

ON-LINE CAPACITOR SWITCHING FOR DISTRIBUTION SYSTEM

A DISSERTATION

*Submitted in partial fulfilment of the
requirements for the award of the degree*

of

MASTER OF ENGINEERING

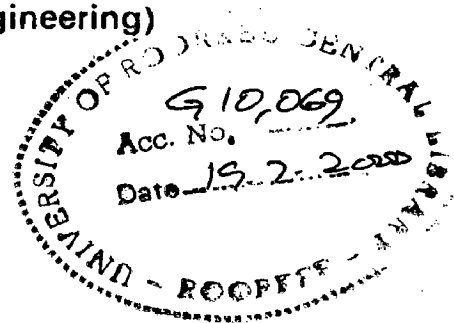
in

ELECTRICAL ENGINEERING

(With Specialization in Power System Engineering)

By

PRADEEP KUMAR VERMA



**DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247 667 (INDIA)**

MARCH, 2000

D D No. 0NT-177/MPP-BD/2000

~~CANDIDATE'S DECLARATION~~


I hereby declare that the work presented in this dissertation entitled "On-line Capacitor Switching for Distribution System", submitted in partial fulfillment of the requirement for the award of the degree of **Master of Engineering, in Electrical Engineering**, with specialization in **Power System Engineering**, in the Department of Electrical Engineering, University of Roorkee, Roorkee, is an authentic record of my own work, carried out with effect from July 1999 to March 2000, under the guidance of **Dr. B. Das** and **Dr. N. P. Padhy**, Department of Electrical Engineering, University of Roorkee, Roorkee.


The matter embodied in this thesis has not been submitted for the award of any other degree.

DATE : 27th March, 2000.


(PRADEEP KUMAR VERMA)

It is certified that the above statement made by the candidate is correct to the best of our knowledge and belief.


(Dr. B. Das) 27.3.2000


(Dr. N. P. Padhy)

Department of Electrical Engineering

University of Roorkee

Roorkee - 247667

(INDIA)

ACKNOWLEDGEMENT

I am greatly indebted to my guides **Dr. B. Das**, and **Dr. N. P. Padhy**, **Department of Electrical Engineering, University of Roorkee, Roorkee**, for their kind support and guidance during my work. Their co-operation and in-depth knowledge have made my work possible.

I am also thankful to **Sri Bharat Gupta**, O.C. Power System Simulation Lab, for providing me the best of facilities, which enabled my work to roll faster.

Last but not least, I am also grateful to Power System Engg. Faculty and my classmates whose presence have made a workable atmosphere in the lab, conducive for my work.



(PRADEEP KUMAR VERMA)

ABSTRACT

In this thesis, an artificial neural network based algorithm for determining optimal capacitor switching pattern for a given loading condition in a radial distribution system has been developed. Traditionally , combinatorial methods have been used to decide the optimal switching patterns. However, these combinatorial methods take significant amount of time for practical size power distribution system involving thousands of feeders. Hence, these algorithms may not be very suitable for on-line application in a modern distribution automation system. A multi-layer perceptron neural network with error-back-propagation training algorithm has been used to determine the optimal capacitor switching pattern. It has been found that the time taken by ANN based method is quite less compared to the time taken by the traditional combinatorial methods.

CONTENTS

	Page no.
CANDIDATE'S DECLARATION	I
ACKNOWLEDGEMENT	II
ABSTRACT	III
CHAPTER 1 : Introduction	1
CHAPTER 2 : ANN (Artificial Neural Network) Algorithm	5
2.1 : Brief Review of ANN	5
2.2 : Error-Back-Propagation Learning Algorithm	7
2.3 : Conclusion	16
CHAPTER 3 : Optimal Capacitor Switching Algorithm	17
3.1 : Objective Function and Constraints	17
3.2 : Solution Methodology	19
3.3 : Conclusion	22
CHAPTER 4 : Results and Discussion	23
CHAPTER 5 : Conclusion	32
REFERENCES	33
APPENDIX A	34
APPENDIX B	36

INTRODUCTION

Due to rapid industrialization and population growth, the demand for electrical power is ever increasing. To supply this ever-increasing load demand, total power generation in India has grown tremendously during the past decades. However, to supply quality power to the ultimate customers, it is not sufficient to merely increase the generation. Adequate transmission and distribution facilities also need to be built-up and consequently, there have also been substantial developments in the power transmission and distribution system.

