CHAPTER 11

Perceptron and Adaline

11.1 INTRODUCTION

This chapter describes single layer neural networks, including some of the classical approaches to the neural computing and learning problem. In the first part of this chapter we discuss the representational power of the single layer networks and their learning algorithms and will give some examples of using the networks. In the second part we will discuss the representational limitations of single layer networks.

Two 'classical' models will be described in the first part of the chapter: the Perceptron, proposed by Rosenblatt and the Adaline, presented by Widrow and Hoff.

11.2 NETWORKS WITH THRESHOLD ACTIVATION FUNCTIONS

A single layer feed-forward network consists of one or more output neurons *o*, each of which is connected with a weighting factor w_{io} to all of the inputs *i*. In the simplest case the network has only two inputs and a single output, as sketched in Fig. 11.1 (we leave the output index *o* out). The input of the neuron is the weighted sum of the inputs plus the bias term. The output of the network is formed by the activation of the output neuron, which is some function of the input:

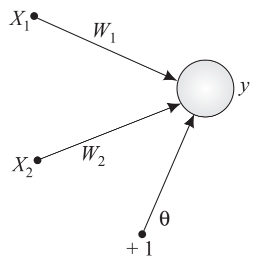

Fig. 11.1 Single layer network with one output and two inputs.

$$y = F\left(\sum_{i=1}^{2} w_i x_i + \theta\right) \qquad \text{...(11.1)}$$

The activation function F can be linear so that we have a linear network, or non-linear. In this section we consider the threshold (sgn) function:

$$F(s) = \begin{cases} +1 & \text{if } s > 0 \\ -1 & \text{otherwise} \end{cases} \qquad \text{...(11.2)}$$

The output of the network thus is either +1 or –1, depending on the input. The network can now be used for a classification task: it can decide whether an input pattern belongs to one of two classes. If the total input is positive, the pattern will be assigned to class +1, if the total input is negative, the sample will be assigned to class +1. The separation between the two classes in this case is a straight line, given by the equation:

$$w_1 x_1 + w_2 x_2 + \theta = 0 \qquad \text{...(11.3)}$$

The single layer network represents a linear discriminant function.

A geometrical representation of the linear threshold neural network is given in Fig. 11.2. Equation (11.3) can be written as

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{\theta}{w_2} \qquad \text{...(11.4)}$$

and we see that the weights determine the slope of the line and the bias determines the 'offset', i.e. how far the line is from the origin. Note that also the weights can be plotted in the input space: the weight

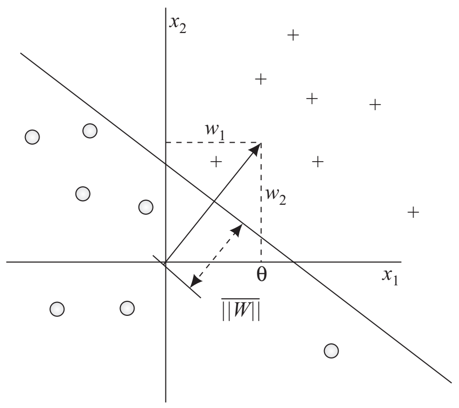

Fig. 11.2 Geometric representation of the discriminant function and the weights.

vector is always perpendicular to the discriminant function.

Now that we have shown the representational power of the single layer network with linear threshold units, we come to the second issue: how do we learn the weights and biases in the network? We will describe two learning methods for these types of networks: the 'perceptron' learning rule and the `delta' or `LMS' rule. Both methods are iterative procedures that adjust the weights. A learning sample is presented to the network. For each weight the new value is computed by adding a correction to the old value. The threshold is updated in a same way:

$$w_i(t+1) = w_i(t) + \Delta w_i(\text{t}) \qquad \text{...(11.5)}$$

$$\theta(t+1) = \theta(t) + \Delta\theta(t) \qquad \text{...(11.6)}$$

The learning problem can now be formulated as: how do we compute $\Delta w_i(t)$ and $\Delta\theta(t)$ in order to classify the larning patterns correctly?

