# 4
# Proposed Methodology using Artificial Neural Network and Generalized Neural Network Approach

## 4.1 INTRODUCTION

Over the past several years, neural networks (NN) have received a great deal of attention and proposed as a powerful computational tool. Neural Networks have abilities to recognize patterns and provide appropriate outputs. These are the strengths which make neural networks suitable for complex forecasting problems. The two key aspects of neural networks are the processing elements and their interconnections. Neural network resembles the brain in two respects [Martin and Araque, 2005]:

1. Knowledge is acquired from its internal network of neurons through a learning process.
2. Interneuron connection strengths (known as synaptic weights) are used to store the acquired knowledge.

They have been successfully applied in nonlinear problems like handwriting recognition, speech recognition, Robotics and load forecasting problems [Hippert *et al.,* 2001; S. Haykins *et al.,* 1999; Colter *et al.,* 1990; Hornik *et al.,* 1989]. A number of papers have been published in last two decades on applications of neural networks for forecasting. Neural network-based models have following inherent properties [Martin and Araque, 2005]:

- Inherent parallelism
- Similarity of components
- Access to locate information
- Incremental learning

In neural network-based models, parallelism in the execution has always been observed. Any particular node's output will depend exclusively on its current states. All the parameters of neural networks undergo several small changes over time, would settle on to their final values [Chaturvedi *et al.,* 2008]. In last two decades several papers have been published and reported that multilayered feed-forward neural network models perform better compared to conventional regression models like Auto-Regression (AR), Moving Average (MA), Auto-Regression Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) [ John *et. al.,* 2016] and other time series models. Above discussion shows that artificial neural networks (ANN) offer a reasonable alternative to the classical methods of load forecasting. ANN technology provides a more general framework to solve regression analysis, time series prediction and classification problems than other classical statistical methods.

Although, ANNs perform better than other time series models, they have their own limitations. To overcome some of the problems of ANN and to improve performance [Chaturvedi *et al.,* 2008] proposed generalized neural networks. Generalized Neural Network is proposed for load forecasting problems and to improve the efficiency of forecasting accuracy [Chaturvedi *et al.,* 2008]. In this chapter, we discuss the methodology of forecasting using Artificial Neural Network (ANN) and Generalized Neural Network (GNN).

## 4.2 ARTIFICIAL NEURAL NETWORK

The first artificial neuron was introduced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pitts (M&P). It is also a mathematical model of a neuron. Any linear problem can be represented as logic function and can be modeled by an M&P neuron model. These logic functions are basically OR, AND and NOT. With the help of these logic functions, one can easily implement any Boolean function. Because of some limitations, weights and threshold are analytically determined (cannot learn), very difficult to minimize the size of a network. The first learning rule was developed in 1949 [Hebb *et al.,* 1949].

A few years after, Frank Rosenblatt (1951) developed a learning rule that could be useful to decide if a sample belongs to one class or the other. A simple neuron model with learning rule is called a Perceptron. This learning algorithm (weight adjustment) was used in M&P model and it was found that it performed better than the learning rule proposed by [Hebb *et al.,* 1949].

In 1960 Widrow and Hoff developed the least mean square (LMS) learning algorithm for M & P neural network model which is called Adaptive Linear Neuron (ADALINE) model. Windrow-Hoff (delta rule) rule of the ADALINE updates the weights based on a linear activation function rather than a unit step function. ADALINE model is a linear model so it has some limitation in its ability to process data, and is capable of any mapping that is linearly separable.

Dr. Herbert Simon, Allen Newell and Cliff Shaw also discuss the neural network approach in the early 1950s. Over the past few years, the artificial neural network (ANN) has received a great deal of attention and is proposed as a powerful computational tool. In 1969 it is reported that perceptron model is not capable of learning those functions which are not linear in nature [Minsky and Papert, 1969].

In the 1970s to 1980s, neural network research declined because of perceptron model was not able to learn non-linear functions. Then the researchers also developed a back propagation learning algorithm for a multilayered network that could provide a method to solve non-linear functions. They have been successfully applied in pattern recognition classification and nonlinear control problems [Fausett *et al.,* 1994].

A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. It has been demonstrated that multi-layered ANN are universal approximators and they are able to approximate any nonlinear continuous function up to the desired level of accuracy.

The main advantages of the neural network models are given below:

i. Adaptive learning
ii. Self-organization
iii. Real-time operation
iv. Fault tolerance

They are extremely versatile and need to fulfill a specification of any particular model. They can perform better with chaotic components of the non-linear problems compare to traditional methods [Master *et al.,* 1995]. These are some of the reasons that artificial neural networks are widely used in the field of finance, consumption, medicine, meteorology and electrical power system application [Palmer, 2008].

### 4.2.1 Development of Artificial Neural Network

An artificial neural network model is a biologically inspired model and mimics the human brain. In Figure 4.1 shows the direction of the information in the human nervous system and it can be broken down into three main stages where the main task for the receptors is a collection of information from the environment. The effectors generate interactions with the environment, for example, activate muscles.

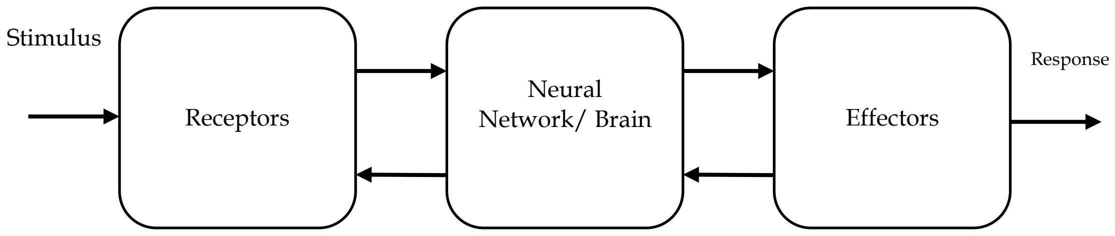

**Figure 4.1:** Block diagram of A Human Nervous System

In this system, the brain plays an important role in processing of information via biological neurons, having a large number (approximately $10^{11}$) of interconnected elements (approximately $10^4$ connections per element). This complete network is called neural network. Figure 4.2 shows the biological neuron and, it consists of three main components cell, body or soma, dendrites or axon. In a biological neural network, dendrites receive the information and information processed in the soma. The final processed information is transferred via the axon.

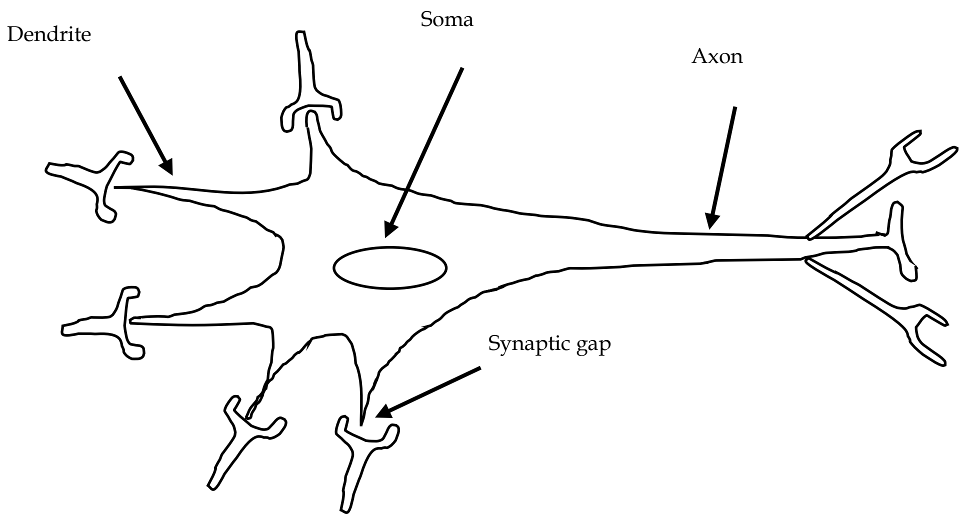

**Figure 4.2:** Structure of biological neuron structure

ANNs have self-adapting capabilities which make them suitable to handle non-linearities, uncertainties, and complexity which occurs in forecasting problems. Data processing of a neuron is shown in Figure 4.3.

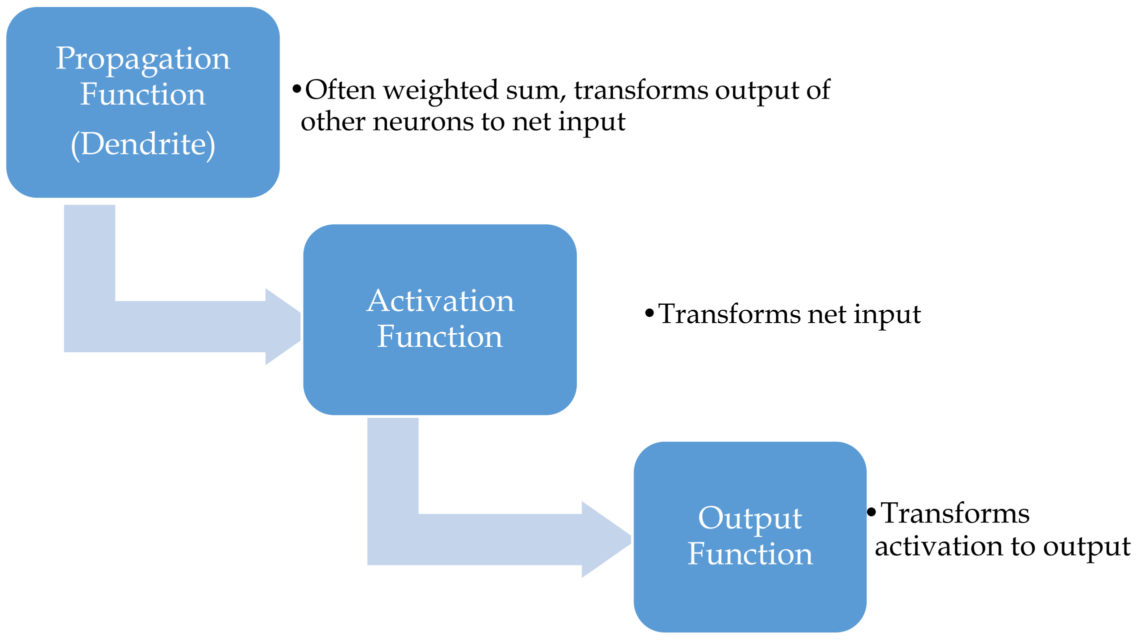
Propagation Function
(Dendrite)
•Often weighted sum, transforms output of other neurons to net input
Activation Function
•Transforms net input
Output Function
•Transforms activation to output

