# 2 FUNDAMENTALS OF NEURAL NETWORKS

*Ali Zilouchian*

## 2.1 INTRODUCTION

For many decades, it has been a goal of science and engineering to develop intelligent machines with a large number of simple elements. References to this subject can be found in the scientific literature of the 19th century. During the 1940s, researchers desiring to duplicate the function of the human brain, have developed simple hardware (and later software) models of biological neurons and their interaction systems. McCulloch and Pitts [1] published the first systematic study of the artificial neural network. Four years later, the same authors explored network paradigms for pattern recognition using a single layer perceptron [2]. In the 1950s and 1960s, a group of researchers combined these biological and psychological insights to produce the first artificial neural network (ANN) [3,4]. Initially implemented as electronic circuits, they were later converted into a more flexible medium of computer simulation. However, researchers such as Minsky and Papert [5] later challenged these works. They strongly believed that intelligence systems are essentially symbol processing of the kind readily modeled on the Von Neumann computer. For a variety of reasons, the symbolic–processing approach became the dominant method. Moreover, the perceptron as proposed by Rosenblatt turned out to be more limited than first expected. [4]. Although further investigations in ANN continued during the 1970s by several pioneer researchers such as Grossberg, Kohonen, Widrow, and others, their works received relatively less attention. The primary factors for the recent resurgence of interest in the area of neural networks are the extension of Rosenblatt, Widrow and Hoff's works dealing with learning in a complex, multi-layer network, Hopfield mathematical foundation for understanding the dynamics of an important class of networks, as well as much faster computers than those of 50s and 60s.

The interest in neural networks comes from the networks' ability to mimic human brain as well as its ability to learn and respond. As a result, neural networks have been used in a large number of applications and have proven to be effective in performing complex functions in a variety of fields. These include pattern recognition, classification, vision, control systems, and prediction [6], [7]. Adaptation or learning is a major focus of neural net research that provides a degree of robustness to the NN model. In predictive modeling, the goal is to map a set of input patterns onto a set of output patterns. NN accomplishes this task by learning from a series of input/output data sets presented to the network. The trained network is then used to apply what it has learned to approximate or predict the corresponding output [8].

This chapter is organized as follows. In section 2.2, various elements of an artificial neural network are described. The Adaptive Linear Element (ADALINE) and single layer perceptron are discussed in section 2.3 and 2.4 respectively. The multi-layer perceptron is presented in section 2.5. Section 2.6 discusses multi-layer perceptron and section 2.7 concludes this chapter.

## 2.2 BASIC STRUCTURE OF A NEURON

### 2.2.1 Model of Biological Neurons

In general, the human nervous system is a very complex neural network. The brain is the central element of the human nervous system, consisting of near $10^{10}$ biological neurons that are connected to each other through sub-networks. Each neuron in the brain is composed of a body, one axon and multitude of dendrites. The neuron model shown in Figure 2.1 serves as the basis for the artificial neuron. The dendrites receive signals from other neurons. The axon can be considered as a long tube, which divides into branches terminating in little endbulbs. The small gap between an endbulb and a dendrite is called a synapse. The axon of a single neuron forms synaptic connections with many other neurons. Depending upon the type of neuron, the number of synapses connections from other neurons may range from a few hundreds to $10^4$.

The cell body of a neuron sums the incoming signals from dendrites as well as the signals from numerous synapses on its surface. A particular neuron will send an impulse to its axon if sufficient input signals are received to stimulate the neuron to its threshold level. However, if the inputs do not reach the required threshold, the input will quickly decay and will not generate any action. The biological neuron model is the foundation of an artificial neuron as will be described in detail in the next section.

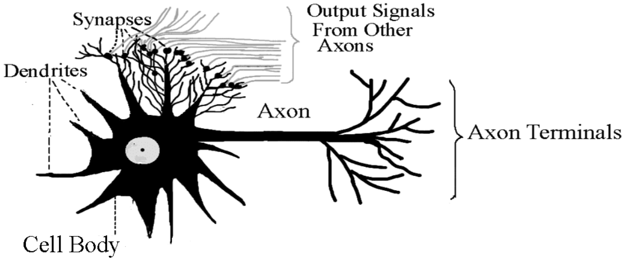

**Figure 2.1:** A Biological Neuron.

### 2.2.2 Elements of Neural Networks

An artificial neuron as shown in Figure 2.2, is the basic element of a neural network. It consists of three basic components that include weights, thresholds, and a single activation function.

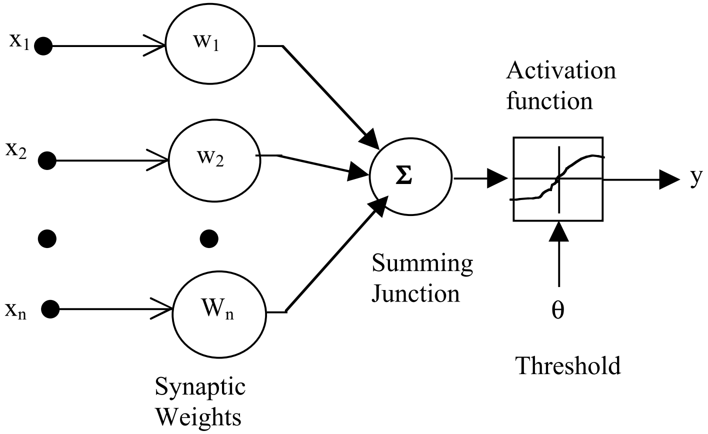

**Figure 2.2:** Basic Elements of an Artificial Neuron.

*2.2.2.1 Weighting Factors*

The values $W_1$ ,$W_2$ ,$W_3$ ,…..,$W_n$ are weight factors associated with each node to determine the strength of input row vector $X = [x_1\ x_2\ x_3.....,x_n]^T$. Each input is multiplied by the associated weight of the neuron connection $X^TW$. Depending upon the activation function, if the weight is positive, $X^TW$ commonly excites the node output; whereas, for negative weights, $X^TW$ tends to inhibit the node output.