11.3 PERCEPTRON LEARNING RULE AND CONVERGENCE THEOREM

11.3.1 Perceptron Learning Rule

Suppose we have a set of learning samples consisting of an input vector x and a desired output $d(x)$. For a classification task the $d(x)$ is usually +1 or –1. The perceptron learning rule is very simple and can be stated as follows:

1. Start with random weights for the connections;
2. Select an input vector x from the set of training samples;
3. If $y \neq d(x)$ (the perceptron gives an incorrect response), modify all connections w_i according to: $\Delta w_i = d(x)x_i$;
4. Go back to 2.

Note that the procedure is very similar to the Hebb rule; the only difference is that, when the network responds correctly, no connection weights are modified. Besides modifying the weights, we must also modify the threshold θ. This θ is considered as a connection w_0 between the output neuron and a 'dummy' predicate unit which is always on: $x_0 = 1$. Given the perceptron learning rule as stated above, this threshold is modified according to:

$$\Delta\theta = \begin{cases} 0 & \text{if the perceptron responds correctly} \\ d(x) & \text{otherwise} \end{cases} \qquad \text{...(11.7)}$$

11.3.2 Convergence Theorem

For the learning rule there exists a convergence theorem, which states the following: ***"If there exists a set of connection weights w* which is able to perform the transformation y = d(x), the perceptron learning rule will converge to some solution (which may or may not be the same as w^*) in a finite number of steps for any initial choice of the weights".***

Proof: Given the fact that the length of the vector w^* does not play a role (because of the sgn operation), we take $\|w^*\| = 1$. Because w^* is a correct solution, the value $|w^* \text{ o } x|$, where o denotes dot or inner product, will be greater than 0 or: there exists a $\delta > 0$ such that $|w^* \text{ o } x| > \delta$ for all inputs x.

Now define $\cos \alpha = \dfrac{w \text{ o } w^*}{||w||}$.

When according to the perceptron learning rule, connection weights are modified at a given input x, we know that $\Delta w = d(x)x$, and the weight after modification is $w' = w + \Delta w$. From this it follows that:

$$\begin{aligned}
w' \text{ o } w^* &= w \text{ o } w^* + d(x) \text{ o } w^* \text{ o } x \\
&= w \text{ o } w^* + sgn(w^* \text{ o } x)\, w^* \text{ o } x \\
&> w \text{ o } w^* + \delta \\
||w'||^2 &= ||w + d(x)x||^2 \\
&= w^2 + 2d\,(x)\, w \text{ o } x + x^2 \\
&< w^2 + x^2 \qquad\qquad (\text{because } d(x) = -\text{sgn}\,[w \text{ o } x]) \\
&= w^2 + M
\end{aligned}$$

After t modifications we have:

$$\begin{aligned}
w(t) \text{ o } w^* &> w \text{ o } w^* + t\delta \\
||w(t)||^2 &< w^2 + tM
\end{aligned}$$

such that

$$\begin{aligned}
\cos \alpha(t) &= \frac{w^* \text{ o } w(t)}{||w(t)||} \\
&> \frac{w^* \text{ o } w + t\delta}{\sqrt{w^2 + tM}}
\end{aligned}$$

From this follows that

$$\lim_{t\to\infty} \cos \alpha(t) = \lim_{t\to\infty} \frac{\delta}{\sqrt{M}} \sqrt{t} = \infty \text{ while } \cos \alpha \le 1.$$

The conclusion is that there must be an upper limit $t_{\max}$ for t. the system modifies its connections only a limited number of times. In other words, after maximally $t_{\max}$ modifications of the weights the perceptron is correctly performing the mapping. $t_{\max}$ will be reached when $\cos \alpha = 1$. If we start with connections $w = 0$,

$$t_{\max} = \frac{M}{\delta^2} \qquad ...(11.8)$$

Example 11.1: A perceptron is initialized with the following weights: $w_1 = 1$; $w_2 = 2$; $\theta = -2$. The perceptron learning rule is used to learn a correct discriminant function for a number of samples, sketched in Fig. 11.3.