Apart from building up the infrastructure of transmission and distribution system, it is also very important to continuously monitor the system to ensure safe, reliable and un-interruptible power supply. Towards this objective, supervisory control and data acquisition (SCADA) systems have been employed worldwide to continuously monitor and control the transmission system. Similarly, to continuously monitor and control the distribution system, the concept of “distribution automation” system has emerged quite a few years ago [1]. In a distribution automation system, a lot of sensors and remote terminal units (RTUs) are placed in the power distribution system. The data collected by these sensors and RTUs are sent to a central computer station over a dedicated communication channel. The sent data are then analyzed by appropriate analysis software to assess the present health of the system. If any anomaly in the operation of the distribution system is detected, proper remedial and control decisions are taken at the central computer station by the help of proper software analysis packages. These control decisions are then sent to the field through the same or other dedicated communication channels for proper field implementation.

Obviously even with the presence of accurate sensors, RTUs and robust, reliable communication channels, the effectiveness of a distribution automation system largely depends upon the quality of the analysis software at the central computer station. As all the decisions regarding the present health and remedial control action are to be taken by

the software analysis packages, the algorithms of the software packages must be such that the results produced are highly accurate. Moreover, as the health monitoring functions and control decision functions are to be carried out quickly to minimize the adverse effects of any untoward conditions in a power distribution system, the algorithms of the software analysis packages also need to be very fast.

The most common different control decision functions, which are currently being employed in any modern distribution automation system, are :

- Feeder reconfiguration to minimize the loss in the system.
- Volt-var control.
- Accurate fault location in a distribution system.
- Feeder reconfiguration for service restoration.
- Feeder load balancing.
- Demand-side-management.
- Remote-monitoring.

Among the above listed functions, volt-var control is possibly one of the most important functions in any modern distribution automation system. The objective of volt-var control is to compensate for the reactive load demand locally, preferably at the site of the load itself, such that these reactive loads do not have to be supplied from the substation. Consequently, electric currents corresponding to these reactive currents do not flow over the feeders and hence the voltage-drop and power loss in the feeders decrease. Ideally, all reactive loads in the distribution system should be supplied locally, such that electric currents corresponding to only active loads flow over the feeders (which is anyway unavoidable), and thus voltage drop and power loss in the distribution system become minimum.

The most common approach to supply reactive power is to install capacitors in the system. Ideally, capacitors should be placed at all the load points in the system and if the reactive load demand in the system remains constant, then the rating of the installed capacitors can be made equal to the reactive load demand in the system. Thus, all the reactive load demand in the system would be met locally. However, the real-life situation is far from this scenario. Firstly, because of the cost consideration, it is not possible to install capacitors at each load point in the system. Hence, the capacitors are to be installed

at some strategic locations in the system. Secondly, the load demand in the system changes continuously. Hence it is not possible to compensate adequately for this changing load demand by fixed capacitors, rather the capacitors should be switchable such that the reactive power supplied by them vary with the variation in the load demand.

Thus, the volt-var control in a distribution system has two distinct aspects. First, to place capacitors (preferably switchable) at some strategic locations in the system. The location and sizes (rating of the capacitors in terms of KVAR supplied) of the capacitors are decided based on forecasted load demands (both active and reactive) in the system over a certain period (called the study period). This is known as the “capacitor placement” problem and this is an involved optimization problem. Second, once the capacitors are placed in the system based on “capacitor placement” study, it is necessary to determine their optimal switching pattern (which capacitor to be switched ‘ON’ and which capacitor to be switched ‘OFF’) at any loading condition such that the power loss and the voltage drop in the system are minimum. This is known as the “capacitor switching” or “capacitor allocation” problem and indeed, this is also an involved optimization problem.