**Figure 4.3:** Data processing of a neuron

### 4.2.2 Basic Element of Artificial Neuron

The basic structure of an ANN consists of processing elements, called neurons, which are fully interconnected with one-way signal channels called connections. Each input is multiplied by a corresponding weight, which is analogous to biological neuron's synaptic strength. All of the weighted inputs are then summed to determine the activation level of the neuron. The flow of information/activation is represented by arrows. Figure 4.4 represents a structure of artificial neuron network with its inputs, weights, activation function, and outputs [Laurene, 1994; Chaturvedi, 2005].

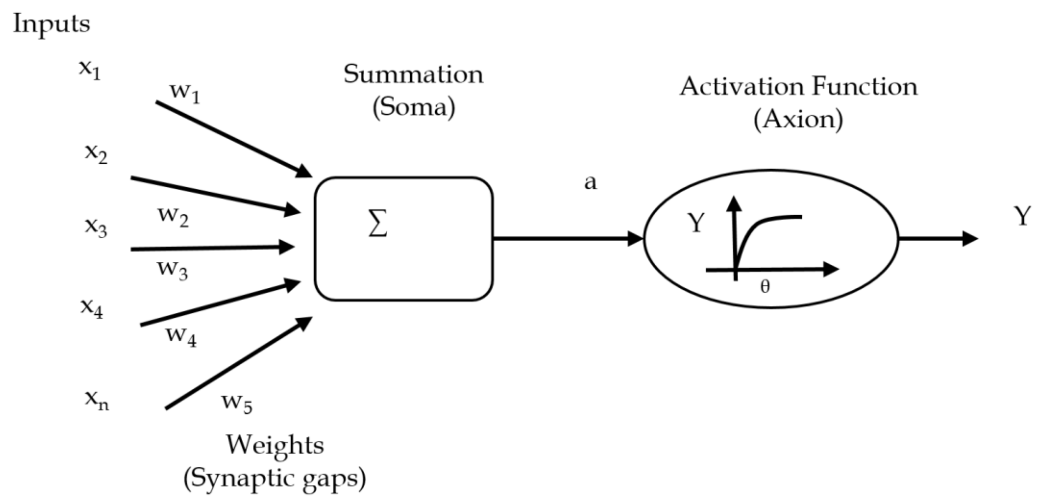

**Figure 4.4:** Artificial neuron structure

***Weight Factor (W):***

A set of input observations X=[$x_1$, $x_2$… $x_n$] are associated with its weights $w_1$ , $w_2$, .., $w_n$. Weighted sum of its each input parameters is processed by summation block through 'an' activation. The '+ve' and '-ve' weight excites and inhibits the output of each node.

$$a = x_1 w_1 + \cdots + x_n w_n + \theta \tag{4.1}$$

***Threshold ($\theta$):***

Internal threshold ($\theta$) of the node is the magnitude offset. It affects the output of a node which is represented as:

$$a = \sum_{i=1}^{n} x_i w_i + \theta \tag{4.2}$$

Then final output Y is a function of processed output from activation block and it may be written as:

$$Y = f(a) \tag{4.3}$$

For computing the final output $Y$, the node output is passed on to a non-linear filter which is $f$ and it is called transfer function or activation function which releases the final output $Y$.

$$Y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right) \tag{4.4}$$

***Threshold of a Neuron:***

In computation, neurons usually do not activate unless and until their total input goes above a threshold value. Total input for every neuron is the weighted sum of its inputs plus its threshold value then its passed through the activation function. For example if activation function ($a_f$) is sigmoidal function then it can represented as:

$$a_f = \frac{1}{1 + \exp^{(-a)}} \tag{4.5}$$

Where;

$$a = \sum_{i=1}^{n} x_i w_i + \theta \tag{4.6}$$

$$a_f = activation\ function$$

$$\theta = value\ of\ threshold$$

***Activation Function:***

The activation rate of the action potential is modeled using activation function (transfer function). It is a mathematical operation on a signal output. It helps in activating the neuron and calculates the final output using the following activation functions. In the last two decades researchers introduced some linear and non-linear activation functions which are helpful for nonlinear and complex problem shown in Figure 4.5 [Laurene, 1994].

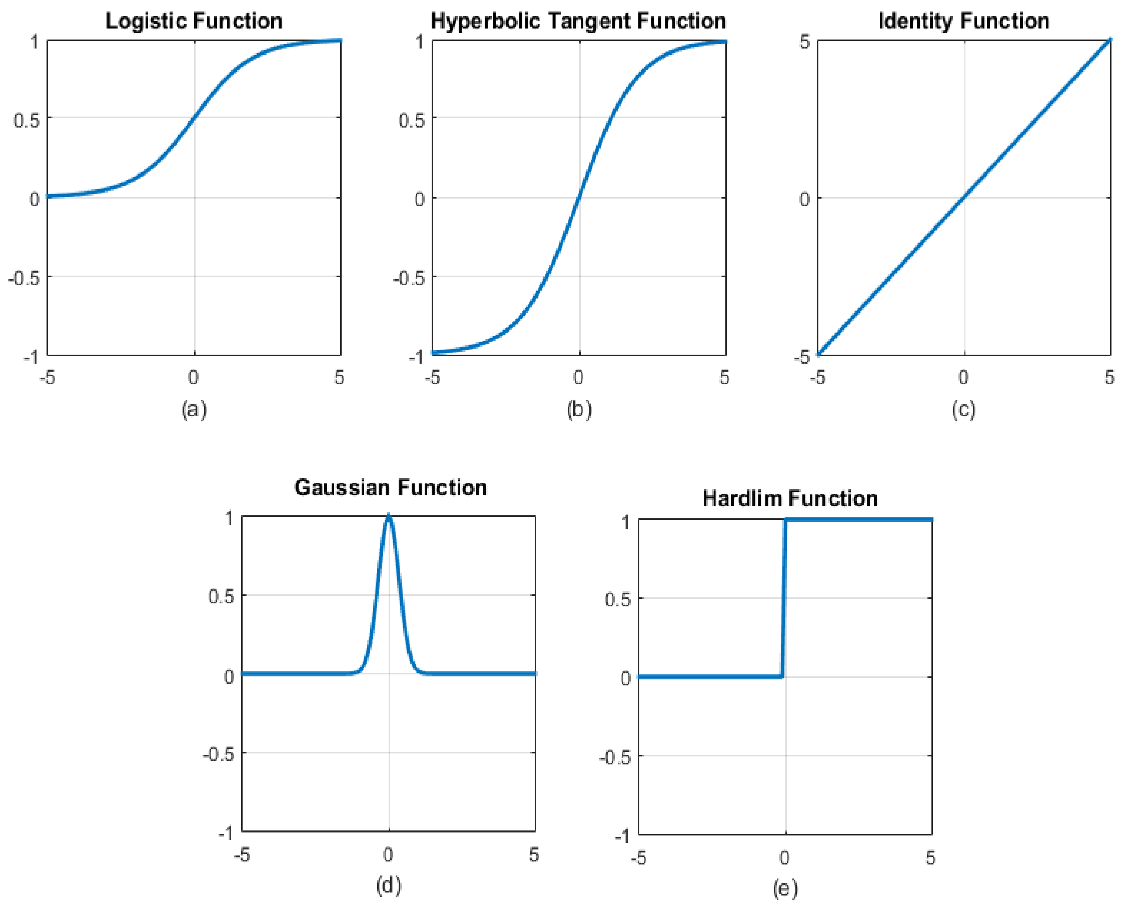

**Figure 4.5:** Outcome of different activation functions

The most common activation functions are:

- Logistic Function
- Hyperbolic Tangent Function
- Identity Function
- Gaussian Function
- Hard limit Function

In Table 4.1 Mathematical expressions of these activation functions are given [Laurene, 1994].

**Table 4.1:** Mathematical equations of different activation functions

| S. No. | Activation Function | Mathematical Equation | Limit |
|---|---|---|---|
| 1. | Sigmoidal | $a = \frac{1}{1 + e^{(-x)}}$ | (0, 1) |
| 2. | Hyperbolic tangent | $a = \left(\frac{1 - \exp(-x)}{1 + \exp(-x)}\right)$ | (-1, 1) |
| 3. | Identify (Linear) | $a = x$ | (-∞, ∞) |
| 4. | Gaussian | $a = \exp(-x)^2$ | (0, 1) |
| 5. | Hardlimit (Step) | $a = f(x)$ | (0, 1) |

### 4.2.3 Architecture of Artificial Neural Network

Neural Networks are generally classified into two main categories: Feed–Forward Neural Network (FFNNs) and Recurrent Neural Network. FFNNs do not have any feedback loop in the network. The flow of Information is only in the forward direction. Present input is responsible for neural network activation. In a recurrent neural network, feed forward neural network with output is fed back to the input. In this section, we discuss different types of neural network architectures.

