3 Single-Layer Networks

Single-layer networks (figure 3.1) have just one layer of active units. Inputs connect directly to the outputs through a single layer of weights. The outputs do not interact so a network with N_{out} outputs can be treated as N_{out} separate single-output networks. Each unit (figure 3.2) produces its output by forming a weighted linear combination of its inputs which it then passes through a saturating nonlinear function

$$u = \sum_j w_j x_j \tag{3.1}$$

$$y = f(u). \tag{3.2}$$

This can be expressed more compactly in vector notation as

$$y(\mathbf{x}) = f\left(\mathbf{w}^T \mathbf{x}\right) \tag{3.3}$$

where $\mathbf{x}$ and $\mathbf{w}$ are column vectors with elements x_j and w_j, and the superscript T denotes the vector transpose. In general, f is chosen to be a bounded monotonic function. Common choices include the sigmoid function $f(u) = 1/(1 + e^{-u})$ and the tanh functions. When f is a discontinuous step function, the nodes are often called *linear threshold units* (LTU). Appendix D mentions other possibilities.

3.1 Hyperplane Geometry

Equations 3.1 and 3.2 are fundamental to most of the networks considered later so it is useful to examine them more closely. The locus of points $\mathbf{x}$ with a constant sum $\sum_j w_j x_j$ defines a hyperplane perpendicular to the vector $\mathbf{w}$. The Euclidean vector norm $\|\mathbf{x}\| = \sqrt{\sum_i x_i^2}$ measures vector length. Because $\mathbf{w}^T\mathbf{x} = \|\mathbf{w}\|\,\|\mathbf{x}\| \cos\phi$, where ϕ is the angle between $\mathbf{w}$ and $\mathbf{x}$, u is proportional to the projection $\|\mathbf{x}\| \cos\phi$ of $\mathbf{x}$ onto $\mathbf{w}$ and all points with equivalent projections produce equivalent outputs (figure 3.3). The locus of points with equivalent projections on $\mathbf{w}$ are hyperplanes orthogonal to $\mathbf{w}$, so the output y is a function of the distance from $\mathbf{x}$ to the hyperplane defined by $\mathbf{w}$. The constant-output surfaces of (3.2) are hyperplanes perpendicular to $\mathbf{w}$.

Orientation The orientation of the node hyperplane is determined by the direction of $\mathbf{w}$. This depends on the relative sizes of the weights w_i but not on the overall magnitude of $\mathbf{w}$. Let $\mathbf{e}_i$ be the unit vector aligned with the ith coordinate axis, for example, $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)$ (w_i will still be used to refer to the ith component of a vector, however). The angle ϕ_1 between the hyperplane normal and the ith coordinate axis is then

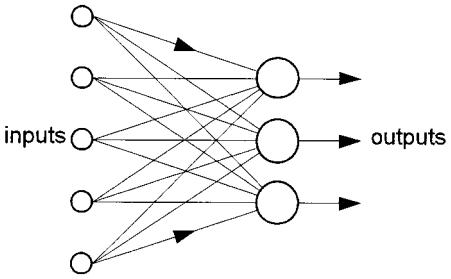

Figure 3.1
A single-layer perceptron has no hidden layers. One layer of weights connects the inputs directly to the outputs. The outputs are independent so this network can be treated as three separate networks.

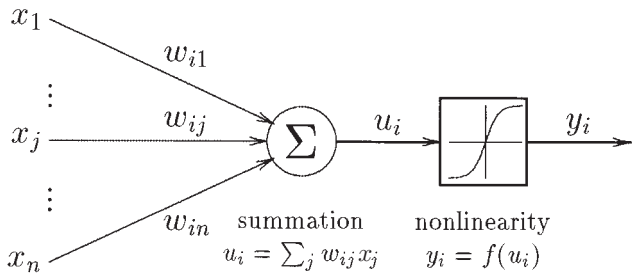

Figure 3.2
Node function. Each node i computes a weighted sum of its inputs and passes the result through a nonlinearity, typically a bounded monotonic function such as the sigmoid.

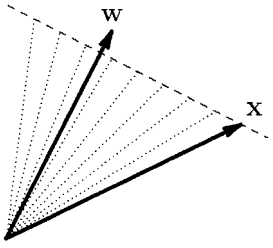

Figure 3.3
Projection of **x** onto **w**. The output of a single unit is determined by the inner product of the input vector **x** and the weight vector **w**: $u = \mathbf{w}^T\mathbf{x} = \|\mathbf{w}\|\ \|\mathbf{x}\| \cos\phi$. All inputs **x** with the same projection onto **w** produce the same output. The locus of points with equivalent projections onto **w** defines a hyperplane perpendicular to **w**.

