CHAPTER 3

ANALYSIS OF THE PERFORMANCE OF INSULATION USING NEURAL NETWORK

3.1 INTRODUCTION TO NEURAL NETWORK

An artificial neural network is an information processing system that has certain performance characteristics in common with biological neural networks. Laurence Fausett (2004) explains that the artificial neural networks have been developed as generalization of mathematical models of human cognition or neural biology, based on the assumptions that:

- Information processing occurs at many simple elements called neurons
- Signals are passed between neurons over connection links
- Each connection link has as an associated weight, which, in a typical neural net, multiplies the signal transmitted
- Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal

A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: its dendrites, soma and axon. Dendrites receive signal from other neurons. The signals are electrical impulses that are transmitted across a synaptic gap by means of a
chemical process. The soma or cell body sums the incoming signals. When sufficient input is received, the cell fires; that is, it transmits a signal over its axon to other cells. Figure 3.1 shows the structure of the biological neuron.

Figure 3.1 Structure of Biological Neuron

An Artificial Neural Network is characterized by,

- Its pattern of connections between the neurons (called its architecture)
- Its method of determining the weights on the connections (called its training or learning, algorithm), and
- Its activation function

The network function is determined largely by the connections between elements. Therefore, a neural network can be trained to perform a particular function by adjusting the values of the connections (weight) between the elements. Commonly neural networks (Zurada 1997) are adjusted or trained, so that a particular input leads to a specific target output. Figure 3.2 shows the basic operation of a neural network. The network weight
is adjusted based on a comparison of the output and the target, until the network output matches the measured value.

**Figure 3.2 Basic Operation of Neural Network**

### 3.1.1 Neuron Model

Figure 3.3 shows a neuron with a single scalar input with no bias. The scalar input p, is transmitted through a connection that multiplies its strength by the scalar weight w, to form the product wp, again a scalar. Here the weighted input wp is the only argument of the activation function f and n is the net input, which produces the scalar output a.

\[ a = f(wp) \]

**Figure 3.3 Single Input Neuron without Bias**

Figure 3.4 shows a neuron with a scalar input, with scalar bias. The bias is much like a weight, except that it has a constant input of 1. The activation function net input n, again a scalar, is the sum of the weighted input wp and the bias b, this sum is the argument of the activation function f.
Here $f$ is an activation function, typically a step function or a sigmoid function, that takes the argument $n$ and produces the output $a$. Here $w$ and $b$ are both adjustable parameters of the neuron.

$$a = f(wp + b)$$

**Figure 3.4 Single Input Neuron with Bias**

The central idea of neural networks is that such parameters ($w$ and $b$) can be adjusted so that the network exhibits some desired or interesting behaviour. Thus, we can train the network to do a particular job by adjusting the weight or bias parameters, or perhaps the network itself will adjust these parameters (weight and bias) to achieve the desired target.

### 3.1.2 Activation Functions

An activation function may be linear or a non-linear function. A particular activation function is chosen to satisfy some specification of a problem that the neuron is attempting to solve. There are three most commonly used activation functions. They are:

(a) Hard limit activation function

(b) Linear activation function

(c) Log-sigmoid activation function
(a) **Hard limit activation function**

Figure 3.5 shows the graphical representation of the hard limit activation function. The hard limit activation function sets the output of the neuron to 0, if the function argument is less than 0, and to 1, if its argument is greater than or equal to 0.

![Graph of the hard limit activation function](image)

\[ a = \text{hardlim} \left( n \right) \]

**Figure 3.5 Hard Limit Activation Function**

(b) **Linear activation function**

The output of a linear activation function is equal to its input. The output \( a \) versus input \( p \) characteristic of a single-input linear neuron is shown in Figure 3.6.

![Graph of the linear activation function](image)

\[ a = \text{purelin} \left( n \right) \]

**Figure 3.6 Linear Activation Function**
(c) Log-sigmoid activation function

Figure 3.7 shows the log-sigmoid activation function. This activation function takes the input, which may have any value between plus infinity and minus infinity and squashes the output into the range 0 to 1,

According to the expression

\[ f(a) = \frac{1}{1+e^{-n}} \]  

(3.1)

![Log-Sigmoid Activation Function](image)

\[ a = \text{logsig} (n) \]

Figure 3.7 Log-Sigmoid Activation Function

This activation function is commonly used in multilayer networks that are trained using the back-propagation algorithm, in part because this function is differentiable.

