# 3

# SINGLE-LAYER NETWORKS

In Chapter 1 we showed that the optimal decision rule for minimizing the probability of misclassification requires a new pattern to be assigned to the class having the largest posterior probability. We also showed how the posterior probabilities can be related to class-conditional densities through Bayes' theorem, and in Chapter 2 we described several techniques for estimating these densities. An alternative approach, which circumvents the determination of probability densities, is based on the idea of a discriminant function, also introduced in Chapter 1. In a practical application of discriminant functions, specific parametrized functional forms are chosen, and the values of the parameters are then determined from a set of training data by means of a suitable learning algorithm.

The simplest choice of discriminant function consists of a linear combination of the input variables, in which the coefficients in the linear combination are the parameters of the model, and has been considered widely in the literature on conventional approaches to pattern recognition. This simple discriminant can be generalized by transforming the linear combination with a non-linear function (called an activation function) which leads to concepts such as logistic regression and the perceptron. Another extension involves transforming the input variables with fixed non-linear functions before forming the linear combination, to give generalized linear discriminants. As we shall see, these various forms of linear discriminant can be regarded as forms of neural network in which there is a single layer of adaptive weights between the inputs and the outputs.

Various techniques exist for determining the weight values in single-layer networks, and in this chapter we shall consider several of them in detail. In particular, we shall study perceptron learning, least-squares methods and the Fisher discriminant. As well as forming an important class of techniques in their own right, single-layer networks provide many useful insights into the properties of more complex multi-layer networks. Single-layer networks were widely studied in the 1960's, and the history of such networks is reviewed in Widrow and Lehr (1990). Two useful books from this period are Nilsson (1965) and Lewis and Coates (1967).

## 3.1 Linear discriminant functions

In Chapter 1 we saw that optimal discriminant functions can be determined from class-conditional densities via Bayes' theorem. Instead of performing density estimation, however, we can postulate specific parametrized functional forms for

the discriminant functions and use the training data set to determine suitable values for the parameters. In this section we consider various forms of linear discriminant, and discuss their properties.

### 3.1.1 *Two classes*

We begin by considering the two-category classification problem. In Chapter 1 we introduced the concept of a discriminant function $y(\mathbf{x})$ such that the vector $\mathbf{x}$ is assigned to class $C_1$ if $y(\mathbf{x}) > 0$ and to class $C_2$ if $y(\mathbf{x}) < 0$. The simplest choice of discriminant function is one which is linear in the components of $\mathbf{x}$, and which can therefore be written as

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \tag{3.1}$$

where we shall refer to the $d$-dimensional vector $\mathbf{w}$ as the *weight vector* and the parameter $w_0$ as the *bias*. Sometimes $-w_0$ is called a *threshold*. Note that the use of the term bias here is quite distinct from the concept of statistical bias which is discussed briefly on page 41, and at length in Section 9.1. From Section 2.1.3 we know that, for class-conditional densities having normal distributions with equal covariance matrices, a linear discriminant of the form (3.1) is optimal.

The expression in (3.1) has a simple geometrical interpretation (Duda and Hart, 1973) as follows. We first note that the decision boundary $y(\mathbf{x}) = 0$ corresponds to a $(d-1)$-dimensional hyperplane in $d$-dimensional $\mathbf{x}$-space. For the case of a two-dimensional input space, $d = 2$, the decision boundary is a straight line, as shown in Figure 3.1. If $\mathbf{x}^A$ and $\mathbf{x}^B$ are two points on the hyperplane, then $y(\mathbf{x}^A) = 0 = y(\mathbf{x}^B)$ and so, using (3.1), we have $\mathbf{w}^{\mathrm{T}}(\mathbf{x}^B - \mathbf{x}^A) = 0$. Thus, $\mathbf{w}$ is normal to any vector lying in the hyperplane, and so we see that $\mathbf{w}$ determines the orientation of the decision boundary. If $\mathbf{x}$ is a point on the hyperplane then the normal distance from the origin to the hyperplane is given by

$$l = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{x}}{\|\mathbf{w}\|} = -\frac{w_0}{\|\mathbf{w}\|} \tag{3.2}$$

where we have used $y(\mathbf{x}) = 0$ together with (3.1). Thus, the bias $w_0$ determines the position of the hyperplane in $\mathbf{x}$-space, as indicated in Figure 3.1.

There is a slightly different notation which we can adopt which will often prove convenient. If we define new $(d+1)$-dimensional vectors $\widetilde{\mathbf{w}} = (w_0, \mathbf{w})$ and $\widetilde{\mathbf{x}} = (1, \mathbf{x})$, then we can rewrite (3.1) in the form

$$y(\mathbf{x}) = \widetilde{\mathbf{w}}^{\mathrm{T}}\widetilde{\mathbf{x}}. \tag{3.3}$$

With this notation we can interpret the decision boundary $y(\mathbf{x}) = 0$ as a $d$-dimensional hyperplane which passes through the origin in $(d+1)$-dimensional $\widetilde{\mathbf{x}}$-space.

We can represent the linear discriminant function in (3.1) or (3.3) in terms

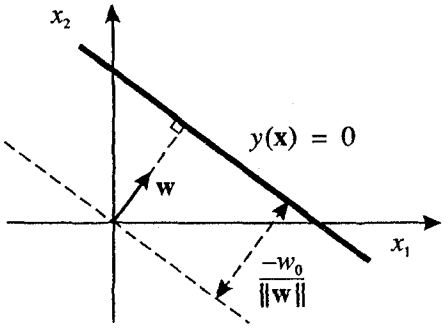


Figure 3.1. A linear decision boundary, corresponding to $y(\mathbf{x}) = 0$, in a two-dimensional input space $(x_1, x_2)$. The weight vector $\mathbf{w}$, which can be represented as a vector in x-space, defines the orientation of the decision plane, while the bias $w_0$ defines the position of the plane in terms of its perpendicular distance from the origin.

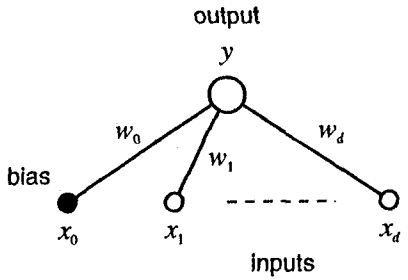


Figure 3.2. Representation of a linear discriminant function as a neural network diagram. Each component in the diagram corresponds to a variable in the linear discriminant expression. The bias $w_0$ can be considered as a weight parameter from an extra input whose activation $x_0$ is permanently set to $+1$.

of a network diagram as shown in Figure 3.2. Inputs $x_1, \ldots, x_d$ are shown as circles, which are connected by the weights $w_1, \ldots, w_d$ to the output $y(\mathbf{x})$. The bias $w_0$ is represented as a weight from an extra input $x_0$ which is permanently set to unity.

