Chapter 6

NEURAL NETWORK – AN OVERVIEW

6.1 NEURAL NETWORKS AND THEIR CAPABILITIES

Artificial Neural Networks (ANNs) are collection of mathematical models that emulate some of the observed properties of the biological nervous system and draw on the analogies of adaptive biological learning. ANNs are configured by employing massive number of simple processing elements (Neurons) with a high degree of connectivity between them to achieve high computation rates. These elements usually undergo a learning process which automatically updates network parameters in response to a possibly evolving input environment.

Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true for ANN as well. Learning typically occurs through training or exposure to a truth table set of input/output data where the training algorithm iteratively adjusts the connection weights (Synapses). These connection weights store the knowledge acquired during learning process.

Neural Network offer several advantages over the more conventional procedural and symbolic approaches to computing. The most frequently cited of these are their ability to develop a generalized solution to a problem from a set of examples, and to continue development and adapt to changing circumstances with exposure to new variations of a problem. The attribute of generalization permits them to be applied to situations other than those used for training and to produce valid solutions even when there are errors in the training data or in the description of an instance of a problem requiring a solution. These factors combine to make ANNs a powerful tool for modeling problems in which functional relationships between dependent and independent variables are poorly understood, subject to uncertainty, or likely to vary with the passage of time. Such problems are common place throughout the various disciplines of civil engineering.

Problems where the time required to generate solutions is critical, such as real-time applications that require many solutions in quick succession, mark another important area that can benefit form the neural network approach. In particular, the ability of neural

networks to produce solutions in a fraction of a second, irrespective of the complexity of the problem, makes them valuable even when alternative techniques are available that can produce more optimal or more accurate solutions. For example, many intelligent-search optimization algorithms operate extremely slowly when the problem to be solved comprises many variables. In such situations, significant time can be saved using a neural network to find a good initial solution that act as the starting point for the optimization algorithm. On the other hand, a neural network may be used as a quick check on the solution developed by more time-consuming in depth analysis. They are often good at solving problems that do not have an algorithmic solution or for which an algorithmic solution is too complex to be found.

6.2 BIOLOGICAL NEURON

Artificial neurons, or nodes, are built on very simple models of biological neurons. The cerebral cortex is the largest part of the human-brain and is composed of numerous interconnected neurons arranged in very thin sheets that are highly convoluted. There are approximately 10^{11} neurons in the human brain, and each is connected to a thousand to ten thousand other neurons. The amount of interconnectivity in the average human brain is therefore in the order of 10^{14} or 10^{15}. The artificial neural networks studied thus far in the literature have a very small fraction of the number of neurons in the brain. Likewise, the interconnectivity of current artificial systems is far below the massive interconnections in the brain.

Figure 6.1 shows the schematic diagram of the structure of the biological neuron. A typical neuron has a cell body called the *soma*, and root-like structures through which input signals are received, called *dendrites*. These dendrites connect to other neurons through what is called a *synapse*. Signals collected through the numerous dendrites are accumulated (summed up) in the soma. If the accumulated signal exceeds the neuron's threshold, then the cell is activated (fired off). This results in an electrical spike that is sent down the output channel called the *axon*. Otherwise, the cell remains inactive. The efficiency by which signals from one neuron is sent to another neuron via a particular synapse is called the *synaptic efficiency*. Synapses are truly channels through which electrical impulses cross. These electrical impulses are regulated by chemicals known as *neuro-transmitters*.