*2.2.2.2 Threshold*

The node's internal threshold $\theta$ is the magnitude offset that affects the activation of the node output y as follows:

$$y = \sum_{i=1}^{n} (X_i W_i) - \theta_k \qquad (2.1)$$

*2.2.2.3 Activation Function*

In this subsection, five of the most common activation functions are presented. An activation function performs a mathematical operation on the signal output. More sophisticated activation functions can also be utilized depending upon the type of problem to be solved by the network. All the activation functions as described herein are also supported by MATLAB package.

*Linear Function*

As is known, a linear function satisfies the superposition concept. The function is shown in Figure 2.3.

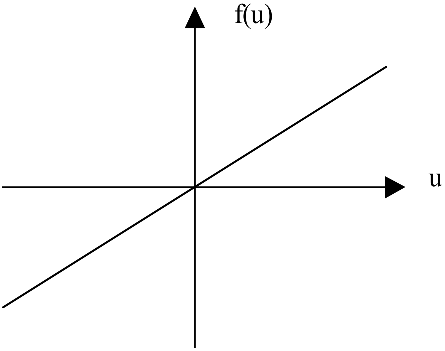

**Figure 2.3:** Linear Activation Function.

The mathematical equation for the above linear function can be written as

$$y = f(u) = a.u \tag{2.2}$$

where α is the slope of the linear function 2.2. If the slope α is 1, then the linear activation function is called the identity function. The output (y) of identity function is equal to input function (u). Although this function might appear to be a trivial case, nevertheless it is very useful in some cases such as the last stage of a multilayer neural network.

*Threshold Function*

A threshold (hard-limiter) activation function is either a *binary* type or a *bipolar* type as shown in Figures 2.4 and 2.5, respectively. The output of a *binary* threshold function can be written as:

$$y = f(u) = \begin{cases} 0 & if \quad u < 0 \\ 1 & if \quad u \geq 0 \end{cases} \tag{2.3}$$

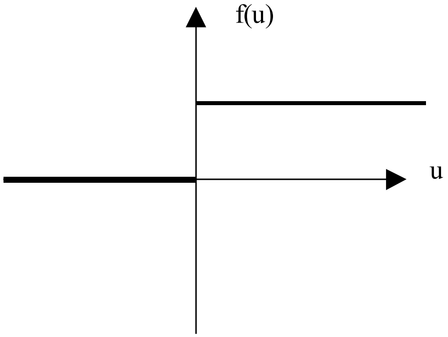

**Figure 2.4:** Binary Threshold Activation Function.

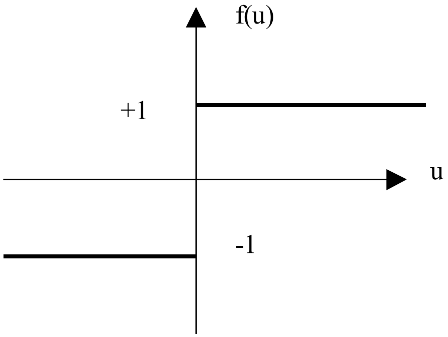

**Figure 2.5:** Bipolar Threshold Activation Function.

The neuron with the hard limiter activation function is referred to as the McCulloch-Pitts model.

*Piecewise Linear Function*

This type of activation function is also referred to as saturating linear function and can have either a binary or bipolar range for the saturation limits of the output. The mathematical model for a symmetric saturation function (Figure 2.6) is described as follows:

$$y = f(u) = \begin{cases} -1 & if \quad u < -1 \\ u & if \quad -1 \geq u \geq 1 \\ 1 & if \quad u \geq 1 \end{cases} \tag{2.4}$$

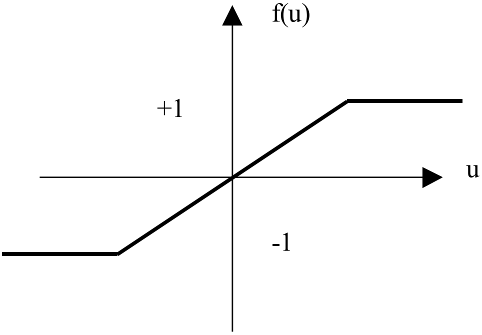

**Figure 2.6:** Piecewise Linear Activation Function.

S*igmoidal (S shaped) function*

This nonlinear function is the most common type of the activation used to construct the neural networks. It is mathematically well behaved, differentiable and strictly increasing function. A sigmoidal transfer function can be written in the following form:

$$f(x) = \frac{1}{1 + e^{-\alpha x}}\ ,\ 0 \le f(x) \le 1 \qquad (2.5)$$

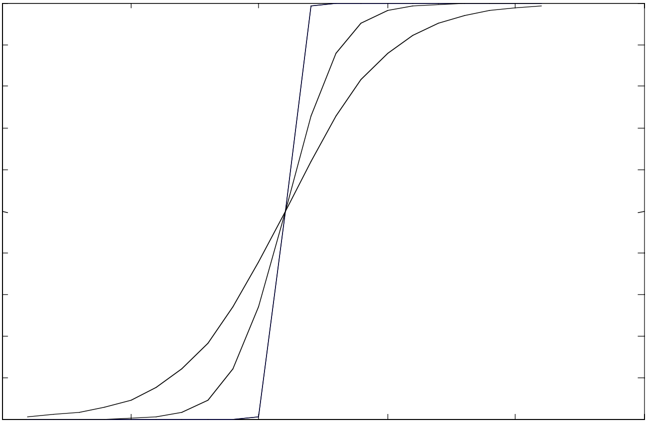

**Figure 2.7:** A Sigmoid Activation Function.

where α is the shape parameter of the sigmoid function. By varying this parameter, different shapes of the function can be obtained as illustrated in Figure 2.7. This function is continuous and differentiable.

*Tangent hyperbolic function*

This transfer function is described by the following mathematical form:

$$f(x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}} \quad -1 \le f(x) \le 1 \qquad (2.6)$$

It is interesting to note that the derivatives of Equations 2.5 and 2.6 can be expressed in terms of the individual function itself (please see problems appendix). This is important for the learning development rules to train the networks as shown in the next chapter.

