Chapter 3

Neural Network Fundamentals

This chapter provides fundamental principles of neural networks. The emphasis is on multilayer feedforward networks. Neural network fundamentals explained consists of four sections. Section 3.1 briefs the origin and development of neural networks. Types of neural networks classified by their structures and their learning paradigms are described in section 3.2. Multilayer feedforward network architecture and functions of a neuron are given in section 3.3 and backpropagation algorithm is provided in the final section.

3.1 Origin and Development of Neural Networks

Artificial neural networks are mathematical structures having the ability to learn from examples presented. Neural networks acquired their name from their similarity to the highly connected structure of the human nervous system. The neural network paradigm emerged from attempts to simulate and understand the working of the human brain. The human brain is composed of networks of neurons. There are about 10^{10} neurons in the brain and each neurons is randomly connected to approximately 10^{4} other neurons. Today 's neural network models are only simplified structures and in no way similar to the complexities of the human brain.

The early developments in the field of neural networks occurred in the 1940s. McCulloch and Pitts (1943) proposed the model of a simple neuron, which seemed appropriate for modeling symbolic logic, perceptron, and behavior. The McCulloch-Pitts neuron is a simple unit having a linear activation function with a threshold value to produce an output. Later in the sixties, Rosenblatt (1959) proposed the perceptron and demonstrated that perceptrons can generalize and learn. The perceptrons consisted of neuron-like processing units with linear thresholds, and were arranged in layers similar to biological systems. They used the Hebbian learning rule for training. This rule reinforces active connections only - weights are increased when the outputs are active and decreased when the outputs are inactive. Later the Adaline, or ADAptive LInear NEuron, was invented by Widrow and Hoff. This is not a network but a single neuron. Adaline used a different learning method called the delta rule. The Adaline 's method of learning is supervised learning, in which the neuron was given a target value. Adaline uses this target value to calculate the prediction error and moves the weight values in the direction of the negative gradient of the error. Still the Adaline is a linear neuron (having a linear transfer function) and is limited to learning linear separable classes. Minsky and Papert (1969) studied these perceptrons in detail and concluded that perceptrons are incapable of solving even simple problems such as the XOR problem, which is linearly inseparable. This dampened the neural network research for some years.

Interests in connectionist systems resurged with the discovery of the "backpropagation method" of training multilayer neural networks. The basic neuron model was changed by including a nonlinear activation function instead of the simple thresholding function and the neurons were arranged in many layers. Multilayer perceptrons with their nonlinear activation functions were capable of solving nonlinear problems that were previously impossible to solve using simple perceptrons. The training method, backpropagation, was developed independently by a number of researchers (Werbos, 1974; Rumelhart, Hinton, and William, 1986b, etc). The Parallel Distributed Processing research group published their research work results about neural networks (for example, see Rumelhart, Hinton, and William, 1986a), which made the interests in neural network research became active again. Backpropagation is based on the generalization of the delta rule used in Adaline. Since a number of variants of backpropagation, and other improved algorithms have been reported for the efficient training of multilayer neural networks. One such method is based on the conjugate gradient algorithm (Leonard and Kramer, 1990).

According to the review of the neural network applications in this decade, multilayer feedforward networks have been popularly applied to most researches as presented in Chapter 2. However researchers have attempted to develop new types of neural networks as well as the optimization techniques for training the networks to improve the efficiency of the neural network applications. Although many types of neural networks and their training algorithms have been improved and developed, they are based on the basis as described the next section.

3.2 Types of Neural Networks

Neural networks generally consist of a number of interconnected processing elements or neurons. How the inter-neuron connections are arranged and the nature of the connections determines the structure of a network. How the strengths of the connections are adjusted or trained to achieve a desired overall behavior of the network is governed by its learning algorithms.

The structure of interconnection and the learning algorithms employed by any neural network are generally lumped together as the *paradigm* (the model or pattern) of the network. The selected structures and the learning algorithms are related and are chosen by the theoretician or experimenter to implement a particular paradigm.

3.2.1 Structural Categorization

In terms of their structures, neural networks can be divided into two types: feedforward networks and feedback networks.