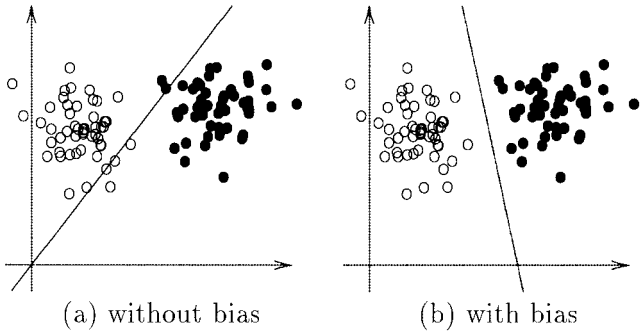

Figure 3.4
Effect of bias weights. A linear threshold unit divides its input space into two half-spaces: (a) without bias, the dividing surface must pass through the origin and certain data sets will not be separable; (b) with bias, the dividing surface can be offset from the origin to obtain better classification.

$$\begin{aligned}
&\mathbf{w}^T \mathbf{e}_i = \|\mathbf{w}\| \ \|\mathbf{e}_i\| \ \cos \phi_i \\
&w_i = \|\mathbf{w}\| \cos \phi_i \qquad (\|\mathbf{e}_i\| = 1) \\
&\cos \phi_i = w_i / \|\mathbf{w}\|. \qquad (3.4)
\end{aligned}$$

The orientation of the plane is independent of the magnitude of $\mathbf{w}$ because the ratios $w_i/\|\mathbf{w}\|$ remain constant when $\mathbf{w}$ is multiplied by a constant.

Distance from the Origin As noted previously, the constant-output surfaces of (3.2) are hyperplanes perpendicular to $\mathbf{w}$. More specifically, the weighted sum $\sum_j w_j x_j = 0$ defines a hyperplane *through the origin*. Inclusion of a *threshold*, or *bias*, term θ

$$u = \mathbf{w}^T \mathbf{x} - \theta \qquad (3.5)$$

shifts the hyperplane along $\mathbf{w}$ to a distance $d = \theta/\|\mathbf{w}\|$ from the origin. To see this, let $\mathbf{v}$ be the vector from the origin to the closest point on the plane. It must be normal to the plane, and thus parallel to $\mathbf{w}$, so $\mathbf{v} = d\mathbf{w}/\|\mathbf{w}\|$. The node hyperplane is the locus of points where $\mathbf{w}^T \mathbf{x} - \theta = 0$ so we have

$$\begin{aligned}
&\mathbf{w}^T \mathbf{v} - \theta = 0 \\
&d\mathbf{w}^T \mathbf{w}/\|\mathbf{w}\| - \theta = 0 \\
&d = \theta/\|\mathbf{w}\|. \qquad (3.6)
\end{aligned}$$

Figure 3.4 illustrates the utility of the bias term. Without bias, the decision surface must pass through the origin and so will be unable to separate some data sets. Addition of a bias allows the surface to be shifted from the origin to obtain better classification. To simplify

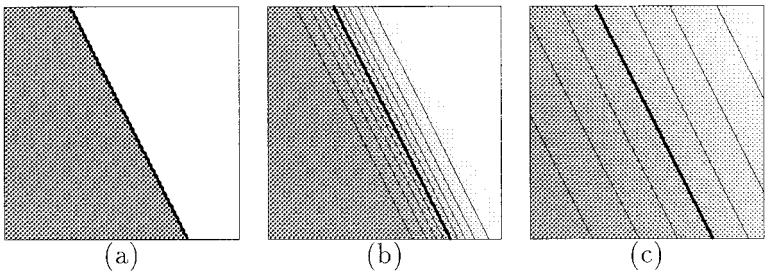

Figure 3.5
Graded responses: (a) a hard-limiter divides the input space with a hyperplane, (b) a sigmoid gives a similar response, but has a smoother transition at the boundary, and (c) a sigmoid with small weights gives an almost linear response for a wide range of inputs.

analyses, the threshold is usually absorbed into the weight vector by assuming that one of the inputs is constant, $x_{bias} = 1$. The constant input is called the *bias node*.

Gradation The node nonlinearity f in (3.2) controls how the output varies as the distance from $\mathbf{x}$ to the node hyperplane changes. As noted, f is usually chosen to be a bounded monotonic function. When f is a binary hard-limiting function as in a linear threshold unit, the node divides the input space with a hyperplane, producing 0 for inputs on one side of the plane and 1 for inputs on the other side. With a softer nonlinearity such as the sigmoid, the transition from 0 to 1 is smoother but other properties are similar.

The magnitude of $\mathbf{w}$ in equation 3.3 plays the role of a scaling parameter that can be varied to obtain transitions of varying steepness. The slope of the transition is $\|\partial y/\partial \mathbf{x}\| = f'(u)\ \|\mathbf{w}\|$, which is proportional to $\|\mathbf{w}\|$, the magnitude of the weight vector. For large $\|\mathbf{w}\|$, the slope is steep and the sigmoid approximates a step function. For small $\|\mathbf{w}\|$, the slope is small and $y(\mathbf{x})$ is nearly linear over a wide range of inputs. Figure 3.5 illustrates functions with various degrees of gradation. In any case, the output is solely a function of the distance of the input from the hyperplane.