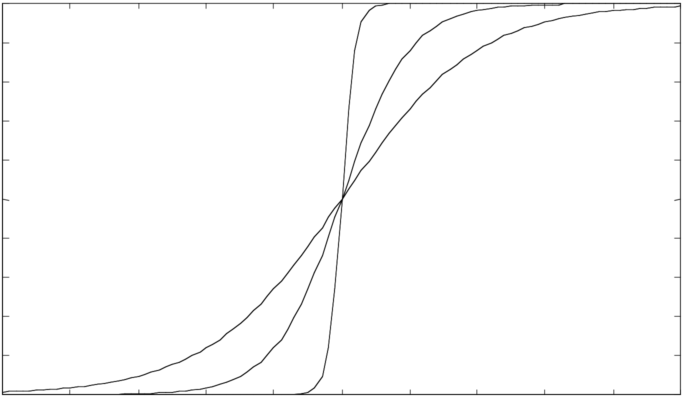

**Figure 2.8:** A Tangent Hyperbolic Activation Function.

**Example 2.1:**

Consider the following network consists of four inputs with the weights as shown

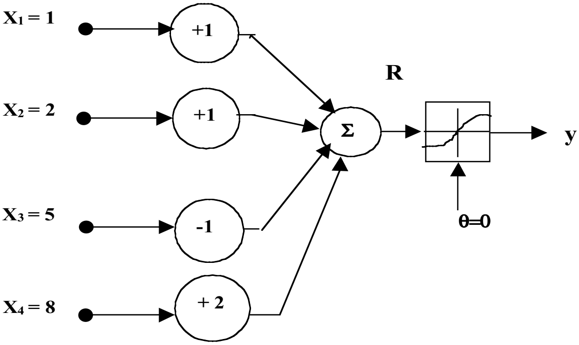

**Figure 2.9:** Neuron Structure of Example 2.1.

The output R of the network, prior to the activation function stage, is calculated as follows:

$$R = W^T.X = \begin{bmatrix} 1 & 1 & -1 & 2 \end{bmatrix}.\begin{bmatrix} 1 \\ 2 \\ 5 \\ 8 \end{bmatrix} = 14 \tag{2.7}$$

With a binary activation function, and a sigmoid function, the outputs of the neuron are respectively as follow:

$$\mathbf{y}(\mathit{Threshold}) = 1;$$

$$\mathbf{y}(\mathit{Sigmoid}) = 1.5*2^{-8}$$

## 2.3 ADALINE

An ADAptive LINear Element (ADALINE) consists of a single neuron of the McCulloch-Pitts type, where its weights are determined by the normalized least mean square (LMS) training law. The LMS learning algorithm was originally proposed by Widrow and Hoff [6]. This learning rule is also referred to as delta rule. It is a well-established supervised training method that has been used over a wide range of diverse applications [7]- [11]. Curve fitting approximations can also be used for training a neural network [10]. The learning objective of curve fitting is to find a surface that best fits to the training data. In the next chapter the implementation of LMS algorithms for backpropagation, and curve fitting algorithms for radial basis function network, will be described in detail.

The architecture of a simple ADALINE is shown In Figure 2.10. It is observed that the basic structure of an ADALINE is similar to a linear neuron (Figure 2.2) with the activation function f(.) to be a linear one with an extra feedback loop. Since ADALINE is a linear device, any combination of these units can be accomplished with the use of a single unit.

During the training phase of ADALINE, the input vector $X \in R^n$: $X = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \end{bmatrix}^T$ as well as desired output are presented to the network. The weights are adaptively adjusted based on delta rule. After the ADALINE is trained, an input vector presented to the network with fixed weights will result in a scalar output. Therefore, the network performs a mapping of an *n* dimensional mapping to a scalar value. The activation function is not used during the training phase. Once the weights are properly adjusted, the response of the trained unit can be tested by applying various inputs, which are not in the training set. If the network produces consistent responses to a high degree with the test inputs, it said that the network could *generalize*. Therefore, the process of training and generalization are two important attributes of the network.

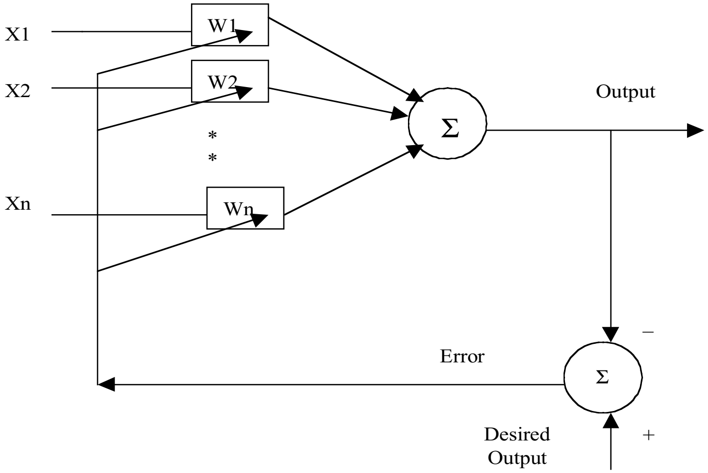

**Figure 2.10:** ADALINE.

In practice, an ADALINE is usually used to make binary decisions. Therefore, the output is sent through a binary threshold as shown in Figure 2.4. Realizations of several logic gates such as AND, NOT and OR are common applications of ADALINE. Only those logic functions that are linearly separable can be realized by the ADALINE, as is explained in the next section.

## 2.4 LINEAR SEPARABLE PATTERNS

For a single ADALINE to function properly as a classifier, the input pattern must be linearly separable. This implies that the patterns to be classified must be sufficiently apart from each other to ensure the decision surface consists of a single hyperplane such as a single straight line in two-dimensional space. This concept is illustrated in Figure 2.11 for a two-dimensional pattern.