**i. Single layered feedforward neural network:**

In a feed forward layered neural network the neurons are organized in the form of layers. In a single layered feed-forward neural network (SLFFNN) source nodes are in the input layer. In a network, 'single layer' refers to the output layer of computation nodes. There is only one input and one output layer. The input layer is not counted as a layer since no mathematical calculations takes place at this layer [Cornelius *et al.,* 1990]. This is shown in Figure 4.6.

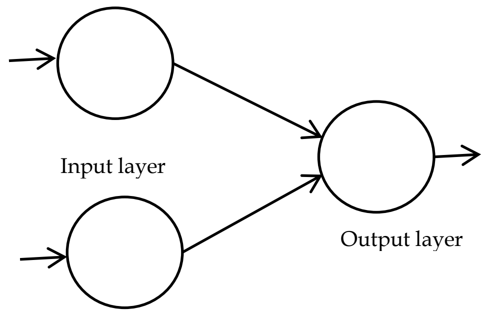

**Figure 4.6:** Single layered feed-forward neural network

**ii. Multi-layered feed forward neural network:**

A multi-layered feed forward neural network has one or more hidden layers as shown in Figure 4.7. The computational nodes are hidden neurons. The output signal of the second layer is used as inputs to the third layer for the rest of the network [Hornik *et al.,* 1989].

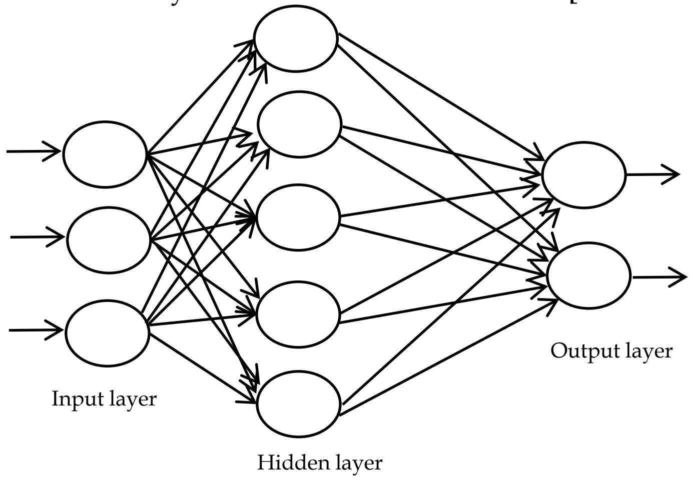

**Figure 4.7** : Multi--layered feed-forward neural network

**iii. Recurrent networks**

A recurrent network consists of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons at input layer. It has at least one feedback loop. Self-feedback refers to a situation where the output of a neuron is fed back into its own input, shown in Figure 4.8. The presence of feedback loops has a profound impact on the learning capability of the network and on its performance [Senjyu *et al.,* 2006].

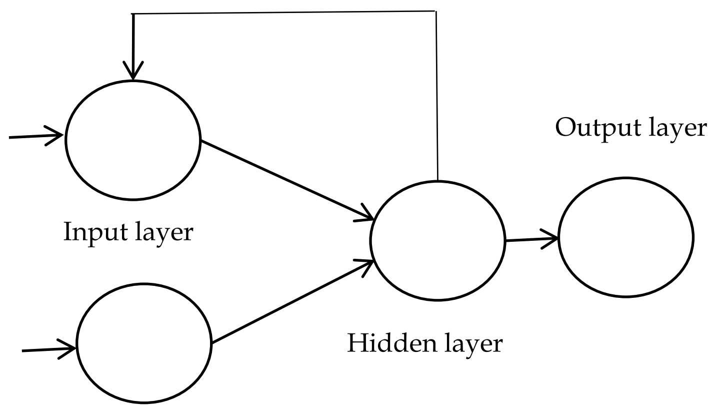

**Figure 4.8** : Recurrent neural network

The idea is that the input-output relationship in an ANN model provides more flexibility and can accommodate an increased level of complexity. As far as the question of complex and dynamic problem like forecasting is concerned, the main concern is to improve the accuracy of forecasting procedures. Artificial Neural Networks can be one of the tools for improving the accuracy of forecasting. This gives the idea that accuracy of solar forecast may also be further improved by modifying the conventional artificial neural network itself, as far as possible. Very few researchers have considered changes in the internal structure of a neuron of ANN [Chaturvedi *et al.,* 2002].

### 4.2.4 Learning Paradigms of Artificial Neural Network

In the computation of neural network model, the learning process plays an important role. A learning process is a method of adaptation in which computing units self-organize to implement the desired behavior and also systematically changes the weight matrix which makes the network capable of performing a useful task by understanding the internal structure of the data.

- ***Back Propagation Learning***

Back-Propagation (BP) with multi-layered feed forward neural network (MLFFNN) is one of the most recognized neural network architecture due to its ability of non-linear mapping. Amari( 1967) proposed gradient descent to the training of MLFFNNs using a single hidden layer to solve a non-linear problem. Dynamic feedback and Learning logic are published by Werbos and Parker in 1974 and 1987 respectively. After decades, Rumelhart, Hilton, and Williams published the concept of back-propagation algorithm, which shows its great impact on the complex and dynamic problems. The combination of the input set and target set is called training pair. Following steps are required for training of back-propagation [Chaturvedi D. K. *et al.,* 2008] as shown in Figure 4.9.

(a) Choose the training pair from the training set; apply the input vector to the network input.
(b) Compute the output of the network.
(c) Compute the error between the network output and the desired output.
(d) Adjust the weights of the network in a way that minimize the error.
(e) If an error is not acceptable then repeat from step (a) to step (e).

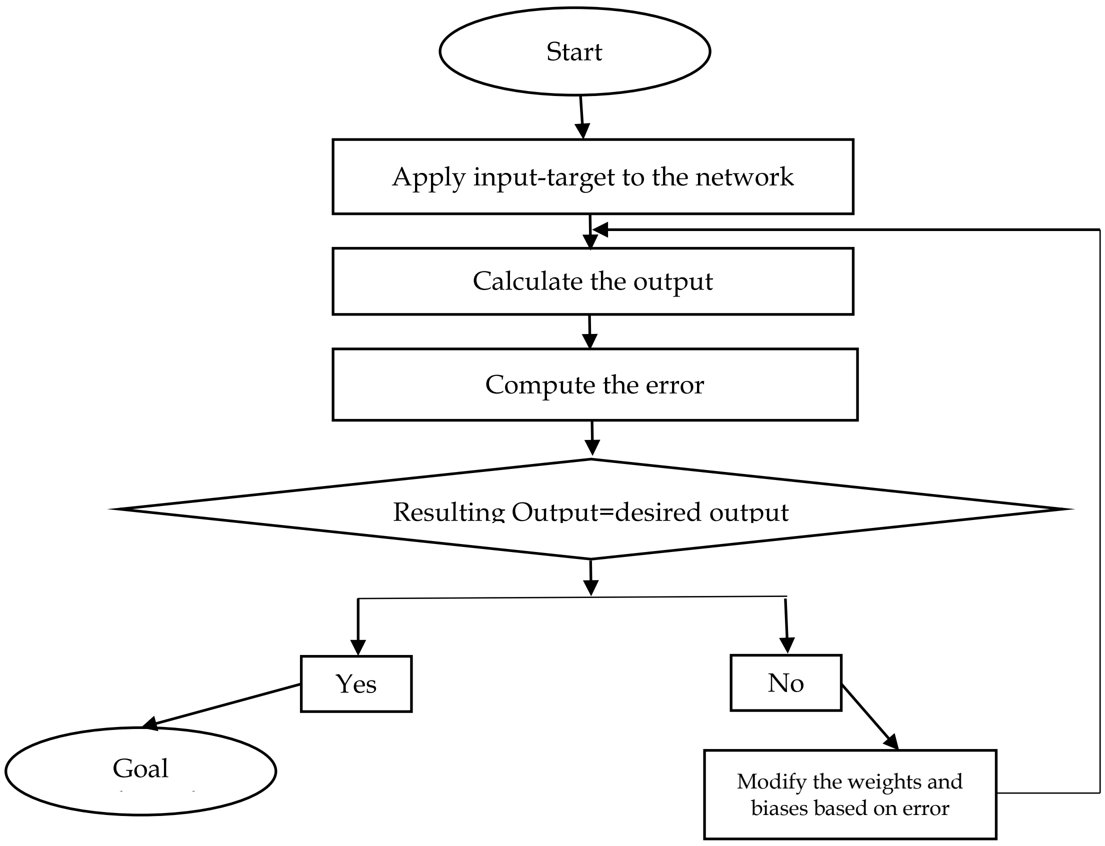
Start
Apply input-target to the network
Calculate the output
Compute the error
Resulting Output=desired output
Yes
No
Goal
Modify the weights and biases based on error

**Figure 4.9 :** line diagram for a supervised back-propagation learning

***Back-propagation Algorithm:***

The main objective of back propagation is to obtain the global minimum on the error surface. Gradient descent algorithm helps in finding the global minimum and network weights which are adjusted according to the steepest downhill slope. The algorithm finds the nearest local minima using many small steps and the possibility to find the global minima is shown in Figure 4.10.