3.2 Linear Separability

An important limitation of single-layer perceptrons was pointed out by Minsky and Papert [268]: a single-layer perceptron can correctly classify only data sets which are *linearly separable*. Classes A and B are linearly separable if they can be separated by a hyperplane, i.e., if a plane exists such that classes A and B lie on opposite sides (figure 3.6). In two dimensions, the hyperplane reduces to a line and classes are linearly separable if a line can be drawn between them. Section 3.3 discusses the probability that a random set of patterns is linearly separable. The exclusive-OR function (figure 3.7) is a well-known example of

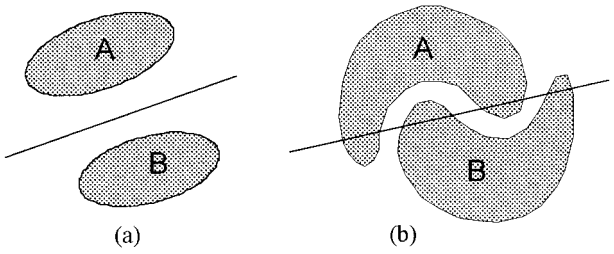

Figure 3.6
(a) Linearly separable classes can be separated by a hyperplane. In two dimensions, the hyperplane is a line. (b) Classes that are not linearly separable cannot be separated by a hyperplane.

a	b	XOR(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

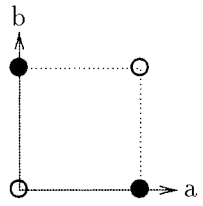

Figure 3.7
The two-input exclusive-OR function is a simple binary function that is not linearly separable and thus not learnable by a single-layer perceptron. In more than two dimensions, it generalizes to the parity function, which is 1 when the number of active inputs is odd and 0 when it is even.

a simple function that is not linearly separable and thus not computable by single-layer perceptrons.

This result is significant because many classification problems are not linearly separable. There are 2^{2^d} Boolean functions of d Boolean input variables, for example, only $O(2^{d^2})$ of which are linearly separable [278]. When d is large, the fraction of Boolean functions that are linearly separable and thus computable by a single-layer net becomes very small. For $d = 2$, a total of 14 of the 16 Boolean functions are linearly separable. (Exclusive-OR and its complement are the exceptions.) But for $d = 4$, only 1,882 out of 65536 are linearly separable [278; 273].

A large part of the appeal of neural networks is that they "learn" from examples, apparently imitating human abilities. The letdown when it was demonstrated that single-layer networks could not compute functions even as simple as exclusive-OR is said to have nearly stopped research in neural networks. Although it was known that multilayer nets are more powerful than single-layer nets, effective multilayer learning procedures were not well known at the time. In the 1980s, learning algorithms for multilayer networks became widely known and research interest revived.

In spite of their limitations, single-layer networks are significant for reasons beyond historical interest. There are significant signal processing applications that can be handled by single-layer networks. Understanding of single-layer networks is also useful and necessary for insight into multilayer networks, which are, after all, simply cascades of single-layer networks.

3.3 Hyperplane Capacity

What is the probability that a random set of points with random binary labels is linearly separable? Cover [87] provided the following result. Given N points in a d-dimensional input space, there are 2^N possible ways of labeling the points 0 or 1. Each of these forms a *dichotomy*—a division of the N points into two classes. A dichotomy is linearly separable if all the 0s can be separated from all the 1s by a hyperplane. It is *homogeneously linearly separable* if the points can be separated by a hyperplane passing through the origin. A linear separation of N points in d dimensions is a homogeneous linear separation in $d+1$ dimensions because the offset of a hyperplane that does not pass through the origin can be absorbed into an extra bias weight.

Cover [87] defines the capacity of a hyperplane as the number of dichotomies it can separate. For N points in general position (defined below) in a Euclidean space of dimension d, the number $C(N,d)$ of homogeneously linearly separable dichotomies is

$$C(N,d) = \begin{cases} 2^N & N \le d+1 \\ 2\sum_{k=0}^{d} \binom{N-1}{k} & N > d+1. \end{cases} \tag{3.7}$$

This result is for N points in *general position*. If the N points are not in general position, the number of linearly separable dichotomies may be much lower. A set of d-dimensional vectors is in general position if all possible subsets of $d+1$ or fewer vectors are linearly independent. For $N > d$, this requires that no set of $d+1$ points lie on a $(d-1)$-dimensional hyperplane. In $d=3$ dimensions, for example, no set of 4 points may be coplanar. For $N \le d$, N points are in general position if no $(d-2)$-dimensional hyperplane contains them all.