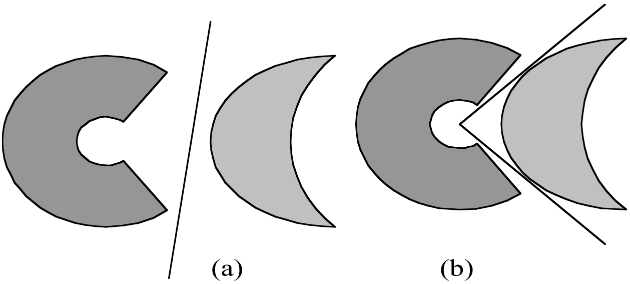

**Figure 2.11:** A Pair of Linearly Separable (a), and Non-Linearly Separable Patterns (b).

A classic example of a mapping that is not separable is XOR (the exclusive or) gate function. Table 2.1 shows the input-output pattern of this problem. Figure 2.12 shows the locations of the symbolic outputs of XOR function corresponding to four input patterns in X1-X2 plane. There is no way to draw a single straight line so that the circles are on one side of the line and the triangular sign on the other side. Therefore, an ADALINE cannot realize this function.

**Table 2.1:** Inputs/Outputs Relationship for XOR.

| *X1* | *X2* | *Output* |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

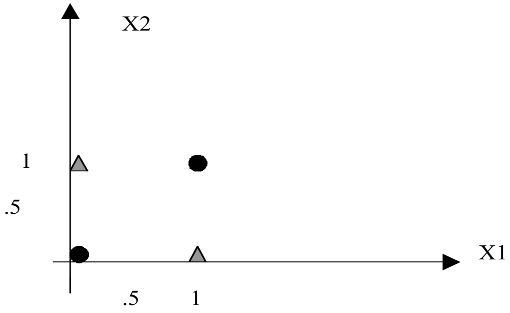

**Figure 2.12:** The Output of XOR in X1-X2 Plane.

One approach to solve this nonlinear separation problem is to use MADALINE (Multiple ADALINE) networks. The basic structure of a MADALINE network consists of combining several ADALINE with their correspondence activation functions into a single forward structure. When suitable weights are chosen, the network is capable of implementing complicated and nonlinear separable mapping such as XOR gate problems. We will address this issue later in this chapter.

## 2.5 SINGLE LAYER PERCEPTRON

### 2.5.1 General Architecture

The original idea of the perceptron was developed by Rosenblatt in the late 1950s along with a convergence procedure to adjust the weights. In Rosenblatt's perceptron, the inputs were binary and no bias was included. It was based on the McCulloch-Pitts model of the neuron with the hard limitation activation function. The single layer perceptron as shown in Figure 2.13 is very similar to ADALINE except for the addition of an activation function.

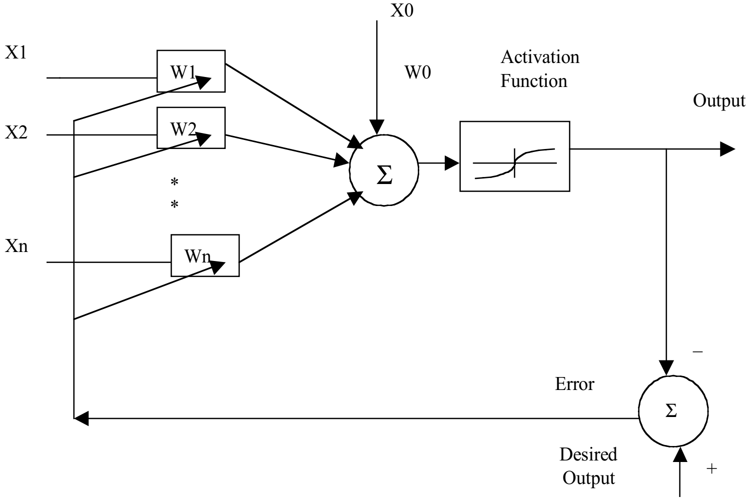

**Figure 2.13:** A Perceptron with a Sigmoid Activation Function.

Connection weights and threshold in a perceptron can be fixed or adapted using a number of different algorithms. Here the original perceptron convergence procedure as developed by Minsky and Papert[5] is described. First, connection weights $W_1$, $W_2$,…,$W_n$ and the threshold value $W_0$ are initialized to small non-zero values. Then, a new input set with N values received through sensory units (measurement devices) and the input is computed. Connection weights are only adapted when an error occurs. This procedure is repeated until the classification of all inputs is completed.

### 2.5.2 Linear Classification

For clarification of the above concept, consider two input patterns classes C1 and C2. The weight adaptation at the kth training phase can be formulated as follow:

1. If k member of the training vector x(k) is correctly classified, no correction action is needed for the weight vector. Since the activation function is selected as a hard limiter, the following conditions will be valid:

   $W(k+1)=W(k)$ *if* $output>0$ *and* $x(k)\in C1$ *, and*

   $W(k+1)=W(k)$ *if* $output<0$ *and* $x(k)\in C2.$

2. Otherwise, the weight should be updated in accordance with the following rule:

   $W(k+1)=W(k)+\eta\, x(k)$ *if* $output\geq 0$ *and* $x(k)\varepsilon\, C1$

   $W(k+1)=W(k)-\eta\, x(k)$ *if* $output\leq 0$ *and* $x(k)\varepsilon\, C2$

Where $\eta$ is the learning rate parameter, which should be selected between 0 and 1.