1) Feedforward Networks

In a feedforward network, the neurons are generally grouped into layers. Signals flow from the input layer through to the output layer via unidirectional connections, the neurons being connected from one layer to the next, but not within the same layer. Examples of feedforward networks include the multilayer perceptron (MLP) or multilayer feedforward network (Rumelhart and McClelland, 1986), the Learning Vector Quantization (LVQ) (Kohonen, 1989) and the Group Method of Data Handling (GMDH) network (Hecht-Nielsen, 1990). Feedforward networks can most naturally perform static mapping between an input space and an output space: the output at a given instant is a function only of the input at that instant.

2) Feedback Networks

In a feedback network or a recurrent network, the outputs of some neurons are fed back to the same neurons or to neurons in preceding layers. Thus, signals can flow in both forward and backward directions. Examples of recurrent networks include the Hopfield network (Hopfield, 1982), the Elman network (Elman, 1990) and the Jordan network (Jordan, 1986). Recurrent networks have a dynamic memory: their outputs at a given instant reflect the current input as well as previous inputs and outputs.

3.2.2 Learning Algorithm Categorization

Neural networks are trained rather than programmed. "Training" or "Learning" means modifying the values of the weights in the interconnections to achieve some target criteria for the output layer of nodes. Information is stored and distributed throughout the network via the interconnection weights.

Many learning rules have been developed, but there is a common feature in those learning rules. Therefore, the learning methods can be grouped into two types: supervised and unsupervised learning algorithm.

1) Supervised Learning

A supervised learning algorithm adjusts the strengths or weights of the inter-neuron connections according to the difference between the desired and actual network outputs corresponding to a given input. Thus, supervised learning requires a "teacher" or "supervisor" to provide desired or target output signals. Examples of supervised learning algorithms include the delta rule (Widrow and Hoff, 1960), the generalized delta rule or backpropagation algorithm (Rumelhart and McClelland, 1986) and the LVQ algorithm (Kohonen, 1989). *Reinforcement learning*, a special case of supervised learning, is also included in this type of learning. Instead of giving the correct output to the network, the network is only told whether the produced output is good or bad (correct or incorrect). Thus, the feedback output is only qualitative. An example of a reinforcement learning is the genetic algorithm (GA) (Holland, 1975; Goldberg, 1989).

2) Unsupervised Learning

Unsupervised learning algorithms do not require the desired outputs to be known. Therefore, the only available information is in the correlation among the input data. The network develops its own classification by extracting the correlation of the input data, and produces an output corresponding to the input category much like Cluster Analysis except that metric relationships can be maintained. During training, only input patterns are presented to the neural network, which automatically adapts the weights of its connection to cluster the input patterns into groups with similar features. Examples of unsupervised learning algorithm include the Kohonen (Kohonen, 1989) and Carpenter Grossberg Adaptive Resonance Theory (ART) (Carpenter and Grossberg, 1988) competitive learning algorithms.

Multilayer feed forward networks with one hidden layer can approximate any nonlinear function (Hornik, Stinchcombe, and White, 1989). Consequently, they are used in this work while the error backpropagation is utilized to train the neural networks.

3.3 Multilayer Feedforward Networks

The feedforward networks, in which the signals flow only from input to output. The mapping relationship between input and output vectors may be *static*, where each application of a given input vector always produces the same output vector, or it may be a *dynamic*, where the output produced depends upon previous, as well as current, inputs and/or outputs. Since feedforward networks have no memory, they are only capable of implementing static mappings. Adding feedback allows the network to produce dynamic mappings. Feedforward network architecture and functions of a neuron are described respectively as follows.

3.3.1 Feedforward Network Architecture

A general structure of a feedforward neural network is shown in Figure 3.1. Processing units, or neurons are arranged in layer. The network consists of three layers: the input, the hidden, and the output layer, respectively. The neurons in the input layer take in the independent variables and the neurons in the output layer compute the dependent variables. Every neuron is connected to every other neuron in the subsequent layer. The hidden layer neurons are "hidden" from the outside world- they are connected with the input layer on one side and with the output layer on the other. In addition to these, there are bias neurons with a constant input of unity that connect to neurons in the hidden and output layers, as shown by the square boxes in Figure 3.1.