### 3.1.2 *Several classes*

Linear discriminants can easily be extended to the case of $c$ classes by following the ideas introduced in Chapter 1 and using one discriminant function $y_k(\mathbf{x})$ for each class $C_k$ of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^{\mathrm{T}}\mathbf{x} + w_{k0}. \tag{3.4}$$

A new point $\mathbf{x}$ is then assigned to class $C_k$ if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$. The decision boundary separating class $C_k$ from class $C_j$ is given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ which, for linear discriminants, corresponds to a hyperplane of the form

$$(\mathbf{w}_k - \mathbf{w}_j)^{\mathrm{T}}\mathbf{x} + (w_{k0} - w_{j0}) = 0. \tag{3.5}$$

By analogy with our earlier results for the single discriminant (3.1), we see that the normal to the decision boundary is given by the difference between the two weight vectors, and that the perpendicular distance of the decision boundary from the origin is given by

$$l = -\frac{(w_{k0} - w_{j0})}{\|\mathbf{w}_k - \mathbf{w}_j\|}. \tag{3.6}$$

The multiclass linear discriminant function (3.4) can be expressed in terms of a neural network diagram as shown in Figure 3.3. The circles at the top of the diagram, corresponding to the functions $y_k(\mathbf{x})$ in (3.4) are sometimes called *processing units*, and the evaluation of the discriminant functions can be viewed as a flow of information from the inputs to the outputs. Each output $y_k(\mathbf{x})$ is associated with a weight vector $\mathbf{w}_k$ and a bias $w_{k0}$. We can express the network outputs in terms of the components of the vectors $\{\mathbf{w}_k\}$ to give

$$y_k(\mathbf{x}) = \sum_{i=1}^{d} w_{ki}x_i + w_{k0}. \tag{3.7}$$

Then each line in Figure 3.3 connecting an input $i$ to an output $k$ corresponds to a weight parameter $w_{ki}$. As before, we can regard the bias parameters as being weights from an extra input $x_0 = 1$, so that

$$y_k(\mathbf{x}) = \sum_{i=0}^{d} w_{ki}x_i. \tag{3.8}$$

Once the network is trained, a new vector is classified by applying it to the inputs of the network, computing the output unit activations, and assigning the vector to the class whose output unit has the largest activation. This leads to a set of decision regions which are always simply connected and convex. To see

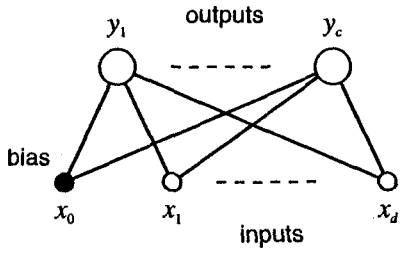


Figure 3.3. Representation of multiple linear discriminant functions $y_k(\mathbf{x})$ as a neural network diagram having $c$ output units. Again, the biases are represented as weights from an extra input $x_0 = 1$.

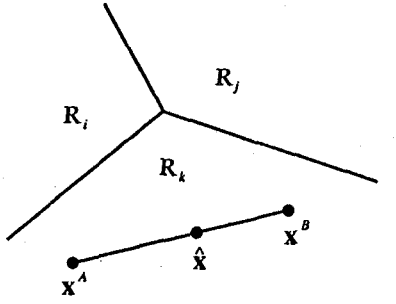


Figure 3.4. Example of decision boundaries produced by a multiclass linear discriminant. If two points $\mathbf{x}^A$ and $\mathbf{x}^B$ both lie in decision region $\mathcal{R}_k$ then every point $\widehat{\mathbf{x}}$ on the line connecting them must also lie in region $\mathcal{R}_k$. It therefore follows that the decision regions must be simply connected and convex.

this, consider two points $\mathbf{x}^A$ and $\mathbf{x}^B$ which both lie in the region $\mathcal{R}_k$ as shown in Figure 3.4. Any point $\widehat{\mathbf{x}}$ which lies on the line joining $\mathbf{x}^A$ and $\mathbf{x}^B$ can be written as

$$\widehat{\mathbf{x}} = \alpha \mathbf{x}^A + (1 - \alpha)\mathbf{x}^B \tag{3.9}$$

where $0 \leq \alpha \leq 1$. Since $\mathbf{x}^A$ and $\mathbf{x}^B$ both lie in $\mathcal{R}_k$, they must satisfy $y_k(\mathbf{x}^A) > y_j(\mathbf{x}^A)$ and $y_k(\mathbf{x}^B) > y_j(\mathbf{x}^B)$ for all $j \neq k$. Using (3.4) and (3.9) it follows that $y_k(\widehat{\mathbf{x}}) = \alpha y_k(\mathbf{x}^A) + (1 - \alpha) y_k(\mathbf{x}^B)$ and hence $y_k(\widehat{\mathbf{x}}) > y_j(\widehat{\mathbf{x}})$ for all $j \neq k$. Thus, all points on the line connecting $\mathbf{x}^A$ and $\mathbf{x}^B$ also lie in $\mathcal{R}_k$ and so the region $\mathcal{R}_k$ must be simply connected and convex.

### 3.1.3 *Logistic discrimination*

So far we have considered discriminant functions which are simple linear functions of the input variables. There are several ways in which such functions can be generalized, and here we consider the use of a non-linear function $g(\cdot)$ which acts on the linear sum to give a discriminant function for the two-class problem of the form

$$y = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0) \tag{3.10}$$

where $g(\cdot)$ is called an *activation function* and is generally chosen to be monotonic. The form (3.10) is still regarded as a linear discriminant since the decision boundary which it generates is still linear, as a consequence of the monotonic nature of $g(\cdot)$.

As a motivation for this form of discriminant, consider a two-class problem in which the class-conditional densities are given by Gaussian distributions with equal covariance matrices $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = \mathbf{\Sigma}$, so that

$$p(\mathbf{x}|C_k) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right\}. \tag{3.11}$$

Using Bayes' theorem, the posterior probability of membership of class $C_1$ is given by

$$P(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_1)P(C_1) + p(\mathbf{x}|C_2)P(C_2)} \tag{3.12}$$

$$= \frac{1}{1 + \exp(-a)} \tag{3.13}$$

$$= g(a) \tag{3.14}$$

where

$$a = \ln \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_2)P(C_2)} \tag{3.15}$$

and the function $g(a)$ is the *logistic sigmoid* activation function given by

$$g(a) \equiv \frac{1}{1 + \exp(-a)} \tag{3.16}$$

which is plotted in Figure 3.5. If we now substitute expressions for the class-conditional densities from (3.11) into (3.15) we obtain

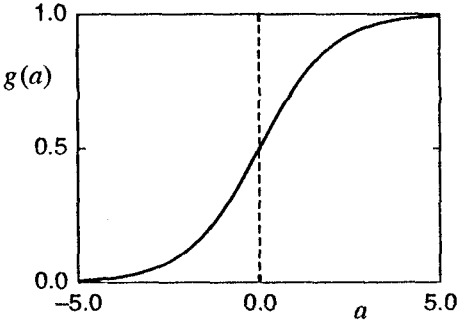


Figure 3.5. Plot of the logistic sigmoid activation function given by (3.16).

$$a = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \tag{3.17}$$

where

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{3.18}$$

$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{P(C_1)}{P(C_2)}. \tag{3.19}$$

Thus, we see that the use of the logistic sigmoid activation function allows the outputs of the discriminant to be interpreted as posterior probabilities. This implies that such a discriminant is providing more than simply a classification decision, and is potentially a very powerful result. The importance of interpreting the outputs of networks in terms of probabilities is discussed at much greater length in Chapter 6.