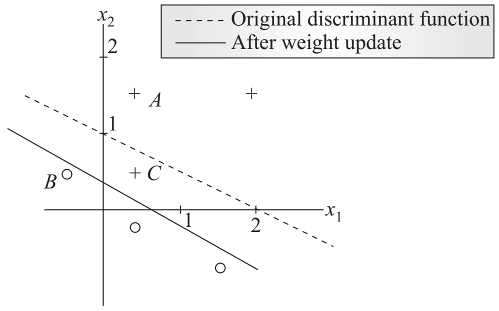

Fig. 11.3 Discriminant function before and after weight update.

The first sample A, with values $x = (0{:}5;\ 1{:}5)$ and target value $d(x) = +1$ is presented to the network. From equation (11.1) it can be calculated that the network output is +1, so no weights are adjusted. The same is the case for point *B*, with values $x = (-0{:}5;\ 0{:}5)$ and target value $d(x) = -1$; the network output is negative, so no change. When presenting point *C* with values $x = (0{:}5;\ 0{:}5)$ the network output will be –1, while the target value $d(x) = +1$.

According to the perceptron learning rule, the weight changes are:

$$\Delta w_1 = 0{:}5,\ \Delta w_2 = 0{:}5,\ \theta = 1.$$

The new weights are now:

$$w_1 = 1{:}5,\ w_2 = 2{:}5,\ \theta = -1,$$

and sample *C* is classified correctly.

In Fig. 11.3 the discriminant function before and after this weight update is shown.

11.4 ADAPTIVE LINEAR ELEMENT (Adaline)

An important generalisation of the perceptron training algorithm was presented by Widrow and Hoff as the 'least mean square' (LMS) learning procedure, also known as the delta rule. The main functional di erence with the perceptron training rule is the way the output of the system is used in the learning rule. The perceptron learning rule uses the output of the threshold function (either –1 or +1) for learning. The delta-rule uses the net output without further mapping into output values –1 or +1.

The learning rule was applied to the 'adaptive linear element', also named Adaline, developed by Widrow and Hoff. In a simple physical implementation (Fig. 11.4) this device consists of a set of controllable resistors connected to a circuit, which can sum up currents caused by the input voltage signals. Usually the central block, the summer, is also followed by a quantiser, which outputs either +1 or –1, depending on the polarity of the sum.

Although the adaptive process is here exemplified in a case when there is only one output, it may be clear that a system with many parallel outputs is directly implementable by multiple units of the above kind.

If the input conductances are denoted by w_i, $i = 0; 1,..., n$, and the input and output signals by x_i and y, respectively, then the output of the central block is defined to be

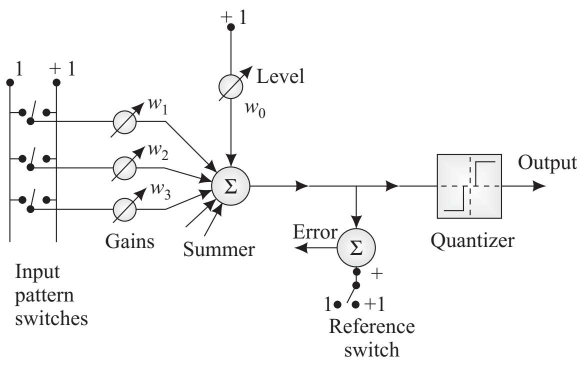

Fig. 11.4 The adaline.

$$y = \sum_{i=1}^{n} w_i x_i + \theta \qquad \text{...(11.9)}$$

where $\theta = w_0$. The purpose of this device is to yield a given value $y = d^p$ at its output when the set of values x_i^p, $i = 1; 2, \ldots, n$, is applied at the inputs. The problem is to determine the coeficients w_i, $i = 0, 1, \ldots, n$, in such a way that the input-output response is correct for a large number of arbitrarily chosen signal sets. If an exact mapping is not possible, the average error must be minimised, for instance, in the sense of least squares. An adaptive operation means that there exists a mechanism by which the w_i can be adjusted, usually iteratively, to attain the correct values. For the Adaline, Widrow introduced the delta rule to adjust the weights.