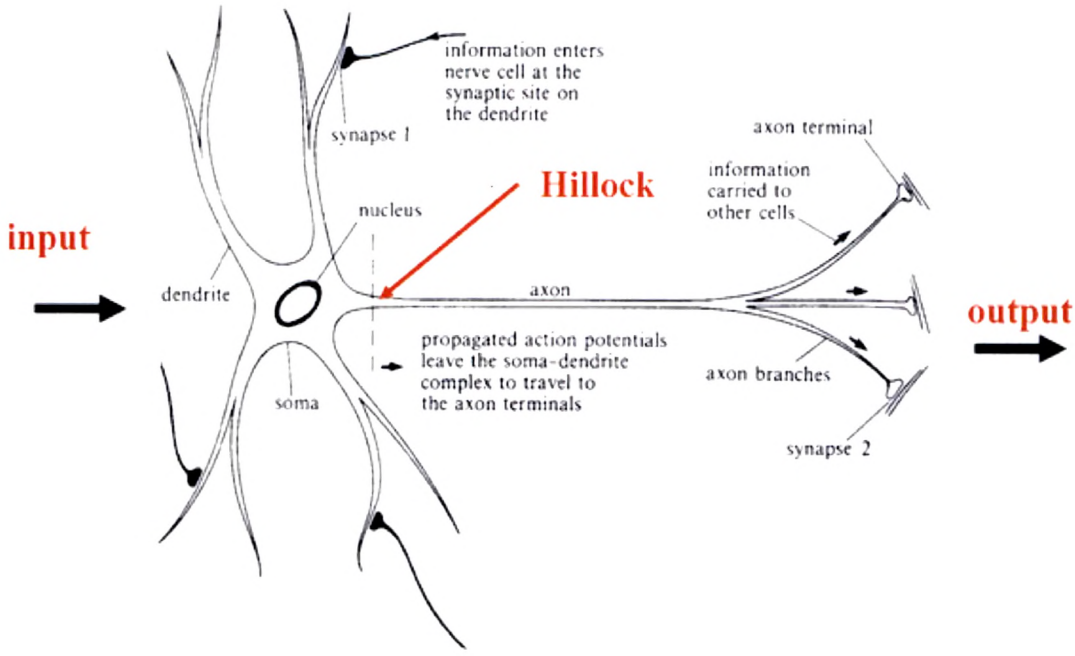

Fig. 6.1 Biological Neuron

6.3 MODEL OF ARTIFICIAL NEURON

A general model of artificial neuron is shown in Fig. 6.2. It is a simplified version of the real neuron. Functioning of this artificial neuron is based on three assumptions, that the position on the node of the incoming synapse (connection) is irrelevant, each node has a single output value, distributed to other nodes via outgoing links and all inputs come in at the same time. Suppose i^{th} neuron receives three inputs (X_1, X_2, X_3) from first, second and third node of the preceding layer, strength of connections between neurons (1, 2, 3) and i^{th} neuron is W_{i1}, W_{i2}, W_{i3}. Thus each artificial neuron is characterized by a function that combines all inputs for the node after multiplying each input value by corresponding weight.

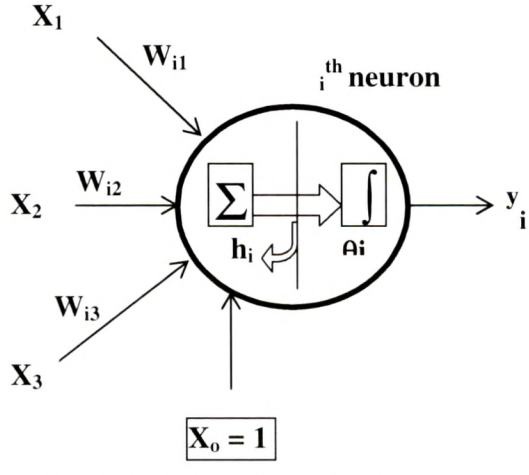

Fig. 6.2 An Artificial Neuron

Strength of various connections is referred as Weight Matrix. When strength value is zero, no connection exists. For Wij value greater than 0, it is an excitatory connection that increases the potential of cell body while in case value less than 0, inhibitory connection that decreases the potential of the cell body. θ_I is the threshold of the neurons. Unless the total input of neuron (weighted sum of input) exceeds θ_I, neuron does not fire. Summation function finds the sum of weighted input. It is taken as,

$$h_i = \sum_{j=1}^{N} W_{ij} \cdot X_j \qquad \ldots (6.1)$$

The transfer function is also known as activation function or gain function or squashing function. Weighted sum passes through the transfer function and the output is fetched by neuron using the threshold.

6.4 NEURAL NETWORK TERMINOLOGY

⊙ **Network:** A ***network*** consists of perceptrons of elementary processing units and weighted links (connections) between them. In its analogy for neuron activation, each unit receives a net input that is computed from the weighted outputs of units in the preceded perceptron with connections leading to this unit. Figure 6.3 shows small network architecture.

