CHAPTER 2

THEORETICAL BACKGROUND

2.1 Introduction to Artificial Neural Networks

2.1.1 Artificial neural network (ANNs)

ANNs is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1) Information processing occurs at many simple elements called neurons.

2) Signals are passed between neurons over connection links.

3) Each connection link has an associated weight, which, in atypical neural net, multiplies the signal transmitted.

4) Each neuron applies an activation function (usually non-linear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by the pattern of connections between the neurons (called its architecture), the method of determining the weights on the connections (called its training or learning, algorithm) and the activation function. A neural net consists of a large number of simple processing elements called neurons, units, cells or nodes. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve a problem.

Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

2.1.2 Biological neural networks

A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: its dendrites, soma and axon. The many dendrites receive signals from other neurons. The signals are electric impulses that are transmitted across a synaptic gap by means of a chemical process. The action of the chemical transmitter modifies the incoming signal (typical, by scaling the frequency of the signals that are received) in a manner similar to action of the weights in an artificial neural network.

The soma, or cell membrane, sums the incoming signals. When sufficient input is received, the cell fires; that is, it transmits a signal over its axon to other cells. It is often supposed that a cell either fires or does not at any instant of time, so that

transmitted signals can be treated as binary. However, the frequency of firing varies and can be viewed as a signal of either greater or lesser magnitude. This corresponds to looking at discrete time steps and summing all activity (signals received or signals sent) at a particular point in time.

A generic biological neuron is illustrated in Fig. 2.1, together with axons from other neurons (from which the illustrated neuron could receive signals) and dendrites for other neurons. Another important characteristic that artificial neural networks share with biological neural systems is fault tolerance. Biological neural systems are fault tolerant in two respects. First, we are able to recognize many input signals that are somewhat different from any signal we have seen before. An example of this is our ability to recognize a person in a picture we have not seen before or to recognize a person after a long period of time.

Second, we are able to tolerate damage to the neural system itself. Humans are born with as 100 billion neurons. Most of these are in the brain, and most are not replaced when they die.

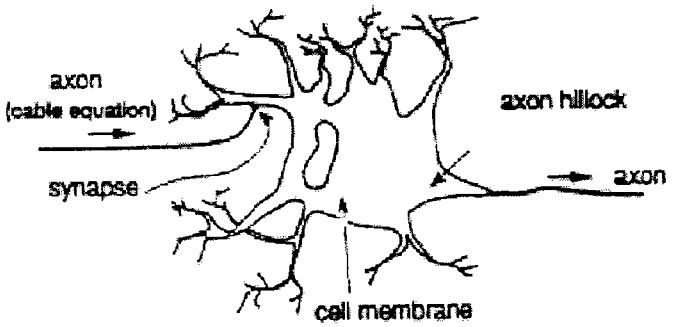

Fig. 2.1 Biological neuron

2.2 Neural Network Architectures

Each unit can be combined into a network in numerous fashions. The most common of these is the *multilayer perceptron* (MLP) network. The basic MLP-network architecture is arrangement of neurons into layers of input units and the connection patterns weight between layers, which are often classified as:

2.2.1 Single layer neural network

A single layer network has only one layer of connection weight. The input units receive signals from the outside world, multiplies an associated weight, sum its weight input signal, apply activation function to output unit. The weight for one output unit does not influence the weights for other output unit.

Typically, the input units of single layer net are fully connected to output and these output units are not connected to the other output unit. The weight for one output unit does not influence the weights for other output unit.

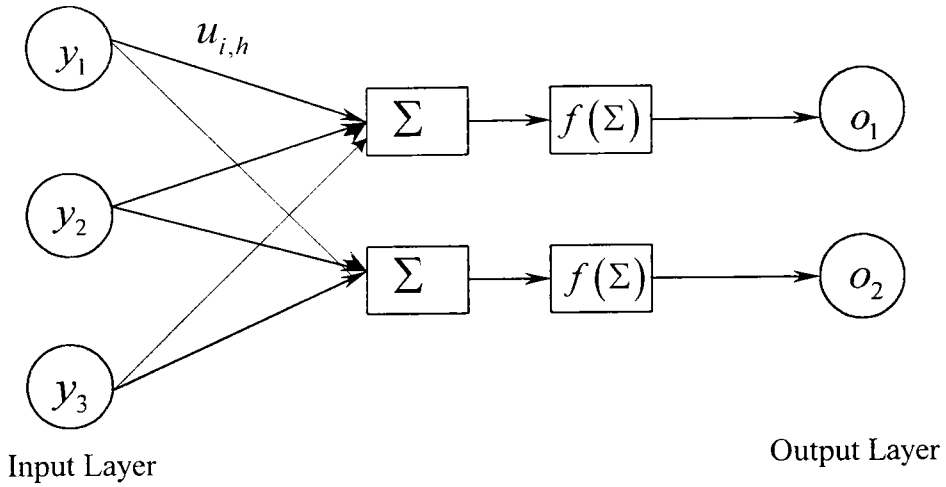

Fig. 2.2 Single layer network

This neural network is very simple, the presence of a hidden unit together with a non-linear activation function gives it the ability to many more complicated problems. That is discussed in Multi-Layer network below.

