sensed images into a standard orientation. In unconstrained situations, normalizing for size and orientation can become a truly challenging task, requiring the automated detection of images features (as discussed in Chapter 11) that can be used as spatial cues.

12.2.2 Optimum Statistical Classifiers

In this section we develop a probabilistic approach to recognition. As is true in most fields that deal with measuring and interpreting physical events, probability considerations become important in pattern recognition because of the randomness under which pattern classes normally are generated. As shown in the following discussion, it is possible to derive a classification approach that is

optimal in the sense that, on average, its use yields the lowest probability of committing classification errors (see Problem 12.10).

Foundation

The probability that a particular pattern $\mathbf{x}$ comes from class ω_i is denoted $p(\omega_i/\mathbf{x})$. If the pattern classifier decides that $\mathbf{x}$ came from ω_j when it actually came from ω_i, it incurs a loss, denoted L_{ij}. As pattern $\mathbf{x}$ may belong to any one of W classes under consideration, the average loss incurred in assigning $\mathbf{x}$ to class ω_j is

$$r_j(\mathbf{x}) = \sum_{k=1}^{W} L_{kj} p(\omega_k/\mathbf{x}) \tag{12.2-9}$$

This equation often is called the *conditional average risk* or *loss* in decision-theory terminology.

Consult the book Web site for a brief review of probability theory.

From basic probability theory, we know that $p(A/B) = [p(A)p(B/A)]/p(B)$. Using this expression, we write Eq. (12.2-9) in the form

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{W} L_{kj} p(\mathbf{x}/\omega_k) P(\omega_k) \tag{12.2-10}$$

where $p(\mathbf{x}/\omega_k)$ is the probability density function of the patterns from class ω_k and $P(\omega_k)$ is the probability of occurrence of class ω_k (sometimes these probabilities are referred to as *a priori*, or simply *prior*, *probabilities*). Because $1/p(\mathbf{x})$ is positive and common to all the $r_j(\mathbf{x})$, $j = 1, 2, \ldots, W$, it can be dropped from Eq. (12.2-10) without affecting the relative order of these functions from the smallest to the largest value. The expression for the average loss then reduces to

$$r_j(\mathbf{x}) = \sum_{k=1}^{W} L_{kj} p(\mathbf{x}/\omega_k) P(\omega_k) \tag{12.2-11}$$

The classifier has W possible classes to choose from for any given unknown pattern. If it computes $r_1(\mathbf{x}), r_2(\mathbf{x}), \ldots, r_W(\mathbf{x})$ for each pattern $\mathbf{x}$ and assigns the pattern to the class with the smallest loss, the total average loss with respect to all decisions will be minimum. The classifier that minimizes the total average loss is called the *Bayes classifier*. Thus the Bayes classifier assigns an unknown pattern $\mathbf{x}$ to class ω_i if $r_i(\mathbf{x}) < r_j(\mathbf{x})$ for $j = 1, 2, \ldots, W$; $j \neq i$. In other words, $\mathbf{x}$ is assigned to class ω_i if

$$\sum_{k=1}^{W} L_{ki} p(\mathbf{x}/\omega_k) P(\omega_k) < \sum_{q=1}^{W} L_{qj} p(\mathbf{x}/\omega_q) P(\omega_q) \tag{12.2-12}$$

for all j; $j \neq i$. The "loss" for a correct decision generally is assigned a value of zero, and the loss for any incorrect decision usually is assigned the same nonzero value (say, 1). Under these conditions, the loss function becomes

$$L_{ij} = 1 - \delta_{ij} \tag{12.2-13}$$

where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$. Equation (12.2-13) indicates a loss of unity for incorrect decisions and a loss of zero for correct decisions. Substituting Eq. (12.2-13) into Eq. (12.2-11) yields

$$\begin{aligned} r_j(\mathbf{x}) &= \sum_{k=1}^{W} (1 - \delta_{kj}) p(\mathbf{x}/\omega_k) P(\omega_k) \\ &= p(\mathbf{x}) - p(\mathbf{x}/\omega_j) P(\omega_j) \end{aligned} \tag{12.2-14}$$

The Bayes classifier then assigns a pattern $\mathbf{x}$ to class ω_i if, for all $j \neq i$,

$$p(\mathbf{x}) - p(\mathbf{x}/\omega_i) P(\omega_i) < p(\mathbf{x}) - p(\mathbf{x}/\omega_j) P(\omega_j) \tag{12.2-15}$$

or, equivalently, if

$$p(\mathbf{x}/\omega_i) P(\omega_i) > p(\mathbf{x}/\omega_j) P(\omega_j) \qquad j = 1, 2, \ldots, W; j \neq i \tag{12.2-16}$$

With reference to the discussion leading to Eq. (12.2-1), we see that the Bayes classifier for a 0-1 loss function is nothing more than computation of decision functions of the form

$$d_j(\mathbf{x}) = p(\mathbf{x}/\omega_j) P(\omega_j) \qquad j = 1, 2, \ldots, W \tag{12.2-17}$$

where a pattern vector $\mathbf{x}$ is assigned to the class whose decision function yields the largest numerical value.

The decision functions given in Eq. (12.2-17) are optimal in the sense that they minimize the average loss in misclassification. For this optimality to hold, however, the probability density functions of the patterns in each class, as well as the probability of occurrence of each class, must be known. The latter requirement usually is not a problem. For instance, if all classes are equally likely to occur, then $P(\omega_j) = 1/W$. Even if this condition is not true, these probabilities generally can be inferred from knowledge of the problem. Estimation of the probability density functions $p(\mathbf{x}/\omega_j)$ is another matter. If the pattern vectors, $\mathbf{x}$, are n-dimensional, then $p(\mathbf{x}/\omega_j)$ is a function of n variables, which, if its form is not known, requires methods from multivariate probability theory for its estimation. These methods are difficult to apply in practice, especially if the number of representative patterns from each class is not large or if the underlying form of the probability density functions is not well behaved. For these reasons, use of the Bayes classifier generally is based on the assumption of an analytic expression for the various density functions and then an estimation of the necessary parameters from sample patterns from each class. By far the most prevalent form assumed for $p(\mathbf{x}/\omega_j)$ is the Gaussian probability density function. The closer this assumption is to reality, the closer the Bayes classifier approaches the minimum average loss in classification.

Bayes classifier for Gaussian pattern classes

To begin, let us consider a 1-D problem ($n = 1$) involving two pattern classes ($W = 2$) governed by Gaussian densities, with means m_1 and m_2 and standard deviations σ_1 and σ_2, respectively. From Eq. (12.2-17) the Bayes decision functions have the form

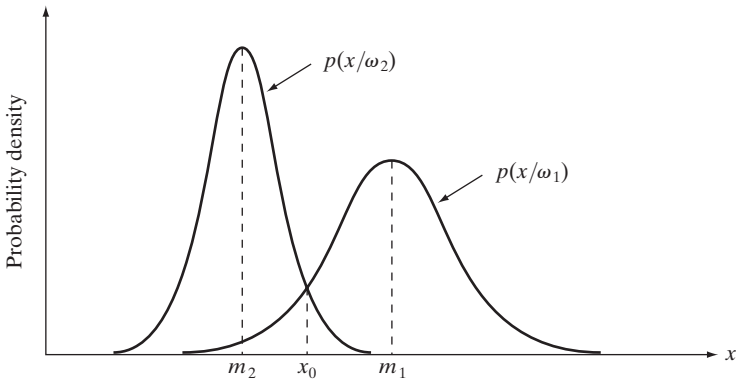


FIGURE 12.10 Probability density functions for two 1-D pattern classes. The point x_0 shown is the decision boundary if the two classes are equally likely to occur.

$$d_j(x) = p(x/\omega_j)P(\omega_j) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x-m_j)^2}{2\sigma_j^2}} P(\omega_j) \qquad j = 1, 2 \tag{12.2-18}$$

where the patterns are now scalars, denoted by x. Figure 12.10 shows a plot of the probability density functions for the two classes. The boundary between the two classes is a single point, denoted x_0 such that $d_1(x_0) = d_2(x_0)$. If the two classes are equally likely to occur, then $P(\omega_1) = P(\omega_2) = 1/2$, and the decision boundary is the value of x_0 for which $p(x_0/\omega_1) = p(x_0/\omega_2)$. This point is the intersection of the two probability density functions, as shown in Fig. 12.10. Any pattern (point) to the right of x_0 is classified as belonging to class ω_1. Similarly, any pattern to the left of x_0 is classified as belonging to class ω_2. When the classes are not equally likely to occur, x_0 moves to the left if class ω_1 is more likely to occur or, conversely, to the right if class ω_2 is more likely to occur. This result is to be expected, because the classifier is trying to minimize the loss of misclassification. For instance, in the extreme case, if class ω_2 never occurs, the classifier would never make a mistake by always assigning all patterns to class ω_1 (that is, x_0 would move to negative infinity).

See the remarks at the end of this section regarding the fact that the Bayes classifier for one variable is an optimum *thresholding* function, as mentioned in Section 10.3.3.

In the n-dimensional case, the Gaussian density of the vectors in the jth pattern class has the form

$$p(\mathbf{x}/\omega_j) = \frac{1}{(2\pi)^{n/2}|\mathbf{C}_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{m}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j)} \tag{12.2-19}$$

where each density is specified completely by its mean vector $\mathbf{m}_j$ and covariance matrix $\mathbf{C}_j$, which are defined as

$$\mathbf{m}_j = E_j\{\mathbf{x}\} \tag{12.2-20}$$

and

$$\mathbf{C}_j = E_j\{(\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T\} \tag{12.2-21}$$

where $E_j\{\cdot\}$ denotes the expected value of the argument over the patterns of class ω_j. In Eq. (12.2-19), n is the dimensionality of the pattern vectors, and

$|\mathbf{C}_j|$ is the determinant of the matrix $\mathbf{C}_j$. Approximating the expected value E_j by the average value of the quantities in question yields an estimate of the mean vector and covariance matrix:

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x} \tag{12.2-22}$$

and

$$\mathbf{C}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x}\mathbf{x}^T - \mathbf{m}_j \mathbf{m}_j^T \tag{12.2-23}$$

where N_j is the number of pattern vectors from class ω_j, and the summation is taken over these vectors. Later in this section we give an example of how to use these two expressions.

Consult the book Web site for a brief review of vectors and matrices.

The covariance matrix is symmetric and positive semidefinite. As explained in Section 11.4, the diagonal element c_{kk} is the variance of the kth element of the pattern vectors. The off-diagonal element c_{jk} is the covariance of x_j and x_k. The multivariate Gaussian density function reduces to the product of the univariate Gaussian density of each element of $\mathbf{x}$ when the off-diagonal elements of the covariance matrix are zero. This happens when the vector elements x_j and x_k are uncorrelated.

According to Eq. (12.2-17), the Bayes decision function for class ω_j is $d_j(\mathbf{x}) = p(\mathbf{x}/\omega_j)P(\omega_j)$. However, because of the exponential form of the Gaussian density, working with the natural logarithm of this decision function is more convenient. In other words, we can use the form

$$\begin{aligned} d_j(\mathbf{x}) &= \ln\big[p(\mathbf{x}/\omega_j)P(\omega_j)\big] \\ &= \ln p(\mathbf{x}/\omega_j) + \ln P(\omega_j) \end{aligned} \tag{12.2-24}$$

This expression is equivalent to Eq. (12.2-17) in terms of classification performance because the logarithm is a monotonically increasing function. In other words, the numerical *order* of the decision functions in Eqs. (12.2-17) and (12.2-24) is the same. Substituting Eq. (12.2-19) into Eq. (12.2-24) yields

$$d_j(\mathbf{x}) = \ln P(\omega_j) - \frac{n}{2}\ln 2\pi - \frac{1}{2}\ln|\mathbf{C}_j| - \frac{1}{2}\big[(\mathbf{x} - \mathbf{m}_j)^T\mathbf{C}_j^{-1}(\mathbf{x} - \mathbf{m}_j)\big] \tag{12.2-25}$$

The term $(n/2)\ln 2\pi$ is the same for all classes, so it can be eliminated from Eq. (12.2-25), which then becomes

$$d_j(\mathbf{x}) = \ln P(\omega_j) - \frac{1}{2}\ln|\mathbf{C}_j| - \frac{1}{2}\big[(\mathbf{x} - \mathbf{m}_j)^T\mathbf{C}_j^{-1}(\mathbf{x} - \mathbf{m}_j)\big] \tag{12.2-26}$$

for $j = 1, 2, \ldots, W$. Equation (12.2-26) represents the Bayes decision functions for Gaussian pattern classes under the condition of a 0-1 loss function.

The decision functions in Eq. (12.2-26) are hyperquadrics (quadratic functions in n-dimensional space), because no terms higher than the second degree in the components of $\mathbf{x}$ appear in the equation. Clearly, then, the best that a

Bayes classifier for Gaussian patterns can do is to place a general second-order decision surface between each pair of pattern classes. If the pattern populations are truly Gaussian, however, no other surface would yield a lesser average loss in classification.

If all covariance matrices are equal, then $\mathbf{C}_j = \mathbf{C}$, for $j = 1, 2, \ldots, W$. By expanding Eq. (12.2-26) and dropping all terms independent of j, we obtain

$$d_j(\mathbf{x}) = \ln P(\omega_j) + \mathbf{x}^T\mathbf{C}^{-1}\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{C}^{-1}\mathbf{m}_j \qquad (12.2\text{-}27)$$

which are linear decision functions (*hyperplanes*) for $j = 1, 2, \ldots, W$.