**Example 2.2:**

Let us consider pattern classes C1 and C2, where C1: {(0,2), (0,1)} and C2: {(1,0), (1,1)}. The objective is to obtain a decision surface based on perceptron learning. The 2-D graph for the above data is shown in Figure 2.14

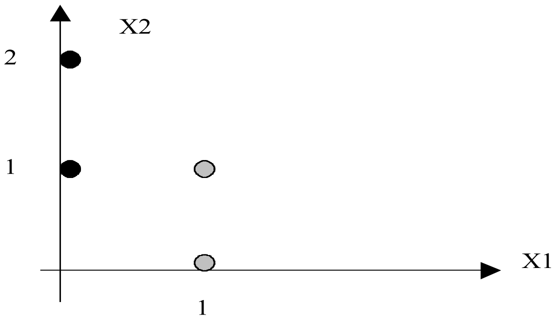

**Figure 2.14:** 2-D Plot of Input Data Sets for Example 2.2.

Since, the input vectors consist of two elements , the perceptron structure is simply as follows:

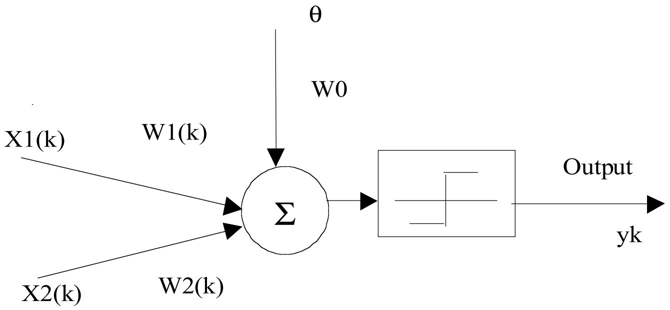

**Figure 2.15:** Perceptron Structure for Example 2.2.

For simplicity, let us assume η=1 and initial weight vector W(1)=[0 0]. The iteration weights are as follow:

*Iteration 1*: $W^T(1).x(1) = \begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}0\\2\end{bmatrix} = 0$

*Weight Update*: $W(2) = W(1) + x(1) = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0\\2\end{bmatrix} = \begin{bmatrix}0\\2\end{bmatrix}$

*Iteration 2*: $W^T(2).x(2) = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 2 > 0$

*Weight Update*: $W(3) = W(2)$

*Iteration 3:* $W^T(3).x(3) = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$

*Weight Update*: $W(4) = W(3) - x(3) = \begin{bmatrix} 0 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

*Iteration 4:* $W^T(4).x(4) = \begin{bmatrix} -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1$

*Weight Update:* $W(5) = W(4) - x(4) = \begin{bmatrix} -1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$

Now if we continue the procedure, the perceptron classifies the two classes correctly at each instance. For example for the fifth and sixth iterations:

*Iteration 5:* $W^T(5).x(5) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 2 > 0$:*Correct Classification*

*Iteration 6:* $W^T(6).x(6) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 > 0$:*Correct Classification*

In a similar fashion for the seventh and eighth iterations, the classification results are indeed correct.

*Iteration 7:* $W^T(7).x(7) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -2 < 0$:*Correct Classification*

*Iteration 8:* $W^T(8).x(8) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -1 < 0$:*Correct Classification*

Therefore, the algorithm converges and the decision surface for the above perceptron is as follows:

$$d(x) = -2X_1 + X_2 = 0 \tag{2.8}$$

Now, let us consider the input data {1,2}, which is not in the training set. If we calculate the output:

$$Y = W^T.X = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = -3 < 0 \tag{2.9}$$

The output Y belongs to the class C2 as is expected.

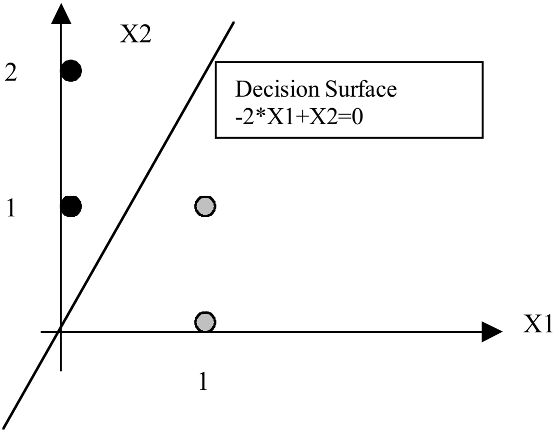

**Figure 2.16:** Decision Surface for Example 2.2.

### 2.5.3 Perceptron Algorithm

The perceptron learning algorithm (Delta rule) can be summarized as follows:

**Step 1**: Initialize the weights $W_1, W_2 \ldots W_n$ and threshold θ to small random values.
**Step 2**: Present new input X1, X2,..Xn and desired output $d_k$.
**Step 3**: Calculate the actual output based on the following formula:

$$y_k = f_h\left(\sum_{i=1}^{n} (X_i Wi) - \theta_k\right) \tag{2.10}$$

**Step 4**: Adapt the weights according to the following equation:

$$W_i(new) = W_i(old) + \eta(d_k - y_k)x_i, 0 \leq i \leq N \tag{2.11}$$

Where η is a positive gain fraction less than 1 and $d_k$ is the desired output. Note that the weights remain the same if the network makes the correct decision.
**Step 5**: Repeat the procedures in steps 2–4 until the classification task is completed.

Similar to ADALINE, if the presented inputs pattern is linearly separable, then the above perceptron algorithm converges and positions the decision hyperplane between two separate classes. On the other hand, if the inputs are not separable and their distribution overlaps, then the decision boundary may oscillate continuously. A modification to the perceptron convergence procedure is the utilization of Least Mean Square (LMS) in this case. The algorithm that forms the LMS solution is also called the Widrow-Hoff. The LMS algorithm is similar to the procedure above except a threshold logic nonlinearity, replaces the hard limited non-linearity. Weights are thus corrected on every trail by an amount that depends on the difference between the desired and actual values. Unlike the learning in the ADALINE, the perceptron learning rule has been shown to be capable of separating any linear separable set of the training patterns.