2.2.2 Multiple layers neural network

A multi-layer network is a network with one or more layers of hidden units between the input units and the output units. There is a layer of weights between hidden layers, too. As shown in Fig. 2.3, The output layer is a layer whose output is the network output. The other layers are hidden layers.

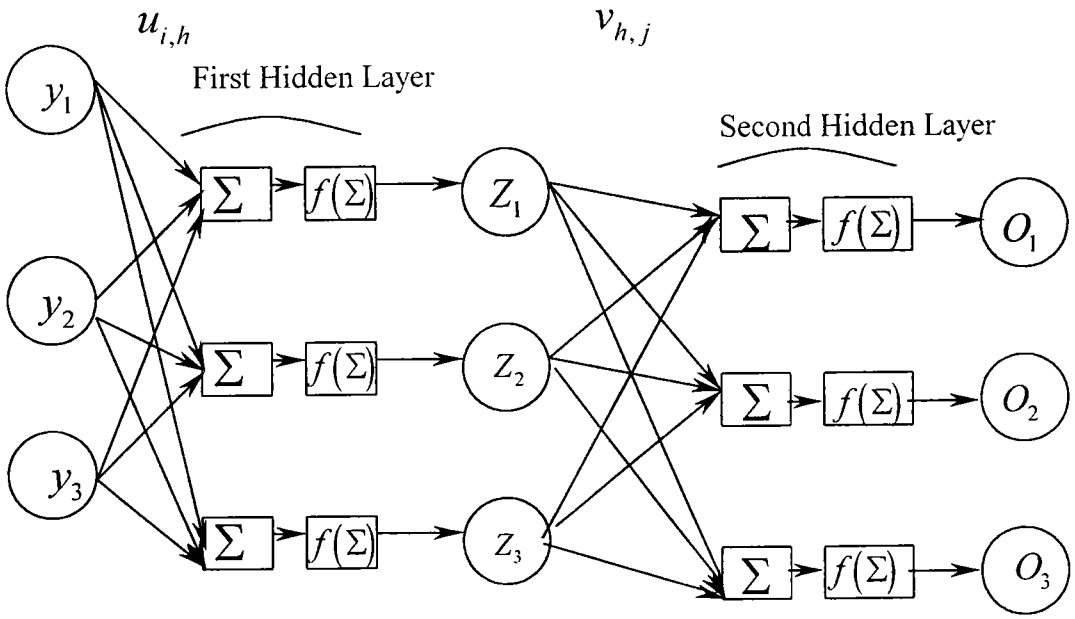

Fig. 2.3 Multiple layers network

Multi-layer networks are more powerful than single layer networks. For instance, multi-layer network can be trained to approximate most functions arbitrarily well. Single-layer network cannot do this. The most practical neural networks have just two or three layers.

2.2.3 Other neural network architectures

The previous describe network architecture is just one out of many neural networks, but it is the only one of covered here. Several textbooks are available that offer introductions to the most common network types. Most of these books also provide the historical background for neural networks and how they relate to biological neural networks. See for example Hertz et al. (1991), Haykin (1998), and Zurada (1992). Not all networks are equally suitable for modeling and control of dynamic systems. For these applications the most common alternative to the multilayer percepteron (MLP) network is probably the Radial Basis Function (RBF) networks (Sanner and Slotine 1992, Tzirkel-Hancock and Fallside 1992). Studying and comparing the performance of different network types is however beyond the scope of this study.

2.3 Activation Function

The activation function is the same for all neurons in any particular layer of a neural net. In most cases, a non-linear activation function is chosen to satisfy some specification of the problem that the neuron is attempting to solve. For the results of feeding a signal through two or more layers of linear processing element.

In identification-related problem. the useful activation function is sigmoid function (*S*-shape curves). The logistic function and the hyperbolic tangent functions are the most common, which are *Binary sigmoid function* and *Bipolar sigmoid function*.