Each connection is associated with a parameter called "the connection weight". The weights represent the strength of the connections and can have either negative or positive values. The exact value of these weights are determined as a result of the learning procedure. If there are n_i, n_h, and n_o neurons in the input, hidden, and output layers respectively, then the total number of parameters in a neural network model, including the bias weights are (n_w),

$$n_w = n_i n_h + n_h + n_h n_o + n_o \tag{3.1}$$

In the normal operation the network gets a set of independent variables in the input layer. The values are fed forward through the hidden layer to the output layer. The neurons in the output layer predict the values of the dependent variables. The computations involved in the neurons are described in the following section.

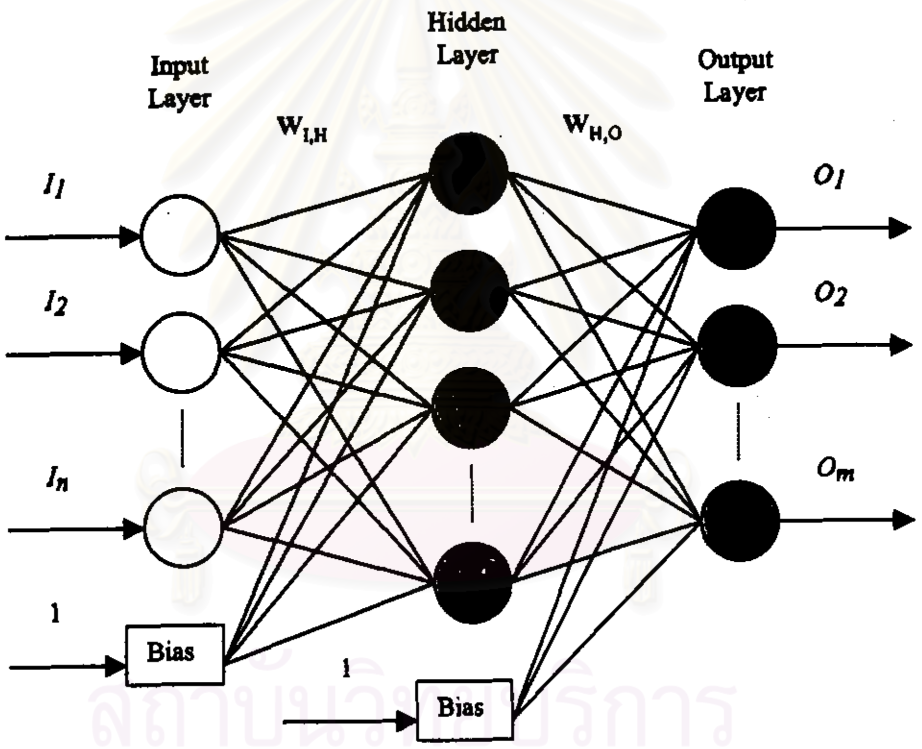

Figure 3.1: General structure of feedforward network with one hidden layer.

$W_{I,H}$ represents the vector of weights between input and hidden layers; $W_{H,O}$ represents the vector of weights between hidden and output layers; I, H, and O represent the input vector, hidden layer output vector, and output layer output vector respectively.

3.3.2 Functions of a Neuron

Neurons are processing elements (PEs) and all the computations of the network are done in these units as illustrated in Figure 3.2. The neurons in the input layer are not associated with any computations. They just act as distribution nodes and receive the values of the independent variables to pass it on to the hidden layer.

The hidden and output layer neurons can have different activation functions. However it is convenient and most common to have the same function in all the neurons in the hidden and output layers. Each neuron receives input from all the neurons in the previous layer. Considering a single neuron in the hidden or output layer, the calculations involved in that particular neuron may be represented as follows: I_1, I_2, ..., I_n, are the inputs to the neuron from the previous layer, and w_1, w_2, ..., w_n, are their respective weights, and b is the bias neuron weight, define S as,

$$S = \sum_{i=1}^{n} w_i I_i + bZ \tag{3.2}$$

If a sigmoid neuron transfer function (see Figure 3.3) is used, then,

$$O = \sigma(S) = \frac{1}{1 + \exp(-S)} \tag{3.3}$$

or, if a hyperbolic tangent function is used, then,

$$O = \sigma(S) = \tanh(S) \tag{3.4}$$

where S is the weighted sum of the inputs, O is the output from the neuron, and Z is the constant input to the bias neuron. The value of Z is usually fixed at unity. Other nonlinear functions can also be used as the activations, as long as they are differentiable and bounded.