If, in addition, $\mathbf{C} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, and also $P(\omega_j) = 1/W$, for $j = 1, 2, \ldots, W$, then

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{m}_j \qquad j = 1, 2, \ldots, W \qquad (12.2\text{-}28)$$

These are the decision functions for a minimum distance classifier, as given in Eq. (12.2-5). Thus the minimum distance classifier is optimum in the Bayes sense if (1) the pattern classes are Gaussian, (2) all covariance matrices are equal to the identity matrix, and (3) all classes are equally likely to occur. Gaussian pattern classes satisfying these conditions are spherical clouds of identical shape in n dimensions (called *hyperspheres*). The minimum distance classifier establishes a hyperplane between every pair of classes, with the property that the hyperplane is the perpendicular bisector of the line segment joining the center of the pair of hyperspheres. In two dimensions, the classes constitute circular regions, and the boundaries become lines that bisect the line segment joining the center of every pair of such circles.

EXAMPLE 12.3: A Bayes classifier for three-dimensional patterns.

■ Figure 12.11 shows a simple arrangement of two pattern classes in three dimensions. We use these patterns to illustrate the mechanics of implementing the Bayes classifier, assuming that the patterns of each class are samples from a Gaussian distribution.

Applying Eq. (12.2-22) to the patterns of Fig. 12.11 yields

$$\mathbf{m}_1 = \frac{1}{4}\begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{m}_2 = \frac{1}{4}\begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}$$

Similarly, applying Eq. (12.2-23) to the two pattern classes in turn yields two covariance matrices, which in this case are equal:

$$\mathbf{C}_1 = \mathbf{C}_2 = \frac{1}{16}\begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}$$

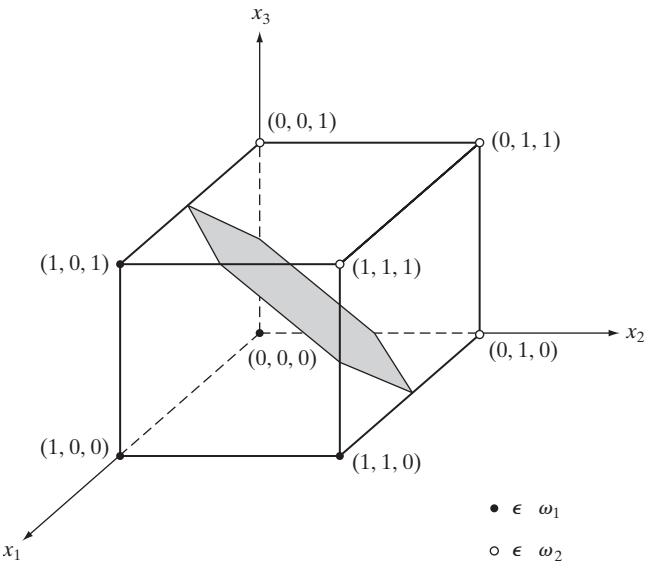


FIGURE 12.11 Two simple pattern classes and their Bayes decision boundary (shown shaded).

Because the covariance matrices are equal the Bayes decision functions are given by Eq. (12.2-27). If we assume that $P(\omega_1) = P(\omega_2) = 1/2$, then Eq. (12.2-28) applies, giving

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{C}^{-1}\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{C}^{-1}\mathbf{m}_j$$

in which

$$\mathbf{C}^{-1} = \begin{bmatrix} 8 & -4 & -4 \\ -4 & 8 & 4 \\ -4 & 4 & 8 \end{bmatrix}$$

Carrying out the vector-matrix expansion for $d_j(\mathbf{x})$ provides the decision functions:

$$d_1(\mathbf{x}) = 4x_1 - 1.5 \quad \text{and} \quad d_2(\mathbf{x}) = -4x_1 + 8x_2 + 8x_3 - 5.5$$

The decision surface separating the two classes then is

$$d_1(\mathbf{x}) - d_2(\mathbf{x}) = 8x_1 - 8x_2 - 8x_3 + 4 = 0$$

Figure 12.11 shows a section of this surface, where we note that the classes were separated effectively. ■

One of the most successful applications of the Bayes classifier approach is in the classification of remotely sensed imagery generated by multispectral

scanners aboard aircraft, satellites, or space stations. The voluminous image data generated by these platforms make automatic image classification and analysis a task of considerable interest in remote sensing. The applications of remote sensing are varied and include land use, crop inventory, crop disease detection, forestry, air and water quality monitoring, geological studies, weather prediction, and a score of other applications having environmental significance. The following example shows a typical application.

EXAMPLE 12.4: Classification of multispectral data using a Bayes classifier.

■ As discussed in Sections 1.3.4 and 11.4, a multispectral scanner responds to selected bands of the electromagnetic energy spectrum; for example, 0.45–0.52, 0.52–0.60, 0.63–0.69, and 0.76–0.90 microns. These ranges are in the visible blue, visible green, visible red, and near infrared bands, respectively. A region on the ground scanned in this manner produces four digital images of the region, one for each band. If the images are registered spatially, a condition generally met in practice, they can be visualized as being stacked one behind the other, as Fig. 12.12 shows. Thus, just as we did in Section 11.4, every point on the ground can be represented by a 4-element pattern vector of the form $\mathbf{x} = (x_1, x_2, x_3, x_4)^T$, where x_1 is a shade of blue, x_2 a shade of green, and so on. If the images are of size 512×512 pixels, each stack of four multispectral images can be represented by 266,144 four-dimensional pattern vectors. As noted previously, the Bayes classifier for Gaussian patterns requires estimates of the mean vector and covariance matrix for each class. In remote sensing applications, these estimates are obtained by collecting multispectral data whose class is known from each region of interest. The resulting vectors are then used to estimate the required mean vectors and covariance matrices, as in Example 12.3.

Figures 12.13(a) through (d) show four 512×512 multispectral images of the Washington, D.C. area taken in the bands mentioned in the previous paragraph. We are interested in classifying the pixels in the region encompassed by the images into one of three pattern classes: water, urban development, or vegetation. The masks in Fig. 12.13(e) were superimposed on the images to extract

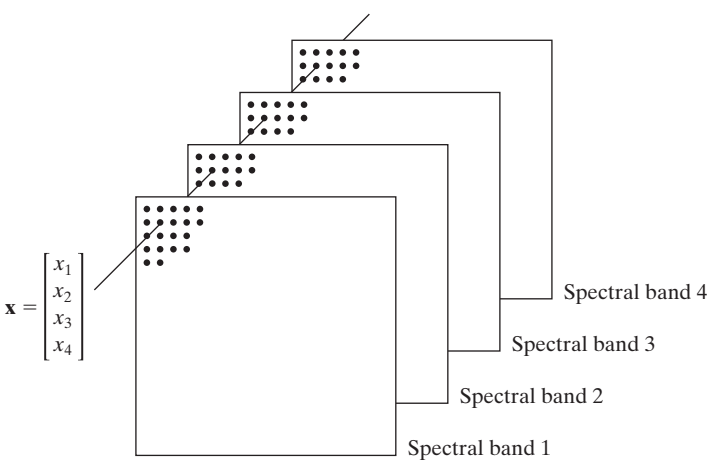
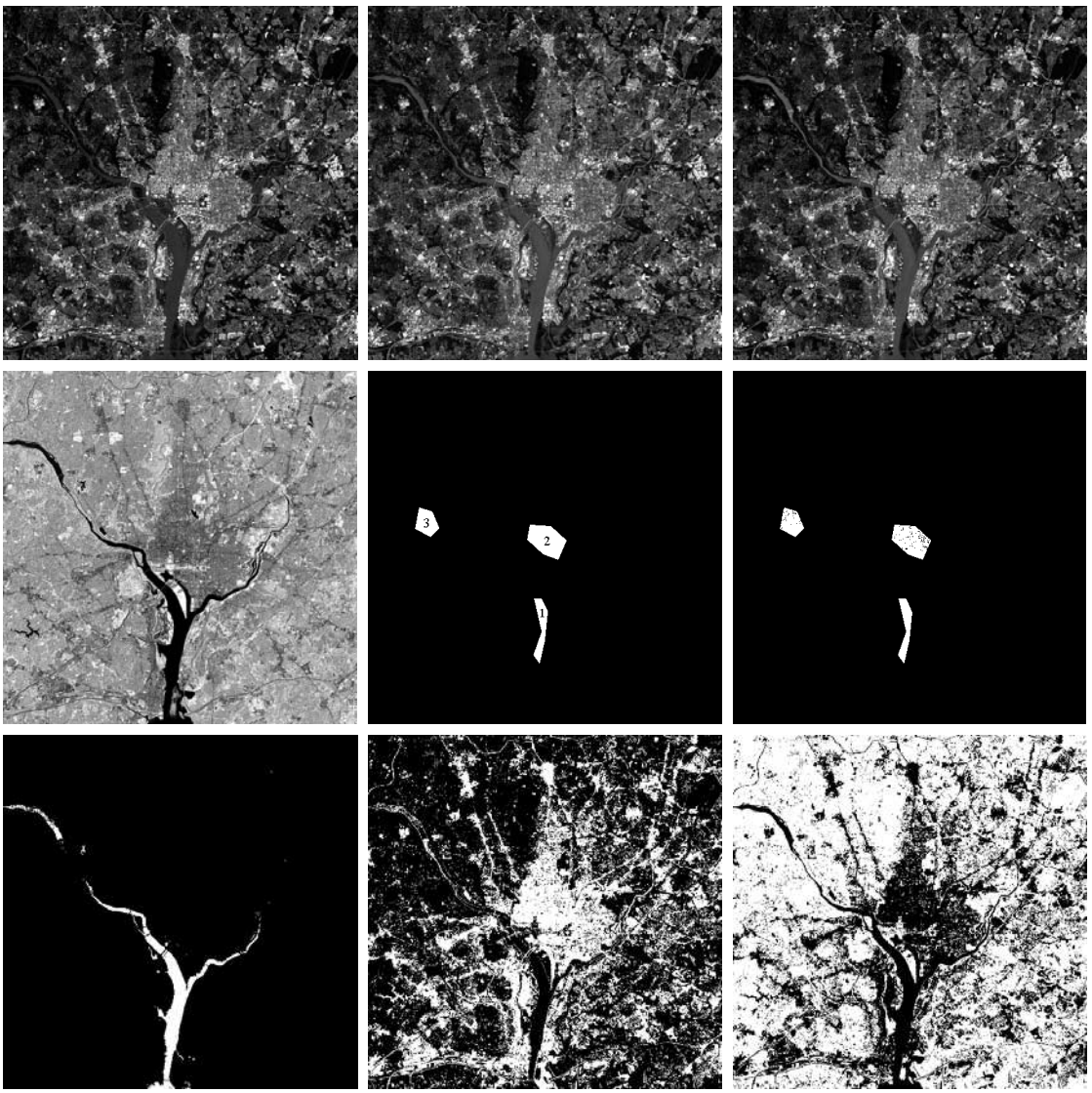


FIGURE 12.12 Formation of a pattern vector from registered pixels of four digital images generated by a multispectral scanner.



a b c
d e f
g h i

FIGURE 12.13 Bayes classification of multispectral data. (a)–(d) Images in the visible blue, visible green, visible red, and near infrared wavelengths. (e) Mask showing sample regions of water (1), urban development (2), and vegetation (3). (f) Results of classification; the black dots denote points classified incorrectly. The other (white) points were classified correctly. (g) All image pixels classified as water (in white). (h) All image pixels classified as urban development (in white). (i) All image pixels classified as vegetations (in white).

TABLE 12.1
Bayes classification of multispectral image data.

Training Patterns						Independent Patterns					
		Classified into Class						Classified into Class			
Class	No. of Samples	1	2	3	% Correct	Class	No. of Samples	1	2	3	% Correct
1	484	482	2	0	99.6	1	483	478	3	2	98.9
2	933	0	885	48	94.9	2	932	0	880	52	94.4
3	483	0	19	464	96.1	3	482	0	16	466	96.7

samples representative of these three classes. Half of the samples were used for training (i.e., for estimating the mean vectors and covariance matrices), and the other half were used for independent testing to assess preliminary classifier performance. The a priori probabilities, $P(\omega_i)$, seldom are known in unconstrained multispectral data classification, so we assume here that they are equal: $P(\omega_i) = 1/3, i = 1, 2, 3$.

Table 12.1 summarizes the recognition results obtained with the training and independent data sets. The percentage of training and independent pattern vectors recognized correctly was about the same with both data sets, indicating stability in the parameter estimates. The largest error in both cases was with patterns from the urban area. This is not unexpected, as vegetation is present there also (note that no patterns in the vegetation or urban areas were misclassified as water). Figure 12.13(f) shows as black dots the patterns that were misclassified and as white dots the patterns that were classified correctly. No black dots are readily visible in region 1, because the 7 misclassified points are very close to the boundary of the white region.

Figures 12.13(g) through (i) are much more interesting. Here, we used the mean vectors and covariance matrices obtained from the training data to classify *all* image pixels into one of the three categories. Figure 12.13(g) shows in white all pixels that were classified as water. Pixels not classified as water are shown in black. We see that the Bayes classifier did an excellent job of determining which parts of the image were water. Figure 12.13(h) shows in white all pixels classified as urban development; observe how well the system performed in recognizing urban features, such as the bridges and highways. Figure 12.13(i) shows the pixels classified as vegetation. The center area in Fig. 12.13(h) shows a high concentration of white pixels in the downtown area, with the density decreasing as a function of distance from the center of the image. Figure 12.13(i) shows the opposite effect, indicating the least vegetation toward the center of the image, when urban development is at its maximum. ■

We mentioned at the beginning of Section 10.3.3 that thresholding may be viewed as a Bayes classification problem, which optimally assigns patterns to two or more classes. In fact, as the previous problem shows, pixel-by-pixel classification is really a segmentation problem that partitions an image into two or

more possible types of regions. If only one single variable (e.g., intensity) is used, then Eq. (12.2-17) becomes an optimum function that similarly partitions an image based on the intensity of its pixels, as we did in Section 10.3. Keep in mind that optimality requires that the PDF and a priori probability of each class be known. As we have mentioned previously, estimating these densities is not a trivial task. If assumptions have to be made (e.g., as in assuming Gaussian densities), then the degree of optimality achieved in segmentation is proportional to how close the assumptions are to reality.