$C(N,d)$ can be computed recursively with the following relations.

$$\begin{aligned} C(1,d) &= 2 \qquad (d \ge 1) \\ C(N,1) &= 2N \\ C(N+1,d) &= C(N,d) + C(N,d-1) \end{aligned} \tag{3.8}$$

The first relation says a single point can be classified in two ways. The second relation says N points on a line can be classified by a linear separator in $2N$ ways: the separator can fall

in any of the $N-1$ intervals between points and it can be oriented in 2 ways so the zeros fall on one side or the other. Adding the two cases where all points are either zero or one gives the total $2N$.

The recurrence relation is obtained as follows. Suppose N points in d dimensions form $C(N,d)$ dichotomies and we add another point p. This divides the existing dichotomies into two classes:

- Some of the existing dichotomies cannot be obtained by a hyperplane through p. When p is added, each of these continues to be a single dichotomy in the new system with p taking whichever value the existing dichotomy dictates.
- The rest of the existing dichotomies could have been obtained by a hyperplane passing through p. Each of these gives rise to two new dichotomies where p is either 0 or 1 because the hyperplane can be shifted infinitesimally to either side of p without changing the classification of the remaining points.

If M_1 and M_2 are the number of dichotomies in the two classes, then the number of dichotomies in the new system is $C(N+1,d) = M_1 + 2M_2$. But $M_1 + M_2 = C(N,d)$ so $C(N+1,d) = C(N,d) + M_2$. However, $M_2 = C(N, d-1)$ because constraining a hyperplane to pass through p (as well as the origin) is equivalent to reducing the dimension d to $d-1$. Substitution gives the recurrence relation $C(N+1,d) = C(N,d) + C(N, d-1)$.

Repeated application of the recurrence to the terms on the right yields

$$C(N,d) = \sum_{k=0}^{N-1} \binom{N-1}{k} C(1, d-k), \tag{3.9}$$

and (3.7) follows on noting that [87]

$$C(1,m) = \begin{cases} 2, & m \geq 1 \\ 0, & m < 1. \end{cases} \tag{3.10}$$

Returning to Equation 3.7, all dichotomies on $N \leq d+1$ points in general position are linearly separable in d dimensions. (All possible labelings of 3 points in 2 dimensions are linearly separable, for example.) For $N > d+1$, only $C(N,d)$ of the 2^N possible dichotomies are linearly separable. The probability that a randomly chosen dichotomy is linearly separable is then

$$f(N,d) = \begin{cases} 1 & N \leq d+1 \\ \frac{2}{2^N} \sum_{k=0}^{d} \binom{N-1}{k} & N > d+1. \end{cases} \tag{3.11}$$

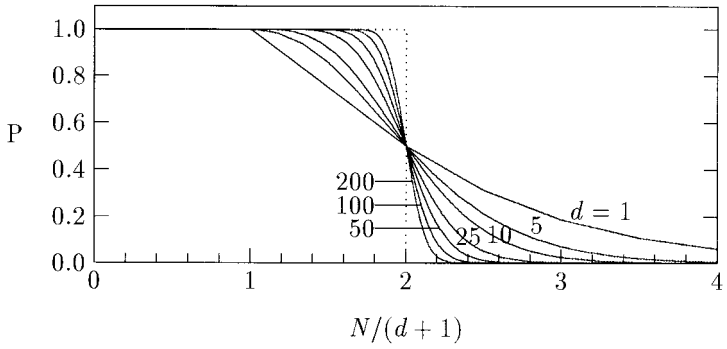

Figure 3.8
Capacity of a hyperplane. Shown is the probability that a random dichotomy on N points in d-dimensions is linearly separable, for various values of d. The probability is greater than $1/2$ for $N < 2(d+1)$, and less than $1/2$ for $N > 2(d+1)$. As d increases, the transition from high to low probability becomes steeper and approaches a step function.

This probability is plotted as a function of $N/(d+1)$ in figure 3.8 for a few values of d. It can be shown that $C(N,d) = 2^{N-1}$ and so $f(N,d) = 1/2$ at $N = 2(d+1)$. When d is large, $f(N,d)$ is greater than 1/2 as long as $N < 2(d+1)$ and less than 1/2 for $N > 2(d+1)$. As d becomes large, the transition becomes steeper and almost all dichotomies of $N < 2(d+1)$ points are linearly separable while almost all dichotomies of more points are not. At $N = 2(d+1)$, one-half of the dichotomies are linearly separable, which suggests $N = 2(d+1)$ points as the separating capacity of the hyperplane.

On the average, a single linear threshold unit with d weights and a threshold can be made to correctly classify up to $2(d+1)$ random binary patterns before the probability of error falls to $1/2$ [87]. For large N, only a tiny fraction of the possible mappings are linearly separable. In terms of generalization, this means that a d-input linear threshold unit can implement any dichotomy on $d+1$ or fewer patterns (in general position) and, when d is large, has a high probability of being able to learn a random dichotomy on up to $2d$ patterns. When $N < 2d$, generalization may be poor because there is a high probability that the training points will be linearly separable even if the function generating the labels is not; the network is underconstrained and the solution is unlikely to generalize well to new patterns. In linear regression, a common heuristic is to require $N \geq 3d$ or more training patterns to avoid overfitting.