Thus, the capacitor-switching algorithm pre-supposes the locations of the installed capacitors and it strives to find out the optimal switching pattern of these capacitors for minimum loss in the system at any given loading condition. Although significant amount of work has been reported in the literature for capacitor placement problem [2-4], not much work has been published in the literature in the area of optimal capacitor switching problem. An approach for determining optimal capacitor switching pattern based on sensitivity approach has been reported in [5]. This method calculates the sensitivity factors based on the repeated load-flow of the system.

However, as it has been mentioned earlier, for this information of optimal switching pattern to be of any use in the modern distribution automation system, it has to be obtained very quickly. Now, by the approach in [5], for a practical power distribution system involving thousands of feeders, it takes a comparatively long time duration (few minutes depending upon the system size) to perform the capacitor switching analysis and compute the optimal switching pattern. In this case, it may be desirable to reduce the computation time by some alternative technique such that the optimal capacitor switching pattern can be obtained quickly. Artificial intelligence (AI) techniques, such as artificial

neural network (ANN) offers such a possibility of computation of optimal switching pattern within a very short duration.

Artificial neural networks are computation tools, which try to mimic the operation of human brain. Analogous to the operation of the human brain, ANNs operate on the principle of parallel processing and consequently, they are quite fast, especially while dealing with large volume of data without any known mathematical correlation among the data. Clearly, optimal capacitor switching algorithm based on ANN technique is also expected to be quite fast and hence, it is expected that they will be quite suitable for on-line application in modern distribution automation system.

In this thesis, a methodology for computing optimal capacitor switching pattern in a distribution system based on ANN technique is developed. Essentially, the development of the methodology comprises of two steps :

1. A suitably chosen ANN structure is to be trained first with a set of training data (input : loading pattern, output : corresponding optimal switching pattern), with the help of a suitable training algorithm. The training data would be obtained from an optimal capacitor switching software package based on a conventional capacitor switching algorithm.
2. Once the ANN is trained, with sufficiently large number of training data, the ANN 'learns' the implicit correlation between the loading patterns and the optimal switching patterns. Next, new loading patterns (which have not been used to train the ANN) would be fed to the network and the network would provide the optimal switching pattern at its output within a very short span.

In this work, the standard multi-layer perceptron neural network with error back propagation training algorithm [6] has been used. The switching algorithm reported in [5] has been used for optimal capacitor switching analysis.

This thesis report is organized as follows. Chapter 2 discusses about the artificial neural network algorithm used in this thesis. This chapter discusses elaborately the multi-layer feed forward perceptron network and the error-back propagation training algorithm. In Chapter 3, conventional optimal capacitor switching algorithm is discussed. Chapter 4 presents the important results of this work. Chapter 5 delineates the main conclusions and gives a brief suggestion for further work.

ANN (ARTIFICIAL NEURAL NETWORK) ALGORITHM

The recent resurgence of interest in neural network has its roots in the recognition that the brain performs computation in a different manner than the conventional digital computers do. The foundation of an artificial neural system is nothing but man made neural systems, which results in major potential gain in the direction of information processing by digital computers. Man has endeavored to build an artificial neural system because of the fact that even an animal can process a visual information much better than the most modern, fastest computer. And of course, humans are more efficient in this regard than the other species. Hence, the gap between the information processing power of the human brain and even the most modern, fastest computer is huge. Artificial neural network is an effort towards bridging this gap.

Jurada [6] gave a brief but nice discussion on biological neuron systems and their correspondence with the man made artificial neural network systems. Hence, these fundamental concepts are not discussed here.

2.1 BRIEF REVIEW OF ANN

A neural network consists of simple processing units called 'neurons' or 'nodes', which bear only a passing resemblance to actual biological neurons. Each neuron is connected to other neurons in the network by unidirectional connections of different strengths or weights. The neurons are usually arranged in a series of layer bounded by input and output layers encompassing a variable number of hidden layers, connected in a structure which depends on the complexity of the problem to be solved.