The term sigmoid means 'S-shaped', and the logistic form of the sigmoid maps the interval $(-\infty, \infty)$ onto $(0, 1)$. If $|a|$ is small, then the logistic sigmoid function $g(a)$ can be approximated by a linear function, and so in this sense a network with sigmoidal activation functions contains a linear network as a special case. If there are more than two classes then an extension of the previous analysis leads to a generalization of the logistic sigmoid called a *normalized exponential* or *softmax*, which is discussed in detail in Section 6.9.

Linear discriminants with logistic activation functions have been widely used in the statistics literature under the name *logistic discrimination* (Anderson, 1982). Sigmoidal activation functions also play a crucial role in multi-layer neural networks, as discussed in Chapter 4.

Another form of linear discriminant was introduced by McCulloch and Pitts (1943) as a simple mathematical model for the behaviour of a single neuron in

a biological nervous system. Again this takes the form (3.10) with an activation function which is the Heaviside step function

$$g(a) = \begin{cases} 0 & \text{when } a < 0 \\ 1 & \text{when } a \geq 0. \end{cases} \tag{3.20}$$

In this model the inputs $x_i$ represent the level of activity of other neurons which connect to the neuron being modelled, the weights $w_i$ represent the strengths of the interconnections, called synapses, between the neurons, and the bias $w_0$ represents the threshold for the neuron to 'fire'. Although this model has its origins in biology, it is clear that it can equally well be motivated within the framework of statistical pattern recognition. Networks of threshold units were studied by Rosenblatt (1962) under the name *perceptrons* and by Widrow and Hoff (1960) who called them *adalines*. They will be discussed in detail in Section 3.5.

Note that it is sometimes convenient to regard the linear discriminant (3.1) as a special case of the more general form (3.10). In this case the model is said to have a linear activation function, which in fact is just the identity $g(a) \equiv a$.

### 3.1.4 *Binary input vectors*

Linear discriminants, and the logistic activation function, also arise in a natural way when we consider input patterns in which the variables are binary (so that each $x_i$ can take only the values 0 or 1). Let $P_{ki}$ denote the probability that the input $x_i$ takes the value +1 when the input vector is drawn from the class $\mathcal{C}_k$. The corresponding probability that $x_i = 0$ is then given by $1 - P_{ki}$. We can combine these together to write the probability for $x_i$ to take either of its allowed values in the form

$$p(x_i|\mathcal{C}_k) = P_{ki}^{x_i}(1 - P_{ki})^{1-x_i} \tag{3.21}$$

which is called a *Bernoulli* distribution. If we now assume that the input variables are statistically independent, we obtain the probability for the complete input vector as the product of the probabilities for each of the components separately:

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^{d} P_{ki}^{x_i}(1 - P_{ki})^{1-x_i}. \tag{3.22}$$

We now recall from Chapter 1 that we can write a discriminant function which minimizes the probability of misclassifying new inputs in the form

$$y_k(\mathbf{x}) = \ln P(\mathbf{x}|\mathcal{C}_k) + \ln P(\mathcal{C}_k). \tag{3.23}$$

Substituting (3.22) into (3.23) we obtain a linear discriminant function given by

$$y_k(\mathbf{x}) = \sum_{i=1}^{d} w_{ki} x_i + w_{k0} \tag{3.24}$$

in which the weights and bias are given by

$$w_{ki} = \ln P_{ki} - \ln(1 - P_{ki}) \tag{3.25}$$

$$w_{k0} = \sum_{i=1}^{d} \ln(1 - P_{ki}) + \ln P(\mathcal{C}_k). \tag{3.26}$$

We have already seen that, for two classes with normally distributed class-conditional densities, the posterior probabilities can be obtained from the linear discriminant by applying a logistic activation function. A similar result holds also for the Bernoulli distribution. Consider a set of independent binary variables $x_i$, having Bernoulli class-conditional densities given by (3.22). If we substitute (3.22) into (3.12) we again obtain a single-layer network structure, with a logistic activation function, of the form

$$P(\mathcal{C}_1|\mathbf{x}) = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0) \tag{3.27}$$

where $g(a)$ is given by (3.16) and

$$w_0 = \sum_i \ln \frac{1 - P_{1i}}{1 - P_{2i}} + \ln \frac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} \tag{3.28}$$

$$w_i = \ln \frac{P_{1i}}{P_{2i}} - \ln \frac{1 - P_{1i}}{1 - P_{2i}}. \tag{3.29}$$

We have shown that, both for normally distributed and Bernoulli distributed class-conditional densities, the posterior probabilities are obtained by a logistic single-layer network. In fact these are particular instances of a much more general result, which is derived in Section 6.7.1.

## 3.2 Linear separability

So far in this chapter we have discussed discriminant functions having a decision boundary which is linear, or more generally hyperplanar in higher dimensions. Clearly this is a very restricted class of decision boundary, and we might well expect such systems to have less than optimal performance for many practical applications. Indeed, this provides the principal motivation for using multi-layer networks of the kind discussed in Chapters 4 and 5. The particular nature of the limitation inherent in single-layer systems warrants some careful discussion, however.

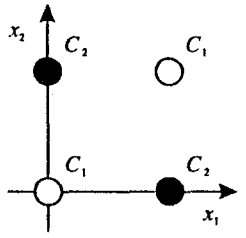


Figure 3.6. The exclusive-OR problem consists of four patterns in a two-dimensional space as shown. It provides a simple example of a problem which is not linearly separable.

Consider for the moment the problem of learning to classify a given data set exactly, where each input vector has been labelled as belonging to one of two classes $C_1$ and $C_2$. If all of the points can be a classified correctly by a linear (i.e. hyperplanar) decision boundary, then the points are said to be *linearly separable*. For such a data set there exist weight and bias values such that a linear discriminant will lead to perfect classification. A simple example of a data set which is not linearly separable is provided by the two-dimensional *exclusive-OR* problem, also known as XOR, illustrated in Figure 3.6. The input vectors $\mathbf{x} = (0,0)$ and $(1,1)$ belong to class $C_1$, while the input vectors $(0,1)$ and $(1,0)$ belong to class $C_2$. It is clear that there is no linear decision boundary which can classify all four points correctly. This problem can be generalized to $d$-dimensions when it is known as the $d$-bit parity problem. In this case the data set consists of all possible binary input vectors of length $d$, which are classified as class $C_1$ if there is an even number of 1's in the input vector, and as class $C_2$ otherwise.