12.2.3 Neural Networks

The approaches discussed in the preceding two sections are based on the use of sample patterns to estimate statistical parameters of each pattern class. The minimum distance classifier is specified completely by the mean vector of each class. Similarly, the Bayes classifier for Gaussian populations is specified completely by the mean vector and covariance matrix of each class. The patterns (of *known* class membership) used to estimate these parameters usually are called *training patterns*, and a set of such patterns from each class is called a *training set.* The process by which a training set is used to obtain decision functions is called *learning* or *training.*

In the two approaches just discussed, training is a simple matter. The training patterns of each class are used to compute the parameters of the decision function corresponding to that class. After the parameters in question have been estimated, the structure of the classifier is fixed, and its eventual performance will depend on how well the actual pattern populations satisfy the underlying statistical assumptions made in the derivation of the classification method being used.

The statistical properties of the pattern classes in a problem often are unknown or cannot be estimated (recall our brief discussion in the preceding section regarding the difficulty of working with multivariate statistics). In practice, such decision-theoretic problems are best handled by methods that yield the required decision functions directly via training. Then, making assumptions regarding the underlying probability density functions or other probabilistic information about the pattern classes under consideration is unnecessary. In this section we discuss various approaches that meet this criterion.

Background

The essence of the material that follows is the use of a multitude of elemental nonlinear computing elements (called *neurons*) organized as networks reminiscent of the way in which neurons are believed to be interconnected in the brain. The resulting models are referred to by various names, including *neural networks, neurocomputers, parallel distributed processing* (PDP) *models, neuromorphic systems, layered self-adaptive networks*, and *connectionist models.* Here, we use the name *neural networks*, or *neural nets* for short. We use these networks as vehicles for adaptively developing the coefficients of decision functions via successive presentations of training sets of patterns.

Interest in neural networks dates back to the early 1940s, as exemplified by the work of McCulloch and Pitts [1943]. They proposed neuron models in the form of binary threshold devices and stochastic algorithms involving sudden 0-1 and 1-0 changes of states in neurons as the bases for modeling neural systems. Subsequent work by Hebb [1949] was based on mathematical models that attempted to capture the concept of learning by reinforcement or association.

During the mid-1950s and early 1960s, a class of so-called *learning machines* originated by Rosenblatt [1959, 1962] caused significant excitement among researchers and practitioners of pattern recognition theory. The reason for the great interest in these machines, called *perceptrons*, was the development of mathematical proofs showing that perceptrons, when trained with linearly separable training sets (i.e., training sets separable by a hyperplane), would converge to a solution in a finite number of iterative steps. The solution took the form of coefficients of hyperplanes capable of correctly separating the classes represented by patterns of the training set.

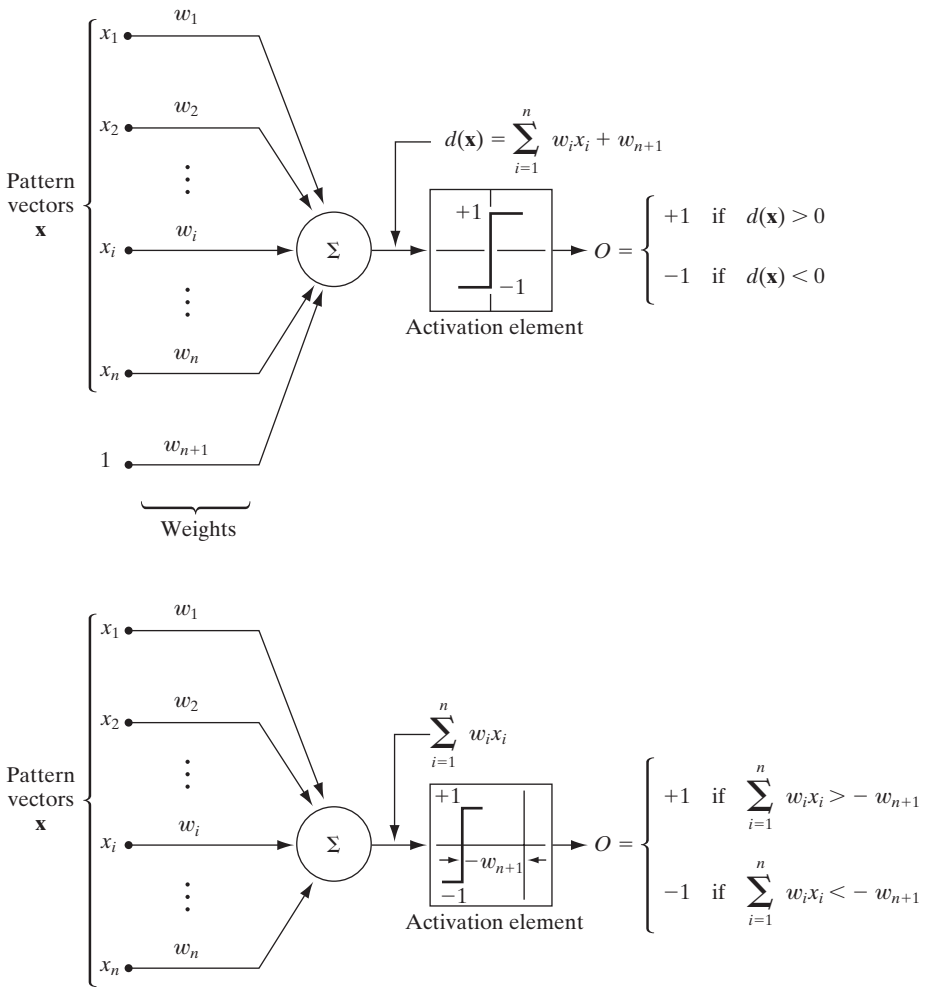
Unfortunately, the expectations following discovery of what appeared to be a well-founded theoretic model of learning soon met with disappointment. The basic perceptron and some of its generalizations at the time were simply inadequate for most pattern recognition tasks of practical significance. Subsequent attempts to extend the power of perceptron-like machines by considering multiple layers of these devices, although conceptually appealing, lacked effective training algorithms such as those that had created interest in the perceptron itself. The state of the field of learning machines in the mid-1960s was summarized by Nilsson [1965]. A few years later, Minsky and Papert [1969] presented a discouraging analysis of the limitation of perceptron-like machines. This view was held as late as the mid-1980s, as evidenced by comments by Simon [1986]. In this work, originally published in French in 1984, Simon dismisses the perceptron under the heading "Birth and Death of a Myth."

More recent results by Rumelhart, Hinton, and Williams [1986] dealing with the development of new training algorithms for multilayer perceptrons have changed matters considerably. Their basic method, often called the *generalized delta rule for learning by backpropagation*, provides an effective training method for multilayer machines. Although this training algorithm cannot be shown to converge to a solution in the sense of the analogous proof for the single-layer perceptron, the generalized delta rule has been used successfully in numerous problems of practical interest. This success has established multilayer perceptron-like machines as one of the principal models of neural networks currently in use.

Perceptron for two pattern classes

In its most basic form, the perceptron learns a linear decision function that dichotomizes two linearly separable training sets. Figure 12.14(a) shows schematically the perceptron model for two pattern classes. The response of this basic device is based on a weighted sum of its inputs; that is,

$$d(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + w_{n+1} \tag{12.2-29}$$



a
b **FIGURE 12.14** Two equivalent representations of the perceptron model for two pattern classes.

which is a linear decision function with respect to the components of the pattern vectors. The coefficients w_i, $i = 1, 2, \ldots, n, n + 1$, called *weights*, modify the inputs before they are summed and fed into the threshold element. In this sense, weights are analogous to synapses in the human neural system. The function that maps the output of the summing junction into the final output of the device sometimes is called the *activation function*.

When $d(\mathbf{x}) > 0$, the threshold element causes the output of the perceptron to be +1, indicating that the pattern $\mathbf{x}$ was recognized as belonging to class ω_1. The reverse is true when $d(\mathbf{x}) < 0$. This mode of operation agrees with the comments made earlier in connection with Eq. (12.2-2) regarding the use of a single decision function for two pattern classes. When $d(\mathbf{x}) = 0$, $\mathbf{x}$ lies on the

decision surface separating the two pattern classes, giving an indeterminate condition. The decision boundary implemented by the perceptron is obtained by setting Eq. (12.2-29) equal to zero:

$$d(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + w_{n+1} = 0 \tag{12.2-30}$$

or

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_{n+1} = 0 \tag{12.2-31}$$

which is the equation of a hyperplane in n-dimensional pattern space. Geometrically, the first n coefficients establish the orientation of the hyperplane, whereas the last coefficient, w_{n+1}, is proportional to the perpendicular distance from the origin to the hyperplane. Thus if $w_{n+1} = 0$, the hyperplane goes through the origin of the pattern space. Similarly, if $w_j = 0$, the hyperplane is parallel to the x_j-axis.

The output of the threshold element in Fig. 12.14(a) depends on the sign of $d(\mathbf{x})$. Instead of testing the entire function to determine whether it is positive or negative, we could test the summation part of Eq. (12.2-29) against the term w_{n+1}, in which case the output of the system would be

$$O = \begin{cases} +1 & \text{if } \sum_{i=1}^{n} w_i x_i > -w_{n+1} \\ -1 & \text{if } \sum_{i=1}^{n} w_i x_i < -w_{n+1} \end{cases} \tag{12.2-32}$$

This implementation is equivalent to Fig. 12.14(a) and is shown in Fig. 12.14(b), the only differences being that the threshold function is displaced by an amount $-w_{n+1}$ and that the constant unit input is no longer present. We return to the equivalence of these two formulations later in this section when we discuss implementation of multilayer neural networks.

Another formulation used frequently is to augment the pattern vectors by appending an additional $(n + 1)$st element, which is always equal to 1, regardless of class membership. That is, an augmented pattern vector $\mathbf{y}$ is created from a pattern vector $\mathbf{x}$ by letting $y_i = x_i, i = 1, 2, \ldots, n$, and appending the additional element $y_{n+1} = 1$. Equation (12.2-29) then becomes

$$\begin{aligned} d(\mathbf{y}) &= \sum_{i=1}^{n+1} w_i y_i \\ &= \mathbf{w}^T \mathbf{y} \end{aligned} \tag{12.2-33}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_n, 1)^T$ is now an *augmented pattern vector*, and $\mathbf{w} = (w_1, w_2, \ldots, w_n, w_{n+1})^T$ is called the *weight vector*. This expression is usually more convenient in terms of notation. Regardless of the formulation used, however, the key problem is to find $\mathbf{w}$ by using a given training set of pattern vectors from each of two classes.

Training algorithms

The algorithms developed in the following discussion are representative of the numerous approaches proposed over the years for training perceptrons.

Linearly separable classes: A simple, iterative algorithm for obtaining a solution weight vector for two linearly separable training sets follows. For two training sets of augmented pattern vectors belonging to pattern classes ω_1 and ω_2, respectively, let $\mathbf{w}(1)$ represent the initial weight vector, which may be chosen arbitrarily. Then, at the kth iterative step, if $\mathbf{y}(k) \in \omega_1$ and $\mathbf{w}^T(k)\mathbf{y}(k) \leq 0$, replace $\mathbf{w}(k)$ by

$$\mathbf{w}(k+1) = \mathbf{w}(k) + c\mathbf{y}(k) \tag{12.2-34}$$

where c is a positive correction increment. Conversely, if $\mathbf{y}(k) \in \omega_2$ and $\mathbf{w}^T(k)\mathbf{y}(k) \geq 0$, replace $\mathbf{w}(k)$ with

$$\mathbf{w}(k+1) = \mathbf{w}(k) - c\mathbf{y}(k) \tag{12.2-35}$$

Otherwise, leave $\mathbf{w}(k)$ unchanged:

$$\mathbf{w}(k+1) = \mathbf{w}(k) \tag{12.2-36}$$

This algorithm makes a change in $\mathbf{w}$ only if the pattern being considered at the kth step in the training sequence is misclassified. The correction increment c is assumed to be positive and, for now, to be constant. This algorithm sometimes is referred to as the *fixed increment correction rule*.

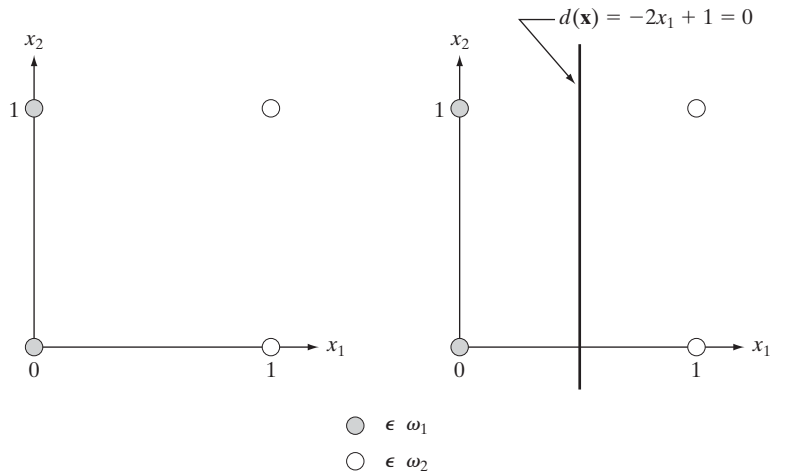
Convergence of the algorithm occurs when the entire training set for both classes is cycled through the machine without any errors. The fixed increment correction rule converges in a finite number of steps if the two training sets of patterns are linearly separable. A proof of this result, sometimes called the *perceptron training theorem*, can be found in the books by Duda, Hart, and Stork [2001]; Tou and Gonzalez [1974]; and Nilsson [1965].

EXAMPLE 12.5: Illustration of the perceptron algorithm.