The most important features of the network are their ability "to learn" or "to be trained" from examples. For many multi-input multi-output systems, the exact mathematical relationship between the inputs and the outputs is not often known in advance. Hence, for this kind of systems, it is often very difficult to determine the correct output pattern for a given input pattern. On the other hand, ANNs, by use of proper training algorithm, are able to identify or "learn" this implicit relationship. For the training

purpose, a number of input patterns and the corresponding output patterns are presented to the ANN. By virtue of suitable training algorithms, the ANN is able to “learn” the implicit co-relationship between the input and output patterns. As the learnt information is stored across the network weights, the network is able to generalize. This means that appropriate output pattern will be generated even for input pattern not actually included in the training. Because of their ability to learn and generalize, the neural networks have the potential for solving the required problem instantaneously, which takes much less time than the conventional methods.

In the literature, a large variety of neural network architecture and their training algorithms have been reported. A nice introduction to artificial neural network (architecture and training algorithm) is given in Jurada [6]. Essentially, all neural network architectures can be classified into two-categories [6] ;

- Feedforward network
- Feedback network

Similarly, the training algorithm for neural network can be divided into two broad categories [6] ;

- Supervised learning
- Unsupervised learning

Examples of supervised learning algorithms are [6] ;

- Perceptron learning algorithm
- Delta learning algorithm
- Widrow-Hoff learning algorithm
- Correlation learning algorithm
- Outstar learning algorithm

Examples of unsupervised learning algorithms are [6] ;

- Hebbian learning algorithm
- Winner-take-all learning algorithm

Among the above-mentioned learning algorithms, one of the most widely applied one is “delta learning algorithm”. In the literature, it is slightly modified and is named as “generalized delta learning algorithm” which ultimately gives rise to very popular “error-back-propagation learning algorithm”.

For the application of ANN for on-line capacitor switching, feedforward network with supervised learning has been used. Specifically, multi-layer feedforward network with error-back-propagation learning algorithm has been used in this work. Hence, in the next section, the error-back propagation (EBP) learning algorithm for multi-layer feedforward network is described [6].

2.2 ERROR-BACK-PROPAGATION LEARNING ALGORITHM

Fig. 2.1 shows a three-layer feedforward neural network. The layers are numbered 1, 2 and 3 from the leftmost side to the rightmost side as shown in the figure. At layer 1, which is called the 'input' layer, the input patterns are presented at the time of training. Similarly at layer 3 the output patterns are presented at the time of training of the network and hence, this layer is known as the 'output' layer. Layer 2, which is intermediate between the input and output layer, is known as the 'hidden' layer. Hence the above three-layer feedforward neural network is comprised of one input layer, one hidden layer and one output layer. It is to be noted that, it is possible to have more than one hidden layer in the neural network. However, in our study, only one hidden layer is used.

Each layer consists of several nodes. Number of nodes in the input layer is equal to the number of inputs. Similarly number of nodes at the output layer is equal to the number of outputs. There is no strict guideline, however, available for deciding the number of nodes in the hidden layer. In most cases, the optimum number of nodes in the hidden layer is dependent on the problem tackled by the neural network and most often this number is decided by trial and error.

There is, however, one important distinction between the input layer and the other layers (hidden and output). At the nodes of the input layer, no processing is done on the data or information presented to them. On the other hand, actual processing is done at the nodes of the hidden and output layers. Hence, essentially, the nodes of the hidden and output layers are the actual processing units or neurons, whereas the nodes of the input layer are just junction points where the input data are presented.