11.5 THE DELTA RULE

For a single layer network with an output unit with a linear activation function the output is simply given by

$$y = \sum_{j} w_j x_j + \theta \qquad \text{...(11.10)}$$

Such a simple network is able to represent a linear relationship between the value of the output unit and the value of the input units. By thresholding the output value, a classifier can be constructed (such as Adaline), but here we focus on the linear relationship and use the network for a function approximation task. In high dimensional input spaces the network represents a (hyper) plane and it will be clear that also multiple output units may be defined.

Suppose we want to train the network such that a hyperplane is fitted as well as possible to a set of training samples consisting of input values x^p and desired (or target) output values d^p. For every given input sample, the output of the network differs from the target value d^p by $(d^p - y^p)$, where y^p is the actual output for this pattern. The delta-rule now uses a cost-or error-function based on these differences to adjust the weights.

The error function, as indicated by the name least mean square, is the summed squared error. That is, the total error E is defined to be

$$E = \sum_p E^p = \frac{1}{2}\sum_p (d^p - y^p)^2 \qquad ...(11.11)$$

where the index p ranges over the set of input patterns and E^p represents the error on pattern p. The LMS procedure finds the values of all the weights that minimize the error function by a method called gradient descent. The idea is to make a change in the weight proportional to the negative of the derivative of the error as measured on the current pattern with respect to each weight:

$$\Delta_p w_j = -\gamma \frac{\partial E^p}{\partial w_j} \qquad ...(11.12)$$

where γ is a constant of proportionality. The derivative is

$$\frac{\partial E^p}{\partial w_j} = \frac{\partial E^p}{\partial y^p}\,\frac{\partial y^p}{\partial w_j}. \qquad ...(11.13)$$

Because of the linear units, eq. (11.10),

$$\frac{\partial y^p}{\partial w_j} = x_j \qquad ...(11.14)$$

and

$$\frac{\partial E^p}{\partial y^p} = -(d^p - y^p) \qquad ...(11.15)$$

such that

$$\Delta_p w_j = \gamma \delta^p x_j \qquad ...(11.16)$$

where $\delta^p = d^p - y^p$ is the difference between the target output and the actual output for pattern p.

The delta rule modifies weight appropriately for target and actual outputs of either polarity and for both continuous and binary input and output units. These characteristics have opened up a wealth of new applications.

11.6 EXCLUSIVE-OR PROBLEM

In the previous sections we have discussed two learning algorithms for single layer networks, but we have not discussed the limitations on the representation of these networks.

Table 11.1 Exclusive or truth table.

x_0	x_1	d
−1	−1	−1
−1	1	1
1	−1	1
1	1	1

One of Minsky and Papert's most discouraging results shows that a single layer perceptron cannot represent a simple exclusive-or function. Table 3.1 shows the desired relationships between inputs and output units for this function.

In a simple network with two inputs and one output, as depicted in Fig. 11.1, the net input is equal to:

$$s = w_1x_1 + w_2x_2 + \theta \qquad ...(11.17)$$

According to eq. (11.1), the output of the perceptron is zero when s is negative and equal to one when s is positive. In Fig. 11.5 a geometrical representation of the input domain is given. For a constant θ, the output of the perceptron is equal to one on one side of the dividing line which is defined by:

$$w_1x_1 + w_2x_2 = -\theta \qquad ...(11.18)$$

and equal to zero on the other side of this line.

To see that such a solution cannot be found, take a loot at Fig. 11.5. The input space consists of four points, and the two solid circles at (1, –1) and (–1, 1) cannot be separated by a straight line from the two open circles at (–1, –1) and (1, 1). The obvious question to ask is: How can this problem be overcome? Minsky and Papert prove that for binary inputs, any transformation can be carried out by adding a layer of predicates which are connected to all inputs. The proof is given in the next section.

