5.5 Learning Objective for TLNs

Observe that a TLN (redrawn for convenience in Fig. 5.8) is actually a linear neuron whose output is directed into a unit step or signum function. The neuron is adaptive when its weights are allowed to change in accordance with a well defined learning law. Widrow called this neuron an *adaptive linear element* or Adaline [592]. Commonly used adaptive algorithms for such threshold logic neurons are the Perceptron learning algorithm and the least-mean-square (LMS) algorithm [594]. These will be the subject of our study in this chapter.

Recall from Chapter 4 that the TLN is a basic building block for more complicated networks. Threshold logic neurons when layered, can generate arbitrarily complex decision regions.

Rosenblatt's α–Perceptron model [486] employed fixed, sparse, and random connections from the input layer of neurons to an intermediate layer of binary threshold neurons. These connections effectively mapped the analog input space into a Boolean space using a fixed transformation. The output neuron activations were set-up in the usual way—by taking an inner product of the binary vector with an adaptive input weight vector. The output neuron signal was binary (or bipolar). Other than the fact that its inputs were restricted to be binary, and that it had no bias weight, the α–Perceptron was operationally equivalent to the standard binary threshold neuron (TLN).

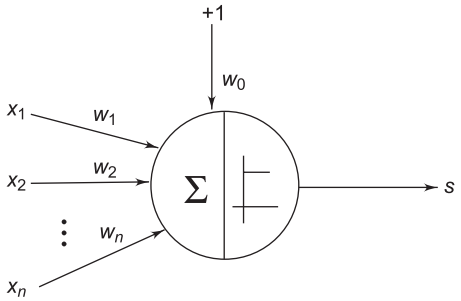

Fig. 5.8 A model for the generic binary threshold logic neuron

As before, the neuron input and weight vectors are assumed to be in augmented form to accommodate the neuron bias as an adjustable weight:

$$X_k = \left(x_0, x_1^k, \ldots, x_n^k\right)^T \quad X_k \in \mathbb{R}^{n+1} \tag{5.1}$$

$$W_k = \left(w_0^k, \ldots, w_n^k\right)^T \quad W_k \in \mathbb{R}^{n+1} \tag{5.2}$$

X_k is the pattern presented at iteration k; W_k is the neuron weight vector at iteration k. Note that $x_0 = 1$ always, and therefore does not carry an iteration index.

where the iteration index k has been introduced to explicitly indicate the temporal nature of these vectors. The neuronal activation, $y_k = X_k^T W_k$, determines whether the neuron fires a 1 or a 0 signal in accordance with the threshold neuronal signal function:

$$\mathcal{S}(y_k) = \begin{cases} 1 & y_k > 0 \\ 0 & y_k < 0 \end{cases} \tag{5.3}$$

In the forthcoming discussion we assume that the condition $y_k = 0$ translates to an ambiguous signal value, and therefore treat this activation condition as a misclassification. Weights will be designed to avoid such ambiguous signal values.

Let us now put our broad objective in place. We wish to design the weights of a TLN to correctly classify a given set of patterns. For this, we assume we are given a training set $\mathcal{T} = \{X_k, d_k\}_{k=1}^{Q}$, $X_k \in \mathbb{R}^{n+1}$, $d_k \in \{0, 1\}$ where each pattern X_k is tagged to one of two classes $\mathcal{C}_0$ or $\mathcal{C}_1$ denoted by the

desired output d_k being 0 or 1 respectively. Note that since this is a two class classification problem the training patterns can be divided into two subsets $\mathcal{X}_0, \mathcal{X}_1$ respectively comprising patterns that belong to $\mathcal{C}_0, \mathcal{C}_1$. In the discussion of the Perceptron and Pocket learning algorithms, we will see that the use of the desired values will become implicit. Since our focus will be on the input patterns explicitly, we will refer to $\mathcal{X} = \mathcal{X}_0 \cup \mathcal{X}_1$ as the training set rather than using the symbol $\mathcal{T}$ which includes desired values also. The two classes will be identified by the two possible signal states of the TLN—$\mathcal{C}_0$ patterns by a signal $\mathcal{S}(y_k) = 0$, and $\mathcal{C}_1$ patterns by a signal $\mathcal{S}(y_k) = 1$.

Recall that one can design such simple Boolean classifiers by hand—through the suitable placement of the separating hyperplane (that represents the neuronal discriminant function). Pattern points are to be separated into two categories by suitable design of a discriminant function. However, such design techniques are well suited to problems of low dimensionality, and prove to be of no use when the classification problem is in higher dimensions. There is thus the need to design weight update procedures or learning algorithms that can automatically search out a weight space solution that can solve the classification problem.

The basic objective is to design an incremental weight update procedure that can search for a solution vector that defines a discriminant function which provides a clear cut separation of data points in $\mathcal{C}_0$ from those in $\mathcal{C}_1$.

The broad objective of the present exercise is to design a weight adjustment procedure for TLN weights such that, given two sets of vectors $\mathcal{X}_0$ and $\mathcal{X}_1$ belonging to classes $\mathcal{C}_0$ and $\mathcal{C}_1$ respectively, it searches a solution weight vector W_S, that correctly classifies the vectors into their respective classes. In the context of TLNs this translates to saying: Find a weight vector W_S such that for all $X_k \in \mathcal{X}_1$, $\mathcal{S}(y_k) = 1$; and for all $X_k \in \mathcal{X}_0$, $\mathcal{S}(y_k) = 0$. Now since positive inner products translate to a +1 signal and negative inner products to a 0 signal, the above requirement translates to saying that for all $X_k \in \mathcal{X}_1$, $X_k^T W_S > 0$; and for all $X_k \in \mathcal{X}_0$, $X_k^T W_S < 0$.

We learnt in Chapter 3 that a single threshold logic neuron can solve such a classification problem only for pattern sets that are linearly separable. If the pattern sets are not linearly separable we cannot hope to solve the problem using a single neuron, and more complicated networks will be required. For the present discussion, we therefore assume that the pattern sets $\mathcal{X}_0$ and $\mathcal{X}_1$ are linearly separable. This assumption guarantees the existence of a separating hyperplane which is represented by the weight vector W_S of the neuron. With this guarantee in place, the next question is: how do we find W_S ?