■ Consider the two training sets shown in Fig. 12.15(a), each consisting of two patterns. The training algorithm will be successful because the two training sets are linearly separable. Before the algorithm is applied the patterns are augmented, yielding the training set $\{(0, 0, 1)^T, (0, 1, 1)^T\}$ for class ω_1 and $\{(1, 0, 1)^T, (1, 1, 1)^T\}$ for class ω_2. Letting $c = 1$, $\mathbf{w}(1) = \mathbf{0}$, and presenting the patterns in order results in the following sequence of steps:

$$\mathbf{w}^T(1)\mathbf{y}(1) = [0, 0, 0]\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0 \qquad \mathbf{w}(2) = \mathbf{w}(1) + \mathbf{y}(1) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{w}^T(2)\mathbf{y}(2) = [0, 0, 1]\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 1 \qquad \mathbf{w}(3) = \mathbf{w}(2) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



a b

FIGURE 12.15
(a) Patterns belonging to two classes.
(b) Decision boundary determined by training.

$$\mathbf{w}^T(3)\mathbf{y}(3) = [0, 0, 1]\begin{bmatrix}1\\0\\1\end{bmatrix} = 1 \qquad \mathbf{w}(4) = \mathbf{w}(3) - \mathbf{y}(3) = \begin{bmatrix}-1\\0\\0\end{bmatrix}$$

$$\mathbf{w}^T(4)\mathbf{y}(4) = [-1, 0, 0]\begin{bmatrix}1\\1\\1\end{bmatrix} = -1 \qquad \mathbf{w}(5) = \mathbf{w}(4) = \begin{bmatrix}-1\\0\\0\end{bmatrix}$$

where corrections in the weight vector were made in the first and third steps because of misclassifications, as indicated in Eqs. (12.2-34) and (12.2-35). Because a solution has been obtained only when the algorithm yields a complete error-free iteration through all training patterns, the training set must be presented again. The machine learning process is continued by letting $\mathbf{y}(5) = \mathbf{y}(1)$, $\mathbf{y}(6) = \mathbf{y}(2)$, $\mathbf{y}(7) = \mathbf{y}(3)$, and $\mathbf{y}(8) = \mathbf{y}(4)$, and proceeding in the same manner. Convergence is achieved at $k = 14$, yielding the solution weight vector $\mathbf{w}(14) = (-2, 0, 1)^T$. The corresponding decision function is $d(\mathbf{y}) = -2y_1 + 1$. Going back to the original pattern space by letting $x_i = y_i$ yields $d(\mathbf{x}) = -2x_1 + 1$, which, when set equal to zero, becomes the equation of the decision boundary shown in Fig. 12.15(b). ■

Nonseparable classes: In practice, linearly separable pattern classes are the (rare) exception, rather than the rule. Consequently, a significant amount of research effort during the 1960s and 1970s went into development of techniques designed to handle nonseparable pattern classes. With recent advances in the training of neural networks, many of the methods dealing with nonseparable behavior have become merely items of historical interest. One of the early methods, however, is directly relevant to this discussion: the original delta rule. Known as the *Widrow-Hoff*, or *least-mean-square* (LMS) *delta rule* for training perceptrons, the method minimizes the error between the actual and desired response at any training step.

Consider the criterion function

$$J(\mathbf{w}) = \frac{1}{2}(r - \mathbf{w}^T\mathbf{y})^2 \tag{12.2-37}$$

where r is the desired response (that is, $r = +1$ if the augmented training pattern vector $\mathbf{y}$ belongs to class ω_1, and $r = -1$ if $\mathbf{y}$ belongs to class ω_2). The task is to adjust $\mathbf{w}$ incrementally in the direction of the negative gradient of $J(\mathbf{w})$ in order to seek the minimum of this function, which occurs when $r = \mathbf{w}^T\mathbf{y}$; that is, the minimum corresponds to correct classification. If $\mathbf{w}(k)$ represents the weight vector at the kth iterative step, a general gradient descent algorithm may be written as

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha\left[\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right]_{\mathbf{w}=\mathbf{w}(k)} \tag{12.2-38}$$

where $\mathbf{w}(k+1)$ is the new value of $\mathbf{w}$, and $\alpha > 0$ gives the magnitude of the correction. From Eq. (12.2-37),

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -(r - \mathbf{w}^T\mathbf{y})\mathbf{y} \tag{12.2-39}$$

Substituting this result into Eq. (12.2-38) yields

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha\left[r(k) - \mathbf{w}^T(k)\mathbf{y}(k)\right]\mathbf{y}(k) \tag{12.2-40}$$

with the starting weight vector, $\mathbf{w}(1)$, being arbitrary.

By defining the change (delta) in weight vector as

$$\Delta\mathbf{w} = \mathbf{w}(k+1) - \mathbf{w}(k) \tag{12.2-41}$$

we can write Eq. (12.2-40) in the form of a *delta correction algorithm:*

$$\Delta\mathbf{w} = \alpha e(k)\mathbf{y}(k) \tag{12.2-42}$$

where

$$e(k) = r(k) - \mathbf{w}^T(k)\mathbf{y}(k) \tag{12.2-43}$$

is the error committed with weight vector $\mathbf{w}(k)$ when pattern $\mathbf{y}(k)$ is presented.

Equation (12.2-43) gives the error with weight vector $\mathbf{w}(k)$. If we change it to $\mathbf{w}(k+1)$, but leave the pattern the same, the error becomes

$$e(k) = r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k) \tag{12.2-44}$$

The change in error then is

$$\begin{aligned}\Delta e(k) &= \left[r(k) - \mathbf{w}^T(k+1)\mathbf{y}(k)\right] - \left[r(k) - \mathbf{w}^T(k)\mathbf{y}(k)\right] \\ &= -\left[\mathbf{w}^T(k+1) - \mathbf{w}^T(k)\right]\mathbf{y}(k) \\ &= -\Delta\mathbf{w}^T\mathbf{y}(k)\end{aligned} \tag{12.2-45}$$

But $\Delta \mathbf{w} = \alpha e(k)\mathbf{y}(k)$, so

$$\begin{aligned} \Delta e &= -\alpha e(k)\mathbf{y}^T(k)\mathbf{y}(k) \\ &= -\alpha e(k)\|\mathbf{y}(k)\|^2 \end{aligned} \tag{12.2-46}$$

Hence changing the weights reduces the error by a factor $\alpha\|\mathbf{y}(k)\|^2$. The next input pattern starts the new adaptation cycle, reducing the next error by a factor $\alpha\|\mathbf{y}(k+1)\|^2$, and so on.

The choice of α controls stability and speed of convergence (Widrow and Stearns [1985]). Stability requires that $0 < \alpha < 2$. A practical range for α is $0.1 < \alpha < 1.0$. Although the proof is not shown here, the algorithm of Eq. (12.2-40) or Eqs. (12.2-42) and (12.2-43) converges to a solution that minimizes the mean square error over the patterns of the training set. When the pattern classes are separable, the solution given by the algorithm just discussed may or may not produce a separating hyperplane. That is, a mean-square-error solution does not imply a solution in the sense of the perceptron training theorem. This uncertainty is the price of using an algorithm that converges under both the separable and nonseparable cases in this particular formulation.

The two perceptron training algorithms discussed thus far can be extended to more than two classes and to nonlinear decision functions. Based on the historical comments made earlier, exploring multiclass training algorithms here has little merit. Instead, we address multiclass training in the context of neural networks.

Multilayer feedforward neural networks

In this section we focus on decision functions of multiclass pattern recognition problems, independent of whether or not the classes are separable, and involving architectures that consist of layers of perceptron computing elements.

Basic architecture: Figure 12.16 shows the architecture of the neural network model under consideration. It consists of layers of structurally identical computing nodes (neurons) arranged so that the output of every neuron in one layer feeds into the input of every neuron in the next layer. The number of neurons in the first layer, called layer A, is N_A. Often, $N_A = n$, the dimensionality of the input pattern vectors. The number of neurons in the output layer, called layer Q, is denoted N_Q. The number N_Q equals W, the number of pattern classes that the neural network has been trained to recognize. The network recognizes a pattern vector $\mathbf{x}$ as belonging to class ω_i if the ith output of the network is "high" while all other outputs are "low," as explained in the following discussion.

As the blowup in Fig. 12.16 shows, each neuron has the same form as the perceptron model discussed earlier (see Fig. 12.14), with the exception that the hard-limiting activation function has been replaced by a soft-limiting "sigmoid" function. Differentiability along all paths of the neural network is required in the development of the training rule. The following sigmoid activation function has the necessary differentiability:

$$h_j(I_j) = \frac{1}{1 + e^{-(I_j+\theta_j)/\theta_o}} \tag{12.2-47}$$

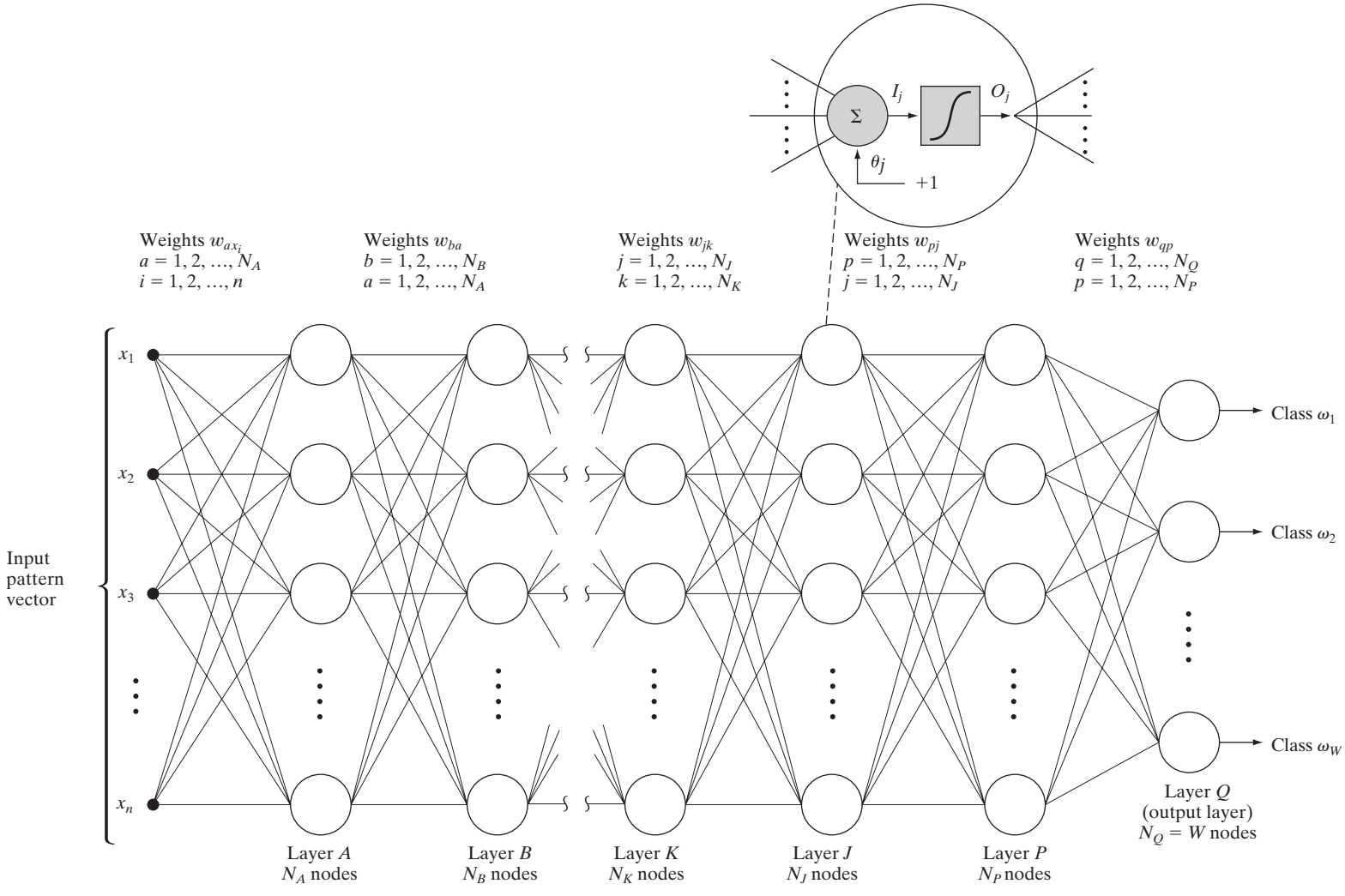


FIGURE 12.16 Multilayer feedforward neural network model. The blowup shows the basic structure of each neuron element throughout the network. The offset, θ_j, is treated as just another weight.

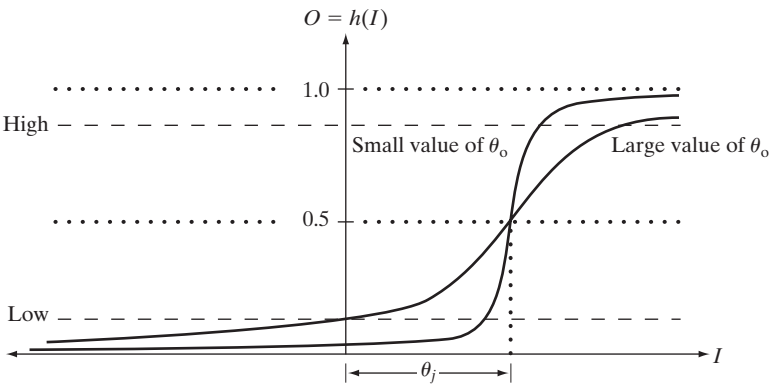


FIGURE 12.17 The sigmoidal activation function of Eq. (12.2-47).

where I_j, $j = 1, 2, \ldots, N_J$, is the input to the activation element of each node in layer J of the network, θ_j is an offset, and θ_o controls the shape of the sigmoid function.