⊙ **Units /Nodes:** Depending on their function in the net, one can distinguish three types of units: The units whose activation's are the problem input for the net are called **input units**, the units whose output represent the output of the net are called **output units**. The remaining units are called hidden units, because they are not visible from the outside. In most neural network models the type correlates with the topological position of the unit in the net: If unit does not have input connections but only responds output signals, then it is *input unit.* If it lacks output connections but has input connections, it is an *output unit.* If it has both types of connections it is a *hidden unit.*

⊙ **Connections (Links):** The direction of a connection shows the direction of the transfer of activation. The unit from which the connection starts is called the *source unit* or *source* while the other is called the *target unit* or *target.* Connections where source and target are identical (recursive connections) are also possible. Multiple connections between one unit and the same input port of another unit are redundant and are therefore prohibited. Each connection has a *weight* assigned to it. The effect of the output of one unit on the successor

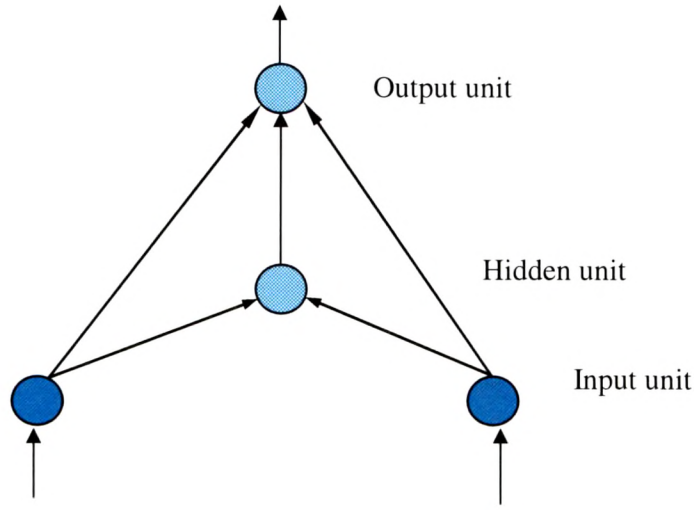

Fig. 6.3 A Small Network with Three Layers of Nodes

unit is defined by this value. If it is negative, then the connection is inhibitory, i.e. decreasing the activity of the target unit and if it is positive, it has an excitatory, i.e. activity enhancing effect. The most frequently used network architecture is built hierarchically bottom-up. The input into a unit comes only from the units of preceding layers. Because of the unidirectional flow of information within the net, such nets are called feed-forward nets. In many models a full connectivity between all units of adjoining levels is assumed.

⊙ **Training of an ANN:** The process of updating the connection strengths is basically known as ***training*** or ***learning*** of a network. A network is trained by presenting various facts (or input data) to the network. The connection weights are then updated using one of the many techniques available. Based on the manner in which the connection strengths are updated, the training can be classified into two types, namely supervised and un-supervised.

⊙ **Supervised Training**: In this type of training of an ANN, the training is ***supervised*** or controlled through adjustment of some parameters in the learning rule. The learning rule is based upon some type of error minimization technique. Backpropagation neural network technique is example of network training rule that employ supervised training.

⊙ **Un-Supervised Training**: In this type of training of an ANN, the training is not supervised. The neurons internally organize themselves from what are known as cluster

(a group of similar neurons). 'Self-organizing' or Kohonen networks are examples of such training method.

⊙ **Threshold or Activation Functions**

(i) Sigmoid Function: A sigmoid function (Fig. 6.4) is used to set the output value of any given neuron by processing the sum of the weighted input values and any bias applied. The sigmoid function outputs a value that is close to zero for a low total in put value and close to one for a high input value. The slope of the function curve can be adjusted by including a threshold value that can make the slope between zero and one steeper or more shallow.