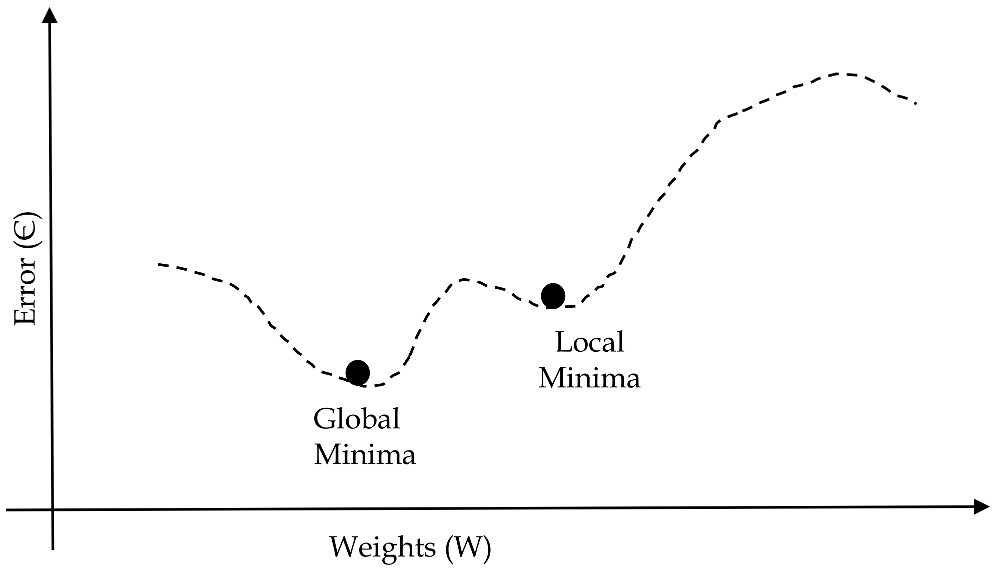

**Figure 4.10 :** Error Curve with respect to weight

Learning is divided into following four main steps [Kriesel, 2007]:

(a) Feed Forward computation
(b) Back Propagation to the output layer
(c) Back Propagation to the hidden layer
(d) Weight updates

The derivation of back-propagation is given below;
Notation:

**Table 4.2:** Notation used in back propagation

| Notation | Description |
|---|---|
| $E(n)$ | Instantaneous sum of error squares at iteration |
| $e_j(n)$ | Error signal at output of neuron j for iteration |
| $d_j(n)$ | Desired response for neuron j used to compute $e_j(n)$ |
| $y_j(n)$ | Activation signal appearing at the output of neuron j for iteration n |
| $w_{ij}(n)$ | Synaptic weight connecting neuron I to neuron j at iteration n |
| $\Delta w_{ij}(n)$ | The correction applied to the synaptic weight at iteration n |
| $v_j(n)$ | The net internal activity level of neuron j at iteration n |
| $\varphi_j(.)$ | The activation function associated with neuron j |
| $\theta_j$ | Threshold value |
| $x_i(n)$ | The ith element of the input vector |
| $o_k(n)$ | The kth element of the overall output vector |
| $\eta$ | Learning rate |
| $\delta_j(n)$ | Local gradient |

The error signal at the output of neuron j at iteration

$$e_j(n) = d_j(n) - y_j(n) \tag{4.7}$$

Here, j is an output node and instantaneous sum of squared error of the network at the output of neuron j can be written as:

$$E(n) = \frac{1}{2}\sum_{j \in c} e_j^2\,(n), \tag{4.8}$$

where, c is a set of all neurons in the output layer of the network.
If N is the total number of input from training set, the average squared error is given by:

$$E_{av} = \frac{1}{n}\sum_{n=1}^{N} E(n) \tag{4.9}$$

Here, $E_{av}$ indicates the cost function of the training process, which modifies the free parameters of weights and value of threshold and errors calculated for each pattern shown into the network Figure 4.11.

$$v_j(n) = \sum_{i=0}^{P} w_{ij}(n) y_i(n) \tag{4.10}$$

Here p indicates the total number of inputs excluding the threshold applied on neuron j.

$$y_j(n) = \varphi_j(v_j\,(n)) \tag{4.11}$$

The learning process applies a correction $\Delta w_{ij}$ (n) to weights $w_{ij}$ (n), which is proportional to gradient $\frac{\partial E(n)}{\partial w_{ij}}$. We may express the gradient on the basis of chain rule of partial derivatives as below [Rojas, 1996] .

$$\frac{\partial E(n)}{\partial w_{ij}} = \frac{\partial E(n)}{\partial e_j(n)} * \frac{\partial e_j(n)}{\partial y_j(n)} * \frac{\partial y_j(n)}{\partial v_j(n)} * \frac{\partial v_j(n)}{\partial w_{ij}(n)} \tag{4.12}$$

Using equation (4.7), (4.8),(4.9) & (4.11) we get,

$$\frac{\partial E(n)}{\partial w_{ij}} = -e_j(n)\varphi'_j(v_j\,(n))y_j(n) \tag{4.13}$$

After correction of the weights using delta learning rule

$$\Delta w_{ij} = -\eta \frac{\partial E(n)}{\partial w_{ij}(n)} \tag{4.14}$$

Where η positive constant is called the learning rate, from Eq. (4.13-4.14).

$$\Delta w_{ij} = \eta \delta_j(n) y_i(n) \tag{4.15}$$

Where $\delta_j(n)$ is called the global gradient at neuron j and it is an equal to the product of corresponding error signal $e_j(n)$ and the derivatives $\varphi'_j(v_j\,(n))$ of the associated activation function.

In back propagation error correction and weights, adjustments are the key factor so there are two cases of adjustment given below [Krose, 1996; Kriesel, 2007].

(a) **When Neuron j is an output node**: With the help on Eq.(4.7) to calculate error signal $e_j(n)$ which is associated with the neuron j and from Eq.(4.15) we can calculate the local gradient.

**(b) When Neuron j is a hidden node:** In this case, when neuron j is located at the hidden layer, there is no specific desired response for that neuron j shown in Figure 4.11. Local gradient can be defined as:

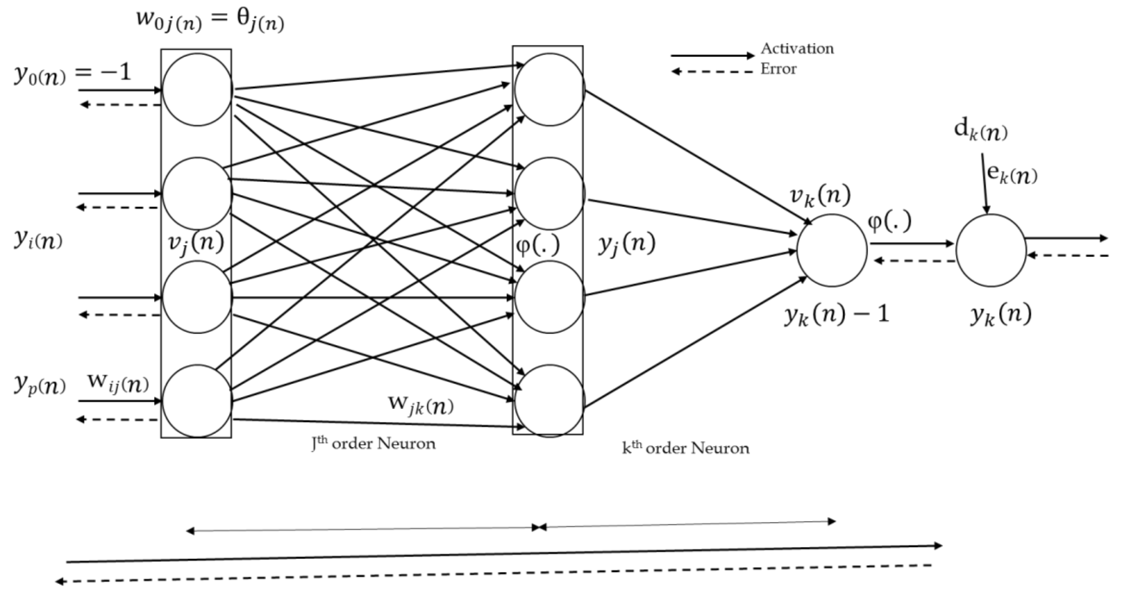

**Figure 4.11 :** Signal flow in multi-layered neural network

$$\delta_j(n) = e_j(n)\varphi'_j(v(n)) \tag{4.16}$$

$$\delta_j(n) = \frac{\partial E(n)}{\partial y_j(n)} * \frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial E(n)}{\partial y_j(n)} * \varphi'_j\left(v_j(n)\right) \tag{4.17}$$

To calculate the partial derivative $\frac{\partial E(n)}{\partial y_j(n)}$, the average error is:

$$E(n) = \frac{1}{2}\sum_{k \in c} e_k^2(n) \tag{4.18}$$

Neuron k is an output node

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} \tag{4.19}$$

Using the chain rule of partial derivatives we can rewrite the Eq.(4.19) as:

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)} * \frac{\partial v_k(n)}{\partial y_j(n)} \tag{4.20}$$

So, Computed error is:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n)) \tag{4.21}$$

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k\left(v_k(n)\right) \tag{4.22}$$

The internal output of neuron k is:

$$v_k(n) = \sum_{j=0}^{q} w_{jk}(n) y_{j(n)} \tag{4.23}$$

Where q = a total number of input applied to neuron k.
After differentiating Eq.(4.23) with respect to $y_j(n)$ then yield is:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{jk}(n) \tag{4.24}$$

So, using Eq.(4.22) and Eq.(4.24), we get:

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n)\varphi'_k(v_k(n))w_{jk}(n) = -\sum_k \delta_k(n)w_{jk}(n) \tag{4.25}$$

So, finally, the local gradient for the hidden neuron j is [Hagan, 1996; Laurene, 1994] :

$$\delta_j(n) = \varphi'_j\left(v_j(n)\right)\sum_k \delta_k(n)w_{jk}(n) \tag{4.26}$$

So, it can summarise the relations as follows [Krose, 1996; Kriesel, 2007]:

$$\left(Weight\ Correction(\Delta w_{ij}(n)\right) = (learning\ rate\ parameter\ (\eta))\left(Loacal\ gradient\left(\delta_j(n)\right)\right) * \left(Input\ signal\ of\ neuron\ j(y_i(n))\right)$$

**The rate of learning and momentum:** The BP algorithm helps to compute the approximation and error-weight space (which is computed by steepest descent). According to this, weights are modified with respect to error surface. During the learning process, the rate of learning η is adjusting the gradient of the error surface which is used for weight adjustment. The smaller learning rate can provide the smoother trajectory in error–weight space. If it is too large then trajectory will be unstable and oscillatory.
Here one another method to increasing the learning rate:

$$\Delta w_{ij}(n) = \alpha\, \Delta w_{ij}(n-1) + \eta\delta_j(n)y_i(n) \tag{4.27}$$

Where $\alpha$ =momentum constant.