2.3.1 Binary sigmoid function

The binary sigmoid function is shown in Fig. 2.4. This activation function takes input, which may have any value between plus and minus infinity and squashes the output into the range 0 to 1.

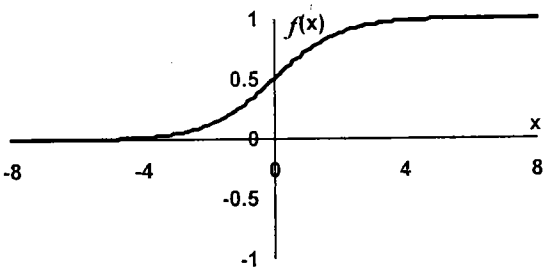

Fig. 2.4 Binary sigmoid function

The binary sigmoid function is the logistic function. which commonly used in multi-layer networks that are trained using the backpropagation algorithm, in part because this function is differentiable.

2.3.2 Bipolar sigmoid function

The bipolar sigmoid is closely related to the hyperbolic tangent function. The application of this function is similar as the binary sigmoid function; this function is illustrated in Fig. 2.5. The input can be any value between plus and minus infinity. But the desire ranges of output values are between –1 and 1, (-1,1). The bipolar sigmoid output is wider than output from binary sigmoid function (0,1).

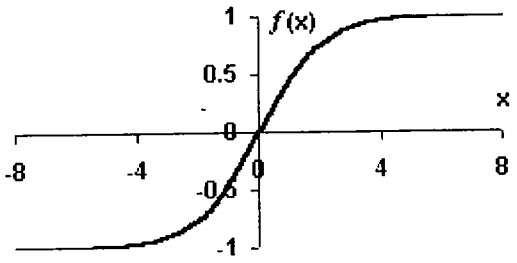

Fig. 2.5 Bipolar sigmoid functions

2.4 Training Algorithm

2.4.1 Learning rule

Learning rule means a procedure for modifying the weights and biases of the network. The purpose of the learning rule is to train to perform some task. There are many types of neural network learning rules. They fall into two broad categories: supervised learning and unsupervised learning.

In *supervised learning,* the learning rule is provided with a set of examples (the training set) of proper network behavior:

$$(y_1, o_1), (y_2, o_2), \ldots, (y_n, o_n)$$

where y_n is input to the network and o_n is the corresponding correct (target) output. As the inputs are applied to the network, the network outputs are comparing to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets.

In *unsupervised learning,* the weights and biases are modified in response to network inputs only. There are no target outputs available. Most of these algorithms perform some kind of clustering operation. They learn to categorize the input patterns into a finite number of classes. This is especially useful in such applications as vector quantization.

2.4.2 Generalized delta rule

In the Widrow-Hoff rule, or delta rule, the amount of learning is represented as the difference (or delta) between the desired and computed outputs. Purposed by Rumelhart, Hinton, and Williams (1986), backpropagation (BP) is an error-correcting learning procedure that generalized the delta rule to multi-layer feedforward neural networks with hidden units between the input and output units. The goal of the learning procedure is to update the weights of the links connecting the nodes, and to minimize the average squared system error between the desired and the computed outputs. The error term from one output neuron is defined as:

$$E = \frac{1}{2}(t - o)^2 \tag{2.1}$$

where t is the desired output, o is the network output.

We adjust the weights and threshold value in proportion to $\frac{-\partial E}{\partial w_j}$ or in order to reduce the system error:

$$\Delta w_j = -\mu \frac{\partial E}{\partial w_j} \tag{2.2}$$

where μ is called the learning rate, which determines what amount of the calculated error sensitivity to, weight change will be used for the weight correction.

2.5 The Backpropagation Network

The backpropagation network (BPN) is currently the most general-purpose and commonly used neural- network paradigm. The BPN achieves its generality because of the gradient- descent technique used to train the network.

Gradient descent is analogous to an error- minimization process. Error minimization, as the term implies, is an attempt to fit a closed-form solution to a set of empirical data points, such that the solution deviates from the exact value by a minimal amount. Fig. 2.6 illustrates the error-minimization concept.