For the case of continuous input variables it is interesting to consider the probability that a random set of patterns will be linearly separable. Suppose we have $N$ data points distributed at random in $d$ dimensions. Note that the particular distribution used to generate the random points is not relevant. All that we require is that there are no accidental degeneracies, i.e. that there is no subset of $d$ or fewer points which are linearly dependent. The points are then said to be in *general position*. Having chosen the points, imagine that we then randomly assign each of the points to one of the two classes $C_1$ and $C_2$ with equal probability. Each possible assignment for the complete data set is referred to as a *dichotomy*, and for $N$ points there are $2^N$ possible dichotomies. We now ask what fraction $F(N,d)$ of these dichotomies is linearly separable. It can be shown (Cover, 1965) that this fraction is given by the expression

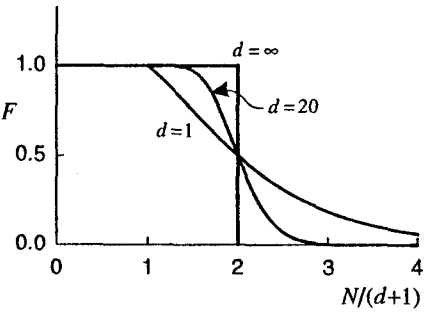


Figure 3.7. Plot of the fraction $F(N, d)$ of the dichotomies of $N$ data points in $d$ dimensions which are linearly separable, as a function of $N/(d+1)$, for various values of $d$.

$$F(N,d) = \begin{cases} 1 & \text{when } N \leq d+1 \\ \dfrac{1}{2^{N-1}} \displaystyle\sum_{i=0}^{d} \binom{N-1}{i} & \text{when } N \geq d+1 \end{cases} \tag{3.30}$$

which is plotted as a function of $N/(d+1)$ in Figure 3.7 for $d = 1$, $d = 20$ and $d = \infty$. Here the symbol

$$\binom{N}{M} \equiv \frac{N!}{(N-M)!M!} \tag{3.31}$$

denotes the number of combinations of $M$ objects selected from a total of $N$. We see from (3.30) that, if the number of data points is fewer than $d+1$, any labelling of the points will always lead to a linearly separable problem. For $N = 2(d+1)$, the probability of linear separability is 0.5 for any value of $d$ (Exercise 3.5). In a practical application, the positions of points from the same class will tend to be correlated, and so the probability that a data set with a much larger number of points than $2(d+1)$ will be linearly separable is higher than (3.30) would suggest.

For the case of binary input patterns, if there are $d$ inputs then there are $2^d$ possible input patterns and hence $2^{2^d}$ possible labellings of those patterns between two classes. Those which can be implemented by a perceptron are called *threshold logic functions* and form an extremely small subset (less than $2^{d^2}/d!$) of the total (Lewis and Coates, 1967).

In the neural computing literature a lot of attention is often paid to the inability of single-layer networks to solve simple problems such as XOR. From our statistical pattern recognition perspective, however, we see that the ability of a particular model to provide an exact representation of a given training set is

largely irrelevant. We are primarily interested in designing systems with good generalization performance, so that they give the greatest accuracy when presented with previously unseen data. Furthermore, problems such as XOR and parity involve learning the complete set of all possible input patterns, so the concept of generalization does not even apply. Finally, they have the property that the smallest possible change in the input pattern produces the largest possible change in the output. Most practical pattern recognition problems have the opposite characteristic, so that small changes in the inputs do not, for the most part, produce large changes in the outputs, and hence the mapping represented by the network should be relatively smooth.

Consider the problem of two normally-distributed classes with equal covariance matrices, discussed in Section 2.1.3. Since the class distributions overlap it is entirely possible that a finite sized data set drawn from these distributions will not be linearly separable. However, we know that the optimal decision boundary is in fact linear. A single-layer network can therefore achieve the best possible classification performance on unseen data, even though it may not separate the training data exactly.

The key consideration concerns the choice of an appropriate discriminant function for the particular problem in hand. This may involve a combination of prior knowledge of the general form which the solution should take, coupled with an empirical comparison of the performance of alternative models. These issues are considered in more detail in Chapters 8, 9 and 10. Here we simply note that single-layer networks correspond to a very narrow class of possible discriminant functions, and in many practical situations may not represent the optimal choice. Nevertheless, single-layer networks remain of considerable practical importance in providing a benchmark against which the performance of more complex multi-layer networks can be assessed. The fact that single-layer networks can often be trained very quickly, as shown in Section 3.4, gives them a particular advantage over more complex network structures which often require considerable computational effort to train.

### 3.3 Generalized linear discriminants

One way to generalize the discriminant functions, so as to permit a much larger range of possible decision boundaries, is to transform the input vector $\mathbf{x}$ using a set of $M$ predefined non-linear functions $\phi_j(\mathbf{x})$, sometimes called *basis functions*, and then to represent the output as a linear combination of these functions

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}. \tag{3.32}$$

This now represents a much larger class of functions $y_k(\mathbf{x})$. In fact, as discussed in Chapters 4 and 5, for a suitable choice of the basis functions $\phi_j(\mathbf{x})$, the function in (3.32) can approximate any continuous functional transformation to arbitrary

accuracy. Again, we can absorb the biases as special cases of the weights by defining an extra basis function $\phi_0 = 1$, so that

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}). \tag{3.33}$$

We have assumed that the basis functions $\phi_j(\mathbf{x})$ are fixed, independently of the data. Chapters 4 and 5 discuss multi-layer neural networks, many of which can be regarded as generalized discriminant functions of the form (3.32), but in which the basis functions themselves can be modified during the training process.

## 3.4 Least-squares techniques

So far in this chapter we have discussed various forms of single-layer network and explored some of their properties. The remainder of the chapter is concerned with techniques for training such networks, and we begin with a discussion of methods based on the minimization of a sum-of-squares error function. This is the simplest form of error function and is most suitable for regression problems. While it can also be used for classification problems, there exist other, more appropriate, error functions, discussed at length in Chapter 6.

### 3.4.1 *Sum-of-squares error function*

For consistency with the discussions in Chapter 5, we shall consider the error minimization problem in the context of the generalized linear network (3.33). This contains the simple linear discriminant of (3.4) as a special case in which the $\phi_j(\mathbf{x})$ simply correspond to the input variables $x_i$. The sum-of-squares error function is given by a sum over all patterns in the training set, and over all outputs, of the form

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\{y_k(\mathbf{x}^n;\mathbf{w}) - t_k^n\}^2 \tag{3.34}$$

where $y_k(\mathbf{x}^n;\mathbf{w})$ represents the output of unit $k$ as a function of the input vector $\mathbf{x}^n$ and the weight vector $\mathbf{w}$, $N$ is the number of training patterns, and $c$ is the number of outputs. The quantity $t_k^n$ represents the target value for output unit $k$ when the input vector is $\mathbf{x}^n$. This error function is a smooth function of the weight parameters $w_{kj}$, and can be minimized by a variety of standard techniques. Since (3.33) is a linear function of the weights, the error function $E(\mathbf{w})$ is a quadratic function of the weights, and hence its derivatives with respect to the weights are linear functions of the weights. The solution for the weight values at the minimum of the error function can therefore be found exactly in closed form, as we shall see in Section 3.4.3.

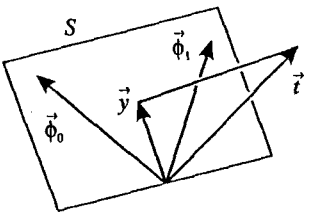


Figure 3.8. Geometrical interpretation of the solution to the least-squares problem, illustrated for the case of 3 training patterns ($N = 3$) and 2 basis functions $\phi_0$ and $\phi_1$ (corresponding to $M = 1$). The target values $t^n$ are grouped together to form an $N$-dimensional vector $\vec{t}$ which lives in an $N$-dimensional Euclidean space. The corresponding network outputs can similarly be represented as a vector $\vec{y}$ which consists of a linear combination of $M + 1$ basis vectors $\vec{\phi}_j$, which themselves span an $(M + 1)$-dimensional Euclidean sub-space $S$. The least-squares solution for $\vec{y}$ is given by the orthogonal projection of $\vec{t}$ onto $S$.