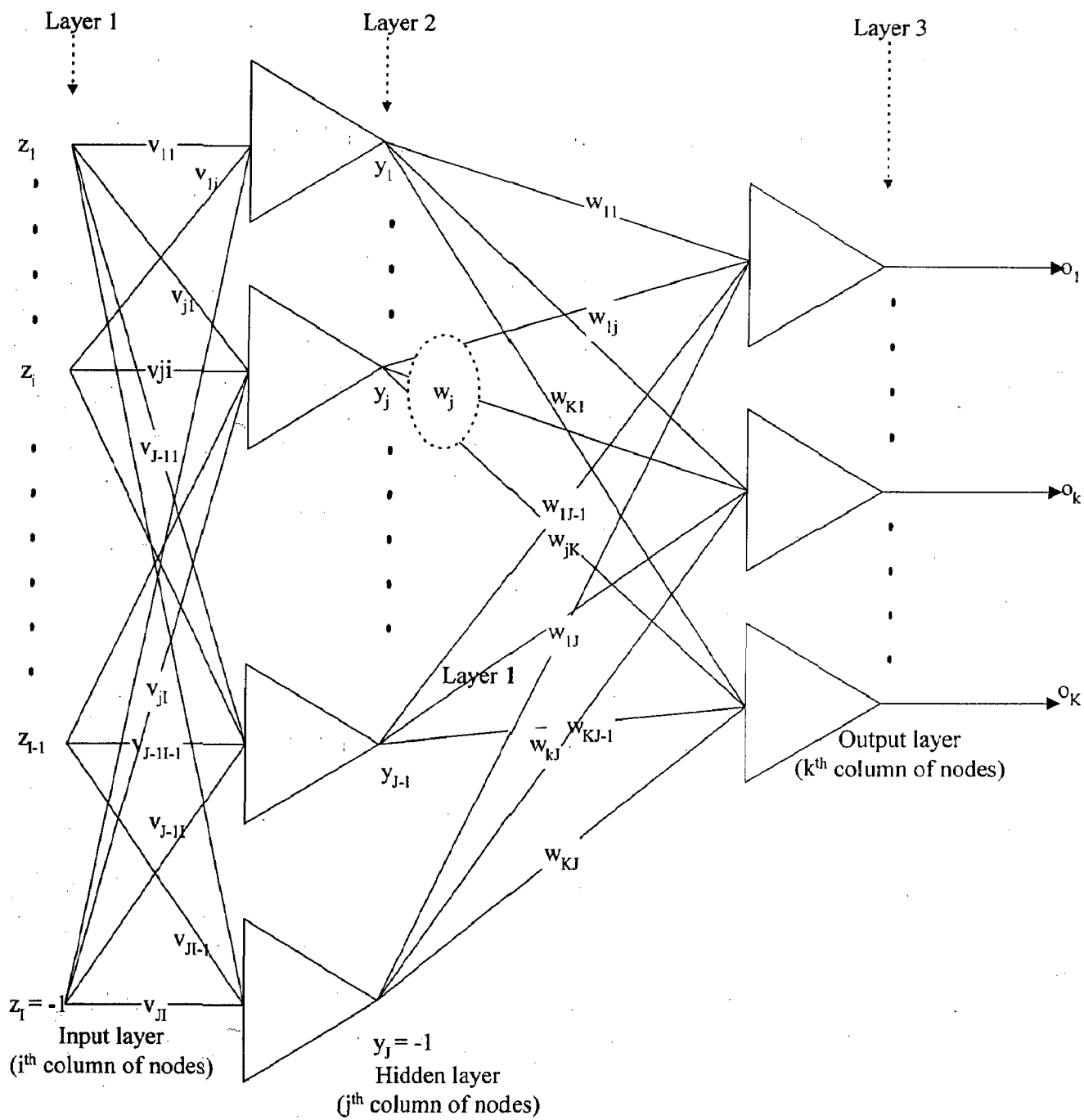


Figure 2.1

Layered feedforward neural network with two continuous perceptron layers

The nodes of the neural network are completely interconnected. That is, each node of the input layer is connected to every node of the hidden layer. Similarly, each node of the hidden layer is connected to every node of the output layer. The information always flows in the forward direction (i.e. from the input layer to the hidden layer and from the hidden layer to the output layer). Hence, this kind of network is known as multi-layer feedforward network.

The interconnection between every pair of nodes is represented by a weight w . Thus, the interconnections between the nodes of the input layer and the hidden layer can be represented by a weight matrix W of the dimension $(J \times I)$, where I is the number of input nodes and J is the number of nodes in the hidden layer

The matrix W is represented by $W = w_{ji}$ for $i = 1, 2, \dots, I$ and $j = 1, 2, \dots, J$ and w_{ji} denotes the weight of the interconnection between i^{th} node of the hidden layer and the j^{th} node of the input layer. Similarly, the interconnection between the nodes of the hidden layer and the nodes of output layer can be represented by a weight matrix $V = v_{kj}$ for $k = 1, 2, \dots, K$ and $j = 1, 2, \dots, J$. Here, it is assumed that there are K number of nodes at the output layer and v_{kj} denotes the weight of the interconnection between the k^{th} node of the output layer and the j^{th} node of the hidden layer.