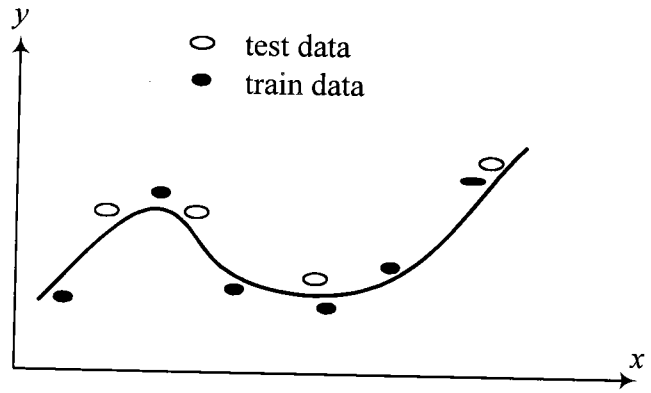

Fig. 2.6 The diagram illustrates the process of minimizing the error of a function through the set of empirical data

The learning process begins with the presentation of an input pattern to the BPN. That input pattern is propagated through the entire network, until an output pattern is produced. The BPN then makes use of what is called the generalized delta rule to determine the error for the current pattern contributed by slightly in a direction that reduces its error signal, and the process is repeated for the next pattern.

In this project, the backpropagation neural network is adapted for the identification of non-linear. The principles of the backpropagation neural network are introduced in the following.

Fig. 2.7 the typical two hidden layers backpropagation neural network: the input layer with n nodes, hidden layer 1, 2 with q, p nodes respectively and the output layer with 1 node. There are weights u_{ih}, v_{hj}, w_j layers, represent the strength of connection of the nodes in the network.

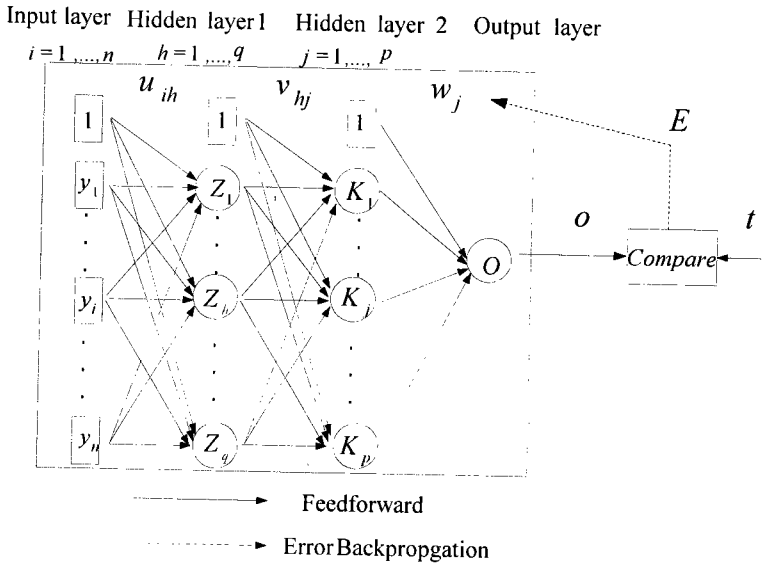

Fig. 2.7 Typical two hidden layers backpropagation neural networks

The first type of operation of bakpropagation neural network is called feed forward and it shown as solid lines with arrows. Starting from an input-output pair, each input neuron receives an input signal and broadcasts this signal to the connected

neurons $Z_1, \ldots, Z_q$ in the first hidden layer. The total input to the Z_h neuron from the input layer is

$$z\_in_h = u_{0h} + \sum_{i=1}^{n} y_i u_{ih} . \tag{2.3}$$

Each of these neurons then computes its activation

$$z_h = f(z\_in_h) \tag{2.4}$$

and sends its result to the connected neurons $K_1, \ldots, K_p$ in the second hidden layer. The total input to the Kj neuron from the first hidden layer is

$$k\_in_j = v_{0j} + \sum_{h=1}^{q} z_h v_{hj} . \tag{2.5}$$

Next, each neuron in the second hidden layer computes its activation

$$k_j = f(k\_in_j) \tag{2.6}$$

and sends its result to the output neuron. The total input to the output neuron O from the second hidden layer is

$$o\_in = w_0 + \sum_{j=1}^{p} k_j w_j . \tag{2.7}$$

Finally, the output neuron yields the network output according to

$$o = f(o\_in). \tag{2.8}$$

In the equations above u_{ih}, v_{hj}, and w_j are the connection weights between the layers, whereas $f(\cdot)$ is the activation function (Haykin 1999). For example, a bipolar sigmoid activation function, Fig. 3.5, is defined as:

$$f(x) = \frac{2}{1+e^{-x}} - 1. \tag{2.9}$$

During training, the network output o is compared with its target or sample output t to determine the error e associated with the output neuron.