### 3.4.2 *Geometrical interpretation of least squares*

Before deriving a solution for the weights, it is instructive to consider a geometrical interpretation of the least-squares problem. To do this we consider a network having a single output $y$. There is no loss of generality in doing this as the same discussion applies separately to each output of the network. For a particular input pattern $\mathbf{x}^n$ we can write the network output as

$$y^n = \sum_{j=0}^{M} w_j \phi_j^n \tag{3.35}$$

where $\phi_j^n \equiv \phi_j(\mathbf{x}^n)$. We now group the target values together to form an $N$-dimensional vector $\vec{t}$ whose elements are given by $t^n$. This vector can be considered to live in an $N$-dimensional Euclidean space, as indicated in Figure 3.8. For each basis function $\phi_j(\mathbf{x})$ we can similarly group the $N$ values of $\phi_j^n$, corresponding to the $N$ data points, to make a vector $\vec{\phi}_j$, also of dimension $N$, which can be drawn in the same space as the vector $\vec{t}$. For the moment we shall assume that the number of basis functions (including the bias) is less than the number of patterns, so that $M + 1 < N$. The $M + 1$ vectors $\vec{\phi}_j$, corresponding to the $M + 1$ basis functions, then form a (non-orthogonal) basis set which spans an $(M + 1)$-dimensional Euclidean sub-space $S$. The network outputs $y^n$ can also be grouped to form a vector $\vec{y}$. From (3.35) we see that $\vec{y}$ is given by a linear combination of the $\vec{\phi}_j$ of the form

$$\vec{y} = \sum_{j=0}^{M} w_j \vec{\phi}_j \tag{3.36}$$

so that $\vec{y}$ is constrained to lie in the sub-space $S$, as shown in Figure 3.8. By changing the values of the weights $w_j$ we can change the location of $\vec{y}$ subject to this constraint.

The sum-of-squares error (3.34) can now be written in the form

$$E = \frac{1}{2} \left\| \sum_{j=0}^{M} w_i \vec{\phi}_j - \vec{t} \right\|^2 \tag{3.37}$$

If we minimize this expression with respect to the weights $w_j$ we find

$$\frac{\partial E}{\partial w_j} = 0 = \vec{\phi}_j^{\mathrm{T}} (\vec{y} - \vec{t}), \qquad j = 1, \ldots, M. \tag{3.38}$$

This represents a set of coupled equations for the weights, known as the *normal equations* of the least-squares problem, for which we shall find an explicit solution shortly. Before doing so, however, it is useful to consider the geometrical interpretation of (3.38). Let us decompose $\vec{t}$ into the sum of two vectors $\vec{t} = \vec{t}_\perp + \vec{t}_\parallel$ where $\vec{t}_\parallel$ is the orthogonal projection of $\vec{t}$ onto the sub-space $S$, and $\vec{t}_\perp$ is the remainder. Then $\vec{\phi}_j^{\mathrm{T}} \vec{t}_\perp = 0$ by definition, and hence from (3.38) we have

$$\vec{\phi}_j^{\mathrm{T}} (\vec{y} - \vec{t}_\parallel) = 0, \qquad j = 1, \ldots, M. \tag{3.39}$$

Since the vectors $\vec{\phi}_i$ form a basis set which span the sub-space $S$, we can solve (3.39) to give

$$\vec{y} = \vec{t}_\parallel \tag{3.40}$$

and so the solution vector is just the projection of the vector of target values onto the sub-space spanned by the basis vectors, as indicated in Figure 3.8. This result is intuitively correct, since the process of learning corresponds to choosing a direction for $\vec{y}$ such as to minimize its distance from $\vec{t}$. Since $\vec{y}$ is constrained to lie in the sub-space, the best we can do is choose it to correspond to the orthogonal projection of $\vec{t}$ onto $S$. This minimizes the length of the error vector $\vec{\epsilon} = \vec{y} - \vec{t}$. Note that the residual error vector $\vec{\epsilon}_{\mathrm{min}} = \vec{t}_\parallel - \vec{t} = -\vec{t}_\perp$ is then orthogonal to $S$, so that $\vec{\phi}_j^{\mathrm{T}} \vec{\epsilon}_{\mathrm{min}} = 0$.

### 3.4.3 *Pseudo-inverse solution*

We now proceed to find an exact solution to the least-squares problem. To do this we return to the case of a network having $c$ outputs. Using the expression (3.33), we can write the sum-of-squares error function (3.34) in the form

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\left\{\sum_{j=0}^{M} w_{kj}\phi_j^n - t_k^n\right\}^2. \tag{3.41}$$

Differentiating this expression with respect to $w_{kj}$ and setting the derivative to zero gives the normal equations for the least-squares problem in the form

$$\sum_{n=1}^{N}\left\{\sum_{j'=0}^{M} w_{kj'}\phi_{j'}^n - t_k^n\right\}\phi_j^n = 0. \tag{3.42}$$

In order to find a solution to (3.42) it is convenient to write it in a matrix notation to give

$$(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi})\mathbf{W}^{\mathrm{T}} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{T}. \tag{3.43}$$

Here $\mathbf{\Phi}$ has dimensions $N \times M$ and elements $\phi_j^n$, $\mathbf{W}$ has dimensions $c \times M$ and elements $w_{kj}$, and $\mathbf{T}$ has dimensions $N \times c$ and elements $t_k^n$. The matrix $\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$ in (3.43) is a square matrix of dimension $M \times M$. Provided it is non-singular we may invert it to obtain a solution to (3.43) which can be written in the form

$$\mathbf{W}^{\mathrm{T}} = \mathbf{\Phi}^{\dagger}\mathbf{T} \tag{3.44}$$

where $\mathbf{\Phi}^{\dagger}$ is an $M \times N$ matrix known as the *pseudo-inverse* of $\mathbf{\Phi}$ (Golub and Kahan, 1965; Rao and Mitra, 1971) and is given by

$$\mathbf{\Phi}^{\dagger} \equiv (\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathrm{T}} \tag{3.45}$$

Since $\mathbf{\Phi}$ is, in general, a non-square matrix it does not itself have a true inverse, but the pseudo-inverse does have the property (as is easily seen from 3.45) that $\mathbf{\Phi}^{\dagger}\mathbf{\Phi} = \mathbf{I}$ where $\mathbf{I}$ is the unit matrix. Note, however, that $\mathbf{\Phi}\mathbf{\Phi}^{\dagger} \neq \mathbf{I}$ in general. If the matrix $\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$ is singular then (3.43) does not have a unique solution. However, if the pseudo-inverse is defined by

$$\mathbf{\Phi}^{\dagger} \equiv \lim_{\epsilon \to 0} \; (\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \epsilon\mathbf{I})^{-1}\mathbf{\Phi}^{\mathrm{T}} \tag{3.46}$$

then it can be shown that the limit always exists, and that this limiting value minimizes $E$ (Rao and Mitra, 1971).