The inputs to each neuron can be thought of as lying in a hyperspace of n dimensions. The neuron draws a hyperplane in that hyperspace. If the neuron produces binary outputs, the two outputs (say 0 and 1) lie on either side of the hyperplane. The actual weights determine where exactly this hyperplane is located in the hyperspace of inputs. When sigmoid functions are used, the neurons produce intermediate values between 0 and 1 and not just the binary outputs. If the bias weight is not present, then it is equivalent to constraining the hyperplane to pass through the origin of the hyperspace. Also, all the neurons in the same layer share the same input space. Without the presence of bias weights, the hyperplanes formed by all these neurons are constrained to pass through the same origin. The presence of bias values offers a threshold for the neurons, and provides the freedom of hyperplanes moving away from the origin.

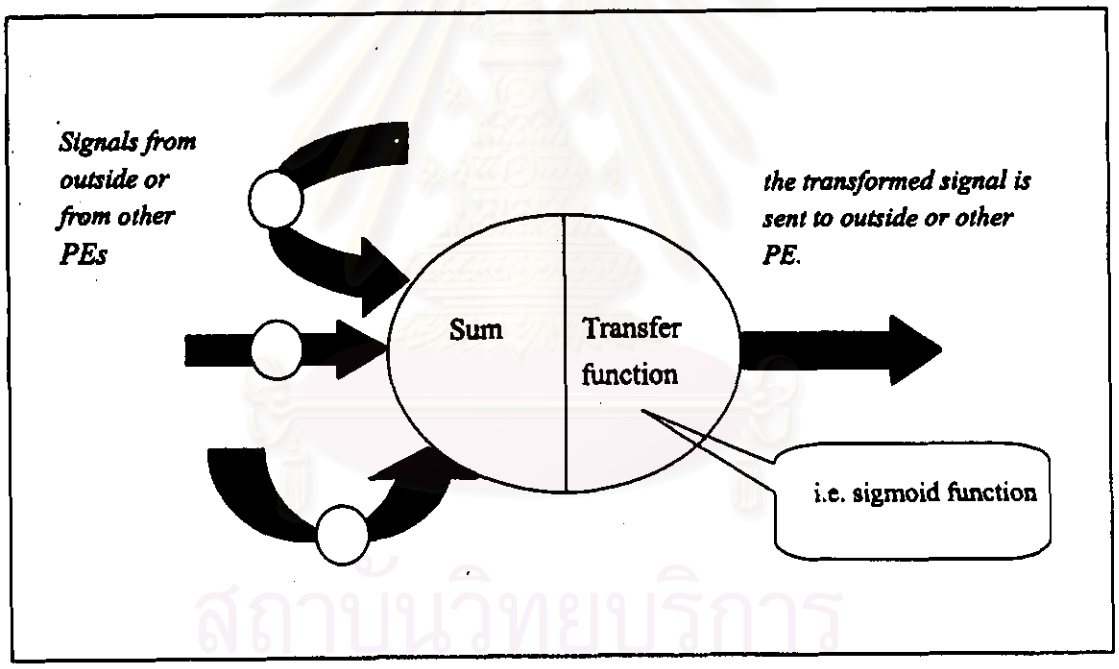
Signals from outside or from other PEs
Sum
Transfer function
the transformed signal is sent to outside or other PE.
i.e. sigmoid function

Figure 3.2: Functions of a neuron.

The outputs of the neurons in the hidden layer form the inputs to the neurons in the output layer. The outputs of the output layer neurons are the dependent variables that are predicted by the neural network. One hidden layer is sufficient to approximate any nonlinear continuous function. For most problems, presently there are no theoretical guidelines to choose the number of neurons in the hidden layer. Determination of the number of hidden layer neurons is a very important issue. Few guidelines exist for certain types of problems (Zurada, 1992, p. 216), however in general it is a trial and error procedure.