3.1.3 Learning Rules

The weights and biases of the network can be modified by means of ‘learning rule’. This procedure may also be referred to as a training algorithm. The purpose of the learning rule is to train the network to perform some task. The Neural networks can be trained to solve problem that are difficult for conventional computers or human beings. There are many types of neural
network learning rules. They fall into three broad categories: supervised learning, unsupervised learning and reinforcement or graded learning.

### 3.1.3.1 Supervised Learning

In supervised learning, the network is provided with inputs and the corresponding correct output. As the inputs are applied to the network, the network outputs are compared 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. An example for the supervised learning is the perceptron-learning rule.

### 3.1.3.2 Reinforcement Learning

This is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade is a measure of the network performance over some sequence of inputs. This type of learning is currently much less common than supervised learning.

### 3.1.3.3 Unsupervised Learning

In unsupervised learning, the weights and biases are modified in response to network inputs only. There are no target outputs available. The network learns to categorize the input patterns into a finite number of classes. An example for unsupervised learning algorithm is the Adaptive Resonance Theory.

### 3.2 BACK-PROPAGATION NEURAL NETWORK

In supervised learning, the first learning rule is perceptron-learning rule, in which the learning rule is provided with a set of examples of proper
network behaviour. As each input is applied to the network, the learning rule adjusts the network parameters so that the network output will move closer to the target. The perceptron learning rule is very simple, but it is also quite powerful. This rule will always converge to a correct solution, if such a solution exists. The perceptron-learning rule forms the basis for understanding the more complex networks (Rajasekaran et al 1995). As with the perceptron rule, the Least Mean Square (LMS) algorithm is an example of supervised learning. The LMS algorithm will adjust the weights and biases to minimize the mean square error, where the error is the difference between the target output and the network output. The perceptron-net is incapable of implementing certain elementary functions. These limitations were overcome with improved (multilayer) perceptron networks.

Performance learning is another important class of the learning law, in which the network parameters are adjusted to optimize the performance of the network. Back propagation (BP) algorithm can be used to train multilayer networks (Laurence 2004). As with the LMS learning law, BP is an approximate steepest descent algorithm, in which the performance index is mean squared error.

The difference between the LMS algorithm and back propagation is only in the way in which the derivatives are calculated. The single-layer perceptron like networks are only able to solve linearly separable classification problems. Multilayer perceptron, trained by BP algorithm were developed to overcome these limitations and is currently the most widely used neural network (Sivanandam and Paulraj 2004). In addition, multi-layer networks can be used as universal function approximators. A two-layer network, with sigmoid-type activation functions in the hidden layer, can approximate any practical function, with enough neurons in the hidden layer. Figure 3.8 shows the Architecture of the BP Neural Network.
The BP algorithm uses the chain rule in order to compute the derivatives of the squared error with respect to the weights and biases in the hidden layers. It is called BP because the derivatives are computed first at the last layer of the network, and then propagated backward through the network, using the chain rule, to compute the derivatives in the hidden layers. The BP training algorithm is an interactive gradient algorithm designed to minimize the mean squared error between the actual output of a feed-forward net and the desired output. Figure 3.9 shows the flowchart of the BP training algorithm (Freeman and Skapura 2000). Let $x_i$, $z_j$, $y_k$ respectively be the input, hidden and output layer neuron, $w_{ok}$ and $v_{oj}$ are the bias of input and hidden layer, $w_{ij}$ and $v_{jk}$ are the weights of the input to hidden and hidden to output layer.
Figure 3.9 Flowchart of BP Training Algorithm
3.2.1 Training Algorithm

Step 1: The weights are initialized randomly between -0.5 and +0.5.

Step 2: While stopping condition is false do steps 3-12.