$$e = (t - o). \tag{2.10}$$

The update of the connection weights aims at minimizing this discrepancy. Consequently, an objective function to be minimized is defined as:

$$E = 0.5(t - o)^2. \tag{2.11}$$

Using the generalized delta rule, the update of the weights connecting the second hidden layer with the output layer is given by

$$w_{j,n} = w_{j,o} + \Delta w_j \tag{2.12}$$

$$\Delta w_j = -\mu \frac{\partial E}{\partial w_j} \tag{2.13}$$

$$= \mu(t - o) f'(o\_in) k_j \tag{2.14}$$

$$\equiv \mu \delta k_j \tag{2.15}$$

where μ = learning rate; $w_{j,n}$ = new weight, $w_{j,o}$ = old weight. In this case, the derivative f' are equal to

$$f'(x) = \frac{(1 + f(x)) \times (1 - f(x))}{2}. \tag{2.16}$$

For the weights connecting the first hidden layer to the second hidden layer, the update is done in the same way

$$v_{hj,n} = v_{hj,o} + \Delta v_{hj} \tag{2.17}$$

$$\Delta v_{hj} = -\mu \frac{\partial E}{\partial v_{hj}} \tag{2.18}$$

$$= \mu f'(k\_in_j) \delta w_j z_h \tag{2.19}$$

$$\equiv \mu \delta_j z_h. \tag{2.20}$$

Finally, the update for the weights connecting the input layer to the first hidden layer is

$$u_{ih,n} = u_{ih,o} + \Delta u_{ih} \tag{2.20}$$

$$\Delta u_{ih} = -\mu \frac{\partial E}{\partial u_{ih}} \tag{2.21}$$

$$= \mu f'(z\_in_h) \sum_{j=1}^{p} \delta_j v_{hj} y_i \tag{2.22}$$

$$\equiv \mu \delta_h y_i . \tag{2.23}$$

After finishing the first sample input-output pair, the procedure repeats from Eq. (2.3) to Eq. (2.8) for each consecutive sample pair. When there is no further improvement in the discrepancy reduction, the procedure is stopped. Besides consecutively updating the connection weights as described, the update can be done in a batch mode manner. In the latter case the objective function Eq. (2.11) is defined as the summation of all discrepancy from individual sample input-output pair.

2.6 Deficiencies of Backpropagation

Despite the apparent success of the backpropagation learning algorithm, there are some aspects, which make the algorithm not guaranteed to be universally useful. Most troublesome is the long training process. This can be a result of a non-optimum learning rate. A lot of advanced algorithm based on backpropagation learning have some optimized method to adapt this learning rate but it is beyond the scope of this study. Outright training failures generally arise from two sources: network paralysis and local minima.

2.6.1 Network paralysis

As the network trains, the weight can be adjusted to very large values. The total input of a hidden unit or output unit can therefore reach very high (either positive or negative) values, and because of the bipolar sigmoid activation function the unit will have an activation very close to minus one or very close to one. As is clear from Eq. (2.16), the weight adjustments which are proportional to $(1+f(x))\times(1-f(x))$ will be close to minus one, and the training process can come to a virtual standstill.

2.6.2 Local minima

The error surface of a complex network is full of hills and valleys. Because of the gradient descent, the network can get trapped in a local minimum when there is a much deeper minimum nearby. Probabilistic methods can help to avoid this trap, but

they tend to be slow. Another suggested possibility is to increase the number of hidden units. Although this will work because of the higher dimensionality of the error space, and the chance to get trapped is smaller, it appears that there is some upper limit of the number of hidden units which, when exceeded, again results in the system being trapped in local minima.