In practice, the direct solution of the normal equations can lead to numerical difficulties due to the possibility of $\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$ being singular or nearly singular. This can arise if two of the basis vectors $\vec{\phi}_j$, shown in Figure 3.8, are nearly collinear. The effects of noise and numerical error can then lead to very large values for the weights which give near cancellation between these vectors. Figure 3.9(a) shows two basis vectors $\vec{\phi}_1$ and $\vec{\phi}_2$ which are nearly orthogonal, together with the component $\vec{y}_{\|}$ of $\vec{y}$ which lies in the plane spanned by $\vec{\phi}_1$ and $\vec{\phi}_2$. The corresponding weight values needed to express $\vec{y}_{\|}$ as a linear combination of $\vec{\phi}_1$ and $\vec{\phi}_2$ have relatively small values. By contrast, Figure 3.9(b) shows the corresponding situation when the vectors $\vec{\phi}_1$ and $\vec{\phi}_2$ are nearly collinear. In this case the weights need to adopt large (positive or negative) values in order to represent $\vec{y}_{\|}$ as a linear combination of the basis vectors. In the case where the two basis vectors are exactly collinear, we can write $\vec{\phi}_2 = \lambda\vec{\phi}_1$ for some constant $\lambda$. Then $w_1\vec{\phi}_1 + w_2\vec{\phi}_2 = (w_1 + \lambda w_2)\vec{\phi}_1$ and only the combination $(w_1 + \lambda w_2)$ is fixed by the least-squares procedure, with the value of $w_2$, say, being arbitrary. Near degeneracies will not be uncommon when dealing with real, noisy data sets. In practice, such problems are best resolved by using the technique of *singular value decomposition* (SVD) to find a solution for the weights. A good introduction to SVD, together with a suggested numerical implementation, can be found in Press *et al.* (1992). Such an approach avoids problems due to the accumulation of numerical roundoff errors, and automatically selects (from amongst a set of nearly degenerate solutions) the one for which the length $\|\mathbf{w}_k\|$ of the $k$th weight vector is shortest.

In the above discussion, the bias parameters were treated as a special case of the weights. We can gain some insight into the role of the biases if we make them explicit. If we consider the minimization of (3.41) with respect to the bias parameters alone we obtain

$$\frac{\partial E}{\partial w_{k0}} = \sum_{n=1}^{N}\left\{\sum_{j=1}^{M} w_{kj}\phi_j^n + w_{k0} - t_k^n\right\} = 0 \tag{3.47}$$

which can be solved for the biases to give

$$w_{k0} = \bar{t}_k - \sum_{j=1}^{M} w_{kj}\bar{\phi}_j \tag{3.48}$$

where

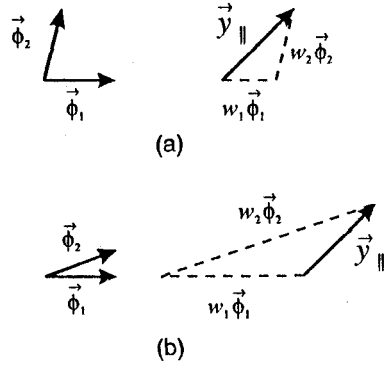


Figure 3.9. In (a) we see two basis vectors $\vec{\phi}_1$ and $\vec{\phi}_2$ which are nearly orthogonal. The least-squares solution vector $\vec{y}_\parallel$ is given by a linear combination of these vectors, with relatively small values for the coefficients $w_1$ and $w_2$. In (b) the basis vectors are nearly collinear, and the magnitudes of the corresponding weight values become very large.

$$\bar{t}_k = \frac{1}{N}\sum_{n=1}^{N} t_k^n, \qquad \bar{\phi}_j = \frac{1}{N}\sum_{n=1}^{N} \phi_j^n. \tag{3.49}$$

This result tells us that the role of the bias parameters is to compensate for the difference between the mean (over the training set) of the output vector for the network and the corresponding mean of the target data.

If $\mathbf{\Phi}^{\mathrm{T}}$ is a square non-singular matrix, the pseudo-inverse reduces to the usual inverse. The matrix is square when $N = M$, so that the number of patterns equals the number of basis functions. If we multiply (3.43) by $(\mathbf{\Phi}^{\mathrm{T}})^{-1}$ we obtain

$$\mathbf{\Phi}\mathbf{W}^{\mathrm{T}} = \mathbf{T}. \tag{3.50}$$

If we write this in index notation we have

$$\sum_{j=0}^{M} w_{kj}\phi_j^n = t_k^n \tag{3.51}$$

and we see that, for each input pattern, the network outputs are exactly equal to the corresponding target values, and hence the sum-of-squares error (3.41) will be zero. The condition for $(\mathbf{\Phi}^{\mathrm{T}})^{-1}$ to exist is that the columns $\boldsymbol{\phi}^n$ of the matrix $\mathbf{\Phi}^{\mathrm{T}}$ be linearly independent. If the vectors $\boldsymbol{\phi}^n$ are not linearly independent, so that the effective value of $N$ is less than $M$, then the least-squares problem

is under-determined. Similarly, if there are fewer patterns than basis functions, so that $N < M$, then the least-squares problem is again under-determined. In such cases, there is a continuum of solutions for the weights, all of which give zero error. Singular value decomposition leads to a numerically well-behaved algorithm which picks out the particular solution for which the magnitude $\|\mathbf{w}_k\|$ of the weight vector for each output unit $k$ is the shortest. As we have already indicated in Chapter 1, it is desirable to have a sufficiently large training set that the weight values are 'over-determined', so that in practice we arrange that $N > M$, which corresponds to the situation depicted in Figure 3.8.

### 3.4.4 *Gradient descent*

We have shown how, for a linear network, the weight values which minimize the sum-of-squares error function can be found explicitly in terms of the pseudo-inverse of a matrix. It is important to note that this result is only possible for the case of a linear network, with a sum-of-squares error function. If a non-linear activation function, such as a sigmoid, is used, or if a different error function is considered, then a closed form solution is no longer possible. However, if the activation function is differentiable, as is the case for the logistic sigmoid in (3.16) for instance, the derivatives of the error function with respect to the weight parameters can easily be evaluated. These derivatives can then be used in a variety of gradient-based optimization algorithms, discussed in Chapter 7, for finding the minimum of the error function. Here we consider one of the simplest of such algorithms, known as gradient descent.

It is convenient to group all of the parameters (weights and biases) in the network together to form a single weight vector $\mathbf{w}$, so that the error function can be expressed as $E = E(\mathbf{w})$. Provided $E$ is a differentiable function of $\mathbf{w}$ we may adopt the following procedure. We begin with an initial guess for $\mathbf{w}$ (which might for instance be chosen at random) and we then update the weight vector by moving a small distance in $\mathbf{w}$-space in the direction in which $E$ decreases most rapidly, i.e. in the direction of $-\nabla_{\mathbf{w}} E$. By iterating this process we generate a sequence of weight vectors $\mathbf{w}^{(\tau)}$ whose components are calculated using

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}} \tag{3.52}$$

where $\eta$ is a small positive number called the *learning rate* parameter. Under suitable conditions the sequence of weight vectors will converge to a point at which $E$ is minimized. The choice of the value for $\eta$ can be fairly critical, since if it is too small the reduction in error will be very slow, while, if it is too large, divergent oscillations can result.