Sigmoid function is often used as transfer function because it introduces non-linearity into the network's calculation by squashing the neuron's activation level into the range [0, 1]. The sigmoid function has the additional benefit of having an extremely simple derivative function as required for back-propagating errors through a feed-forward neural network. If no non-linearity is introduced by squashing or clipping, the network loses much of its computational power, becoming a simple matrix multiplication operation from linear analysis. Sigmoid function is given as,

$$f(x) = 1/(1 + e^{-x}). \quad \ldots (6.2)$$

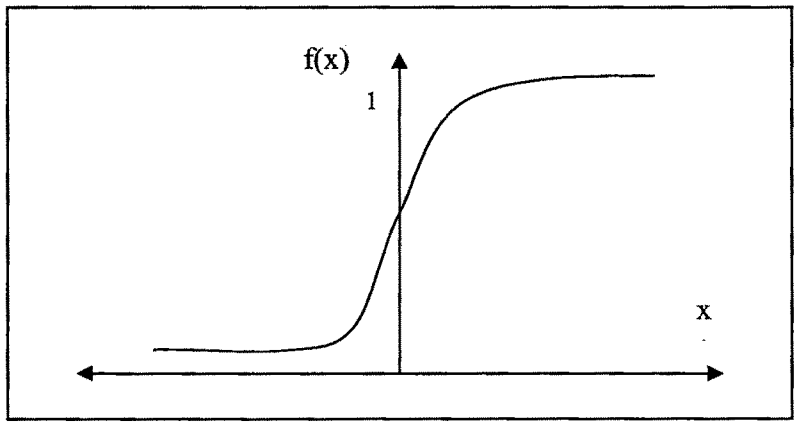

Fig. 6.4 The Sigmoid Function

(ii) The Step Function: The function is 0 to start with and remains so to the left of some threshold value of θ . A jump to 1 occurs for the value of the function to the right of θ, and the function then remains at the level 1 as shown in Fig. 6.5. In general, a step function can have a finite number of points at which jumps of equal or unequal size occur.

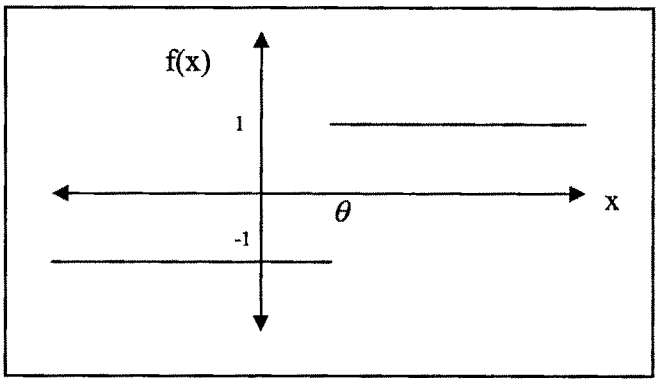

Fig. 6.5 The Step Function

(iii)The Ramp Function: To describe the ramp function simply, first consider a step function that makes a jump from 0 to 1 at some point. Instead of letting it take a sudden jump like that at one pint, let it gradually gain in value, along a straight line (looks like a ramp), over a finite interval reaching from an initial 1 to a final 1 as shown in Fig. 6.6.

(iv)Linear Function: A linear function is simply given as,

$$F(x) = \alpha x + \beta \qquad \ldots (6.3)$$

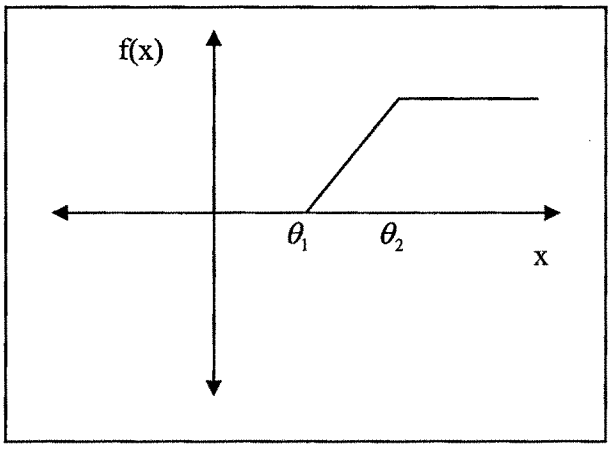

Fig. 6.6 The Ramp Function

When $\alpha = 1$ the application of this threshold function amounts to simply adding a bias equal to β to the sum of the inputs.