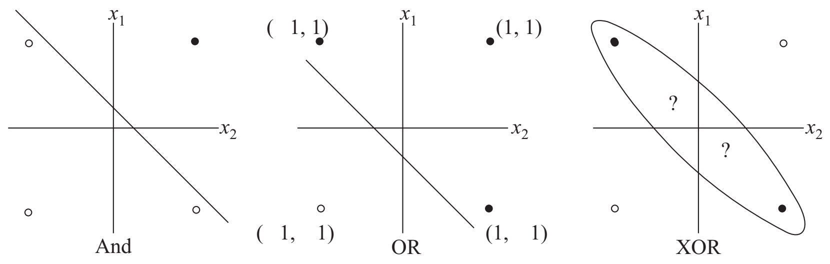

Fig. 11.5 Geometric representation of input space

For the specific *XOR* problem we geometrically show that by introducing hidden units, thereby extending the network to a multi-layer perceptron, the problem can be solved. Fig. 11.6a demonstrates that the four input points are now embedded in a three-dimensional space defined by the two inputs plus the single hidden unit. These four points are now easily separated by a linear manifold (plane) into two groups, as desired. This simple example demonstrates that adding hidden units increases the class of problems that are soluble by feed-forward, perceptron- like networks. However, by this generalization of the basic architecture we have also incurred a serious loss: we no longer have a learning rule to determine the optimal weights.

(a) The perceptron of Fig. 11.1 with an extra hidden unit. With the indicated values of the weights w_{ij} (next to the connecting lines) and the thresholds θ_i (in the circles) this perceptron solves the *XOR* problem. (b) This is accomplished by mapping the four points of Fig. 11.6 onto the four points indicated here; clearly, separation (by a linear manifold) into the required groups is now possible.

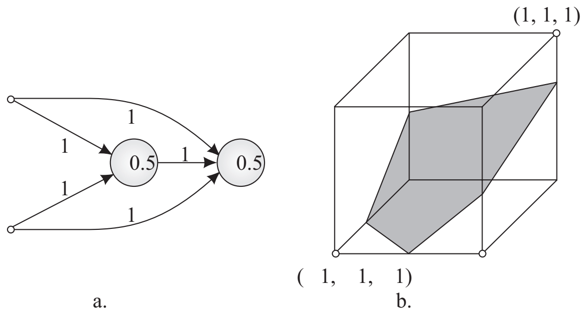

Fig. 11.7 Solution of the XOR problem.

11.7 MULTI-LAYER PERCEPTRONS CAN DO EVERYTHING

In the previous section we showed that by adding an extra hidden unit, the *XOR* problem can be solved. For binary units, one can prove that this architecture is able to perform any transformation given the correct connections and weights. The most primitive is the next one. For a given transformation $y = d(x)$, we can divide the set of all possible input vectors into two classes:

$$X^{+} = \{x|d(x) = 1\} \text{ and } X \;\; = \{x|d(x) - 1\} \qquad ...(11.19)$$

Since there are N input units, the total number of possible input vectors x is $2N$. For every $x^p \in X^+$ a hidden unit h can be reserved of which the activation y_h is 1 if and only if the specific pattern p is present at the input: we can choose its weights w_{ih} equal to the specific pattern x^p and the bias θ_h equal to 1 - N such that

$$y_h^p = \operatorname{sgn}\left(\sum_i w_{ih} x_i^p - N + \frac{1}{2}\right) \qquad ...(11.20)$$

is equal to 1 for $x^p = w_h$ only. Similarly, the weights to the output neuron can be chosen such that the output is one as soon as one of the M predicate neurons is one:

$$y_o^p = \operatorname{sgn}\left(\sum_{h=1}^{M} y_h + M - \frac{1}{2}\right) \qquad ...(11.21)$$

This perceptron will give $y_0 = 1$ only if $x \in X^+$: it performs the desired mapping. The problem is the large number of predicate units, which is equal to the number of patterns in X^+, which is maximally 2^N. Of course we can do the same trick for X , and we will always take the minimal number of mask units, which is maximally 2^{N-1}. A more elegant proof is given by Minsky and Papert, but the point is that for complex transformations the number of required units in the hidden layer is exponential in N.