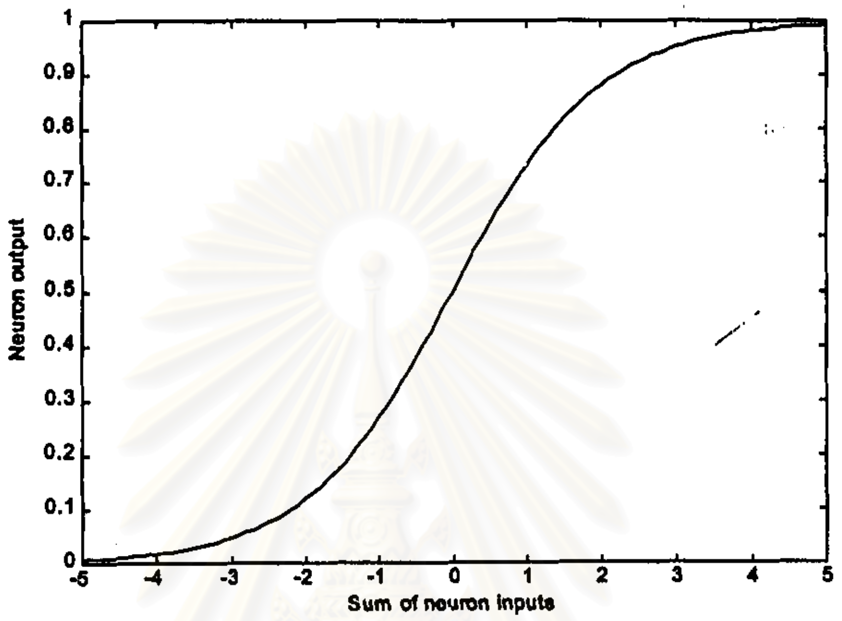

Figure 3.3: Sigmoid function.

3.4 Backpropagation Algorithm

Backpropagation (Rumelhart, Hinton, and William, 1986b) is a gradient descent learning rule, also called the generalized delta rule. In this method, the network predicted output is compared with the actual output (target), and the weights are changed in the negative direction of error to minimize the prediction error. This type of learning is known as "supervised learning". The following description of the training algorithm is summarized from various sources. (See for example, Beale and Jackson 1990; Rumelhart, Hinton, and William, 1986a; and Rumelhart, Hinton, and William, 1986b).

The error function to be minimized is defined as proportional to the square of the difference between the actual and desired output, for all the data patterns to be learned. Let E_p be the prediction error for pattern p. Then,

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \tag{3.5}$$

where t_{pj} and o_{pj} refer to the target (desired) value and the network predicted value of the output, respectively, for pattern p and output j. The overall error is then given by

$$E = \sum E_p \tag{3.6}$$

The delta rule implements a gradient descent rule in which the weights are changed in proportion to the negative gradient of the error.

$$\Delta_p w_{ij} \, \alpha - \frac{\partial E_p}{\partial w_{ij}} \tag{3.7}$$

where w_{ij} is the weight from node i to node j and $\Delta_p w_{ij}$ is the change in w_{ij} due to prediction error in pattern p.

The computations in unit j for a pattern p can be represented by,

$$S_{pj} = \sum_i w_{ij} o_{pi} \tag{3.8}$$

$$o_{pj} = f_j(S_{pj}) \tag{3.9}$$

where o_{pj} is the output of the neuron j for pattern p. The error gradient with respect to the weights can then be determined as follows: Using the chain rule, we can write

$$\frac{\partial E_p}{\partial W_{ij}} = \frac{\partial E_p}{\partial S_{pj}} \frac{\partial S_{pj}}{\partial W_{ij}} \tag{3.10}$$