Equation (12.2-47) is plotted in Fig. 12.17, along with the limits for the "high" and "low" responses out of each node. Thus when this particular function is used, the system outputs a high reading for any value of I_j greater than θ_j. Similarly, the system outputs a low reading for any value of I_j less than θ_j. As Fig. 12.17 shows, the sigmoid activation function always is positive, and it can reach its limiting values of 0 and 1 only if the input to the activation element is infinitely negative or positive, respectively. For this reason, values near 0 and 1 (say, 0.05 and 0.95) define low and high values at the output of the neurons in Fig. 12.16. In principle, different types of activation functions could be used for different layers or even for different nodes in the same layer of a neural network. In practice, the usual approach is to use the same form of activation function throughout the network.

With reference to Fig. 12.14(a), the offset θ_j shown in Fig. 12.17 is analogous to the weight coefficient w_{n+1} in the earlier discussion of the perceptron. Implementation of this displaced threshold function can be done in the form of Fig. 12.14(a) by absorbing the offset θ_j as an additional coefficient that modifies a constant unity input to all nodes in the network. In order to follow the notation predominantly found in the literature, we do not show a separate constant input of +1 into all nodes of Fig. 12.16. Instead, this input and its modifying weight θ_j are integral parts of the network nodes. As noted in the blowup in Fig. 12.16, there is one such coefficient for each of the N_J nodes in layer J.

In Fig. 12.16, the input to a node in any layer is the weighted sum of the outputs from the previous layer. Letting layer K denote the layer preceding layer J (no alphabetical order is implied in Fig. 12.16) gives the input to the activation element of each node in layer J, denoted I_j:

$$I_j = \sum_{k=1}^{N_K} w_{jk} O_k \tag{12.2-48}$$

for $j = 1, 2, \ldots, N_J$, where N_J is the number of nodes in layer J, N_K is the number of nodes in layer K, and w_{jk} are the weights modifying the outputs O_k of the nodes in layer K before they are fed into the nodes in layer J. The outputs of layer K are

$$O_k = h_k(I_k) \tag{12.2-49}$$

for $k = 1, 2, \ldots, N_K$.

A clear understanding of the subscript notation used in Eq. (12.2-48) is important, because we use it throughout the remainder of this section. First, note that I_j, $j = 1, 2, \ldots, N_J$, represents the input to the *activation element* of the jth node in layer J. Thus I_1 represents the input to the activation element of the first (topmost) node in layer J, I_2 represents the input to the activation element of the second node in layer J, and so on. There are N_K inputs to every node in layer J, but *each* individual input can be weighted differently. Thus the N_K inputs to the first node in layer J are weighted by coefficients w_{1k}, $k = 1, 2, \ldots, N_K$; the inputs to the second node are weighted by coefficients w_{2k}, $k = 1, 2, \ldots, N_K$; and so on. Hence a total of $N_J \times N_K$ coefficients are necessary to specify the weighting of the outputs of layer K as they are fed into layer J. An additional N_J offset coefficients, θ_j, are needed to specify completely the nodes in layer J.

Substitution of Eq. (12.2-48) into (12.2-47) yields

$$h_j(I_j) = \frac{1}{1 + e^{-\left(\sum_{k=1}^{N_k} w_{jk}o_k + \theta_j\right)/\theta_o}} \tag{12.2-50}$$

which is the form of activation function used in the remainder of this section.

During training, adapting the neurons in the output layer is a simple matter because the desired output of each node is known. The main problem in training a multilayer network lies in adjusting the weights in the so-called *hidden layers*. That is, in those other than the output layer.

Training by back propagation: We begin by concentrating on the output layer. The total squared error between the desired responses, r_q, and the corresponding actual responses, O_q, of nodes in (output) layer Q, is

$$E_Q = \frac{1}{2}\sum_{q=1}^{N_Q}(r_q - O_q)^2 \tag{12.2-51}$$

where N_Q is the number of nodes in output layer Q and the $\frac{1}{2}$ is used for convenience in notation for taking the derivative later.

The objective is to develop a training rule, similar to the delta rule, that allows adjustment of the weights in each of the layers in a way that seeks a minimum to an error function of the form shown in Eq. (12.2-51). As before, adjusting the

weights in proportion to the partial derivative of the error with respect to the weights achieves this result. In other words,

$$\Delta w_{qp} = -\alpha \frac{\partial E_Q}{\partial w_{qp}} \tag{12.2-52}$$

where layer P precedes layer Q, Δw_{qp} is as defined in Eq. (12.2-42), and α is a positive correction increment.

The error E_Q is a function of the outputs, O_q, which in turn are functions of the inputs I_q. Using the chain rule, we evaluate the partial derivative of E_Q as follows:

$$\frac{\partial E_Q}{\partial w_{qp}} = \frac{\partial E_Q}{\partial I_q} \frac{\partial I_q}{\partial w_{qp}} \tag{12.2-53}$$

From Eq. (12.2-48),

$$\frac{\partial I_q}{\partial w_{qp}} = \frac{\partial}{\partial w_{qp}} \sum_{p=1}^{N_P} w_{qp} O_p = O_p \tag{12.2-54}$$

Substituting Eqs. (12.2-53) and (12.2-54) into Eq. (12.2-52) yields

$$\begin{aligned} \Delta w_{qp} &= -\alpha \frac{\partial E_Q}{\partial I_q} O_p \\ &= \alpha \delta_q O_p \end{aligned} \tag{12.2-55}$$

where

$$\delta_q = -\frac{\partial E_Q}{\partial I_q} \tag{12.2-56}$$

In order to compute $\partial E_Q / \partial I_q$, we use the chain rule to express the partial derivative in terms of the rate of change of E_Q with respect to O_q and the rate of change of O_q with respect to I_q. That is,

$$\delta_q = -\frac{\partial E_Q}{\partial I_q} = -\frac{\partial E_Q}{\partial O_q} \frac{\partial O_q}{\partial I_q} \tag{12.2-57}$$

From Eq. (12.2-51),

$$\frac{\partial E_Q}{\partial O_q} = -(r_q - O_q) \tag{12.2-58}$$

and, from Eq. (12.2-49),

$$\frac{\partial O_Q}{\partial I_q} = \frac{\partial}{\partial I_q} h_q(I_q) = h'_q(I_q) \tag{12.2-59}$$

Substituting Eqs. (12.2-58) and (12.2-59) into Eq. (12.2-57) gives

$$\delta_q = (r_q - O_q)\, h_q'(I_q) \tag{12.2-60}$$

which is proportional to the error quantity $(r_q - O_q)$. Substitution of Eqs. (12.2-56) through (12.2-58) into Eq. (12.2-55) finally yields

$$\begin{aligned} \Delta w_{qp} &= \alpha(r_q - O_q)\, h_q'(I_q) O_p \\ &= \alpha \delta_q O_p \end{aligned} \tag{12.2-61}$$

After the function $h_q(I_q)$ has been specified, all the terms in Eq. (12.2-61) are known or can be observed in the network. In other words, upon presentation of any training pattern to the input of the network, we know what the desired response, r_q, of each output node should be. The value O_q of each output node can be observed as can I_q, the input to the activation elements of layer Q, and O_p, the output of the nodes in layer P. Thus we know how to adjust the weights that modify the links between the last and next-to-last layers in the network.

Continuing to work our way back from the output layer, let us now analyze what happens at layer P. Proceeding in the same manner as above yields

$$\begin{aligned} \Delta w_{pj} &= \alpha(r_p - O_p)\, h_p'(I_p) O_j \\ &= \alpha \delta_p O_j \end{aligned} \tag{12.2-62}$$

where the error term is

$$\delta_p = (r_p - O_p)\, h_p'(I_p) \tag{12.2-63}$$

With the exception of r_p, all the terms in Eqs. (12.2-62) and (12.2-63) either are known or can be observed in the network. The term r_p makes no sense in an internal layer because we do not know what the response of an internal node in terms of pattern membership should be. We may specify what we want the response r to be only at the outputs of the network where final pattern classification takes place. If we knew that information at internal nodes, there would be no need for further layers. Thus we have to find a way to restate δ_p in terms of quantities that are known or can be observed in the network.

Going back to Eq. (12.2-57), we write the error term for layer P as

$$\delta_p = -\frac{\partial E_p}{\partial I_p} = -\frac{\partial E_p}{\partial O_p}\frac{\partial O_p}{\partial I_p} \tag{12.2-64}$$

The term $\partial O_p / \partial I_p$ presents no difficulties. As before, it is

$$\frac{\partial O_p}{\partial I_p} = \frac{\partial h_p(I_p)}{\partial I_p} = h_p'(I_p) \tag{12.2-65}$$

which is known once h_p is specified because I_p can be observed. The term that produced r_p was the derivative $\partial E_p / \partial O_p$, so this term must be expressed in a way that does not contain r_p. Using the chain rule, we write the derivative as

$$-\frac{\partial E_P}{\partial O_p} = -\sum_{q=1}^{N_Q} \frac{\partial E_P}{\partial I_q}\frac{\partial I_q}{\partial O_p} = \sum_{q=1}^{N_Q}\left(-\frac{\partial E_P}{\partial I_q}\right)\frac{\partial}{\partial O_p}\sum_{p=1}^{N_P} w_{qp} O_p$$

$$= \sum_{q=1}^{N_Q} \left(-\frac{\partial E_P}{\partial I_q} \right) w_{qp}$$
$$= \sum_{q=1}^{N_Q} \delta_q w_{qp} \tag{12.2-66}$$

where the last step follows from Eq. (12.2-56). Substituting Eqs. (12.2-65) and (12.2-66) into Eq. (12.2-64) yields the desired expression for δ_p:

$$\delta_p = h'_p(I_p) \sum_{q=1}^{N_Q} \delta_q w_{qp} \tag{12.2-67}$$

The parameter δ_p can be computed now because all its terms are known. Thus Eqs. (12.2-62) and (12.2-67) establish completely the training rule for layer P. The importance of Eq. (12.2-67) is that it computes δ_p from the quantities δ_q and w_{qp}, which are terms that were computed in the layer immediately following layer P. After the error term and weights have been computed for layer P, these quantities may be used similarly to compute the error and weights for the layer immediately preceding layer P. In other words, we have found a way to propagate the error back into the network, starting with the error at the output layer.

We may summarize and generalize the training procedure as follows. For any layers K and J, where layer K immediately precedes layer J, compute the weights w_{jk}, which modify the connections between these two layers, by using

$$\Delta w_{jk} = \alpha \delta_j O_k \tag{12.2-68}$$

If layer J is the output layer, δ_j is

$$\delta_j = (r_j - O_j) h'_j(I_j) \tag{12.2-69}$$

If layer J is an internal layer and layer P is the next layer (to the right), then δ_j is given by

$$\delta_j = h'_j(I_j) \sum_{p=1}^{N_P} \delta_p w_{jp} \tag{12.2-70}$$

for $j = 1, 2, \ldots, N_j$. Using the activation function in Eq. (12.2-50) with $\theta_o = 1$ yields

$$h'_j(I_j) = O_j(1 - O_j) \tag{12.2-71}$$

in which case Eqs. (12.2-69) and (12.2-70) assume the following, particularly attractive forms:

$$\delta_j = (r_j - O_j) O_j(1 - O_j) \tag{12.2-72}$$

for the output layer, and

$$\delta_j = O_j(1 - O_j) \sum_{p=1}^{N_P} \delta_p w_{jp} \tag{12.2-73}$$

for internal layers. In both Eqs. (12.2-72) and (12.2-73), $j = 1, 2, \ldots, N_J$.

Equations (12.2-68) through (12.2-70) constitute the generalized delta rule for training the multilayer feedforward neural network of Fig. 12.16. The process starts with an arbitrary (but not all equal) set of weights throughout the network. Then application of the generalized delta rule at any iterative step involves two basic phases. In the first phase, a training vector is presented to the network and is allowed to propagate through the layers to compute the output O_j for each node. The outputs O_q of the nodes in the output layer are then compared against their desired responses, r_p, to generate the error terms δ_q. The second phase involves a backward pass through the network during which the appropriate error signal is passed to each node and the corresponding weight changes are made. This procedure also applies to the bias weights θ_j. As discussed earlier in some detail, these are treated simply as additional weights that modify a unit input into the summing junction of every node in the network.

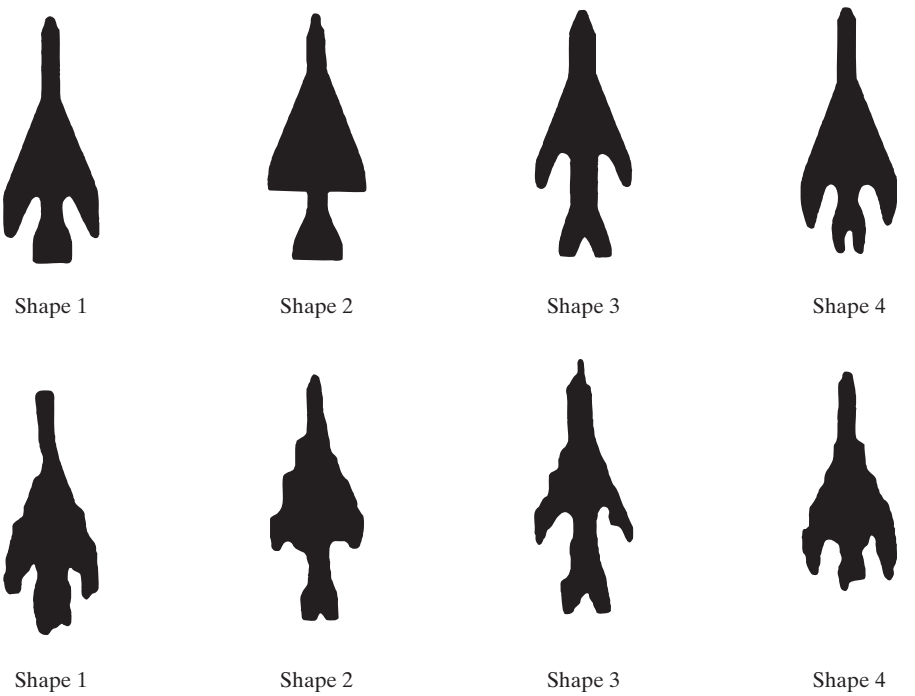
Common practice is to track the network error, as well as errors associated with individual patterns. In a successful training session, the network error decreases with the number of iterations and the procedure converges to a stable set of weights that exhibit only small fluctuations with additional training. The approach followed to establish whether a pattern has been classified correctly during training is to determine whether the response of the node in the output layer associated with the pattern class from which the pattern was obtained is high, while all the other nodes have outputs that are low, as defined earlier.