## 2.6 MULTI-LAYER PERCEPTRON

### 2.6.1 General Architecture

Multi-layer perceptrons represent a generalization of the single-layer perceptron as described in the previous section. A single layer perceptron forms a half–plane decision region. On the other hand multi-layer perceptrons can form arbitrarily complex decision regions and can separate various input patterns. The capability of multi-layer perceptron stems from the non-linearities used within the nodes. If the nodes were linear elements, then a single-layer network with appropriate weight could be used instead of two- or three-layer perceptrons. Figure 2.17 shows a typical multi-layer perceptron neural network structure. As observed it consists of the following layers:

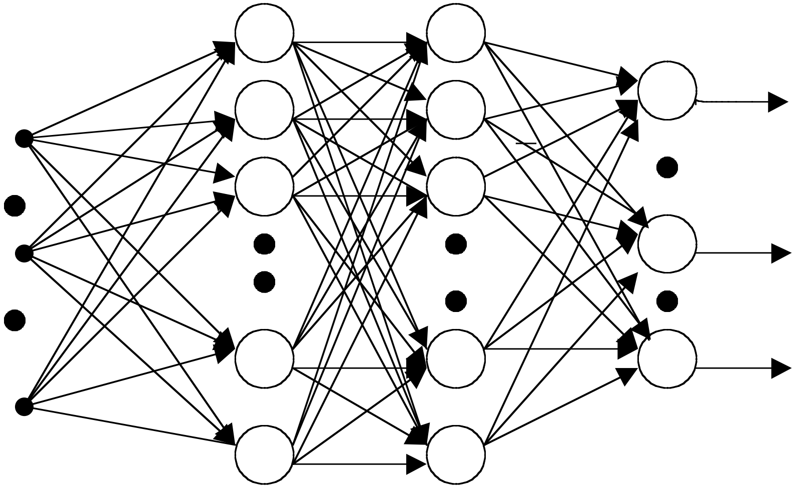

**Figure 2.17:** Multi-layer Perceptron.

*Input Layer*: A layer of neurons that receives information from external sources, and passes this information to the network for processing. These may be either sensory inputs or signals from other systems outside the one being modeled.

*Hidden Layer*: A layer of neurons that receives information from the input layer and processes them in a hidden way. It has no direct connections to the outside world (inputs or outputs). All connections from the hidden layer are to other layers within the system.

*Output Layer*: A layer of neurons that receives processed information and sends output signals out of the system.

*Bias*: Acts on a neuron like an offset. The function of the bias is to provide a threshold for the activation of neurons. The bias input is connected to each of the hidden and output neurons in a network.

### 2.6.2 Input-Output Mapping

The input/output mapping of a network is established according to the weights and the activation functions of their neurons in input, hidden and output layers. The number of input neurons corresponds to the number of input variables in the neural network, and the number of output neurons is the same as the number of desired output variables. The number of neurons in the hidden layer(s) depends upon the particular NN application. For example, consider the following two-layer feed-forward network with three neurons in the hidden layer and two neurons in the second layer:

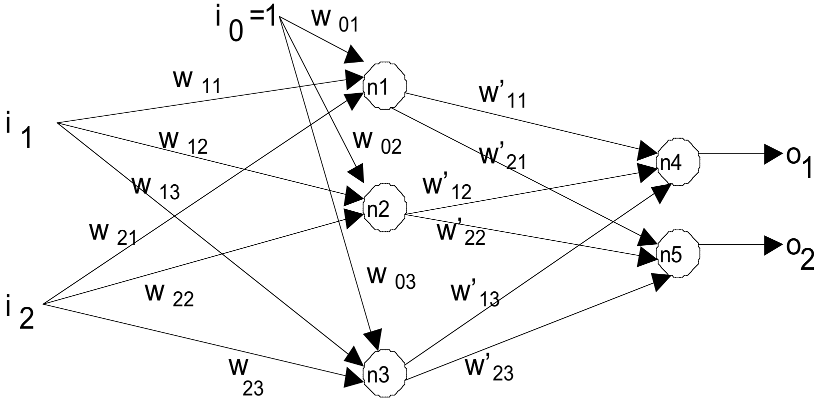

**Figure 2.18:** An Example of Multi-layer Perceptron.

As is shown, the inputs are connected to each neuron in hidden layer via their corresponding weights. A zero weight indicates no connection. For example, if $W_{23} = 0$, it is implied that no connection exists between the second input ($i_2$) and the third neuron ($n_3$). Outputs of the last layer are considered as the outputs of the network.

The structure of each neuron within a layer is similar to the architecture as described in section 2.5. Although the activation function for one neuron could be different from other neurons within a layer, for structural simplicity, similar neurons are commonly chosen within a layer. The input data sets (or sensory information) are presented to the input layer. This layer is connected to the first hidden layer. If there is more than one hidden layer, the last hidden layer should be connected to the output layer of the network. At the first phase, we will have the following linear relationship for each layer:

$$A_1 = W_1 X \tag{2.12}$$

where $A_1$ is a column vector consisting of m elements, $W_1$ is an m×n weight matrix and X is a column input vector of dimension n. For the above example, the linear activity level of the hidden layer (neurons $n_1$ to $n_3$) can be calculated as follows:

$$\begin{cases} a_{11} = w_{11} i_1 + w_{21} i_2 \\ a_{12} = w_{12} i_1 + w_{22} i_2 \\ a_{13} = w_{13} i_1 + w_{23} i_2 \end{cases} \tag{2.13}$$

The output vector for the hidden layer can be calculated by the following formula:

$$O_1 = F \,. A_1 \tag{2.14}$$

where $A_1$ is defined in Equation 2.12, and $O_1$ is the output column vector of the hidden layer with m element. $F$ is a diagonal matrix comprising the non-linear activation functions of the first hidden layer:

$$F = \begin{bmatrix} f_1(.) & 0 & 0 & \ldots & 0 \\ 0 & f_2(.) & & & 0 \\ . & & .. & & .. \\ . & & & .. & 0 \\ 0 & 0 & \ldots & 0 & f_m(.) \end{bmatrix} \tag{2.15}$$

For example, if all activation functions for the neurons in the hidden layer of Figure 2.18 are chosen similarly, then the output of the neurons $n_1$ to $n_3$ can be calculated as follows:

$$\begin{cases} O_{11} = f(a_{11}) \\ O_{12} = f(a_{12}) \\ O_{13} = f(a_{13}) \end{cases} \tag{2.16}$$