In general the error function is given by a sum of terms each of which is calculated using just one of the patterns from the training set, so that

$$E(\mathbf{w}) = \sum_n E^n(\mathbf{w}) \tag{3.53}$$

where the term $E^n$ is calculated using pattern $n$ only. In this case we can update the weight vector using just one pattern at a time

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E^n}{\partial w_{kj}} \tag{3.54}$$

and this is repeated many times by cycling through all of the patterns used in the definition of $E$. This form of sequential, or pattern-based, update is reminiscent of the Robbins–Monro procedure introduced in Section 2.4, and many of the same comments apply here. In particular, this technique allows the system to be used in real-time adaptive applications in which data is arriving continuously. Each data point can be used once and then discarded, and if the value of $\eta$ is chosen appropriately, the system may be able to 'track' any slow changes in the characteristics of the data. If $\eta$ is chosen to decrease with time in a suitable way during the learning process, then gradient descent becomes precisely the Robbins–Monro procedure for finding the root of the regression function $\mathcal{E}[\partial E^n/\partial w_l]$ where $\mathcal{E}$ denotes the expectation. If the value of $\eta$ is chosen to be steadily decreasing with time, so that $\eta^{(\tau)} = \eta_0/\tau$ (which satisfies the conditions for the Robbins–Monro theorem stated in Section 2.4), then the weight matrix $\mathbf{W}$ can be shown to converge to a solution of

$$\mathbf{\Phi}^{\mathrm{T}}(\mathbf{\Phi}\mathbf{W} - \mathbf{T}) = 0 \tag{3.55}$$

where $\mathbf{\Phi}$ is defined on page 92, irrespective of whether or not $\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$ is singular. Gradient descent, and its limitations, are discussed at greater length in Chapter 7, along with a variety of more sophisticated optimization algorithms.

In order to implement gradient descent, we need explicit expressions for the derivatives of the error function with respect to the weights. We consider first the pattern-based form of gradient descent given by (3.54). For a generalized linear network function of the form (3.33) the derivatives are given by

$$\frac{\partial E^n}{\partial w_{kj}} = \{y_k(\mathbf{x}^n) - t_k^n\}\phi_j(\mathbf{x}^n) = \delta_k^n \phi_j^n \tag{3.56}$$

where we have defined

$$\delta_k^n \equiv y_k(\mathbf{x}^n) - t_k^n. \tag{3.57}$$

We see that the derivative with respect to a weight $w_{kj}$ connecting basis function $j$ to output $k$ can be expressed as the product of $\delta_k$ for the output unit and $\phi_j$ for the basis function. Thus, the derivative can be calculated from quantities which are 'local' (in the sense of the network diagram) to the weight concerned. This property is discussed at greater length in the context of multi-layer networks in Section 4.8. Combining (3.54) and (3.56) we see that the change in the weights

due to presentation of a particular pattern is given by

$$\Delta w_{kj} = -\eta \delta_k^n \phi_j^n. \tag{3.58}$$

This rule, and its variants, are known by a variety of names including the LMS (least mean squares) rule, the adaline rule, the Widrow–Hoff rule (Widrow and Hoff, 1960), and the delta rule.

For networks with differentiable non-linear activation functions, such as the logistic sigmoid shown in Figure 3.5, we can write the network outputs in the form

$$y_k = g(a_k) \tag{3.59}$$

where $g(\cdot)$ is the activation function, and

$$a_k = \sum_{j=0}^{M} w_{kj} \phi_j. \tag{3.60}$$

The derivatives of the error function for pattern $n$ again take the form

$$\frac{\partial E^n}{\partial w_{kj}} = g'(a_k) \delta_k^n \phi_j^n \tag{3.61}$$

in which

$$\delta_k^n = g'(a_k)(y_k(\mathbf{x}^n) - t_k^n). \tag{3.62}$$

For the logistic sigmoid given by (3.16), the derivative of the activation function can be expressed in the simple form

$$g'(a) = g(a)(1 - g(a)). \tag{3.63}$$

For gradient descent based on the total error function (summed over all patterns in the training set) given by (3.52), the derivatives are obtained by computing the derivatives for each pattern separately and then summing over all patterns

$$\frac{\partial E}{\partial w_{kj}} = \sum_n \frac{\partial E^n}{\partial w_{kj}}. \tag{3.64}$$

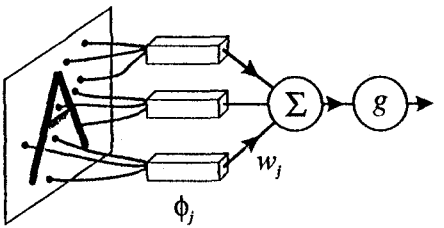


Figure 3.10. The perceptron network used a fixed set of processing elements, denoted $\phi_j$, followed by a layer of adaptive weights $w_j$ and a threshold activation function $g(\cdot)$. The processing elements $\phi_j$ typically also had threshold activation functions, and took inputs from a randomly chosen subset of the pixels of the input image.

## 3.5 The perceptron

Single-layer networks, with threshold activation functions, were studied by Rosenblatt (1962) who called them *perceptrons*. Rosenblatt also built hardware implementations of these networks, which incorporated learning using an algorithm to be discussed below. These networks were applied to classification problems, in which the inputs were usually binary images of characters or simple shapes. The properties of perceptrons are reviewed in Block (1962).

At the same time as Rosenblatt was developing the perceptron, Widrow and co-workers were working along similar lines using systems known as *adalines* (Widrow and Lehr, 1990). The term adaline comes from ADAptive LINear Element, and refers to a single processing unit with threshold non-linearity (Widrow and Hoff, 1960) of essentially the same form as the perceptron.

We have already seen that a network with a single layer of weights has very limited capabilities. To improve the performance of the perceptron, Rosenblatt used a layer of fixed processing elements to transform the raw input data, as shown in Figure 3.10. These processing elements can be regarded as the basis functions of a generalized linear discriminant. They typically took the form of fixed weights connected to a random subset of the input pixels, with a threshold activation function of the form (3.20). We shall again use the convention introduced earlier of defining an extra basis function $\phi_0$ whose activation is permanently set to $+1$, together with a corresponding bias parameter $w_0$. The output of the perceptron is therefore given by

$$y = g\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right) = g(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}) \tag{3.65}$$

where $\boldsymbol{\phi}$ denotes the vector formed from the activations $\phi_0, \ldots, \phi_M$. The output

unit activation function is most conveniently chosen to be an anti-symmetric version of the threshold activation function of the form

$$g(a) = \begin{cases} -1 & \text{when } a < 0 \\ +1 & \text{when } a \geq 0. \end{cases} \tag{3.66}$$

We now turn to a discussion of the procedures used to train the perceptron.

### 3.5.1 *The perceptron criterion*

Since our goal is to produce an effective classification system, it would be natural to define the error function in terms of the total number of misclassifications over the training set. More generally we could introduce a loss matrix (Section 1.10) and consider the total loss incurred as a result of a particular classification of the data set. Such error measures, however, prove very difficult to work with in practice. This is because smooth changes in the values of the weights (and biases) cause the decision boundaries to move across the data points resulting in discontinuous changes in the error. The error function is therefore piecewise constant, and so procedures akin to gradient descent cannot be applied. We therefore seek other error functions which can be more easily minimized.