The second term of Eq. (3.10) can be obtained from Eq. (3.8)

$$\frac{\partial S_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} o_{pk} \quad = \sum_k \frac{\partial w_{jk}}{\partial w_{ij}} o_{pk}$$

$$= o_{pi} \quad (3.11)$$

Defining

$$-\frac{\partial E_p}{\partial S_{pj}} = \delta_{pj} \quad (3.12)$$

Eq. (3.10) can be written as,

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} o_{pi} \quad (3.13)$$

The weight change is proportional to the error gradient with respect to the weights. Therefore from Eq. (3.7) and (3.13),

$$\Delta_p w_{ij} = \eta \delta_{pj} o_{pi} \quad (3.14)$$

The values of δ_{pj} need to be determined for each neuron *j*. Then, the weights of the network can be updated such that the prediction error decreases. Using the chain rule on Eq. (3.12),

$$\delta_{pj} = -\frac{\partial E_p}{\partial S_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial S_{pj}} \quad (3.15)$$

From Eq. (3.5) and (3.9), Eq. (3.15) can be written as,

$$\delta_{pj} = -(t_{pj} - o_{pj}) f_j'(S_{pj}) \quad (3.16)$$

For sigmoid functions as in Eq. (3.3), $f_{pj}'(S_{pj})$ can easily be obtained to be

$$f_j'(S_{pj}) = o_{pj}(1 - o_{pj}) \quad (3.17)$$

The first derivative can easily be calculated for sigmoid function from the output values only as given above.

Eq. (3.16) is useful for calculating δ's for the neurons in the output layer. However, this equation cannot be used for the hidden layer neurons, because the "target value" similar to t_{pj} is not available to define the error for the hidden layer neurons. Therefore, when j refers to a hidden layer neuron, the term $\frac{\partial E_p}{\partial w_{ij}}$ can be obtained as follows: by using the chain rule again,

$$\frac{\partial E_p}{\partial o_{pj}} = \sum \frac{\partial E_p}{\partial S_{pk}} \frac{\partial S_{pk}}{\partial o_{pj}}$$

$$= \sum_k \frac{\partial E_p}{\partial S_{pk}} \frac{\partial}{\partial o_{pj}} \sum_j w_{jk} o_{pj}$$

$$= -\sum_k \delta_{pk} w_{jk} \tag{3.18}$$

where k is the output layer. Therefore, δ for a hidden layer neuron is given by,

$$\delta_{pj} = \sum \delta_{pk} w_{jk} f_j'\left(S_{pj}\right) \tag{3.19}$$

The weight update rule is given by Eq. (3.14) and the δ_{pj} is given by Eq. (3.16) for neuron in the output layer, and by Eq. (3.19) for neurons in the hidden layer. The values of $f_{pj}'(S_{pj})$ for sigmoid functions, required in these equations ((3.16) and (3.19)) can easily be obtained only from the outputs as in Eq. (3.17). Note that the δ_{pj} values of the hidden layer neurons are dependent on all of the δ_{pk} values (k = 1, 2,...) of the output layer neurons. Therefore, the δ values of the output neurons are calculated first and then "propagated back" to the hidden layer, giving the name "backpropagation" to this generalized delta rule.

Implementing the error backpropagation rule as described is very simple. But this method converge slowly to the optimal values. The training algorithm can be significantly improved by using an acceleration method called "momentum algorithm" which is a conventional optimization tool. A description of the algorithm can be found in Rumelhart, Hinton, and William (1986b). In this algorithm, an additional term is added to the weight update rule which gives "a momentum" to the change in weights towards the optimum. The modified training rule with the addition of a momentum term is given by,

$$\Delta_p w_{ij}(t) = \eta \delta_{pj}(t) o_{pi}(t) + \alpha \Delta w_{ij}(t-1) \qquad (3.20)$$

where t refers to an epoch, or an iteration, incremented by 1 for each sweep through the whole set of input-output values. The term α is the momentum parameter which can take a value between 0 and 1, determining the relative contribution of the earlier gradients to the current weight change. This procedure produces a large change in the weights if the changes are currently large, and will decrease as the changes become less. Therefore the training speed increases, and the network is less likely to get stuck in local optima early on, since the momentum term will push the network out of local optima following the overall general trend in weight movement. Figure 3.4 shows the forward flow of the data to the feedforward network and backward flow of the error in such network trained with error backpropagation algorithm.