**The stopping criteria of BP algorithm:** It is a process for improving generalization of the algorithm. The complete data set is divided into three parts, the first one is called training set, which can help to calculate the gradient and updating the network weights. The second subset of data is the validation set and it is monitored during the training process. If a validation error is increased after a number of iteration then training of network is stopped [Krose, 1996].

### 4.2.5 Development of forecasting model using artificial neural network model

Following steps are necessary to develop forecasting model using artificial neural network as shown in Figure 4.12.

*a)* Selection of input parameters
*b)* Selection of neural network
*c)* Selection of perfect training algorithm
*d)* Selection of training parameter

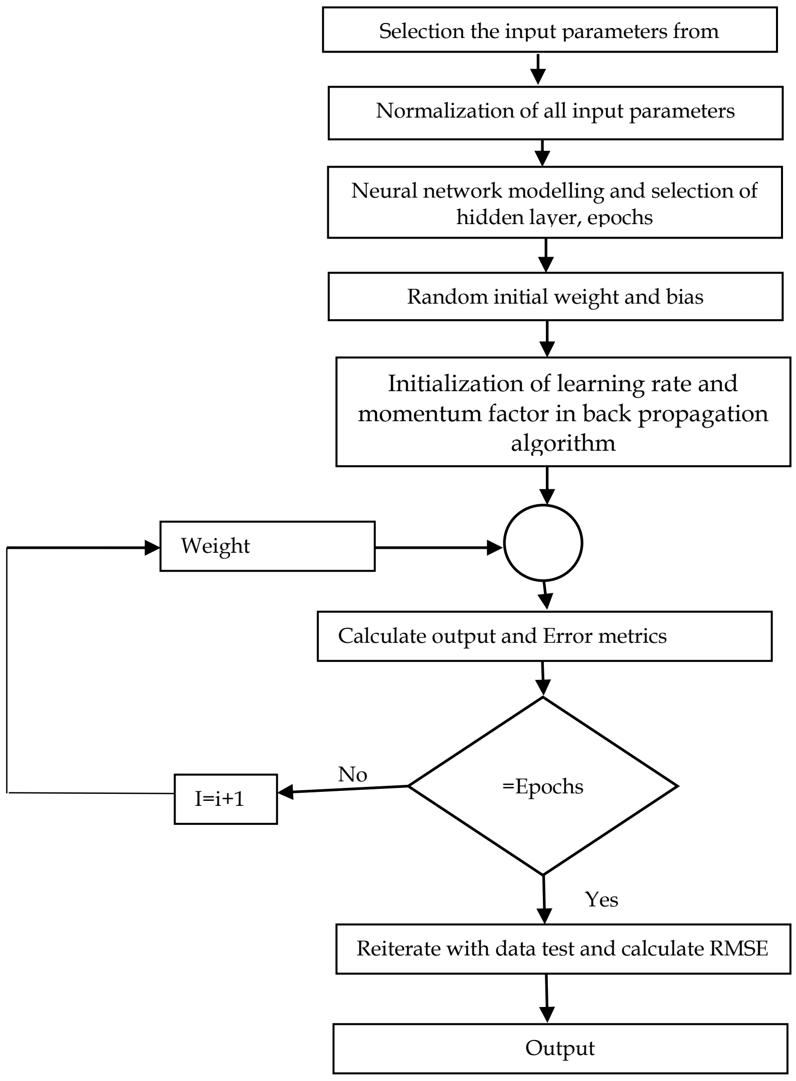
Selection the input parameters from
Normalization of all input parameters
Neural network modelling and selection of hidden layer, epochs
Random initial weight and bias
Initialization of learning rate and momentum factor in back propagation algorithm
Weight
Calculate output and Error metrics
No
I=i+1
=Epochs
Yes
Reiterate with data test and calculate RMSE
Output

**Figure 4.12** Schematic diagram of neural network methodology

***a) Selection of Input Parameters***

Selected input parameters have been used for training, testing, and validation of the model. Here in forecasting models following parameters are taken as input parameters:

- Global horizontal irradiation
- Global tilted irradiation
- Ambient temperature
- Module temperature
- Sun availability

For the 5 MW plant, these are daily average. For the rooftop plant, there are average values of 15-minute time interval.

***b) Selection of Neural Network***

In this paper, multilayered feed-forward neural network is used for solar power forecasting modeling. It is having more than two layers and all the layers are adaptive and there is no cyclic process from later layers back to earlier layers, therefore the name of the network is called "feed-forward". Learning of neural network is defined as any progressive systematic change in the memory (weight matrix) and can be supervised or unsupervised. Supervised learning is used in multi-layered net.

The neural network selected has five nodes in the input layer, six nodes in the hidden layer and a single node in the output layer. All hidden neurons' outputs are connected with single node in the output layer. Each hidden or output neuron of a multi-layered perceptron is designated to perform following two computations [Chaturvedi *et al.,* 2008].

1. The function signal appearing at the output of a neuron which is expressed as a continuous non-linear function of the input signals and synaptic weights.
2. The gradient of the error surface with respect to the weights connected to the inputs of a neuron, which is needed for the backward pass through the network.

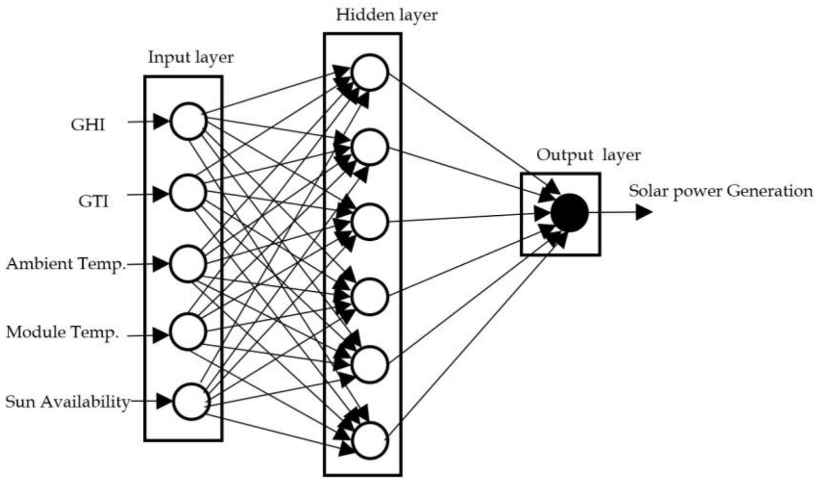

**Fig. 4.13 :** Structure of ANN model

And following the specific selection of neurons and layers provides a better result.

**Table 4.3: Structure of Neural Network**

| S. No | Network parameters | Value |
|---|---|---|
| 1. | Number of input variables | 5 |
| 2. | Number of outputs | 1 |
| 3. | Number of input layers | 1 |
| 4. | Number of hidden layers | 1 |
| 5. | Number of Hidden layer neurons | 6 |

### *c) Selection of Perfect Training Algorithm*

In this work, a back-propagation training algorithm is used for learning of network. The steepest descent minimization method is a back-propagation algorithm. It directly helps in the adjustment of the weight and threshold coefficients. Figure 4.14 shows the signal flow of input parameter toward to hidden and output layer. The successive adjustments to the weights are in the direction of the steepest descent of the error surface. In hidden layer, every node calculated the weighted sum of its inputs to form its scalar net activation, which is denoted simply as a net. Net activation refers to the inner product of the inputs with the weights at the hidden unit. Net activation at hidden layer to output layer can be written as in Eq.(4.28-4.31) [Laurene, 1994]:

Where,

$X_d = Input\ pair$
$d = Number\ of\ net$
$j = hidden\ unit$

$$net_j^d = \sum_{k=1}^{5} w_{jk}^{(1)}\, x_k^d \tag{4.28}$$

$$V_j^d = f(net_j^d) = f\left(\sum_{k=1}^{5} w_{jk}^{(1)}\, x_k^d\right) \tag{4.29}$$

$$net_i^d = \sum_{j=1}^{3} w_{ij}^{(2)} V_j^d = \sum_{j=1}^{3}\left(w_{ij}^{(2)} . f\left(\sum_{k=1}^{5} w_{jk}^{(1)}\, x_k^d\right)\right) \tag{4.30}$$

$$O_i^d = f\left(net_i^d\right) = f\left(\sum_{j=1}^{j} w_{ik}^{(2)} V_j^d\right) = f\left(\sum_{j=1}^{j}\left(w_{ij}^{(2)} . f\left(\sum_{k=1}^{n} w_{jk}^{(1)}\, x_k^d\right)\right)\right) \tag{4.31}$$

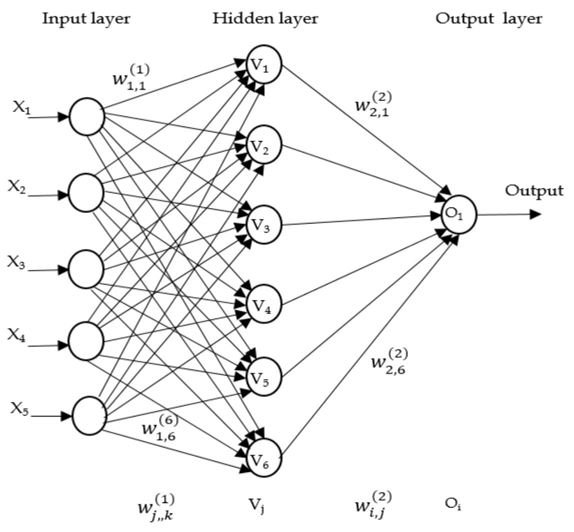

**Fig. 4.14:** Signal flow of the network for computation

In pre-processing stage transfer functions plays an important role in the network. In this work, the input variables are normalized in the range of 0.1 to 0.9 so here log sigmoidal function is used as activation (transfer) function as it generates output in the range 0 to 1. In a multi-layered network, the log-sigmoid transfer function is commonly used and it is trained by the back-propagation algorithm because this function is differentiable. The log sigmoidal function expressed by Eq.(4.32)- (4.33) and shown in Figure 4.16 [Laurene, 1994].