After the system has been trained, it classifies patterns using the parameters established during the training phase. In normal operation, all feedback paths are disconnected. Then any input pattern is allowed to propagate through the various layers, and the pattern is classified as belonging to the class of the output node that was high, while all the others were low. If more than one output is labeled high, or if none of the outputs is so labeled, the choice is one of declaring a misclassification or simply assigning the pattern to the class of the output node with the highest numerical value.

EXAMPLE 12.6: Shape classification using a neural network.

■ We illustrate now how a neural network of the form shown in Fig. 12.16 was trained to recognize the four shapes shown in Fig. 12.18(a), as well as noisy versions of these shapes, samples of which are shown in Fig. 12.18(b).

Pattern vectors were generated by computing the normalized signatures of the shapes (see Section 11.1.3) and then obtaining 48 uniformly spaced samples of each signature. The resulting 48-dimensional vectors were the inputs to the three-layer feedforward neural network shown in Fig. 12.19. The number of neuron nodes in the first layer was chosen to be 48, corresponding to the dimensionality of the input pattern vectors. The four neurons in the third (output) layer correspond to the number of pattern classes, and the number of neurons in the middle layer was heuristically specified as 26 (the average of the number of neurons in the input and output layers). There are no known rules for specifying the number of nodes in the internal layers of a neural network, so this number generally is based either on prior experience or simply chosen arbitrarily and then refined by testing. In the output layer, the four nodes from



a
b

FIGURE 12.18 (a) Reference shapes and (b) typical noisy shapes used in training the neural network of Fig. 12.19. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

top to bottom in this case represent the classes ω_j, $j = 1, 2, 3, 4$, respectively. After the network structure has been set, activation functions have to be selected for each unit and layer. All activation functions were selected to satisfy Eq. (12.2-50) with $\theta_o = 1$ so that, according to our earlier discussion, Eqs. (12.2-72) and (12.2-73) apply.

The training process was divided in two parts. In the first part, the weights were initialized to small random values with zero mean, and the network was then trained with pattern vectors corresponding to noise-free samples like the shapes shown in Fig. 12.18(a). The output nodes were monitored during training. The network was said to have learned the shapes from all four classes when, for any training pattern from class ω_i, the elements of the output layer yielded $O_i \geq 0.95$ and $O_q \leq 0.05$, for $q = 1, 2, \ldots, N_Q$; $q \neq i$. In other words, for any pattern of class ω_i, the output unit corresponding to that class had to be high (≥ 0.95) while, simultaneously, the output of all other nodes had to be low (≤ 0.05).

The second part of training was carried out with noisy samples, generated as follows. Each contour pixel in a noise-free shape was assigned a probability V of retaining its original coordinate in the image plane and a probability $R = 1 - V$ of being randomly assigned to the coordinates of one of its eight neighboring pixels. The degree of noise was increased by decreasing V (that is, increasing R). Two sets of noisy data were generated. The first consisted of 100 noisy patterns of each class generated by varying R between 0.1 and 0.6, giving a total of 400 patterns. This set, called the *test set*, was used to establish system performance after training.

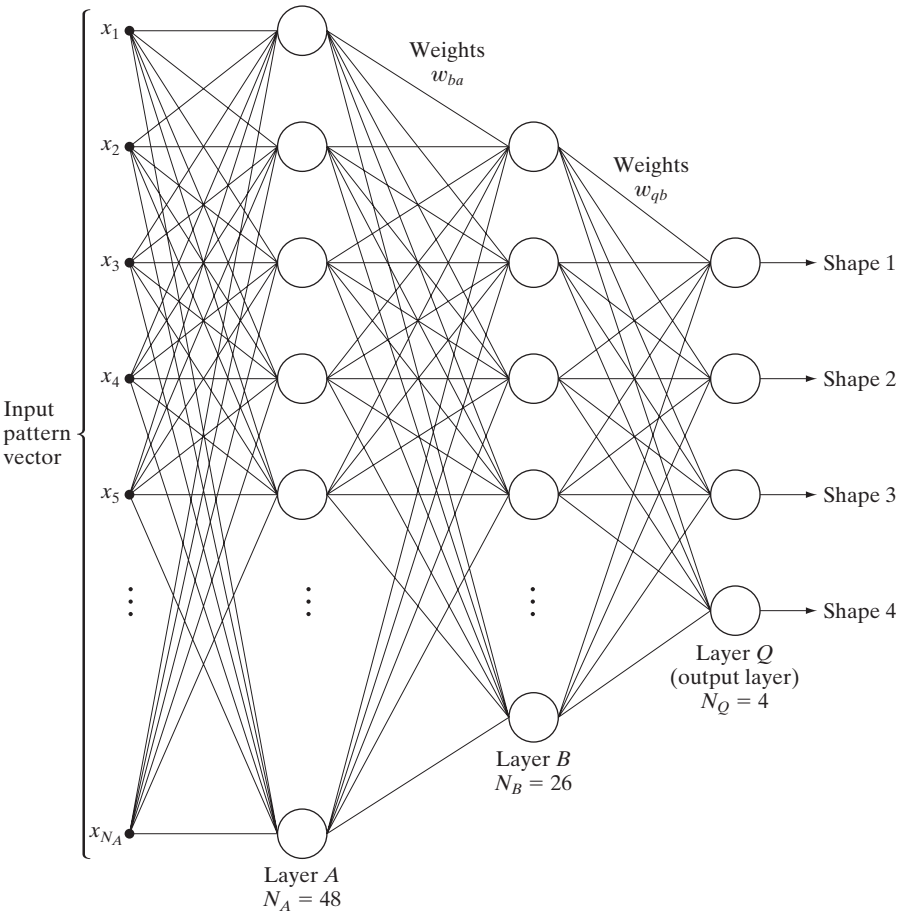


FIGURE 12.19 Three-layer neural network used to recognize the shapes in Fig. 12.18. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

Several noisy sets were generated for training the system with noisy data. The first set consisted of 10 samples for each class, generated by using $R_t = 0$, where R_t denotes a value of R used to generate training data. Starting with the weight vectors obtained in the first (noise-free) part of training, the system was allowed to go through a learning sequence with the new data set. Because $R_t = 0$ implies no noise, this retraining was an extension of the earlier, noise-free training. Using the resulting weights learned in this manner, the network was subjected to the test data set yielding the results shown by the curve labeled $R_t = 0$ in Fig. 12.20. The number of misclassified patterns divided by the total number of patterns tested gives the probability of misclassification, which is a measure commonly used to establish neural network performance.

Next, starting with the weight vectors learned by using the data generated with $R_t = 0$, the system was retrained with a noisy data set generated with $R_t = 0.1$. The recognition performance was then established by running the test samples through the system again with the new weight vectors. Note the significant improvement in performance. Figure 12.20 shows the results obtained by continuing

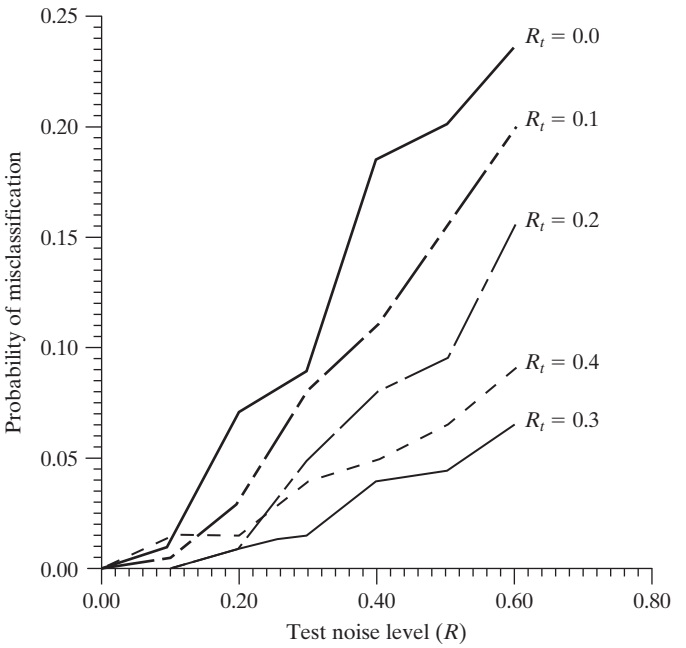


FIGURE 12.20 Performance of the neural network as a function of noise level. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

this retraining and retesting procedure for $R_t = 0.2$, 0.3, and 0.4. As expected if the system is learning properly, the probability of misclassifying patterns from the test set decreased as the value of R_t increased because the system was being trained with noisier data for higher values of R_t. The one exception in Fig. 12.20 is the result for $R_t = 0.4$. The reason is the small number of samples used to train the system. That is, the network was not able to adapt itself sufficiently to the larger variations in shape at higher noise levels with the number of samples used. This hypothesis is verified by the results in Fig. 12.21, which show a lower probability of misclassification as the number of training samples was increased. Figure 12.21 also shows as a reference the curve for $R_t = 0.3$ from Fig. 12.20.

The preceding results show that a three-layer neural network was capable of learning to recognize shapes corrupted by noise after a modest level of training. Even when trained with noise-free data ($R_t = 0$ in Fig. 12.20), the system was able to achieve a correct recognition level of close to 77% when tested with data highly corrupted by noise ($R = 0.6$ in Fig. 12.20). The recognition rate on the same data increased to about 99% when the system was trained with noisier data ($R_r = 0.3$ and 0.4). It is important to note that the system was trained by increasing its classification power via systematic, small incremental additions of noise. When the nature of the noise is known, this method is ideal for improving the convergence and stability properties of a neural network during learning. ■

Complexity of decision surfaces: We have already established that a single-layer perceptron implements a hyperplane decision surface. A natural question at this point is: What is the nature of the decision surfaces implemented by

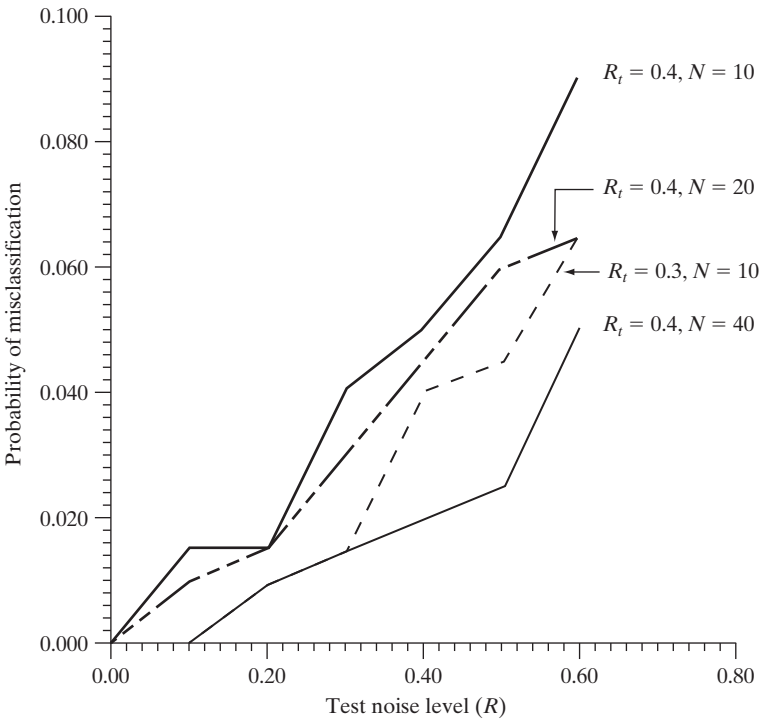


FIGURE 12.21 Improvement in performance for $R_t = 0.4$ by increasing the number of training patterns (the curve for $R_t = 0.3$ is shown for reference). (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

a multilayer network, such as the model in Fig. 12.16? It is demonstrated in the following discussion that a three-layer network is capable of implementing arbitrarily complex decision surfaces composed of intersecting hyperplanes.

As a starting point, consider the two-input, two-layer network shown in Fig. 12.22(a). With two inputs, the patterns are two dimensional, and therefore, each node in the first layer of the network implements a line in 2-D space. We denote by 1 and 0, respectively, the high and low outputs of these two nodes. We assume that a 1 output indicates that the corresponding input vector to a node in the first layer lies on the positive side of the line. Then the possible combinations of outputs feeding the single node in the second layer

a b c

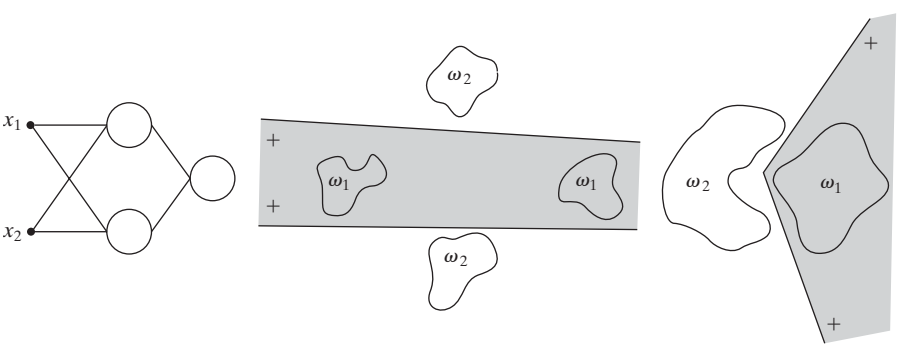


FIGURE 12.22 (a) A two-input, two-layer, feedforward neural network. (b) and (c) Examples of decision boundaries that can be implemented with this network.

are (1, 1), (1, 0), (0, 1), and (0, 0). If we define two regions, one for class ω_1 lying on the positive side of both lines and the other for class ω_2 lying anywhere else, the output node can classify any input pattern as belonging to one of these two regions simply by performing a logical AND operation. In other words, the output node responds with a 1, indicating class ω_1, only when both outputs from the first layer are 1. The AND operation can be performed by a neural node of the form discussed earlier if θ_j is set to a value in the half-open interval (1, 2]. Thus if we assume 0 and 1 responses out of the first layer, the response of the output node will be high, indicating class ω_1, only when the sum performed by the neural node on the two outputs from the first layer is greater than 1. Figures 12.22(b) and (c) show how the network of Fig. 12.22(a) can successfully dichotomize two pattern classes that could not be separated by a single linear surface.