Step 3: Initialize the error and sum of mean squared error (E)

\[ e_k(0) = 0, \text{for}(k = 1, 2, 3, \ldots, p) \]
\[ E = 0 \]  \hspace{1cm} (3.2)

Step 4: For each training pair do steps 5-11

Step 5: Compute the output signals for the hidden units 
\[ z_j = 1, 2, 3, \ldots, m. \]

\[ z_{\text{in}}^j(s) = w_{oj} + \sum_{i=1}^{n} x_i \cdot w_{ij} \]  \hspace{1cm} (3.3)

\[ z_j^s = f(z_{\text{in}}^j(s)) = \frac{2}{1 + e^{-z_{\text{in}}^j(s)}} - 1 \]  \hspace{1cm} (3.4)

Step 6: Compute the output signal for the output units 
\[ y_k = 1, 2, 3, \ldots, p. \]

\[ y_{\text{in}}^k(s) = v_{oj} + \sum_{k=1}^{p} z_j^s \cdot v_{jk}, \]  \hspace{1cm} (3.5)

\[ y_k^s = f(y_{\text{in}}^k(s)) = \frac{2}{1 + e^{-y_{\text{in}}^k(s)}} - 1 \]  \hspace{1cm} (3.6)

Step 7: Compute the error and sum of mean squared error (E), for (k = 1, 2, 3… p)

\[ e_k(s) = t_k(s) - y_k(s), \]  \hspace{1cm} (3.7)

\[ E_{\text{new}} = E_{\text{old}} + [e_k(s)]^2 \]  \hspace{1cm} (3.8)
Step 8: Compute the error gradient and the change in weight for the output neurons, for \((k = 1, 2, 3, \ldots, p)\) and for \((j = 1, 2, 3, \ldots, m)\)

\[
\delta_k = e_k(s)f'(y_{in_k}(s)), \quad (3.9)
\]

\[
\Delta v_{jk} = \alpha \delta_k z_j(s) + \eta \Delta v_{jk} \text{ (old)} \quad (3.10)
\]

Step 9: Compute the error gradient and the change in weight for the hidden neurons, for \((j = 1, 2, 3, \ldots, m)\) and for \((i = 1, 2, 3, \ldots, n)\)

\[
\hat{\delta}_j = \left( \sum_{k=1}^{p} \delta_k v_{jk} \right) f'(z_{in_j}(s)), \quad (3.11)
\]

\[
\Delta w_{ij} = \alpha \hat{\delta}_j x_i(s) + \eta \Delta w_{ij} \text{ (old)} \quad (3.12)
\]

Step 10: Each output unit \(Y_k, k=1, 2, 3, \ldots, p\), update its bias and weight, for \((j = 0, 1, 2, \ldots, m)\) as

\[
v_{jk} \text{ (new)} = \Delta v_{jk} + v_{jk} \text{ (old)} \quad (3.13)
\]

Step 11: Each hidden unit \(Z_j, j=1, 2, 3, \ldots, m\), update its bias and weight, for \((i = 0, 1, 2, \ldots, n)\) as

\[
w_{ij} \text{ (new)} = \Delta w_{ij} + w_{ij} \text{ (old)} \quad (3.14)
\]

Step 12: Test for stopping condition.

The flow chart for the training algorithm as shown in Figure 3.9.
3.3 CHOICE OF PARAMETERS FOR NETWORK TRAINING

When the basic BP algorithm is applied to a practical problem, the training may take days or weeks of computer time. This has encouraged considerable research on methods to accelerate the convergence of the algorithm. The research on faster algorithms falls roughly into two categories; the first category involves the development of heuristic techniques, which arises out of a study of the distinctive performance of the standard BP algorithm. These heuristic techniques include such ideas varying the learning rate, using momentum and rescaling variables. Another category of research has focused on standard numerical optimization techniques.

3.3.1 Learning Rate

The speed of training the BP network is improved by changing the learning rate during training. Increasing the learning rate on flat surfaces and then decreasing the learning rate when slope increases can increase the process of convergence (Sathis Kumar 2004). If the learning rate is too large, it leads to unstable learning. And if it is too small, it leads to incredibly long training times. Hence care has to be taken while deciding learning rate. There are many different approaches for varying the learning rate. The learning rate is varied according to the performance of the algorithm.