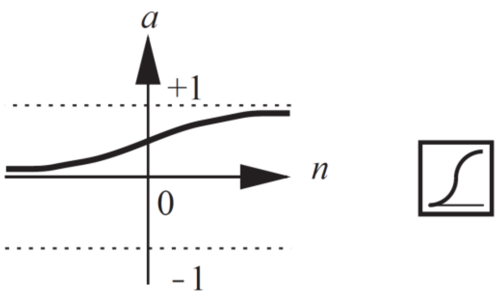

**Fig. 4.15:** Log-Sigmoid Transfer Function

Where

$$a = f(net_j) = logsig(n) \quad (4.32)$$

$$a = f(net_j) = \frac{1}{1+e^{-n}} \quad (4.33)$$

Advantages of proposed back propagation algorithm (Arbib *et al.,* 2003) are:

- Scale the data.
- Direct input–output connections.
- Vary the sharpness (gain) of the activation.
- Use of a different activation.
- Use of better algorithms.

***d) Selection of training parameter***

Back-propagation algorithm involves the use of learning rate, momentum factor because for weight adjustment the rate of learning 'η' decides the scaling of the gradient of the error surface. Here momentum factor is used for the danger of instability in network training and it provides stability into the learning process. In back propagation algorithm the weight adjustment equation is written as in Eq.(4.34) [Laurene, 1994]:

$$W_{new} = W_{old} + \Delta W \quad (4.34)$$

*where,*

$$\Delta W(k) = -\eta * \frac{\delta E_{ss}}{\delta W} + \alpha * \Delta W(k-1) \quad (4.35)$$

$\eta$ = rate of learning
$\alpha$ = momentum factor
$E_{ss}$ = used error function

When the error of validation set reaches a minimum, then network training stops. Error tolerance specifies how close the output value must be to the desired value.

**Table 4.4** Values of training parameters

| S. No. | Parameters | value |
|---|---|---|
| 1. | Number of epochs | 1000 |
| 2. | Error tolerance | 0.001 |
| 3. | Learning rate | 0.9 |
| 4. | Momentum factor | 0.3 |

## 4.3 Generalized Neural Network

In this section generalization of the conventional neural network is discussed which can overcome some of the drawbacks and improve the accuracy of the artificial neural network. In a generalized neural network, the simple neuron is transformed into the generalized neural network. Artificial neural network is commonly constructed using summation (∑) Aggregation function. This has been improved to obtain a generalized neural network model using fuzzy compensatory operators as aggregation operators as proposed by [Mizumoto, 1989] to overcome large number of neurons and layers that are required for nonlinear, dynamic and complex function approximation problems.

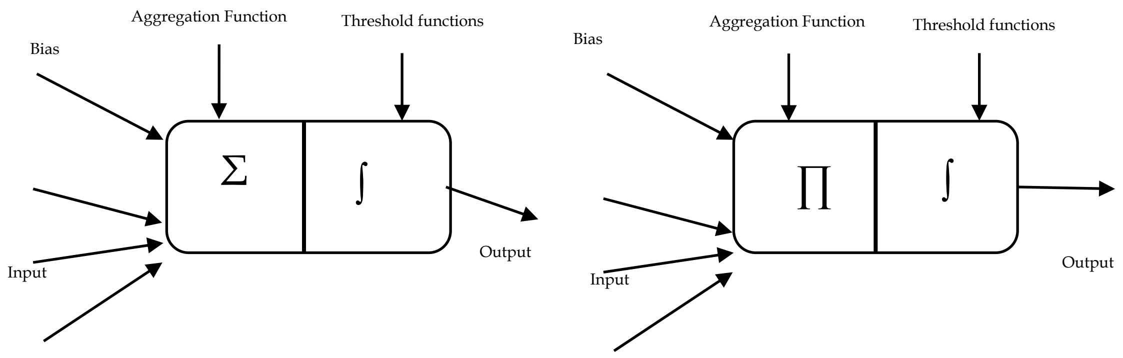

**Figure 4.16:** Simple summation and product type neuron model

Generally, the conventional neural network is constructed by aggregation function with linear or non-linear activation function is shown in Figure 4.16.

### 4.3.1 Conventional Neural Architecture

On the basis of variation in aggregation functions neurons are classified into two major categories which are given below:

(a) Summation Neuron (∑-type)

(b) Product Neuron (∏-type)

***(a) Summation Neuron (∑-type):*** Generally summation neuron is used in a conventional neural network where summation function is used as an aggregation stage and non-linear functions are used as activation stage which is shown in Figure 4.17.

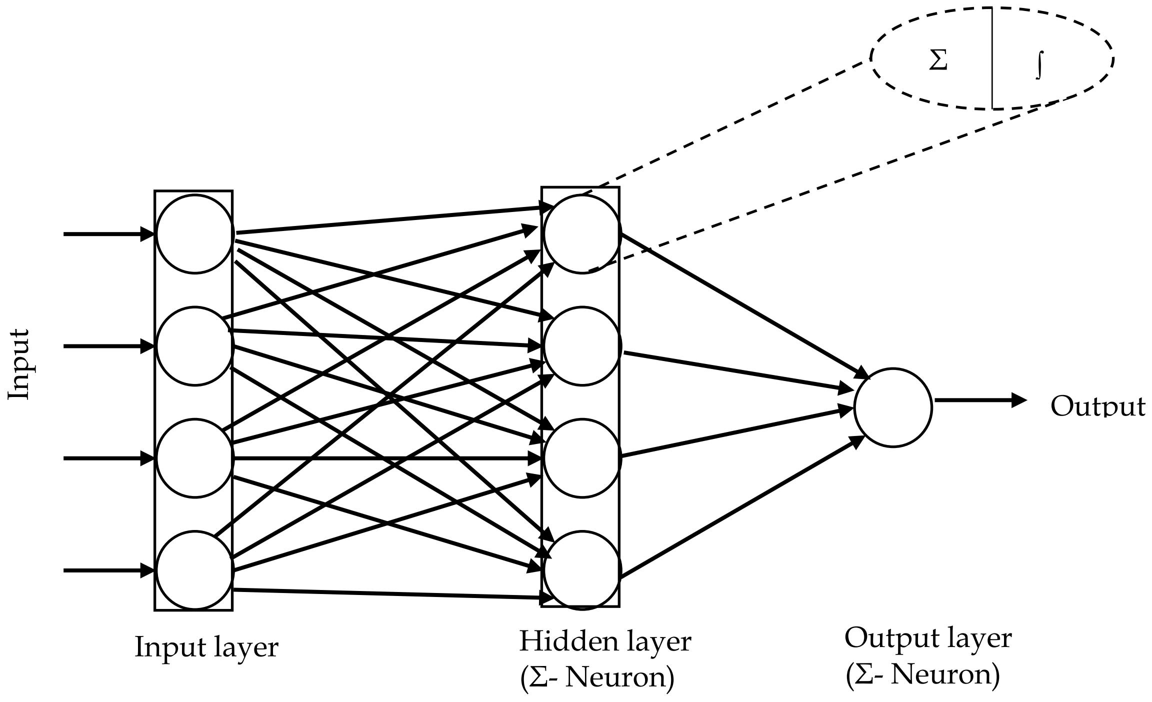

**Figure 4.17:** Summation type Neural Network

***(b) Product Neuron (∏-type):*** It is constructed by product function and nonlinear activation function at aggregation and activation stage respectively as shown in Figure 4.18.

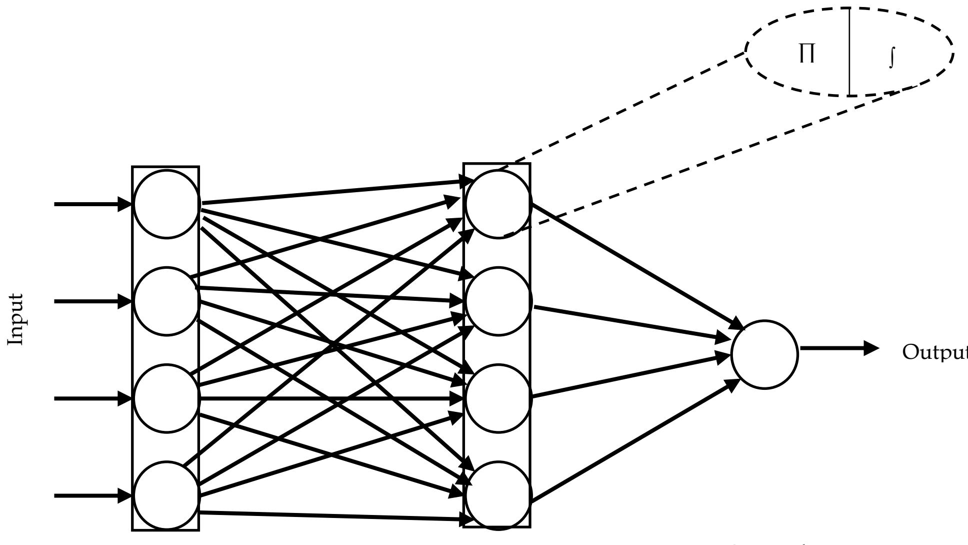

**Figure 4.18:** Product type neural network

With the help of the combination of summation and product type of neuron model following the type of neural network model could be developed:

***(a) Summation types neural network***

It is constructed with summation neuron at hidden layer and output layer as shown in Figure 4.17.