6.5. BACK PROPAGATION NEURAL NETWORK

6.5.1 The Concept

The Feed forward backpropagation network is a very popular model in neural networks. It does not have feedback connections, but errors are back-propagated during training. It learns

by Generalized Delta rule and Least Mean Squared Error is used for learning. Errors in the output determine hidden layer output errors, which are used as basis for adjustment of connection weights between input and hidden layers (Fig. 6.7). Adjusting the weights is the iterative process until the error is reached below tolerance level specified. Learning parameters scale the adjustments to weights. One can also use momentum parameter in the adjustment of weights from previous iteration and by keeping weight change process moving and thereby not get stuck in local minima. Also, noise can be added to inputs in order to get generalization ability.

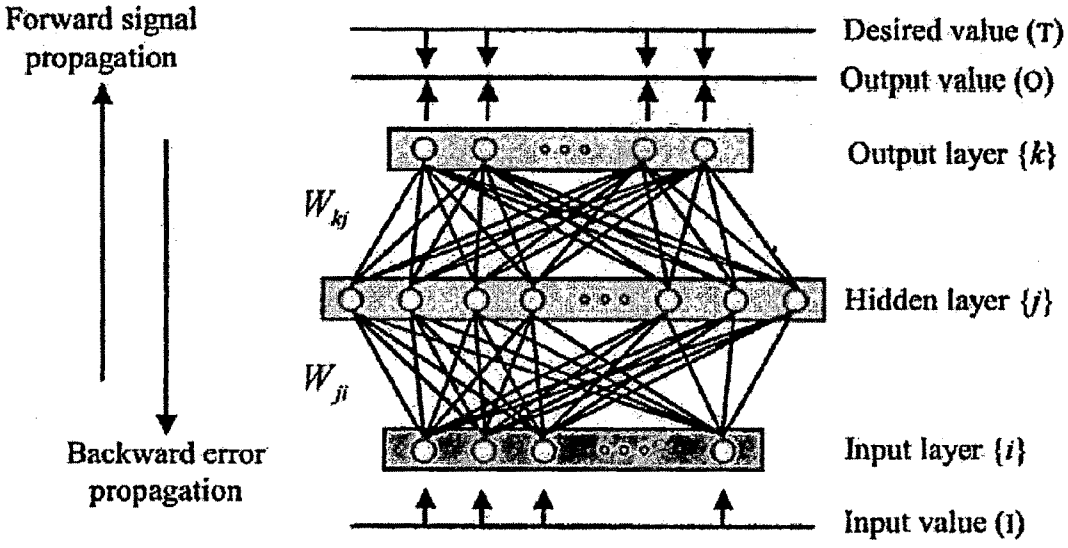

Fig. 6.7 Layout of Feed Forward Back Propagation Neural Network

6.5.2 Forward Signal Propagation

The algorithm starts with the first hidden layer using as input values the independent variables of a case from the training data set. The neuron outputs are computed for all neurons in the first hidden layer by performing the relevant sum and activation function evaluations. These outputs are the inputs for neurons in the second hidden layer. Again the relevant sum and activation function calculations are performed to compute the outputs of second layer neurons. This continues layer by layer until one reaches the output layer and computes the outputs for this layer.

6.5.3 Backward Error Propagation

Backward pass phase begins with the computation of error at each neuron in the output layer. A popular error function is the squared difference between O_k, the output of node k and Y_k,

the target value for that node. These errors are used to adjust the weights of the connections between the last-but-one layer of the network and the output layer. Then finally new weights are calculated. The backward propagation of weight adjustments along these lines continues until one reaches the input layer. At this time one has a new set of weights on which one can make a new forward pass when presented with a training data observation.

Consider network shown in Fig. 6.9 with following notations:

- N_i - Number of input nodes,
- N_j - Number of hidden nodes,
- N_k - Number of output nodes,
- W_{ji} - Connection weights between input and hidden nodes,
- W_{kj} - Connection weights between hidden and output nodes,
- θ_j - Threshold value for hidden layer,
- τ_k - Threshold value for output layer,
- x_i - Inputs applied,
- Y_j - Output at the hidden layer,
- O_k - Output of the output layer, and
- T_k - Target output of the output layer.
- P - Number of training patterns (input-output sets)

6.5.4 Step-by Step Procedure

Step-1: First randomly generate the weights between the input and hidden layers (W_{ji}) and hidden and output layers (W_{kj}). Also generate random threshold value for hidden layer (θ_j) and random threshold value for output layer (τ_k).