The rules of the variable learning rate BP algorithm are:

i. If the squared error increases by more than some set percentage $\xi$ (typically one to five percent) after weight update, then the weight update is discarded, the learning rate is multiplied by some factor $0 < p < 1$, and the momentum coefficient $\gamma$ (if it is used) is set to zero.
ii. If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If $\gamma$ has been previously set to zero, it is reset to its original value.

iii. If the squared error increases by less than $\xi$ then the weight update is accepted but the learning rate is unchanged. If $\gamma$ has been previously set to zero, it is reset to its original value.

3.3.2 Momentum Factor

In BP with momentum, the weight change is in a direction that is a combination of the current gradient and the previous gradient. This is a modification of gradient descent whose advantage arises chiefly when some training data are very different from the majority of the data. By the use of momentum, a larger training rate can be used, while maintaining the stability of the algorithm. Another feature of momentum is that it tends to accelerate convergence when the trajectory is moving in a consistent direction.

The larger the value of $\gamma$, the more the momentum the trajectory has. The momentum coefficient is maintained with the range $[0, 1]$.

3.3.3 Bias

Sigmoid hidden and output units usually use a "bias" or "threshold" term in computing the net input to the unit. For a linear output unit, a bias term is equivalent to an intercept in a regression model.

A bias term can be treated as a connection weight from a special unit with a constant, nonzero activation value. The term "bias" is usually used with respect to a "bias unit" with a constant value of one. The term "threshold" is usually used with respect to a unit with a constant value of
negative one. Regardless of the terminology, whether biases or thresholds are added or subtracted has no effect on the performance of the network.

The single bias unit is connected to every hidden or output unit that needs a bias term. The "universal approximation" property of multilayer networks with most commonly-used hidden-layer activation functions does not hold if you omit the bias terms. But Hornik (1993) shows that a sufficient condition for the universal approximation property without biases is that no derivative of the activation function vanishes at the origin, which implies that with the usual sigmoid activation functions, a fixed nonzero bias term can be used instead of a trainable bias.

In the common type of feed-forward back-propagation artificial neural network using a sigmoid transfer function, the sigmoid function is applied to the sum of the weighted inputs i.e. to the sum of $x(i) * w(i)$ where $w(i)$ is the weight associated with the input $i$ and $x(i)$ is the current value of input $i$. Applying the sigmoid function to just the sum of the weights would mean that the neuron's output didn't change when the inputs changed, which would be useless. The bias is an offset value applied to this sum - in effect; it is the weight of an input whose value $x_{bias}$ is always 1. The bias shifts the centre of the sigmoid function away from the origin (Hornik 1993).

3.4 STRUCTURE OF BPN NETWORK FOR ANALYSING THE INSULATION PERFORMANCE

An artificial neural network is composed of neurons with a deterministic activation function. The neural network, trained by adjusting the numerical value of the weights, will contain the non-linearity of the desired mapping, such that difficulties in the mathematical modeling can be avoided. The BP training algorithm is used to adjust the numerical values of the weights and the internal threshold of each neuron. The network is trained by,
initially selecting small random weights and internal threshold and then presenting all training data. Weights and thresholds are adjusted after every training example and presented to the network, until the weight converges or the error is reduced to an acceptable value. Figure 3.10 shows the structure of BPN Network for analysis of the insulation performance.

Figure 3.10 Structure of BPN Network for Analysis of Insulation Performance

A three layer neural network suitable for training with BPN algorithm as shown in Figure 3.10. The input (first) layer serves only as distribution points; they perform no input summation. The input signal is simply passed to the weights on their outputs. Each neuron in the subsequent layers produces output signals according to the activation function used. A neuron is associated with the set of weights that connects to its input. This network is considered to consist of three layers. The input or distribution layer is designated as layer 0, the second layer called hidden layer is denoted as layer 1 and the output layer as layer 2.
The activation function used in the BPN is the sigmoidal function. Training is generally commenced with randomly chosen weight values. Typically, the weights chosen are small (between -1 and +1 or -0.5 and +0.5), since larger weight magnitudes may drive the output of layer 1 neurons to saturation, requiring more time to come out of the saturated state. The learning begins with the feed forward recall phase.