The assumptions behind this result should be noted: the points are presumed to be in general position with randomly chosen target labels. In real problems, the points will not necessarily be in general position and if the labels are determined by a physical process

there is an assumption that the label of a point can be inferred from its position. Obviously there are sets of $N \gg 2d$ points which are linearly separable (two well separated clusters, for example) and there are sets of as few as 4 points, not in general position, which are not linearly separable (e.g., four coplanar points in an exclusive-OR configuration).

3.4 Learning Rules for Single-Layer Networks

A single-layer network can be trained by many methods. Almost every optimization technique has been applied to neural network training at some time or another. In addition to the general purpose optimization algorithms (some are reviewed in chapter 10) there are methods specialized for single-layer classifiers.

Rosenblatt's original perceptron learning algorithm (described in section 3.4.1) can be used for binary input-output problems that are known to be linearly separable. Statistical procedures can be applied in many cases. Linear discriminant analysis may be used for classification problems where the data form Gaussian clusters. The mathematics of single-layer perceptrons are similar to linear regression. Indeed, when f is linear, the output is a simple linear combination of the inputs and optimal weights can be calculated directly. Logistic regression (e.g., [178]) is very similar to learning binary functions with a single-layer network of sigmoid units using a cross-entropy error function.

When the node nonlinearity is smooth and the error function is differentiable, various gradient-based optimization methods including back-propagation can be used. More sophisticated gradient based techniques such as Newton's method, Levenberg-Marquardt, or conjugate gradient descent may be especially effective for single-layer networks because their error surface does not depart too much from the basic quadratic error function for which these methods are specialized.

3.4.1 The Perceptron Learning Algorithm

The perceptron learning algorithm is suitable for learning linearly separable binary functions of binary inputs and is guaranteed to find a solution if the classes are linearly separable. Unfortunately, if the classes are not linearly separable, the algorithm may not even converge and is unlikely to produce the best solution when it does. (Section 3.4.3 describes a possible way of handling this problem.) Because of these problems and the general limitations of single-layer networks, the method is rarely used except in special circumstances.

Because the outputs are binary, linear threshold units are used. Each unit computes the weighted sum u of its N inputs x_j, $j = 1 \ldots N$, and generates a binary output y

$$u = \sum_{j=0}^{N} w_j x_j \quad = \mathbf{w}^T \mathbf{x} \tag{3.12}$$

$$y = \begin{cases} -1 & u \leq 0 \\ +1 & u > 0. \end{cases} \tag{3.13}$$

A node threshold is absorbed into the weight vector by assuming the presence of a constant value bias unit, $x_{bias} = 1$. Input, output, and target values are assumed to be ± 1.

The weights can be updated by a number of simple learning rules [166]. During training, input patterns $\mathbf{x}$ are presented and the outputs $y(\mathbf{x})$ are compared to the targets $t(\mathbf{x})$. Weights are adapted by

$$\Delta \mathbf{w} = \begin{cases} 2\eta t \mathbf{x} & \text{if } t \not= y \\ 0 & \text{otherwise,} \end{cases} \tag{3.14}$$

where η is a small positive constant controlling the learning rate. Typically $0 < \eta < 1$. Because $t, y \in \{-1, +1\}$, the following are equivalent:

$$\Delta \mathbf{w} = \eta(1 - ty)t\mathbf{x} \tag{3.15}$$

and

$$\Delta \mathbf{w} = \eta(t - y)\mathbf{x}. \tag{3.16}$$

In both cases, no change occurs when the output classification is correct. Otherwise, each element of the weight vector is incremented by 2η when the output is less than the target or decremented by 2η when the output is greater.

For improved reliability, it may be desirable that a unit activates only when the weighted sum $u = \mathbf{w}^T \mathbf{x}$ is greater than a threshold $N\kappa$, where $0 \leq \kappa < 1$. The following rule may be used [325]:

$$\Delta \mathbf{w} = \eta \, \Theta(N\kappa - tu) \, t\mathbf{x}, \tag{3.17}$$

where $\Theta(u)$ is the unit step function

$$\Theta(u) = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0. \end{cases}$$

Note that as the classification accuracy improves, the system makes fewer errors so weight changes become less and less frequent. The effective learning rate therefore slows down so convergence to perfect classification may take a long time.