Step-2: Calculate activation of hidden layer node j. It is calculated as $\sigma_j = \sum_{i=1}^{N_i} w_{ji} x_i$.

Step-3: Sum up threshold value at hidden layer and activation of hidden layer as calculated above i.e. $s = (\sigma_j + \theta_j)$.

Step-4: Using sigmoid function, calculate output at hidden layer as follows: $Y_j = 1/(1 + e^{-s})$.

Step-5: Calculate complement of the hidden layer as, $COH_j = 1 - Y_j$.

Step-6: Calculate activation of output layer as, $\sigma_k = \sum_{j=1}^{N_j} w_{kj} y_j$.

Step-7: Sum up threshold value at output layer and activation of the output layer as calculated above $t = (\sigma_k + \tau_k)$.

Step-8: Using sigmoid function, calculate output at output layer as,

$O_k = 1/(1+e^{-t})$.

Step-9: Calculate complement of the output layer $COP_k = 1 - O_k$.

Step-10: Calculate difference between the calculated output and target output, $e_k = T_k - O_k$.

Step-11: Compute error at output layer as, $EO_k = O_k * COP_k * e_k$.

Step-12: Compute error at hidden layer as, $EH_j = Y_j * COH_j * \sum_{k=1}^{N_k} w_{kj} * EO_k$.

Step-13: Calculate correction in threshold as,

$$\Delta\theta_j = L.P * EH_j$$

$$\Delta\tau_k = L.P * EO_k$$

Step-14: Calculate correction in weights as,

$$\Delta w_{kj} = L.P * Y_j * EO_k$$

$$\Delta w_{ji} = L.P * x_i * EH_j$$

If momentum term is used, then carry out weight correction as,

$$\Delta w_{kj} = L.P * y_j * EO_k + momentum\_term * previous\_weight\_change$$

$$\Delta w_{ji} = L.P * x_i * EH_j + momentum\_term * previous\_weight\_change$$

Step-15: Calculate new threshold vector as,

$$\theta_j = \theta_j + \Delta\theta_j$$

$$\tau_k = \tau_k + \Delta\tau_k$$

Step-16: Calculate new weight matrix as,

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

$$w_{kj} = w_{kj} + \Delta w_{kj}$$

Step-17: Go to **Step-2.**

Step-18: Calculate root mean square error with the help of,

$$RMSE = \sqrt{\sum_{p=1}^{P}\sum_{i=1}^{N_k}\left(T_i^{(p)} - O_i^{(P)}\right)^2} \Big/ (Training\_patterns * output\_units)$$

Step-19: Repeat the process until termination criteria is satisfied i.e. either maximum number of cycles is reached or *RMSE* is less than error tolerance specified.

6.5.5 Design Considerations for BPN

While using feed forward backpropagation neural network approach, following are the important considerations:

1. For a given problem and architecture, the error surface in the multi-dimensional space of weights may be hilly and may have many local minima. The objective is to reach global minimum. If the global minimum is shallow, convergence may be slow.
2. If the weights are large, the output of neurons may have high or low limiting values, where the slope is zero and hence no weight changes are possible. So it is advisable to use small initial weights.
3. Learning rate should not be too large; otherwise solution may jump from one minima to other and may lead to slow convergence.
4. In incremental weight update scheme, for each pattern, weights are modified before presenting other pattern. While in case of batch update scheme, weights are adjusted only after completion of the iteration after presenting all the patterns.
5. For better convergence, often the patterns are presented in random fashion.
6. The input and target vectors may need some transformation so that variation within them is smooth. Sharply falling or rising curve cannot be learned properly by the BPN.
7. Training set should be comprehensive and should represent fully the dynamics to be learned.
8. From the known input-output pairs, 80% patterns are used for training whereas the remaining 20% sets are used to assess the capability of generalization.
9. It is necessary to ensure that the network is not over fitted. Number of weight parameters to be determined should be substantially smaller than the number of data points available.
10. In general, a 3-layer network can represent any complex function. In other words, network with one or two hidden layers can also give good convergence.
11. The number of neurons in the hidden layer must be judiciously chosen so that network may not loose the ability of generalization.