If the number of nodes in the first layer were increased to three, the network of Fig. 12.22(a) would implement a decision boundary consisting of the intersection of three lines. The requirement that class ω_1 lie on the positive side of all three lines would yield a convex region bounded by the three lines. In fact, an arbitrary open or closed convex region can be constructed simply by increasing the number of nodes in the first layer of a two-layer neural network.

The next logical step is to increase the number of layers to three. In this case the nodes of the first layer implement lines, as before. The nodes of the second layer then perform AND operations in order to form regions from the various lines. The nodes in the third layer assign class membership to the various regions. For instance, suppose that class ω_1 consists of two distinct regions, each of which is bounded by a different set of lines. Then two of the nodes in the second layer are for regions corresponding to the same pattern class. One of the output nodes needs to be able to signal the presence of that class when either of the two nodes in the second layer goes high. Assuming that high and low conditions in the second layer are denoted 1 and 0, respectively, this capability is obtained by making the output nodes of the network perform the logical OR operation. In terms of neural nodes of the form discussed earlier, we do so by setting θ_j to a value in the half-open interval [0, 1). Then, whenever at least one of the nodes in the second layer associated with that output node goes high (outputs a 1), the corresponding node in the output layer will go high, indicating that the pattern being processed belongs to the class associated with that node.

Figure 12.23 summarizes the preceding comments. Note in the third row that the complexity of decision regions implemented by a three-layer network is, in principle, arbitrary. In practice, a serious difficulty usually arises in structuring the second layer to respond correctly to the various combinations associated with particular classes. The reason is that lines do not just stop at their intersection with other lines, and, as a result, patterns of the same class may occur on both sides of lines in the pattern space. In practical terms, the second layer may have difficulty figuring out which lines should be included in the AND operation for a given pattern class—or it may even be impossible. The reference to the exclusive-OR problem in the third column of Fig. 12.23 deals with the fact that, if the input patterns were binary, only four different patterns could be constructed in two

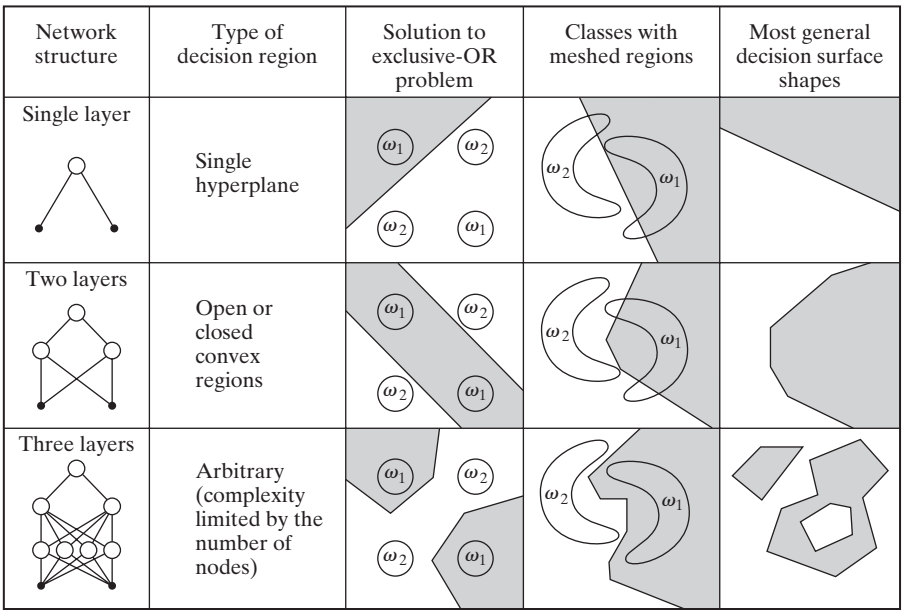


FIGURE 12.23 Types of decision regions that can be formed by single- and multilayer feed-forward networks with one and two layers of hidden units and two inputs. (Lippman.)

dimensions. If the patterns are arranged so that class ω_1 consists of patterns $\{(0, 1), (1, 0)\}$ and class ω_2 consists of the patterns $\{(0, 0), (1, 1)\}$, class membership of the patterns in these two classes is given by the exclusive-OR (XOR) logical function, which is 1 only when one or the other of the two variables is 1, and it is 0 otherwise. Thus an XOR value of 1 indicates patterns of class ω_1, and an XOR value of 0 indicates patterns of class ω_2.

The preceding discussion is generalized to n dimensions in a straightforward way: Instead of lines, we deal with hyperplanes. A single-layer network implements a single hyperplane. A two-layer network implements arbitrarily convex regions consisting of intersections of hyperplanes. A three-layer network implements decision surfaces of arbitrary complexity. The number of nodes used in each layer determines the complexity of the last two cases. The number of classes in the first case is limited to two. In the other two cases, the number of classes is arbitrary, because the number of output nodes can be selected to fit the problem at hand.

Considering the preceding comments, it is logical to ask: Why would anyone be interested in studying neural networks having more than three layers? After all, a three-layer network can implement decision surfaces of arbitrary complexity. The answer lies in the method used to train a network to utilize only three layers. The training rule for the network in Fig. 12.16 minimizes an error measure but says nothing about how to associate groups of hyperplanes with specific nodes in the second layer of a three-layer network of the type discussed earlier. In fact, the problem of how to perform trade-off analyses between the number of layers and the number of nodes in each layer remains unresolved. In practice, the trade-off is generally resolved by trial and error or by previous experience with a given problem domain.

12.3 Structural Methods

The techniques discussed in Section 12.2 deal with patterns quantitatively and largely ignore any structural relationships inherent in a pattern's shape. The structural methods discussed in this section, however, seek to achieve pattern recognition by capitalizing precisely on these types of relationships. In this section, we introduce two basic approaches for the recognition of boundary shapes based on string representations. Strings are the most practical approach in structural pattern recognition.

12.3.1 Matching Shape Numbers

A procedure analogous to the minimum distance concept introduced in Section 12.2.1 for pattern vectors can be formulated for the comparison of region boundaries that are described in terms of shape numbers. With reference to the discussion in Section 11.2.2, the *degree of similarity*, k, between two region boundaries (shapes) is defined as the largest order for which their shape numbers still coincide. For example, let a and b denote shape numbers of closed boundaries represented by 4-directional chain codes. These two shapes have a degree of similarity k if

$$\begin{aligned} s_j(a) &= s_j(b) \qquad \text{for } j = 4, 6, 8, \ldots, k \\ s_j(a) &\neq s_j(b) \qquad \text{for } j = k + 2, k + 4, \ldots \end{aligned} \tag{12.3-1}$$

where s indicates shape number and the subscript indicates order. The *distance* between two shapes a and b is defined as the inverse of their degree of similarity:

$$D(a, b) = \frac{1}{k} \tag{12.3-2}$$

This distance satisfies the following properties:

$$\begin{aligned} D(a, b) &\geq 0 \\ D(a, b) &= 0 \quad \text{iff } a = b \\ D(a, c) &\leq \max\big[D(a, b), D(b, c)\big] \end{aligned} \tag{12.3-3}$$

Either k or D may be used to compare two shapes. If the degree of similarity is used, the larger k is, the more similar the shapes are (note that k is infinite for identical shapes). The reverse is true when the distance measure is used.

EXAMPLE 12.7: Using shape numbers to compare shapes.

■ Suppose that we have a shape f and want to find its closest match in a set of five other shapes (a, b, c, d, and e), as shown in Fig. 12.24(a). This problem is analogous to having five prototype shapes and trying to find the best match to a given unknown shape. The search may be visualized with the aid of the similarity tree shown in Fig. 12.24(b). The root of the tree corresponds to the lowest possible degree of similarity, which, for this example, is 4. Suppose that the shapes are identical up to degree 8, with the exception of shape a, whose degree of similarity with respect to all other shapes is 6. Proceeding down the

a
b c

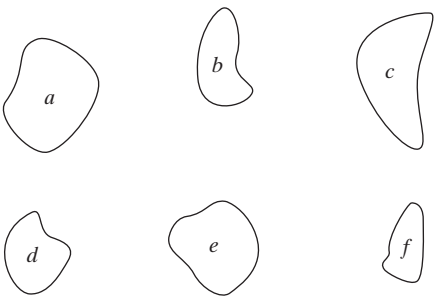
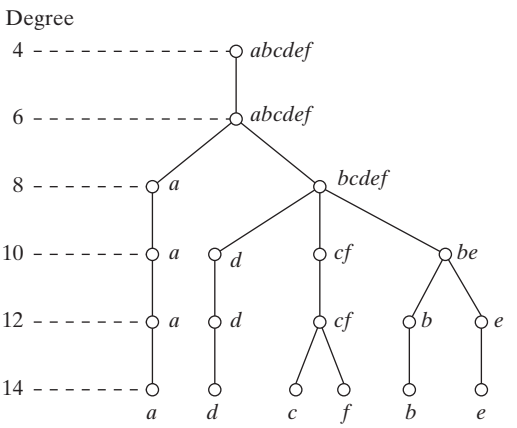


FIGURE 12.24
(a) Shapes.
(b) Hypothetical similarity tree.
(c) Similarity matrix.
(Bribiesca and Guzman.)



	a	*b*	*c*	*d*	*e*	*f*
a	∞	6	6	6	6	6
b		∞	8	8	10	8
c			∞	8	8	12
d				∞	8	8
e					∞	8
f						∞

tree, we find that shape *d* has degree of similarity 8 with respect to all others, and so on. Shapes *f* and *c* match uniquely, having a higher degree of similarity than any other two shapes. At the other extreme, if *a* had been an unknown shape, all we could have said using this method is that *a* was similar to the other five shapes with degree of similarity 6. The same information can be summarized in the form of a *similarity matrix*, as shown in Fig. 12.24(c). ■

12.3.2 String Matching

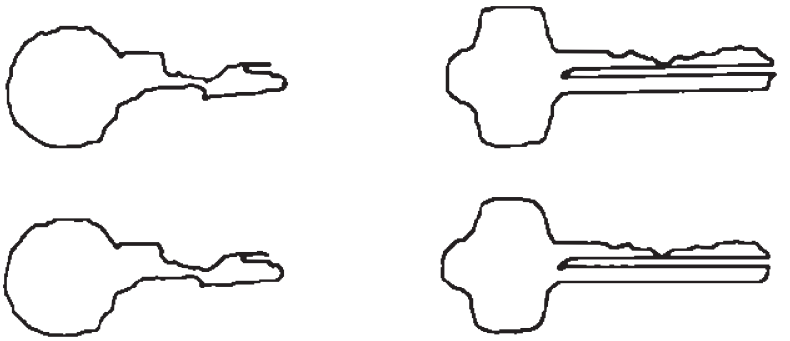
Suppose that two region boundaries, *a* and *b*, are coded into strings (see Section 11.5) denoted $a_1a_2 \ldots a_n$ and $b_1b_2 \ldots b_m$, respectively. Let α represent the number of matches between the two strings, where a match occurs in the kth position if $a_k = b_k$. The number of symbols that do not match is

$$\beta = \max(|a|, |b|) - \alpha \tag{12.3-4}$$

where |arg| is the length (number of symbols) in the string representation of the argument. It can be shown that $\beta = 0$ if and only if *a* and *b* are identical (see Problem 12.21).

A simple measure of similarity between *a* and *b* is the ratio

$$R = \frac{\alpha}{\beta} = \frac{\alpha}{\max(|a|, |b|) - \alpha} \tag{12.3-5}$$



a b
c d
e f
g

FIGURE 12.25 (a) and (b) Sample boundaries of two different object classes; (c) and (d) their corresponding polygonal approximations; (e)–(g) tabulations of R. (Sze and Yang.)

R	1.a	1.b	1.c	1.d	1.e	1.f
1.a	∞					
1.b	16.0	∞				
1.c	9.6	26.3	∞			
1.d	5.1	8.1	10.3	∞		
1.e	4.7	7.2	10.3	14.2	∞	
1.f	4.7	7.2	10.3	8.4	23.7	∞

R	2.a	2.b	2.c	2.d	2.e	2.f
2.a	∞					
2.b	33.5	∞				
2.c	4.8	5.8	∞			
2.d	3.6	4.2	19.3	∞		
2.e	2.8	3.3	9.2	18.3	∞	
2.f	2.6	3.0	7.7	13.5	27.0	∞

R	1.a	1.b	1.c	1.d	1.e	1.f
2.a	1.24	1.50	1.32	1.47	1.55	1.48
2.b	1.18	1.43	1.32	1.47	1.55	1.48
2.c	1.02	1.18	1.19	1.32	1.39	1.48
2.d	1.02	1.18	1.19	1.32	1.29	1.40
2.e	0.93	1.07	1.08	1.19	1.24	1.25
2.f	0.89	1.02	1.02	1.24	1.22	1.18

Hence R is infinite for a perfect match and 0 when none of the corresponding symbols in a and b match ($\alpha = 0$ in this case). Because matching is done symbol by symbol, the starting point on each boundary is important in terms of reducing the amount of computation. Any method that normalizes to, or near, the same starting point is helpful, so long as it provides a computational advantage over brute-force matching, which consists of starting at arbitrary points on each string and then shifting one of the strings (with wraparound) and computing Eq. (12.3-5) for each shift. The largest value of R gives the best match.