In a similar manner, the output of other hidden layers can be computed. The output of a network with only one hidden layer according to Equation 2.14 is as follows:

$$A_2 = W_2 . O_1 \tag{2.17}$$

$$O_2 = G . A_2 \tag{2.18}$$

Where $A_2$ is the vector of activity levels of output layer and $O_2$ is the q output of the network. $G$ is a diagonal matrix consisting of nonlinear activation functions of the output layer:

$$G = \begin{bmatrix} g_1(.) & 0 & 0 & \ldots & 0 \\ 0 & g_2(.) & & & 0 \\ . & & .. & & .. \\ . & & & .. & 0 \\ 0 & 0 & \ldots & 0 & g_q(.) \end{bmatrix} \tag{2.19}$$

For Figure 2.18, the activity level of output neurons n $_4$ and n $_5$ can be calculated as follows:

$$\begin{cases} a_{21} = W'_{11} O_{11} + W'_{12} O_{21} + W'_{13} O_{31} \\ a_{22} = W'_{21} O_{11} + W'_{22} O_{21} + W'_{23} O_{31} \end{cases} \tag{2.20}$$

The two outputs of the network with the similar activation functions can be calculated as follows:

$$\begin{cases} O_1 = g(a_{21}) \\ O_2 = g(a_{22}) \end{cases} \tag{2.21}$$

Therefore, the input-output mapping of a multi-layer perceptron is established according to relationships 2.12–2.22. In sequel, the output of the network can be calculated using such nonlinear mapping and the input data sets.

### 2.6.3 XOR Realization

As it was shown in section 2.4, a single-layer perceptron cannot classify the input patterns that are not linearly separable such as an Exclusive OR (XOR) gate. This problem may be considered as a special case of a more general non-linear mapping problem. In the XOR problem, we need to consider the four corners of the unit square that correspond to the input pattern. We may solve the problem with a multi-layer perceptron with one hidden layer as shown in Figure 2.19.

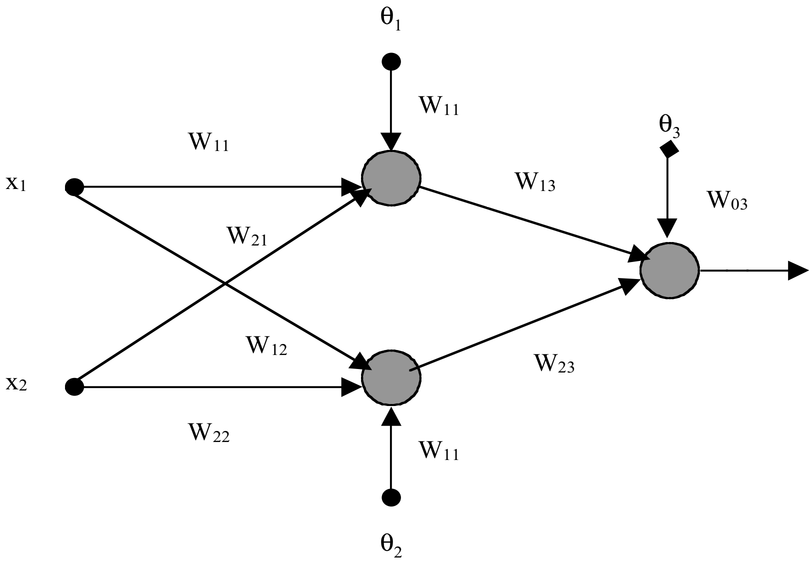

**Figure 2.19:** Neural Network Architecture to Solve XOR Problem.

In the above configuration, a McCulloh-Pitts model represents each neuron, which uses a hard limit activation function. By appropriate selections of the network weights, the XOR could be implemented using decision surfaces as shown in Figure 2.20.

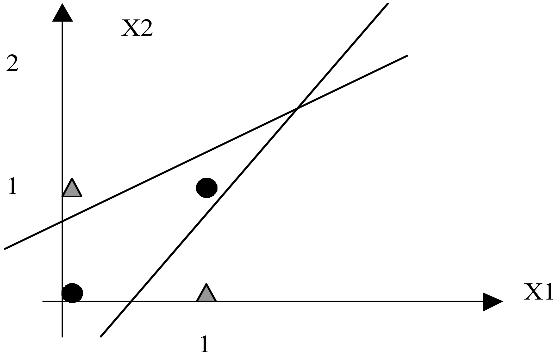

**Figure 2.20:** Decision Surfaces to Solve XOR Problem.

**Example 2.3**:

Suppose weights and biases are selected as shown in Figure 2.21. The McCulloh-Pitts model represents each neuron (binary hard limit activation function). Show that the network solves XOR problem. In addition, draw the decision boundaries constructed by the network.

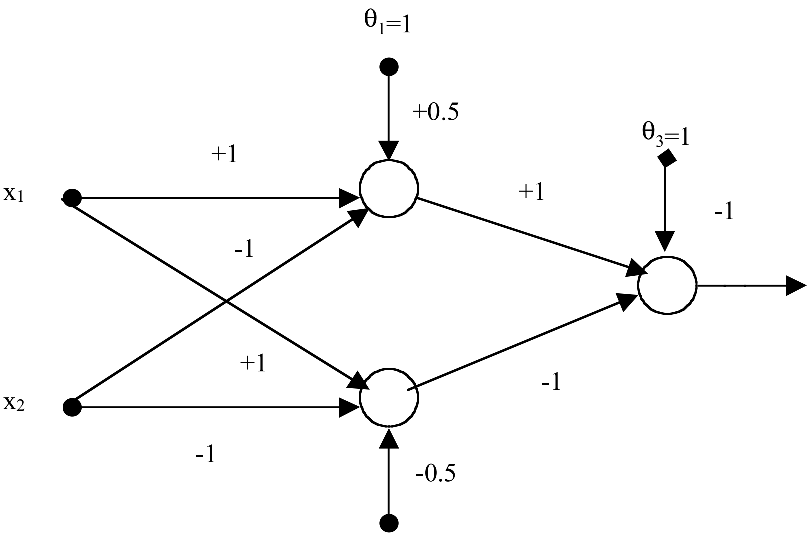

**Figure 2.21:** Neural Network Architecture for Example 2.3.

In Figure 2.21, suppose the outputs of neurons (before activation function) denote as $O_1$, $O_2$, and $O_3$. The outputs of the summing points at the first layer are as follow:

$$O_1 = x_1 - x_2 + 0.5 \tag{2.22}$$

$$O_2 = x_1 - x_2 - 0.5 \tag{2.23}$$

With the binary hard limited functions, the output $y_1$ and $y_2$ are shown in Figures 2.22 and 2.23.