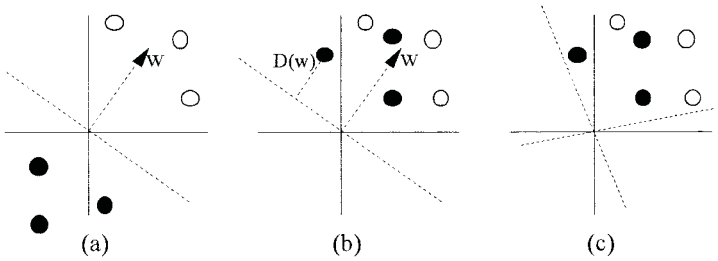

Figure 3.9
The separation problem. (a) A weight vector must be found that separates the positive samples (open circles) from the negative samples (filled circles). (b) The transformation $\mathbf{z} = t\mathbf{x}$ reflects the negative examples across the origin and changes the problem to one of finding a weight vector such that $\mathbf{w} \cdot \mathbf{z} > 0$ for each transformed pattern $\mathbf{z}$. $D(\mathbf{w})$ is the minimum distance from any point $\mathbf{z}$ to the separating hyperplane defined by $\mathbf{w}$. (c) The difficulty of a problem is determined by how much $\mathbf{w}$ can be varied and still meet the separating criterion. Easy problems can be satisfied by many $\mathbf{w}$ vectors.

3.4.2 Perceptron Convergence Proof

With input, output, and target values of ± 1, and using learning rule (3.17), a test of correct classification is that [166]

$$t\mathbf{w} \cdot \mathbf{x} > N\kappa \tag{3.18}$$

($\mathbf{w} \cdot \mathbf{x} \equiv \sum_i w_i x_i$ denotes the inner product of $\mathbf{w}$ and $\mathbf{x}$). As shown in figure 3.9(b), the transformation $\mathbf{z} = t\mathbf{x}$ reflects the negative examples across the origin and changes the problem to one of finding a weight vector such that $\mathbf{w} \cdot \mathbf{z} > 0$ for every pattern $\mathbf{z}$. The points $\mathbf{z}$ will be classified correctly if

$$\mathbf{w} \cdot \mathbf{z} > N\kappa. \tag{3.19}$$

The $N\kappa$ term adds the additional requirement that all $\mathbf{z}$ must be at least a distance $N\kappa/\|\mathbf{w}\|$ from the origin and provides a margin for noise- and fault-tolerance.

The difficulty of a problem is determined by how much $\mathbf{w}$ can be varied and still meet the separating criterion. Easy problems can be satisfied by many hyperplanes while hard problems require very precise placement of the separator (figure 3.9(c)). Let $D(\mathbf{w})$ be the minimum distance from any point $\mathbf{z}$ to the separating hyperplane defined by $\mathbf{w}$

$$D(\mathbf{w}) = \frac{1}{\|\mathbf{w}\|} \min_{\mu} \mathbf{w} \cdot \mathbf{z}^{\mu}, \tag{3.20}$$

where the superscript μ indexes the points. $D(\mathbf{w})$ is positive when all the points are on the positive side of the hyperplane, in which case (3.19) can be satisfied by making $\|\mathbf{w}\|$ large

enough to ensure the safety margin $N\kappa$. There is some $\mathbf{w}_{opt}$ for which $D_{max} = D(\mathbf{w}_{opt})$ is maximized; this defines the *optimal perceptron*.

The following proof of convergence is provided in [166]. Additional results may be found in [284], a good summary of early work in perceptron-like networks. At each step, a pattern is chosen and the weights are updated only if equation 3.19 is not satisfied. Let M^μ denote the number of times that pattern μ has been used to update the weights at some point in the learning process. Then at that time

$$\mathbf{w} = \eta \sum_\mu M^\mu \mathbf{z}^\mu \tag{3.21}$$

if the initial weights are 0.

Assume that a solution vector $\mathbf{w}^*$ exists. Because $\mathbf{w}^*$ is a solution, $D(\mathbf{w}^*) > 0$. The proof computes bounds on $\|\mathbf{w}\|$ and on the overlap $\mathbf{w} \cdot \mathbf{w}^*$. These are used to show that $\mathbf{w} \cdot \mathbf{w}^*/\|\mathbf{w}\|$ would get arbitrarily large if $M = \sum_\mu M^\mu$ kept increasing. This is impossible, however, because $\mathbf{w}^*$ is fixed, so updating must stop after some finite M.

Using (3.20) and (3.19)

$$\mathbf{w} \cdot \mathbf{w}^* = \eta \sum_\mu M^\mu \mathbf{z}^\mu \cdot \mathbf{w}^* \tag{3.22}$$

$$\geq \eta M \min_\mu \mathbf{z}^\mu \cdot \mathbf{w}^* \tag{3.23}$$

$$= \eta M D(\mathbf{w}^*) \|\mathbf{w}^*\|. \tag{3.24}$$

$D(\mathbf{w}^*)\|\mathbf{w}^*\|$ is a constant, so $\mathbf{w} \cdot \mathbf{w}^*$ grows like M.