In this section we consider a continuous, piecewise-linear error function called the *perceptron criterion*. As each input vector $\mathbf{x}^n$ is presented to the inputs of the network it generates a corresponding vector of activations $\boldsymbol{\phi}^n$ in the first-layer processing elements. Suppose we associate with each input vector $\mathbf{x}^n$ a corresponding target value $t^n$, such that the desired output from the network is $t^n = +1$ if the input vector belongs to class $\mathcal{C}_1$, and $t^n = -1$ if the vector belongs to class $\mathcal{C}_2$. From (3.65) and (3.66) we want $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}^n > 0$ for vectors from class $\mathcal{C}_1$, and $\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}^n < 0$ for vectors from class $\mathcal{C}_2$. It therefore follows that for all vectors we want to have $\mathbf{w}^{\mathrm{T}}(\boldsymbol{\phi}^n t^n) > 0$. This suggests that we try to minimize the following error function, known as the perceptron criterion

$$E^{\mathrm{perc}}(\mathbf{w}) = -\sum_{\boldsymbol{\phi}^n \in \mathcal{M}} \mathbf{w}^{\mathrm{T}}(\boldsymbol{\phi}^n t^n) \tag{3.67}$$

where $\mathcal{M}$ is the set of vectors $\boldsymbol{\phi}^n$ which are *misclassified* by the current weight vector $\mathbf{w}$. The error function $E^{\mathrm{perc}}(\mathbf{w})$ is the sum of a number of positive terms, and equals zero if all of the data points are correctly classified. From the discussion in Section 3.1 we see that $E^{\mathrm{perc}}(\mathbf{w})$ is proportional to the sum, over all of the input patterns which are misclassified, of the (absolute) distances to the decision boundary. During training, the decision boundary will move and some points which were previously misclassified will become correctly classified (and vice versa) so that the set of patterns which contribute to the sum in (3.67) will change. The perceptron criterion is therefore continuous and piecewise linear with discontinuities in its gradient.

### 3.5.2 *Perceptron learning*

If we apply the pattern-by-pattern gradient descent rule (3.54) to the perceptron criterion (3.67) we obtain

$$w_j^{(\tau+1)} = w_j^{(\tau)} + \eta \phi_j^n t^n. \tag{3.68}$$

This corresponds to a very simple learning algorithm which can be summarized as follows. Cycle through all of the patterns in the training set and test each pattern in turn using the current set of weight values. If the pattern is correctly classified do nothing, otherwise add the pattern vector (multiplied by $\eta$) to the weight vector if the pattern is labelled class $\mathcal{C}_1$ or subtract the pattern vector (multiplied by $\eta$) from the weight vector if the pattern is labelled class $\mathcal{C}_2$. It is easy to see that this procedure tends to reduce the error since

$$-\mathbf{w}^{(\tau+1)\mathrm{T}}(\boldsymbol{\phi}^n t^n) = -\mathbf{w}^{(\tau)\mathrm{T}}(\boldsymbol{\phi}^n t^n) - \eta(\boldsymbol{\phi}^n t^n)^{\mathrm{T}}(\boldsymbol{\phi}^n t^n) < -\mathbf{w}^{(\tau)\mathrm{T}}(\boldsymbol{\phi}^n t^n) \tag{3.69}$$

since $\|\boldsymbol{\phi}^n t^n\|^2 > 0$ and $\eta > 0$.

For the particular case of the perceptron criterion, we see that the value of $\eta$ is in fact unimportant since a change in $\eta$ is equivalent to a re-scaling of the weights and bias (assuming the initial parameter values are similarly re-scaled). This leaves the location of the decision boundaries unchanged. To see this, recall that the location of the decision boundary is given by (3.2), and is therefore unchanged if all of the weights, including the bias, are rescaled by the same constant. Thus, when minimizing the perceptron criterion, we can take $\eta = 1$ with no loss of generality. This property does not hold, however, for most other forms of error function.

In Figures 3.11–3.13 we give a simple example of learning in a perceptron, for the case of one basis function $\phi_1$, so that, with biases included as special cases of the weights, the data points live in a two-dimensional space $(\phi_0, \phi_1)$ with $\phi_0 = 1$.

### 3.5.3 *Perceptron convergence theorem*

There is an interesting result which states that, for any data set which is linearly separable, the learning rule in (3.68) is guaranteed to find a solution in a finite number of steps (Rosenblatt, 1962; Block, 1962; Nilsson, 1965; Minsky and Papert, 1969; Duda and Hart, 1973; Hand, 1981; Arbib, 1987; Hertz *et al.*, 1991). This is known as the *perceptron convergence theorem*. Here we give a relatively simple proof, based on Hertz *et al.* (1991).

Since we are considering a data set which is linearly separable, we know that there exists at least one weight vector $\widehat{\mathbf{w}}$ for which all training vectors are correctly classified, so that

$$\widehat{\mathbf{w}}^{\mathrm{T}} \boldsymbol{\phi}^n t^n > 0 \qquad \text{for all } n. \tag{3.70}$$

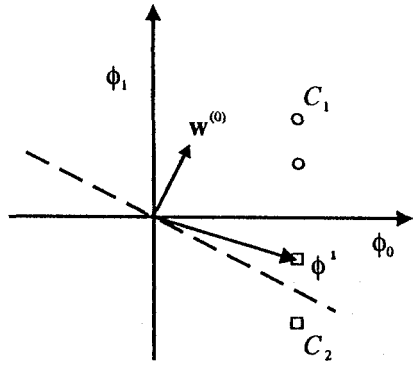


Figure 3.11. A simple example of perceptron learning, for a data set with four patterns. Circles represent patterns belonging to class $C_1$ and squares represent patterns belonging to class $C_2$. The initial decision boundary, corresponding to the weight vector $\mathbf{w}^{(0)}$, shown by the dashed curve, leaves one of the points, at $\boldsymbol{\phi}^1$, incorrectly classified.

The learning process starts with some arbitrary weight vector which, without loss of generality, we can assume to be the zero vector. At each step of the algorithm, the weight vector is updated using

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \boldsymbol{\phi}^n t^n \tag{3.71}$$

where $\boldsymbol{\phi}^n$ is a vector which is misclassified by the perceptron. Suppose that, after running the algorithm for some time, the number of times that each vector $\boldsymbol{\phi}^n$ has been presented and misclassified is $\tau^n$. Then the weight vector at this point will be given by

$$\mathbf{w} = \sum_n \tau^n \boldsymbol{\phi}^n t^n. \tag{3.72}$$

We now take the scalar product of this equation with $\widehat{\mathbf{w}}$ to give

$$\begin{aligned} \widehat{\mathbf{w}}^{\mathrm{T}}\mathbf{w} &= \sum_n \tau^n \widehat{\mathbf{w}}^{\mathrm{T}} \boldsymbol{\phi}^n t^n \\ &\geq \tau \min_n (\widehat{\mathbf{w}}^{\mathrm{T}} \boldsymbol{\phi}^n t^n) \end{aligned} \tag{3.73}$$

where $\tau = \sum_n \tau^n$ is the total number of weight updates, and the inequality results from replacing each update vector by the smallest of the update vectors.