The input parameters used in this network $x_i$ are applied voltage, leakage current and capacitance. The weight function used between the input layer and hidden layer is $W_{ij}$. The net weighted input along with the bias signal $V_{oj}$ is given as the functional parameter $Z_{ij}$ for hidden layer.

$$z_{ij} = v_{oj} + \sum W_{ij} \text{ and } z_j = f(z_{ij}) \quad (3.15)$$

Output of the hidden layer neurons $z_j$ are calculated using the sigmoidal function. This output values are given as the input parameter for the output layer. The net weighted input $Z_j * V_{jk}$ along with the bias is given as the functional parameter for the output layer. The final output value which is the Dissipation factor is calculated using the sigmoidal activation function.

Then, the error signal computation phase follows. The error signal vector must be determined in the output layer first, and then it is propagated toward the network input nodes. Negative gradient descent technique is used for calculation of the error factor and for the calculation of the weight matrix adjustment. First, the weight matrix connects the hidden layer and the output layer is adjusted and then the weight matrix connects the input layer and the hidden layer is adjusted.

The training is stopped if the cumulative error is within the limit or the number of training epoch reaches a maximum set value.
3.4.1 Sigmoidal Activation Function

The sigmoidal function relates the output of a neuron to the weighted input or net input \( y \) as follows:

\[
f(y) = \frac{1}{1 + e^{(-y)}} \quad (3.16)
\]

for binary sigmoidal function.

\[
f(y) = \frac{2}{1 + e^{(-y)}} - 1 \quad (3.17)
\]

for bipolar sigmoidal function (Kishan et al 1997).

The features of sigmoidal functions are as follows (Chin et al 1996)

i. The sigmoidal function is an \( S \) shaped form.

ii. The value of its output is always between -1 and 1 for bipolar and 0 and 1 for binary function.

iii. The function is highly non-linear. For large value of activation, the output of neuron is restricted by the activation function.

iv. The sigmoidal function is continuous and continuously differentiable.

Sigmoid functions are prized because of their derivatives are easy to calculate, which is helpful for calculating the weight updates in training algorithms. Sigmoid-type activation function can approximate any practical function.
This sigmoid function is especially advantageous for use in neural networks trained by back-propagation because it is easy to differentiate, and thus can dramatically reduce the computation burden for training. The activation function used here is bipolar sigmoidal function, because the highest correlation coefficient is obtained when using the bipolar sigmoidal function.

3.4.2 An Overview of Training

The objective of training the network is to adjust the weights so that the application of a set of inputs produces the desired set of outputs. For reasons of brevity, these input-output sets can be referred to as vectors. Training assumes that each input vector is paired with a target vector representing the desired output. Together these are called a training pair. Usually, a network is trained over a number of training pairs. The activation function used for the analysis is bipolar sigmoidal function.

3.4.3 Choice of Learning Rate and Momentum Factor

Weight changes in BPN networks are proportional to the negative gradient of the error; this guideline determines the relative changes that must occur in different weights when a training sample (or a set of samples) is presented, but does not fix the exact magnitudes of the desired weight changes. The magnitude change depends on the appropriate choice of the learning rate $\eta$. A large value of $\eta$ will lead to rapid learning but the weight may then oscillate, while low values imply slow learning. This is typical of all gradient descent methods. The right value of $\eta$ will depend on the applications. Values between 0.001 and 0.9 have been used in many applications (Chin-Teng Lin and George Lee 1996).
Back propagation leads the weights in a neural network to a local minimum of the mean squared error. Possibly substantially different from the global minimum that corresponds to the best choice of weights. This problem can be particularly bothersome if the “error surface” is highly uneven or jagged, with a large number of local minima. To avoid this, Rumelhart, Hinton and Williams suggested that the weight changes in the \( i^{th} \) iteration of the BPN algorithm depend on immediately preceding weight changes, made in the \((i-1)^{th}\) iteration. The implementation of this method is straightforward, and is accomplished by adding a momentum term to the weight update rule,

\[
\Delta w_{jk} = \alpha \Delta w_{jk}(s) + \eta \Delta w_{jk}(old)
\]  

(3.18)

Use of momentum term in the weight update equation introduces yet another parameter \( \alpha \), whose optimal value depends on the application and is not easy to determine a prior. A well-chosen \( \alpha \) can significantly reduce the number of epochs for convergence. A value close to 0 implies that the past history does not have much effect on the weight change, while a value closer to 1 suggests that the current error has little effect on the weight change.