An upper bound on $\|\mathbf{w}\|$ is obtained by considering the change in length of $\mathbf{w}$ at a single update by pattern α

$$\Delta\|\mathbf{w}\|^2 = \|\mathbf{w} + \eta\mathbf{z}^\alpha\|^2 - \|\mathbf{w}\|^2 \tag{3.25}$$

$$= \eta^2\|\mathbf{z}^\alpha\|^2 + 2\eta\mathbf{w} \cdot \mathbf{z}^\alpha \tag{3.26}$$

$$\leq \eta^2 N + 2\eta N\kappa \tag{3.27}$$

$$\leq N\eta(\eta + 2\kappa). \tag{3.28}$$

The inequality comes from the fact that (3.19) is not satisfied when a weight update occurs. Because $z_i^\mu = \pm 1$, $(\mathbf{z}^\alpha)^2 = N$. The upper bound results from summing the increments for M steps

$$\|\mathbf{w}\|^2 \leq MN\eta(\eta + 2\kappa). \tag{3.29}$$

Thus $\|\mathbf{w}\|$ grows no faster than $\sqrt{M}$.

Next, consider the normalized scalar product

$$\phi = \frac{(\mathbf{w} \cdot \mathbf{w}^*)^2}{\|\mathbf{w}\|^2 \|\mathbf{w}^*\|^2}, \tag{3.30}$$

which is the square of the cosine of the angle between $\mathbf{w}$ and $\mathbf{w}^*$ and cannot be greater than one. By (3.24) and (3.29),

$$M \frac{D(\mathbf{w}^*)^2 \eta}{N(\eta + 2\kappa)} \leq \phi \leq 1, \tag{3.31}$$

which gives an upper bound on the number of weight updates (using the best possible $\mathbf{w}^*$)

$$M \leq N \frac{1 + 2\kappa/\eta}{D_{max}^2}, \tag{3.32}$$

where $D_{max} \equiv \max_{\mathbf{w}} D(\mathbf{w})$ is $D(\mathbf{w})$ maximized over all possible $\mathbf{w}$. Thus M has an upper limit that is an unknown, but finite, constant. Because there can be no more than this number of updates, the weights must eventually converge to a final unchanging value.

This upper bound is proportional to N, the number of inputs, but does not depend on the number of patterns. In reality, however, in order to find the patterns that require weight updates, we have to cycle through the patterns, a process that takes time proportional to the number of patterns on the average. Also, D_{max} usually decreases as the number of patterns grows, resulting in a larger M. M also grows linearly with κ, because for larger κ, a larger $\|\mathbf{w}\|$ is needed to satisfy (3.19) along a good weight vector.

Equation 3.32 cannot be used to calculate the time required for convergence because it requires knowledge of a solution vector. Hampson and Volper [151] studied average case upper and lower bounds for M. The number of weight updates required is bounded by

$$M \leq \frac{\|\mathbf{w}^*\|^2 \|\mathbf{x}\|_{max}^2}{a^2}, \tag{3.33}$$

where $\|\mathbf{x}\|_{max}^2$ is the largest squared magnitude of any input vector ($N + 1$ for N bivalent input units), $\mathbf{w}^*$ is any solution vector, and $a \equiv \min_\mu \mathbf{x}^\mu \cdot \mathbf{w} \geq 0$. The worst case upper bound is $M \approx N^N$. An average case upper bound may be $O(4^N)$ for large N. The worst case lower bound is $M \geq 2^N$. The average case lower bound is $O(1.4^N)$ for large N [151]. It is reported that M grows linearly with the number of irrelevant inputs that are uncorrelated with the target outputs.

3.4.3 The Pocket Algorithm

Although the perceptron algorithm is guaranteed to learn pattern sets that are linearly separable, it does not converge when the training data is not linearly separable. Most of the time, the weight vector stays in regions that give small numbers of errors, but it does not always stay there. It is possible for the system to visit a near optimal set of weights and then wander away to a very bad set; the total error may be small at one moment and suddenly become large in the next so there is no guarantee that a weight vector obtained late in the training process will be near optimal.

Gallant's "pocket algorithm" [133] keeps a copy of the best weight vector obtained so far. Two sets of weights are maintained, a working set and the pocket set. Training examples are chosen randomly and whenever the current weights have a run of consecutive correct classifications longer than the best run by the pocket weights, they replace them. The quality of the pocket weights thus tends to improve over time and converge to an optimum set.

3.4.4 Rosenblatt's Perceptron

Single-layer sigmoidal networks are often called perceptrons even though this isn't strictly accurate. The original perceptron described by Rosenblatt [324, 325, 46] actually consisted of a family of networks, most having more than one layer and some having recurrent connections. In most cases, however, only the final layer of weights to the output units were modifiable. Connections in preceding layers had fixed values, either randomly chosen or set by some systematic procedure. The output nodes were often connected in a winner-take-all fashion. The label probably stuck because Minsky and Papert [268] reduced the structure to a single layer for the purpose of analyzing what could be learned by the modifiable units. The well-known result was to show that single-layer networks were adequate only for linearly separable functions.

Unlike a single-layer network, Rosenblatt's perceptron was capable of correctly classifying patterns that were not linearly separable if the preceding units had appropriate responses. But because only the final layer of weights was modifiable, it could not learn certain functions if this required adjustment of the responses of the preceding units.