Conclusion of the backpropagation algorithm and an example demonstrating how to apply such algorithm to solve a problem are provided in Appendix B.

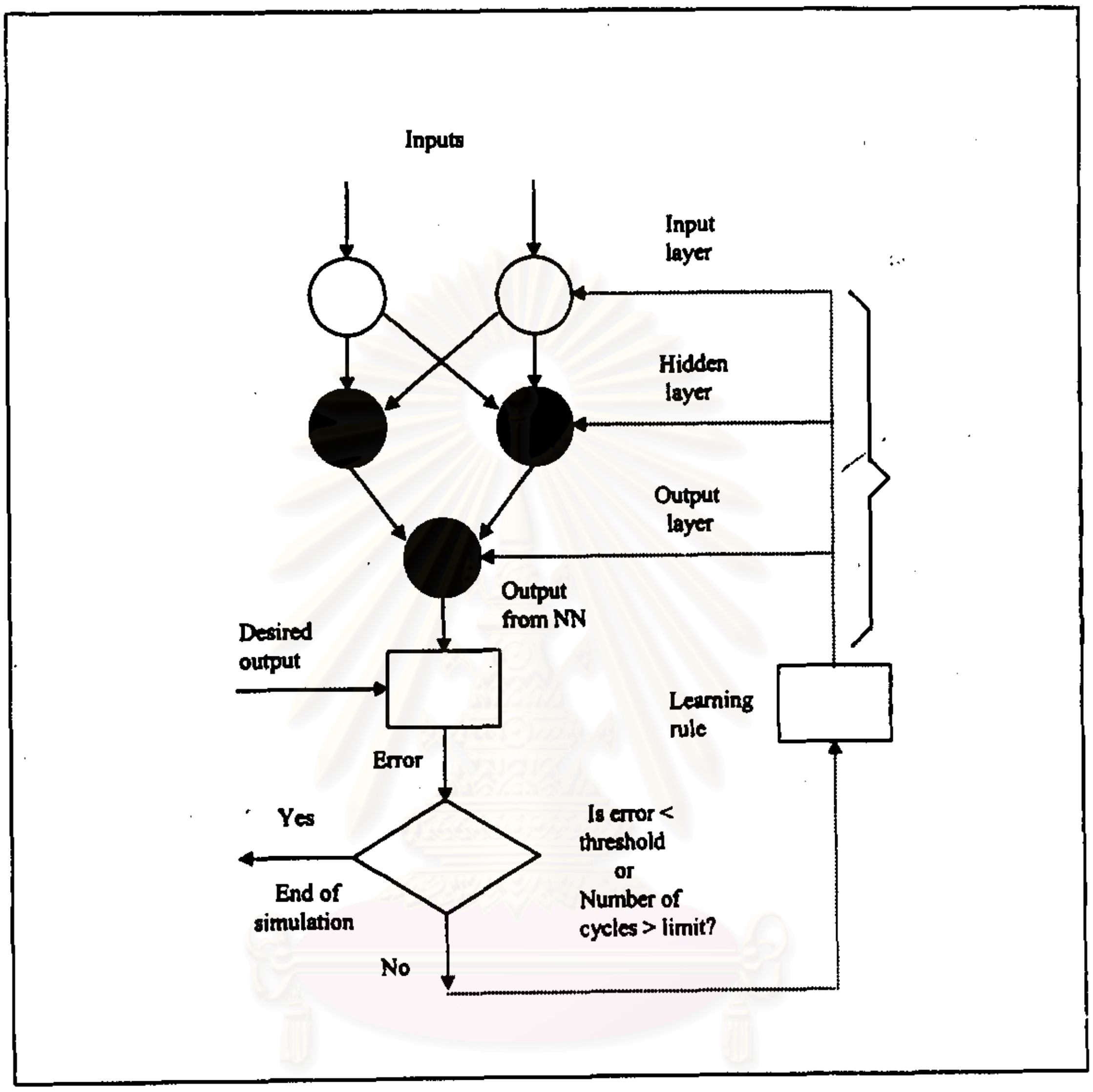

Figure 3.4: Forward flow of information or data (arrows) and backward flow of error (dashed lines) in a back propagation type of neural network