### 3.5 EXPERIMENTAL RESULTS

This section illustrates the performance of the proposed procedure for the Analysis of PD Measurements. For all the experiments, with the chosen learning rate and momentum factor, the bias for both hidden and output layers are set to 1. The initial weights are randomized between -0.5 and +0.5. All the input values and test outputs are normalized between 0 to 1 using equations (3.19).

\[
\text{Normalized value} = \frac{x - \text{minimum value}}{\text{maximum value} - \text{minimum value}}
\]  

(3.19)
PD measurements from seven machines are used for training the network. Three machines chosen are of 11 kV rating and the other four machines are of 6.6 kV rating respectively (Appendix 1). Finally the network is tested with the eighth machine of 11 kV rating (Table 3.1). Three input parameters are used to train the networks which are applied voltage, leakage current and capacitance. Initially two output parameters were considered for training the network which is dissipation factor and PD magnitude. But, it is observed that variation in PD magnitude affects the training of entire network and the network becomes unstable. Minimum void can cause huge variation in PD magnitude which causes the above issue. Hence the network is trained with only one output parameter which is the Dissipation factor.

Table 3.1  PD Measurement of 11 kV Machine (Used to Test the Trained Network)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Grounded Terminals</th>
<th>Applied Voltage in kV</th>
<th>Leakage Current in mA</th>
<th>Capacitance in nF</th>
<th>Dissipation Factor</th>
<th>PD Magnitude in pC</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Y and B</td>
<td>4.40</td>
<td>135.7</td>
<td>99.73</td>
<td>2.458</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.60</td>
<td>202.9</td>
<td>99.85</td>
<td>2.586</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.80</td>
<td>269.5</td>
<td>99.49</td>
<td>2.923</td>
<td>1600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.0</td>
<td>339.5</td>
<td>100.3</td>
<td>3.315</td>
<td>2100</td>
</tr>
<tr>
<td>Y</td>
<td>B and R</td>
<td>4.40</td>
<td>137.8</td>
<td>101.0</td>
<td>2.421</td>
<td>950</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.60</td>
<td>205.8</td>
<td>100.9</td>
<td>2.549</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.80</td>
<td>275.5</td>
<td>101.3</td>
<td>2.856</td>
<td>1700</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.0</td>
<td>341.1</td>
<td>100.5</td>
<td>3.372</td>
<td>2000</td>
</tr>
<tr>
<td>B</td>
<td>R and Y</td>
<td>4.40</td>
<td>136.1</td>
<td>99.97</td>
<td>2.462</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.60</td>
<td>203.6</td>
<td>100.1</td>
<td>2.592</td>
<td>1400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.80</td>
<td>272.1</td>
<td>100.2</td>
<td>2.898</td>
<td>1750</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.0</td>
<td>340.7</td>
<td>100.4</td>
<td>3.343</td>
<td>2200</td>
</tr>
</tbody>
</table>
Dissipation factor and dissipation factor tip-up test is an indirect way of determining if partial discharges are occurring in a high voltage stator winding. Partial discharge is a symptom of many high voltage winding insulation deterioration mechanism, dissipation test can indicate if many failure processes are occurring. If the dissipation factors increase over time, then it indicates that the winding is contaminated or has absorbed moisture (Stone et al 2004).

Hence the variation of the Dissipation factor is sufficient to analyze the partial discharge and so this is considered for training and testing the network. This problem is tested with conventional BPN algorithm.