***(b) Product Type neural network***

It is constructed with Product neuron at hidden layer and output layer as shown in Figure 4.18.

***(c) Mixed type neural network***

- Summation- Product type neural network

Summation neurons and product neurons used in hidden and output layer respectively are as shown in Figure 4.19.

- Product-Summation types neural network

Product neurons and summation neurons used in hidden and output layers respectively are as shown in Figure 4.20.

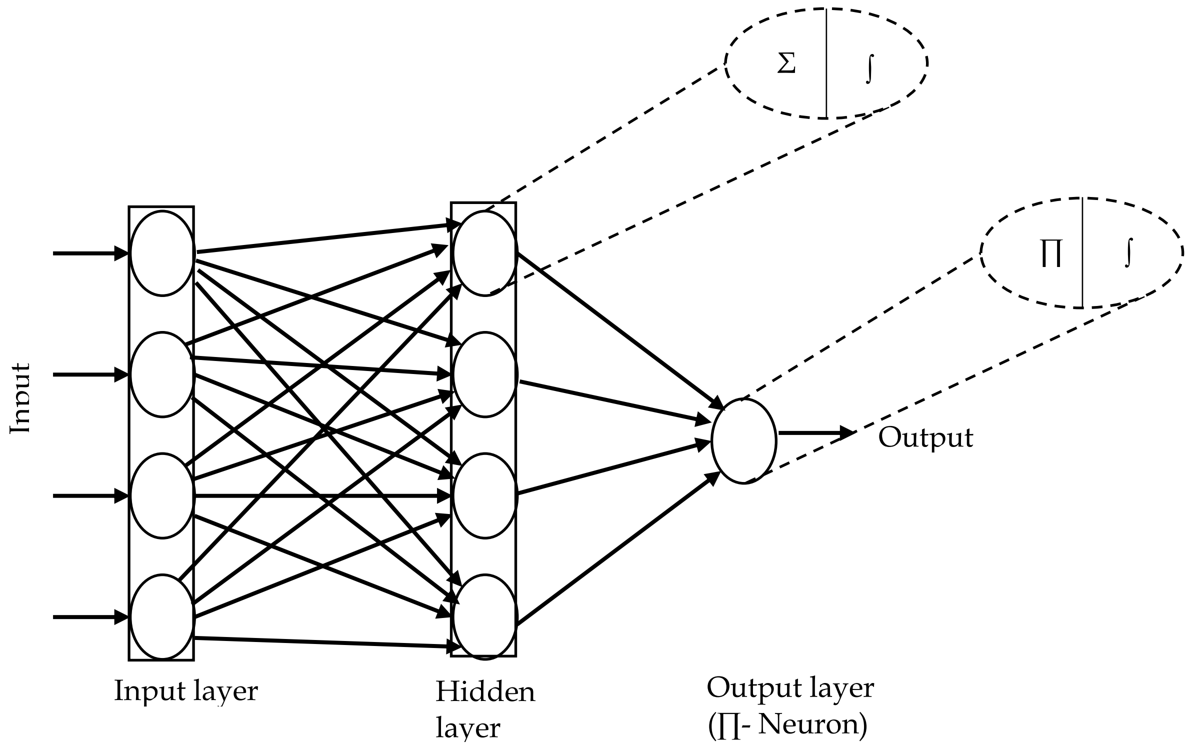

**Figure 4.19:** Summation and Product type neural network (Σ-∏ ANN)

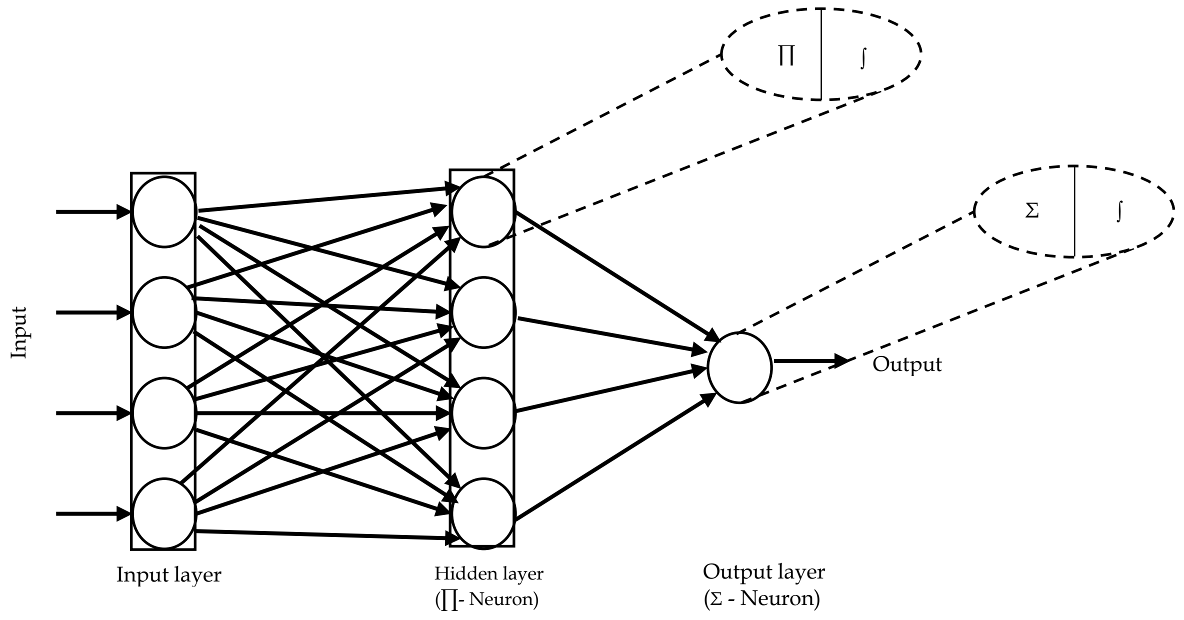

**Figure 4.20:** Product -Summation type neural network (∏- ΣANN)

## 4.3.2 Development Methodology of Generalized Neural Network

As discussed above the combination of neuron model at hidden and output layers helps in the generalization of the conventional neural network. In [Chaturvedi D K, 2008] it is shown that combination of different type of neuron model gives a better result compared to conventional neuron model. The generalized neural network is a combination of conventional neuron models and fuzzy compensatory operator which is proposed by [Mizumoto, 1989] and is given in Table [4.5].

**Table 4.5 Compensatory operators [Mizumoto, 1989]**

| Summation types fuzzy operator | Product type fuzzy operator |
|---|---|
| $[X_1 \cap X_2] * W + [X_1 U X_2] * (1-W)$ | $[X_1 \cap X_2]^W * [X_1 U X_2]^{(1-W)}$ |

Real life problems deal with non-linearity and complexity in nature. So a generalized neural network model has flexibility at aggregation and activation level to use Sigmoidal and Gaussian functions with weights of nodes [Chaturvedi and Malik, 2005]. Here summation and product function are used with sigmoidal and Gaussian functions respectively. So the mathematical expression of the output of summation and product type of generalized neuronal network is given below:

$$O_\Sigma = \frac{1}{1+e^{-\lambda s * s_{net}}} \tag{4.36}$$

$$GNN_{Summation} = O_\Sigma * W + O_\prod * (1-W) \tag{4.37}$$

$$GNN_{Summation} = \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} * W + e^{-\lambda p * \prod W_{\Sigma_i} X_i * X_{O\prod}} * (1-W) \tag{4.38}$$

$$O_\prod = e^{-\lambda p * pi_{net}2} \tag{4.39}$$

$$GNN_{product} = O_{\Sigma}^{W} * \ O_{\Pi}^{(1-W)} \tag{4.40}$$

$$GNN_{Product} = (\frac{1}{1+e^{-\lambda s*\sum W_{\Sigma_i} X_i + X_{O\Sigma}}})^{W} * \left(e^{-\lambda p*\Pi\Sigma_i X_i * X_{O\Pi}}\right)^{(1-W)} \tag{4.41}$$

Here,

$O_{\Sigma} = 0 < O_{\Sigma} < 1$

$O_{\Pi} = 0 < O_{\Pi} < 1$

### 4.3.3 Generalized Neural Network as Universal Approximator

In the previous studies, Multi-layered feed forward neural network has been shown to have the capability to process the input and output data and approximate any continuous and non-linear function [Cybenko, 1989; Hornik, 1991; Barron, 1993]. Accuracy of any universal approximator depends on the measure of closeness with function's output of neural network [Hornik, 1991]. In the previous studies [Barron, 1993; Hornik *et al.,* 1989; Hornik, 1991] authors describe that MLFFNN with activation function as a universal approximator. In the previous studies, it is shown that neural network with any number of hidden layers and activation functions is supposed to be monotone and continuous [Barron, 1993]. Neural network is formed from compositions and super composition of single, simple non-linear activation or response function [Cybenko, 1989]. For activation function, any function should be continuous, bounded. So universal approximation theorem stated that [Cybenko, 1989; Hornik, 1991; Barron, 1993]:

" Let Φ(·) be a non-constant, bounded, and monotone-increasing continuous function. Let $\mathbb{I}^n$ Denotes the n- dimensional unit hypercube$[0,1]^n$, and let the space of continuous function on $\mathbb{I}^n$ be denoted by $C(\mathbb{I}^n)$. Then given any function $F \epsilon C(\mathbb{I})^n$ and $\epsilon > 0$, there exists an integer $p$ and sets of real constants $\alpha_j, \theta_j$, and $w_{ij}$, where *i*=1, … , $n$ and *j*=1, … , $p$ such that we may define:"

$$f(X,W) = \sum_{j=1}^{p} \alpha_j \Phi\left(\sum_{i=1}^{n} w_{ij}\, x_i - \theta_j\right) X\epsilon\ \mathbb{I}^n, W\epsilon\ \mathbb{R}^{n*p} \tag{4.42}$$

$$|f(X,W) - F(X)| \leq \epsilon \tag{4.43}$$

In previous section, it is mentioned that in generalized neural network two activation functions are used with the help of fuzzy compensatory operators. Fuzzy compensatory operators are additive type and it is shown in [Bart, 1994], that any continuous real function can be approximated by fuzzy system. So with the help of mathematical expressions we can describe that a Generalized neural network fulfils the criteria of universal approximation theorem as given below in Eq.(4.44).

$$|f(x) - g(x)| \leq \epsilon \tag{4.44}$$

In the previous section it is mentioned that, sigmoidal and gaussian activation function are used to calculate the GNN output. So output of GNN model is given below:

$$g(x) = O_{\Sigma} * W + \ O_{\Pi} * (1 - W) \tag{4.45}$$

$$O_{\Sigma} = \frac{1}{1 + e^{-\lambda s * s_{net}}} \tag{4.46}$$

$$O_{\Pi} = e^{-\lambda p * pi_{net}2} \tag{4.47}$$

With the help of Eq.(4.45-4.47), Generalized Neural Network model can be rewrite in the form of given Eq.(4.48)

$$g(x) \; = \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} * W + e^{-\lambda p * \prod W_{\Sigma_i} X_i * X_{O\Pi}} * (1-W) \tag{4.48}$$

$$\left| f(x) - \left( \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} * W + e^{-\lambda p * \prod W_{\Sigma_i} X_i * X_{O\Pi}} * (1-W) \right) \right| \leq \epsilon \tag{4.49}$$

Let $f(x)$ be the sigmoidal function [Cybenko, 1989]
Here

$$f(x) = \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} \tag{4.50}$$

Let two cases where
W=0 and 1
Where, 0 worst case and 1 is better case

$$\left| \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} - \left( \left( \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} \right) * W + \left( e^{-\lambda p * \prod W_{\Sigma_i} X_i * X_{O\Pi}} * (1-W) \right) \right) \right| \leq \epsilon \tag{4.51}$$

When w=0 then; (4.52)

$$\left| \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} - \left( e^{-\lambda p * \prod W_{\Sigma_i} X_i * X_{O\Pi}} * (1-W) \right) \right| \leq \epsilon \tag{4.53}$$

When w=1 then;

$$\left| \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} - \left( \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}} \right) \right| \leq \epsilon \tag{4.54}$$

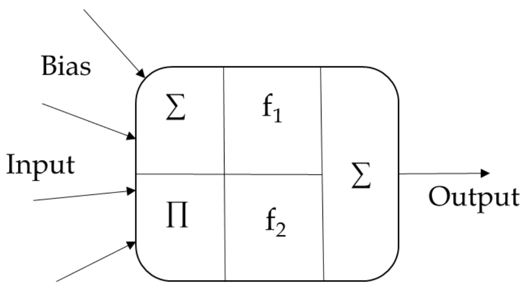

**Figure 4.21 :** Generalized Neural Network model

In the Figure 4.21, $f_1$ and $f_2$ are Sigmoid and Gaussian functions,

$$f_1(O_\Sigma) = \sigma(O_\Sigma) = \frac{1}{1+e^{-\lambda s * \sum W_{\Sigma_i} X_i + X_{O\Sigma}}}$$

The range of Sigmoidal activation function is:

$$\sigma(x) = \begin{cases} 1, & t \to +\infty \\ 0, & t \to -\infty \end{cases}$$

$$f_2(O_\Pi) = \sigma(O_\Pi) = e^{-\lambda p * \prod W_{\Pi_i} X_i * X_{O\Pi}}$$

The range of Gaussian activation function is:

$$\sigma(x) = \begin{cases} 1, & t < 0 \\ 0, & t \geq 0 \end{cases}$$

Above activation functions are universal approximators according to (Cybenko, 1979).
Here $f(x)$ of an n−dimensional real variable, $X_i \in \mathbb{R}^n$, is given by finite linear combination of the form

$$f(x) = \sum_{i=1}^{n} W_{\Sigma i} X_i + X_{O\Sigma}$$

$$g'(x) = \sum_{j=1}^{n} \sigma(f(x))$$

Here, $\sigma$ is any continuous Sigmoidal and Gaussian function.

The output of the network is the value of the function that results from this particular composition. It is enough to approximate any function, thus we can say, GNN works as universal approximator.

### 4.3.4 Learning Paradigms of Generalized Neural Network ($\sum -Type$)

Training of Generalized neural network ($\sum -Type$) includes the following steps to calculate the output of the network and training of network [Chaturvedi, 2008].

**-- Forward calculation**
In forward calculation output of a neuron is calculated in following three steps.

**Step-1**
**Output of summation types neuron**

$$O_\Sigma = \frac{1}{1 + e^{-\lambda s * s_{net}}} \tag{4.55}$$

Where

$$S_{net} = \sum W_{\Sigma i} X_i + X_{o\Sigma} \tag{4.56}$$

**Step-2**
**Output of Product types neuron**

$$O_\Pi = e^{-\lambda p * pi_{net}2} \tag{4.57}$$

Where,

$$pi_{net} = \prod W_{\Pi i} X_i + X_{o\Pi} \tag{4.58}$$

**Step-3**
**Final Output of Generalized Neural Network**

$$O_{pk} = O_\Sigma * W + O_\Pi * (1 - W) \tag{4.59}$$

**Reverse calculation**
After the forward calculation for the output of summation type neuron, the outcome of a neuron is compared with desired output to calculate the error. With the help of backpropagation learning algorithm the generalized neural network is trained to minimize the error which is shown in following steps:

**Step-4**
In this step, the error is calculated for the ith input data set and $E_p$ is a simplified sum-squared error for convergence.

Error $(E_i) = (Y_i - O_i)$ (4.60)

$E_p = 0.5 \sum E_i^2$ (4.61)

**Step-5**

(a) Here summation neuron model is associated with the weights. So updated weights for the network are

$$W(k) = W(k-1) + \Delta W \tag{4.62}$$

Where,

$$\Delta W = \eta \delta_k \, (O_{\Sigma} - O_{\Pi}) \, X_i + \alpha W(k-1)$$

and

$$\delta_k = \sum (Y_i - O_i)$$

(b) When weights are associated with summation part then updated weights at summation part are :

$$W_{\Sigma i}(k) = W_{\Sigma i}(k-1) + \Delta W_{\Sigma i} \tag{4.63}$$

Where,

$$\Delta W_{\Sigma i} = \eta \delta_{\Sigma j} X_i + \alpha W_{\Sigma i}(k-1)$$

and

$$\delta_{\Sigma i} = \sum \delta_k \, W(1 - O_{\Sigma})^* \, O_{\Sigma}$$

(c) When weights are associated with product part then updated weights at product part are:

$$W_{\Sigma \Pi i}(k) = W_{\Sigma \Pi i}(k-1) + \Delta W_{\Sigma \Pi i} \tag{4.64}$$

Where

$$\Delta W_{\Pi i} = \eta \delta_{\Pi j} X_i + \alpha W_{\Pi i}(k-1)$$

and

$$\delta_{\Pi j} = \sum \delta_k \, (1 - W) * (-2 * Pi_{net})^* \, O_{\Pi}$$

Here,

α=momentum factor

η= learning rate

### 4.3.5 Efficacy of Generalized Neural Network Model

In previous sections development, learning rules of generalized neural network are described. GNN model can overcome of drawbacks of the ANN model. ANN model have various drawback as given below:

- Large number of neurons are required in hidden layers.
- Hidden layers are required for non-linear and complex problems [Lippmann, 1987].
- Due to presence of hidden layers in neural network its takes large time to compute the output.
- Higher number of iterations required for desired output [Gorman and Sejnowski, 1988].

Generalized neural network have following advantages compare to multilayered feed forward neural network model [Chaturvedi, 2008].

- Required unknown values of weights are less compare to multilayered feed forward neural network.
- Due to less number of unknown weights, training time of the network is reduced.
- Single neuron based GNN model can solve any non-linear problem.
- Due to generalization of neural network, we can choose the suitable aggregation and activation function on the basis of complexity of the problem.

## 4.4 ERROR METRICS FOR ACCURACY OF FORECASTING MODEL

In this section, Assessment of above forecasting models are discussed which is basically: the root mean square error (RMSE) and mean square error (MSE). In (Marquez *et al.,* 2012) statistical error metrics to characterize the quality of neural network model in Eq. (4.65)-(4.67) are given. These include the coefficient of determination (**$R^2$**) which compares the variance of the errors to the variance of the data which is to be modeled:

$$R^2 = 1 - \frac{Var(\hat{Y}-Y)}{Var(Y)} \tag{4.65}$$

The Root-Mean-Squared Error (RMSE), which is a measure of the average, spread of the errors:

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\hat{Y}_t - Y_t\right)^2} \tag{4.66}$$

The Mean Squared Error (MSE) which is a measure of the average spread of the errors:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left(\hat{Y}_t - Y_t\right)^2 \tag{4.67}$$

…