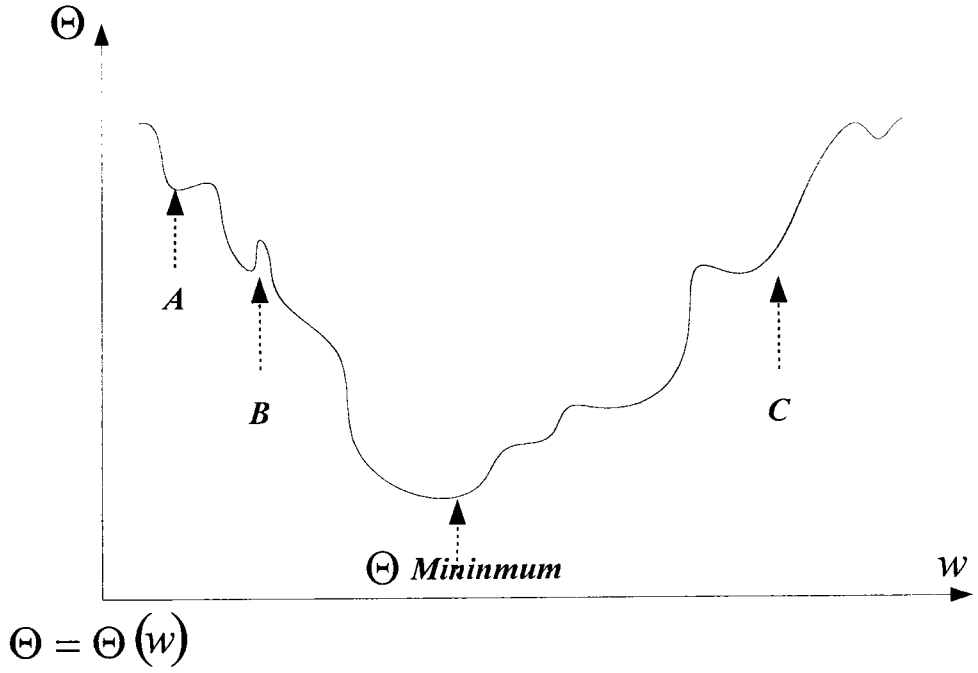

Fig. 2.8 Local minima (A, B, and C) for a one-weight performance index Θ

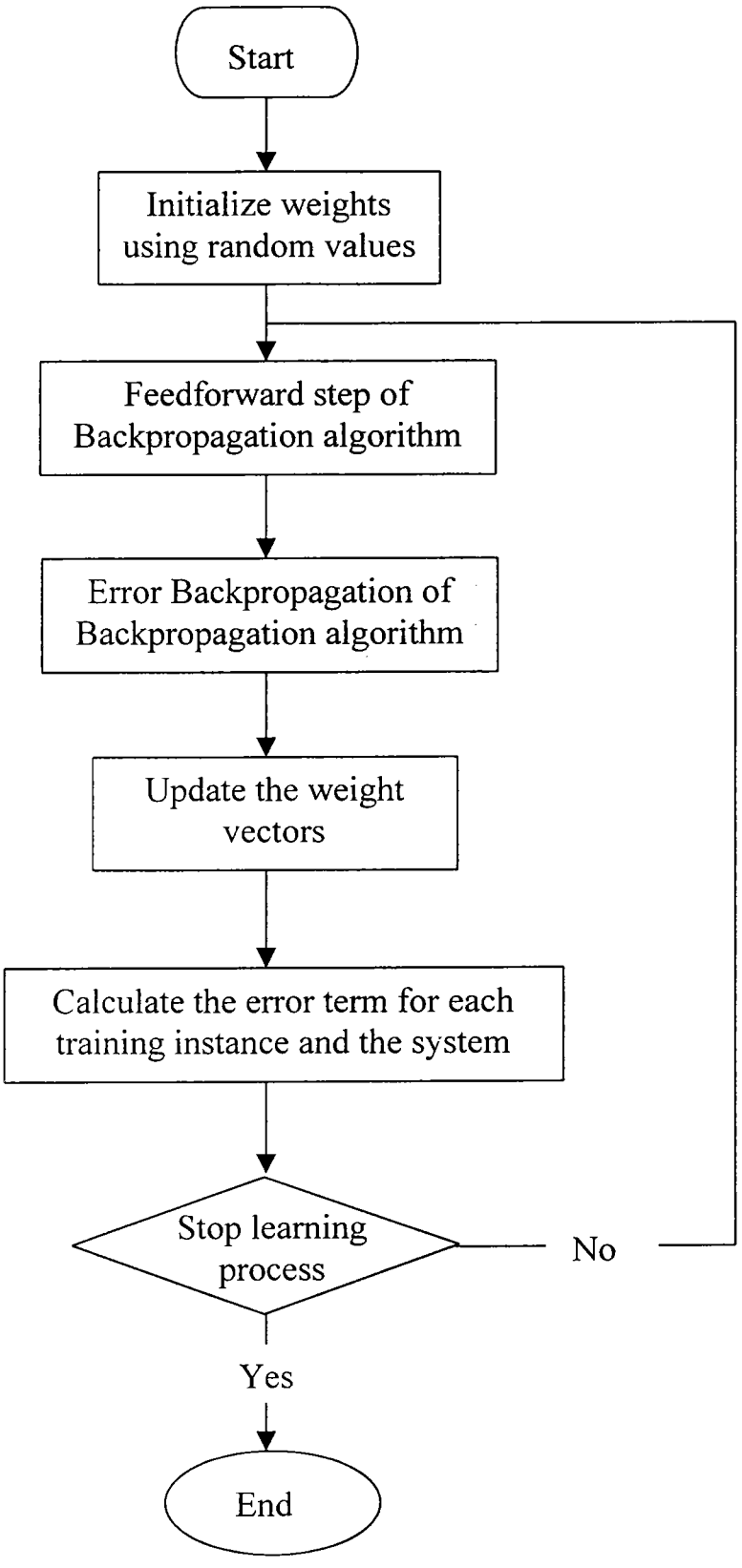

Fig. 2.9 The Backpropagation learning algorithm

2.7 Practical Considerations

2.7.1 Selecting a topology for a network

Every stage of any ANN project requires a little trial and error to establish a suitable and stable network for the project. Trail and error may be extended to building several networks, stopping and testing the network at different stages of learning and initializing the network with different random weights. Each network must be tested, analyzed and the most appropriated network must be chosen for a particular project.

The number of inputs to the network and the number of outputs from the network are defined by external problem specifications. For example if there are four external variables to be used as inputs, there are four input nodes to the network. Similarly, there are to be one output from the network, there must be one output neuron in the output layer. But the chosen of input of the network must consider the relationship of network inputs and outputs value and other constraint as well. In Gunaratnam and Gero (1994) the dimensionality analysis was used in order to reduce the number of independent components required to describe the domain relationship. Instead of the dimensional representation of physical variables their dimensionless products can be used. Such an approach leads not only to reduction in the dimensionality of the space in which the domain relationship is described but makes the regularities in the domain simple and results in simpler relationship. The domain knowledge is thus considering as another important factor to choose the inputs to the network.

There is also no general rule for selecting the number of neurons in a hidden layer. The choice of hidden layer size is mainly problem specific and to some extent depends on the number and the quality of training patterns. The number of neurons in a neural network must be sufficient for the correct modeling of the problem, but it should be sufficiently low to ensure generalization. Too large a number of hidden neurons world encourage over-fitting, i.e. modeling the noise in the data or modeling the data with unnecessarily complex functions. Generally, there is a trade-off between building a network which generalizes well and is robust, and one which is more accurate but more brittle i.e. one in which the network learn the training data well but its performance is poor when tested with unseen data. More complex, accurate, yet brittle networks require more data patterns for training. A more general model describes a smoother curve through the training data points, missing some, but resisting the effect of noise and peculiarities, which might be present. The number of neurons also greatly affects the generalization characteristics of a neural network. To be able to design a stable network, it would be more appropriate to carry out a parametric study be changing the number of neurons in the hidden layer in order to test the stability of the network.