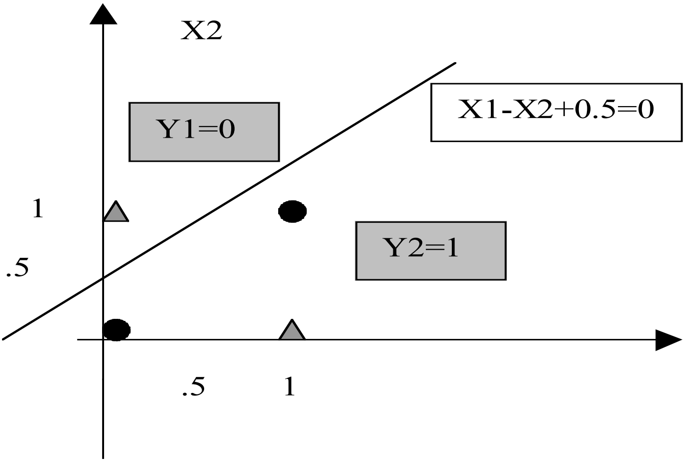

**Figure 2.22:** Decision Surface for Neuron 1 of Example 2.3.

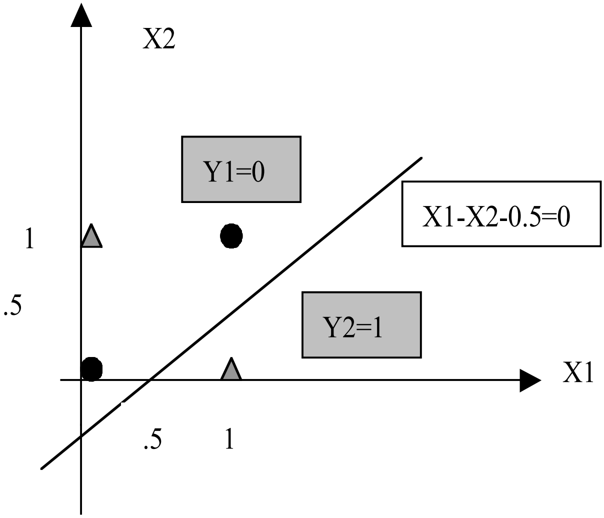

**Figure 2.23:** Decision Surface for Neuron 2 of Example 2.3.

The outputs of the summing points at the second layer are:

$$O_3 = y_1 - y_2 - 1 \tag{2.24}$$

The decision boundaries of the network are shown in Figure 2.24. Therefore, XOR realization can be accomplished by selection of appropriate weights using Figure 2.19.

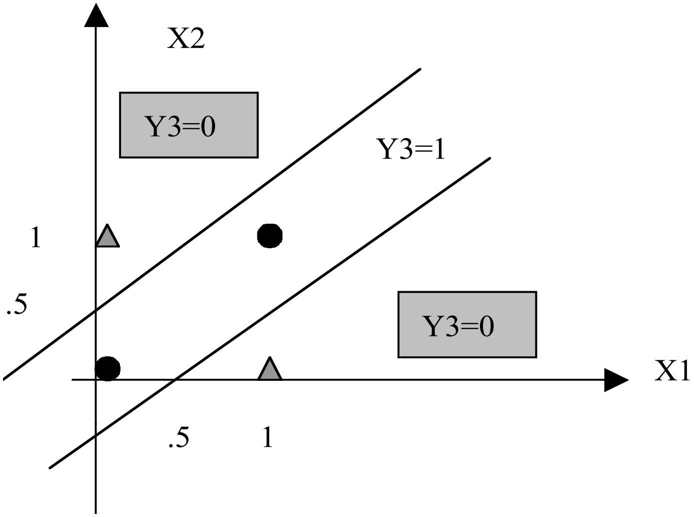

**Figure 2.24:** Decision Surfaces for Example 2.3.

## 2.7 CONCLUSION

In this chapter, the fundamentals of neural networks were introduced. The perceptron is the simplest form of neural network used for the classification of linearly separable patterns. Multi-layer perceptron overcome many limitations of single-layer perceptron. They can form arbitrarily complex decision regions in order to separate various nonlinear patterns. The next chapter is devoted to several neural network architectures. Applications of NN will be presented in Chapters 4–7 and Chapter 15 of the book.

## REFERENCES

1. McCulloch, W.W. and Pitts, W., A Logical Calculus of Ideas Imminent in Nervous Activity. *Bull. Math. Biophys.*, 5, 115–133, 1943.
2. Pitts, W. and McCulloch, W.W., How we Know Universals, Bull. *Math.* 127–147, 1947.
3. McClelland, J.L. and Rumelhart, D.E., *Parallel Distributed Processing -Explorations in the Microstructure of Cognition*, Vol. 2, *Psychological and Biological Models*, MIT Press, Cambridge, MA, 1986.
4. Rosenblatt, F., *Principles of Neurodynamics*, Spartan Press, Washington, DC, 1961.
5. Minsky, M. and Papert, S., *Perceptron*: *An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1969.
6. Widrow, B. and Hoff, M.E, Adaptive Switching Circuits, IRE WESCON Convention Record, Part 4, NY, IRE, 96–104, 1960.
7. Fausett, L., *Fundamentals of Neural Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
8. Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, 1999.
9. Kosko, B., *Neural Network for Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1992.
10. Ham, F. and Kostanic, I., *Principles of Neurocomputing for Science and Engineering*, McGraw Hill, New York, NY, 2001.
11. Lippmann, R.P., An Introduction to Computing with Neural Network, *IEEE Acoustic, Speech, and Sig. Proces. Mag.*, 4, 1987.