The network is trained using the MATLAB programming tool. The initial random weights are chosen using the MATLAB inbuilt random weight generation. One set of random weight may result into training failure. Hence it is necessary to choose multiple sets of random weights for training the network. About 30 random weights are chosen and is divided into 3 trial sets with each trial weight consisting of 10 sets of randomized weight samples. Performance of the training is shown in Table 3.2 for each trial set. Maximum epoch for each set of random weight is fixed as 31000. The minimum cumulative error per set is fixed as 0.59. For a particular set of initial random weight, the training is stopped if the cumulative error reduces below the limit (0.59) or the number of training epoch reaches a maximum set value of 31000. If cumulative error reduces below this set value, then the training of this randomized set of weight will be identified as successful. If the cumulative error is greater than 0.59 even after reaching the maximum epoch then the current training will be treated as failure. The performance results are shown in Table 3.2.
Table 3.2 Analysis of PD Measurements using BPN

<table>
<thead>
<tr>
<th>Analysis of PD Measurements using BPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input neurons: 3  Functional Inputs: 3</td>
</tr>
<tr>
<td>Hidden neurons: 3  Output neurons: 1  Bias: 1</td>
</tr>
</tbody>
</table>

Learning Parameters:
Learning Rate = 0.1  Momentum Factor = 0.85

<table>
<thead>
<tr>
<th>Iterations: 10  Training Tolerance = 0.59  Test Tolerance = 0.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial No.</td>
</tr>
<tr>
<td>Successful iterations</td>
</tr>
<tr>
<td>Minimum Epoch</td>
</tr>
<tr>
<td>Maximum Epoch</td>
</tr>
<tr>
<td>Mean Epoch</td>
</tr>
<tr>
<td>Minimum Time (Sec)</td>
</tr>
<tr>
<td>Maximum Time (Sec)</td>
</tr>
<tr>
<td>Mean Time (Sec)</td>
</tr>
<tr>
<td>% Mean Misclassifications</td>
</tr>
</tbody>
</table>

Failures column shows the number of failures occurred in each trial set. Minimum and maximum epoch column shows the earliest successful training and the longest successful training for the trial set. Mean epoch and standard deviation are also calculated for each trial set. Time taken for training is also calculated.

Finally the network is tested with the test machine. For each successful training, a final weight set will be achieved. For testing the network, each set of input values are applied to the final weight set and the output dissipation factor is calculated. If the difference between the actual output and the network output is within the test tolerance limit (0.04), then there is no misclassification. If the difference exceeds the limit, then
misclassification is set to 1. For example, a set of random weight is chosen and the network is trained. When R-phase is energized and the other two phases (Y and B) are grounded and if the applied voltage is 6.6 kV, the measured dissipation factor is 2.586 and the one which is obtained from the trained network are 2.8001.

The error is calculated as given by the equation (3.20).

\[
\text{Error} = \frac{(\text{Actual} - \text{Target})^2}{2} \tag{3.20}
\]

In this case, the error is 0.02 and is within the tolerance limit 0.04. This is a successful case and no misclassification. Similarly the network is tested with all successful trainings and the misclassifications are calculated. Average misclassification from all the successful training is calculated for each trial set and is also shown in the Table 3.2.

Epoch Vs Error characteristics for one set of randomized weight samples is shown in Figure 3.11. This graph is a result of a successful training. For each epoch, the cumulative error is calculated and is tabulated against the epoch number. At starting the cumulative error is around 2.4 and is gradually reduced when the number of epoch increases. At around 25000\textsuperscript{th} epoch, the network reaches its limit. i.e., the cumulative error reduces to within the set limit of 0.59.

The measured value (target output) and actual output comparison for the test machine of R, Y and B phases respectively are shown in Figures 3.12 - 3.14. For each input value of applied voltage, the measured value of dissipation factor is compared with the dissipation factor obtained from the trained BPN network.
**Figure 3.11** Cumulative Error Vs Epoch

**Figure 3.12** R-Phase Output Response for DF using BPN
Figure 3.13 Y-Phase Output Response for DF using BPN

Figure 3.14 B-Phase Output Response for DF using BPN
3.6 CONCLUSION

i. The performance of the insulation of the stator winding of 11 kV and 6.6 kV high voltage rotating machine were analyzed using the Neural Network with BPN Algorithm.

ii. The Neural Network has been trained by using the applied voltage, leakage current and capacitance as input parameters and Dissipation Factor as an output parameter.

iii. The Neural Network has been trained for three trial sets; each trial set consisting of 10 set of randomized weight samples.

iv. Simulation results showed that the predicted values of the Dissipation Factor are closer to the measured values. The percentage of misclassification parameter confirms the good performance of the trained network.

v. BPN network based analysis of the insulation performance shows fairly acceptable results with an error tolerance of 0.04.