EXAMPLE 12.8: Illustration of string matching.

■ Figures 12.25(a) and (b) show sample boundaries from each of two object classes, which were approximated by a polygonal fit (see Section 11.1.3). Figures 12.25(c) and (d) show the polygonal approximations corresponding to the boundaries shown in Figs. 12.25(a) and (b), respectively. Strings were formed from the polygons by computing the interior angle, θ, between segments as each polygon was traversed clockwise. Angles were coded into one of eight possible symbols, corresponding to 45° increments; that is, α_1: $0° < \theta \leq 45°$; α_2: $45° < \theta \leq 90°$; . . . ; α_8: $315° < \theta \leq 360°$.

Figure 12.25(e) shows the results of computing the measure R for six samples of object 1 against themselves. The entries correspond to R values and, for example, the notation 1.c refers to the third string from object class 1. Figure 12.25(f) shows the results of comparing the strings of the second object class against themselves. Finally, Fig. 12.25(g) shows a tabulation of R values obtained by comparing strings of one class against the other. Note that, here, all R values are considerably smaller than any entry in the two preceding tabulations, indicating that the R measure achieved a high degree of discrimination between the two classes of objects. For example, if the class membership of string 1.a had been unknown, the *smallest* value of R resulting from comparing this string against sample (prototype) strings of class 1 would have been 4.7 [Fig. 12.25(e)]. By contrast, the *largest* value in comparing it against strings of class 2 would have been 1.24 [Fig. 12.25(g)]. This result would have led to the conclusion that string 1.a is a member of object class 1. This approach to classification is analogous to the minimum distance classifier introduced in Section 12.2.1. ■

Summary

Starting with Chapter 9, our treatment of digital image processing began a transition from processes whose outputs are images to processes whose outputs are attributes about images, in the sense defined in Section 1.1. Although the material in the present chapter is introductory in nature, the topics covered are fundamental to understanding the state of the art in object recognition. As mentioned at the beginning of this chapter, recognition of individual objects is a logical place to conclude this book. To go past this point, we need concepts that are beyond the scope we set for our journey back in Section 1.4. Specifically, the next logical step would be the development of image analysis methods whose proper development requires concepts from machine intelligence.

As mentioned in Sections 1.1 and 1.4, machine intelligence and some areas that depend on it, such as scene analysis and computer vision, still are in their relatively early stages of practical development. Solutions of image analysis problems today are characterized by heuristic approaches. While these approaches are indeed varied, most of them share a significant base of techniques that are precisely the methods covered in this book.

Having concluded study of the material in the preceding twelve chapters, you are now in the position of being able to understand the principal areas spanning the field of digital image processing, both from a theoretical and practical point of view. Care was taken throughout all discussions to lay a solid foundation upon which further study of this and related fields could be based. Given the task-specific nature of many imaging problems, a clear understanding of basic principles enhances significantly the chances for their successful solution.

References and Further Reading

Background material for Sections 12.1 through 12.2.2 are the books by Theodoridis and Koutroumbas [2006], by Duda, Hart, and Stork [2001], and by Tou and Gonzalez [1974]. The survey article by Jain et al. [2000] also is of interest. The book by Principe et al. [1999] presents a good overview of neural networks. A special issue of *IEEE Trans. Image Processing* [1998] is worth comparing with a similar special issue ten years earlier (*IEEE Computer* [1988]). The material presented in Section 12.2.3 is introductory. In fact, the neural network model used in that discussion is one of numerous models proposed over the years. However, the model we discussed is representative and also is

used quite extensively in image processing. The example dealing with the recognition of distorted shapes is adapted from Gupta et al. [1990, 1994]. The paper by Gori and Scarselli [1998] discusses the classification power of multilayer neural networks. An approach reported by Ueda [2000] based on using linear combinations of neural networks to achieve minimum classification error is good additional reading in this context.

For additional reading on the material in Section 12.3.1, see Bribiesca and Guzman [1980]. On string matching, see Sze and Yang [1981], Oommen and Loke [1997], and Gdalyahu and Weinshall [1999]. Additional references on structural pattern recognition are Gonzalez and Thomason [1978], Fu [1982], Bunke and Sanfeliu [1990], Tanaka [1995], Vailaya et al. [1998], Aizaka and Nakamura [1999], and Jonk et al. [1999]. See also the book by Huang [2002].

Detailed solutions to the problems marked with a star can be found in the book Web site. The site also contains suggested projects based on the material in this chapter.

Problems

12.1 **(a)** Compute the decision functions of a minimum distance classifier for the patterns shown in Fig. 12.1. You may obtain the required mean vectors by (careful) inspection.

(b) Sketch the decision surfaces implemented by the decision functions in (a).

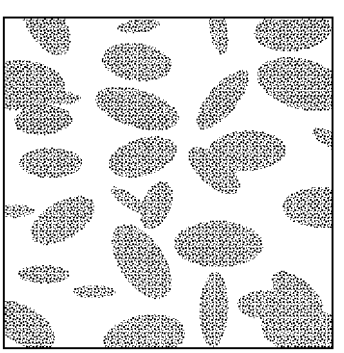
★**12.2** Show that Eqs. (12.2-4) and (12.2-5) perform the same function in terms of pattern classification.

12.3 Show that the surface given by Eq. (12.2-6) is the perpendicular bisector of the line joining the n-dimensional points $\mathbf{m}_i$ and $\mathbf{m}_j$.

★**12.4** Show how the minimum distance classifier discussed in connection with Fig. 12.7 could be implemented by using W resistor banks (W is the number of classes), a summing junction at each bank (for summing currents), and a maximum selector capable of selecting the maximum of W inputs, where the inputs are currents.

12.5 Show that the correlation coefficient of Eq. (12.2-8) has values in the range $[-1, 1]$. (*Hint:* Express $\gamma(x, y)$ in vector form.)

★**12.6** An experiment produces binary images of blobs that are nearly elliptical in shape (see the following figure). The blobs are of three sizes, with the average values of the principal axes of the ellipses being $(1.3, 0.7)$, $(1.0, 0.5)$, and $(0.75, 0.25)$. The dimensions of these axes vary $\pm 10\%$ about their average values. Develop an image processing system capable of rejecting incomplete or overlapping ellipses and then classifying the remaining single ellipses into one of the three size classes given. Show your solution in block diagram form, giving specific details regarding the operation of each block. Solve the classification problem using a minimum distance classifier, indicating clearly how you would go about obtaining training samples and how you would use these samples to train the classifier.



12.7 The following pattern classes have Gaussian probability density functions: ω_1: $\{(0, 0)^T, (2, 0)^T, (2, 2)^T, (0, 2)^T\}$ and ω_2: $\{(4, 4)^T, (6, 4)^T, (6, 6)^T, (4, 6)^T\}$.

(a) Assume that $P(\omega_1) = P(\omega_2) = \frac{1}{2}$ and obtain the equation of the Bayes decision boundary between these two classes.

(b) Sketch the boundary.

★**12.8** Repeat Problem 12.7, but use the following pattern classes: ω_1: $\{(-1, 0)^T, (0, -1)^T, (1, 0)^T, (0, 1)^T\}$ and ω_2: $\{(-2, 0)^T, (0, -2)^T, (2, 0)^T, (0, 2)^T\}$. Observe that these classes are not linearly separable.

12.9 Repeat Problem 12.6, but use a Bayes classifier (assume Gaussian densities). Indicate clearly how you would go about obtaining training samples and how you would use these samples to train the classifier.

★**12.10** The Bayes decision functions $d_j(\mathbf{x}) = p(\mathbf{x}/\omega_j)P(\omega_j)$, $j = 1, 2, \ldots, W$, were derived using a 0-1 loss function. Prove that these decision functions minimize the probability of error. (*Hint:* The probability of error $p(e)$ is $1 - p(c)$, where $p(c)$ is the probability of being correct. For a pattern vector $\mathbf{x}$ belonging to class ω_i, $p(c/\mathbf{x}) = p(\omega_i/\mathbf{x})$. Find $p(c)$ and show that $p(c)$ is maximum [$p(e)$ is minimum] when $p(\mathbf{x}/\omega_i)P(\omega_i)$ is maximum.)

12.11 **(a)** Apply the perceptron algorithm to the following pattern classes: ω_1: $\{(0, 0, 0)^T, (1, 0, 0)^T, (1, 0, 1)^T, (1, 1, 0)^T\}$ and ω_2: $\{(0, 0, 1)^T, (0, 1, 1)^T, (0, 1, 0)^T, (1, 1, 1)^T\}$. Let $c = 1$, and $\mathbf{w}(1) = (-1, -2, -2, 0)^T$.

(b) Sketch the decision surface obtained in (a). Show the pattern classes and indicate the positive side of the surface.

★**12.12** The perceptron algorithm given in Eqs. (12.2-34) through (12.2-36) can be expressed in a more concise form by multiplying the patterns of class ω_2 by -1, in which case the correction steps in the algorithm become $\mathbf{w}(k + 1) = \mathbf{w}(k)$, if $\mathbf{w}^T(k)\mathbf{y}(k) > 0$, and $\mathbf{w}(k + 1) = \mathbf{w}(k) + c\mathbf{y}(k)$ otherwise. This is one of several perceptron algorithm formulations that can be derived by starting from the general gradient descent equation

$$\mathbf{w}(k + 1) = \mathbf{w}(k) - c\left[\frac{\partial J(\mathbf{w}, \mathbf{y})}{\partial \mathbf{w}}\right]_{\mathbf{w}=\mathbf{w}(k)}$$

where $c > 0$, $J(\mathbf{w}, \mathbf{y})$ is a criterion function, and the partial derivative is evaluated at $\mathbf{w} = \mathbf{w}(k)$. Show that the perceptron algorithm formulation is obtainable from this general gradient descent procedure by using the criterion function $J(\mathbf{w}, \mathbf{y}) = \frac{1}{2}(|\mathbf{w}^T\mathbf{y}| - \mathbf{w}^T\mathbf{y})$, where $|\arg|$ is the absolute value of the argument.

(*Note:* The partial derivative of $\mathbf{w}^T\mathbf{y}$ with respect to $\mathbf{w}$ equals $\mathbf{y}$.)

12.13 Prove that the perceptron training algorithm given in Eqs. (12.2-34) through (12.2-36) converges in a finite number of steps if the training pattern sets are linearly separable. [*Hint:* Multiply the patterns of class ω_2 by -1 and consider a nonnegative threshold, T, so that the perceptron training algorithm (with $c = 1$) is expressed as $\mathbf{w}(k + 1) = \mathbf{w}(k)$, if $\mathbf{w}^T(k)\mathbf{y}(k) > T$, and $\mathbf{w}(k + 1) = \mathbf{w}(k) + \mathbf{y}(k)$ otherwise. You may need to use the Cauchy-Schwartz inequality: $\|\mathbf{a}\|^2\|\mathbf{b}\|^2 \geq (\mathbf{a}^T\mathbf{b})^2$.]

★**12.14** Specify the structure and weights of a neural network capable of performing *exactly* the same function as a minimum distance classifier for two pattern classes in n-dimensional space.

12.15 Specify the structure and weights of a neural network capable of performing *exactly* the same function as a Bayes classifier for two pattern classes in n-dimensional space. The classes are Gaussian with different means but equal covariance matrices.

★**12.16** **(a)** Under what conditions are the neural networks in Problems 12.14 and 12.15 identical?

(b) Would the generalized delta rule for multilayer feedforward neural networks developed in Section 12.2.3 yield the particular neural network in (a) if trained with a sufficiently large number of samples?

12.17 Two pattern classes in two dimensions are distributed in such a way that the patterns of class ω_1 lie randomly along a circle of radius r_1. Similarly, the patterns of class ω_2 lie randomly along a circle of radius r_2, where $r_2 = 2r_1$. Specify the structure of a neural network with the minimum number of layers and nodes needed to classify properly the patterns of these two classes.

★**12.18** Repeat Problem 12.6, but use a neural network. Indicate clearly how you would go about obtaining training samples and how you would use these samples to train the classifier. Select the simplest possible neural network that, in your opinion, is capable of solving the problem.

12.19 Show that the expression $h_j'(I_j) = O_j(1 - O_j)$ given in Eq. (12.2-71), where $h_j'(I_j) = \partial h_j(I_j)/\partial I_j$, follows from Eq. (12.2-50) with $\theta_o = 1$.

★**12.20** Show that the distance measure $D(A, B)$ of Eq. (12.3-2) satisfies the properties given in Eq. (12.3-3).

12.21 Show that $\beta = \max(|a|, |b|) - \alpha$ in Eq. (12.3-4) is 0 if and only if a and b are identical strings.

12.22 A certain factory mass produces small American flags for sporting events. The quality assurance team has observed that, during periods of peak production, some printing machines have a tendency to drop (randomly) between one and three stars and one or two entire stripes. Aside from these errors, the flags are perfect in every other way. Although the flags containing errors represent a small percentage of total production, the plant manager decides to solve the problem. After much investigation, he concludes that automatic inspection using image processing techniques is the most economical way to handle the problem. The basic specifications are as follows: The flags are approximately 7.5 cm by 12.5 cm in size. They move lengthwise down the production line (individually, but with a $\pm 15°$ variation in orientation) at approximately 50 cm/s, with a separation between flags of approximately 5 cm. In all cases, "approximately" means $\pm 5\%$. The plant manager hires you to design an image processing system for each production line. You are told that cost and simplicity are important parameters in determining the viability of your approach. Design a complete system based on the model of Fig. 1.23. Document your solution (including assumptions and specifications) in a brief (but clear) written report addressed to the plant manager.