Rosenblatt's perceptron is historically significant because it was one of the first biologically plausible models specified precisely enough to simulate and complex enough to show interesting behavior [46]. It was responsible for much of the initial interest in neural networks during the 1960s.

Now the term "perceptron" is commonly used to refer to any feedforward network of nodes with responses like $f(\mathbf{w}^T\mathbf{x})$ where f is a sigmoid-like "squashing function." Single-

layer networks are often simply called perceptrons and networks with more than one layer are typically called multilayer perceptrons.

3.5 Adalines and the Widrow-Hoff Learning Rule

Adaline networks (short for *adaptive linear neuron*) were described by Widrow et al. in the 1960s [402, 399, 404, 401, 400, 357]. The structure is a single-layer network with a step function as the nonlinearity. Although the perceptron was analyzed assuming binary inputs, Adaline inputs may be continuous. Weights are adjusted with the Widrow-Hoff learning rule to minimize the difference between the output and an externally supplied target.

The Widrow-Hoff learning rule, also called the LMS algorithm or the delta rule, is basically an iterative implementation of linear regression (see appendix A). Both minimize the mean squared error of a linear fit. Ideally, the solutions are the same so it shares many properties with linear regression and succeeds or fails in similar situations.

An input pattern $\mathbf{x}$ is selected at random from the training set, applied to the input, and the output is calculated by equations 3.12 and 3.13. The weights are then updated by

$$\Delta \mathbf{w} = \eta(t - u)\frac{\mathbf{x}}{\|\mathbf{x}\|^2}, \tag{3.34}$$

where η is a small positive constant learning rate. Note that this minimizes the difference between the target and the weighted input sum u, not the output $y = f(u)$. Weight adjustments are repeated with new patterns until the total error (summed over all training patterns) is minimized.

The error is reduced by a factor of η each time the weights are updated with the input pattern fixed. Stability [406] requires that $0 < \eta < 2$ and generally $0.1 < \eta < 1.0$. For $\eta = 1$ the error on the present pattern is completely corrected in one cycle; for $\eta > 1$, it is overcorrected. Because of the linearity of the rule, the error function is quadratic and has only one global minimum; there are no local minima.

If the input patterns are linearly independent, the weights will converge to unique values. If not, the corrections will be oscillatory and η should decrease over time to allow the weights to settle. One possible schedule is $\eta = k^{-1}$, where k indexes the iterations [212].

Vectors $\mathbf{a}$ and $\mathbf{b}$ are *orthogonal* if $\mathbf{a}^T\mathbf{b} = 0$. A set of vectors $\{\mathbf{v}_i\}_{i=1}^n$ are *orthonormal* if any two distinct vectors are orthogonal and each has unit length. For orthonormal input patterns, the optimal weights to which the rule converges iteratively can be computed directly [359]

$$\mathbf{w} = \sum_{p=1}^{P} t_p \mathbf{x}_p. \tag{3.35}$$

Here p indexes training patterns, not vector elements; $\mathbf{x}_p$ is the p-th input pattern, a column vector, and t_p is the p-th output target. This is essentially Hebbian learning. Because the patterns are orthogonal, stored patterns don't interfere with each other so recall is perfect

$$t_p = \mathbf{w}^T \mathbf{x}_p. \tag{3.36}$$

Orthonormality is not, however, a requirement for perfect recall of the stored patterns. The correct output will be produced in response to every stored pattern as long as the patterns are merely linearly independent. Unlike the perceptron algorithm that does not converge when the patterns are not linearly separable, the delta rule converges to the optimal mean-squared-error solution. When the patterns are linearly separable, the minimum MSE solution may separate the classes, but this is not guaranteed. There are linearly separable data sets that a MSE solution does not separate correctly.

3.5.1 Adaline Capacity

An Adaline can store and recall perfectly up to N linearly independent patterns. This is limited by the dimension of the input since N patterns can be linearly independent only if $N \leq d$, where d is the dimension of the patterns (the number of inputs including the bias unit).

For random input patterns and targets, the capacity approaches $2d$ [401]. This is an average capacity for random inputs and targets and is determined by the results discussed in section 3.3. But as few as four patterns can create a not-linearly-separable set that is unlearnable by the Adaline. In a sense, random data is the hardest to learn. In most real problems, however, patterns and targets are not random; the target usually depends on the inputs in a systematic way. When the patterns are linearly separable (or nearly so), the system is able to correctly classify many more than $2d$ patterns [401]. The case of two well-separated clusters is a simple example.

In simulations, it is reported that approximately $5d$ training passes are required to achieve a 20% error level [402], where d is the number of bits per pattern. This generalizes to approximately d/ϵ where ϵ is the acceptable error level. For linearly separable problems, it is sufficient to train only on the border patterns since all patterns farther from the border will be classified correctly if the border patterns are [401].