2.7.2 Pre-processing training data

Neurons like to see data in a particular input range to be most effective. If data contains high variation, the network will not be useful, since the middle layer of neurons have a bipolar sigmoid activation function that squashes large inputs to either -1 or 1. In other word, the data should be chosen to fit a range that does not saturate.

or overwhelm the network neurons. Choosing inputs from -1 to 1 is a good idea. Then within each data set, input data and targets will be scaled to values between -1 to 1 in corresponding to the bipolar sigmoid function. A proposed scaling formula is:

$$del = max - min \tag{2.24a}$$
$$m = 1.8/del \tag{2.24b}$$
$$c = -0.9 - (m \times X_{min}) \tag{2.24c}$$
$$y = mx + c \tag{2.24d}$$

where
- c = y axis-interception
- X_{min} = minimum of input or output value
- x = original / raw input or output value
- y = scale input or output value
- m = slope of linear equation

2.7.3 Data selection for neural network training

In order to ensure that the network has properly mapped input training data to the target output, it is essential that the set of patterns presented to the network is appropriately selected to cover a good sample of the training domain. A well-trained network is one, which is able to respond to any unseen pattern within an appropriate domain. At present, ANN is not good at extrapolating information outside the training domain.

The selection of an adequate number of training patterns is therefore, an extremely important issue. There are no acceptable generalized rules to determine the size of the training data for suitable training. Patterns chosen for training must cover upper and lower boundaries and a sufficient number of samples representing particular features over the entire training domain (Rafiq et al. 2001).

2.7.4 Initial weights and learning parameters

Choice of training parameters (initial weight, learning parameters, etc.) is problem-dependent. The choice of initial weights will influence whether the net reaches a global (or only a local) minimum of the error and, if so, how quickly it converges. It is likely important to avoid choices of initial weights that would lead to zero values of the activation or their derivatives. The value of initial weights must not be too large, or the initial input signals to each hidden or output node will be likely to fall in the region where the derivative of the bipolar sigmoid function has a very small value (the so-called saturation region). On the other hand, if the initial weights are too small, the net input to a hidden or output will be close to zero, which also causes extremely slow learning. The bias or the threshold should add to the activation. The bias gives the network extra variable, and the networks with biases would be more powerful than those without. For instance, that a neuron without a bias will always have a net input of zero when the network inputs are zero. This may not be desirable and can be avoided by the use of bias.

A common procedure is to initialize the weights and bias to random values between –0.5 and 0.5 (Fausett 1994). So it will be adapted herein.

Selection of a value for the learning rate parameter has also a significant effect on the network performance. The large value of learning rate will lead to rapid learning but the weight may then oscillate, while low values imply slow learning. The right value of learning rate will depend on the application.

2.7.5 Duration of network training

Presenting the entire set of training patterns to the network is called an epoch. The number of epochs, number of times that the whole set of patterns is presented to the network, affects the performance of the network. This number depends on many factors of which the following are more important:

1) Number of training data.

2) Number of hidden layers.

3) Number of neurons in hidden layers,

4) Number of dependent output parameters.

If the network is trained for too long, it begins to overfit, e.g. model noise in the training data, or develop an unnecessarily complex function to fit the data. As a result the network will not perform well to unseen patterns. If the training is not carried out long enough, the network will not learn all features and subfeatures and hence the network performance will be unsatisfactory. The numbers of epoch for network to learn some specific problems are vary from case to case. There are a number of reasons for not being able to define a general rule about the number of epochs, which the training patterns should be shown to the network.

1) The back-propagation algorithm does not guarantee a convergence for MLP neural networks. The network may fail to reach a suitable weight configuration, which minimizes the network error.

2) The lowest possible error for the noisy data is not zero.

3) MLP networks are designed to minimized training error. They may not necessarily minimize the generalization error.

For the above reasons, it is practically necessary to carry out a parametric study to determine the optimum number of epochs necessary for network training.

Once suitable network architecture is found and the training patterns are selected and scaling, the network training can be started. The result of the training can be saved at various intervals for network testing purposes. Testing of trained networks at regular intervals during training aims to avoid the important problem of overfitting and hence produce a good generalization network.

2.7.6 Training mode

Generally, there are two different modes of training ANN: batch mode and sequential mode. In a batch mode, when an epoch is completed (i.e. when the entire set of training data is presented to the network) a single average error is calculated and the weights in the network are adjusted according to that error. In a sequential mode, the error is calculated after each pattern is presented to the network, and network weights are adjusted. Choosing between the two modes is generally, problem specific. Swingler (1996) has indicated that the following points should be considered when choosing a training mode:

1) Batch mode requires less weight update and hence it is faster to train.

2) Batch mode provides a more accurate measurement of the required weight changes.

3) Batch mode is more likely, than sequential mode, to become tapped in local optimum.

From experience has also shown that batch mode are much sensitive to initial weights. It would be advisable to train the network using batch mode to start with and test and analyze the network output. If the level of error after testing the network with unseen data was not satisfactory then a sequential mode should be used.

2.7.7 Testing the trained networks

After the network has been trained until a satisfactory minimum error is reached, the weights are fixed and apply this trained network to the test data and note the error. If the error from tested network is lower than or equal to allowable value of error, it means that the trained neural network model is good emulator.

2.8 Application of Artificial Neural Network

The application are expending because neural networks are good at solving problem, not just in engineering, science and mathematics, but in medicine, business, finance and literature as well. Their application to a wide variety of problems in many fields makes them very attractive.

Robert Hecht-Nielsen concentrated to application of neural networks in three main areas. In order of their prevalence, he lists (1) data analysis, (2) pattern recognition, and (3) control function. He goes on the describe sub areas within each of these categories that are most likely to benefit from the neural approach.

Within data analysis, fields of potential application include the processing of loan application; analysis of commodity trading, time series forecasts, crop of activity in public records, and law enforcement, involving the search of criminal records.

In pattern recognition, optical character recognition ranks high as a neural application especially when it involves hand-addressed mail or hand-written checks.

Industrial inspection is another environment where complex pattern recognition problems exist. For example, in silicon wafer inspection, there are unexpected defects that are not easily classified; a neural-network-based novelty filter then becomes a likely tool, with the neural network perceiving, as a human observer might, that "this one looks different".

A control application is suitable for neural networks include such functions as the operation of machine tools. For example, a tool that is rigid enough to hog out metal blocks becomes susceptible to breakage at high speeds of operation. Sensors that detected vibrations and stresses could provide real-time clues to where the boundaries are. The result of intelligent supervision would be a significant rise in productivity.