Hence, the weight matrix W , interconnecting the nodes of the input layer and hidden layer is given by,

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1I} \\ w_{21} & w_{22} & \dots & w_{2I} \\ \dots & \dots & \dots & \dots \\ w_{J1} & w_{J2} & \dots & w_{JI} \end{bmatrix} \dots\dots(2.1)$$

Similarly, the weight matrix V interconnecting the nodes of the hidden layer and output layer is given by,

$$V = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1J} \\ v_{21} & v_{22} & \dots & v_{2J} \\ \dots & \dots & \dots & \dots \\ v_{K1} & v_{K2} & \dots & v_{KJ} \end{bmatrix} \dots\dots(2.2)$$

To train the network, the input-output patterns are presented to the network. Let there is total P number of input-output patterns. In each pattern, let there are I inputs and K outputs. Let the inputs are represented by an input vector \mathbf{z} , where \mathbf{z} is denoted by the symbol,

$$\mathbf{z} = [z_1, z_2, \dots, z_I]^T \quad \dots\dots(2.3)$$

where 'T' denotes the transpose of a vector. Similarly, the outputs are represented by an output vector \mathbf{d} , where

$$\mathbf{d} = [d_1, d_2, \dots, d_K]^T \quad \dots\dots(2.4)$$

As described earlier, the information flow in an ANN proceeds from the input layer to the output layer via the hidden layers. The computation process for this information flow is described below. For a given input pattern \mathbf{z} , input y_j at the j^{th} hidden node is calculated by,

$$y_j = \sum_{i=1}^I w_{ji} z_i; \text{ for } j = 1, 2, \dots, J \quad \dots\dots(2.5)$$

Let the inputs at the hidden layer be represented by the vector \mathbf{y} , where $\mathbf{y} = [y_1, y_2, \dots, y_J]^T$. Then from Equation (2.5), vector \mathbf{y} can also be calculated by the following expression,

$$\mathbf{y} = \mathbf{WZ} \quad \dots\dots(2.6)$$

Output p_j of the j^{th} hidden node is then calculated by,

$$p_j = f(y_j) = \frac{1}{1 + \exp(-\lambda y_j)}; \text{ for } j = 1, 2, \dots, J \quad \dots\dots(2.7)$$

where λ is a constant.

Once the outputs at the nodes of the hidden layer are calculated, the input r_k to k^{th} node at the output layer is given by,

$$r_k = \sum_{j=1}^J v_{kj} p_j; \text{ for } k = 1, 2, \dots, K \quad \dots\dots(2.8)$$

The output o_k of the k^{th} node of the output layer is given by,

$$o_k = f(r_k) = \frac{1}{1 + \exp(-\lambda r_k)}; \text{ for } k = 1, 2, \dots, K \quad \dots\dots(2.9)$$

It is to be noted that the output d_k for the k^{th} node at the output layer is the desired output, whereas o_k is the calculated output by the ANN based on the weight matrices and the input vector. By the process of training, ANN tries to make o_k equal to d_k by modifying the weights of the interconnections suitably. In the error-back propagation algorithm, the weights are modified in such a way that the total error at the output layer between the desired output and the calculated output is minimized.

The total error is defined as,

$$E = \frac{1}{2} \sum_{k=1}^K (d_k - o_k)^2 \quad \text{.....(2.10)}$$

According to the classical optimization theory, to minimize E , the weights should be changed in the negative direction of the gradient of E . Hence, the adjustments of weights Δv_{kj} and Δw_{ji} are given by,

$$\Delta v_{kj} = -\eta \frac{\partial E}{\partial v_{kj}} \quad \text{.....(2.11)}$$

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \quad \text{.....(2.12)}$$

Where η is a constant, called learning constant.

Now,

$$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial r_k} \frac{\partial r_k}{\partial v_{kj}} \quad \text{.....(2.13)}$$

From equation (2.8), we have,

$$\frac{\partial r_k}{\partial v_{kj}} = p_j \quad \text{.....(2.14)}$$

Again,

$$\frac{\partial E}{\partial r_k} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial r_k} \quad \text{.....(2.15)}$$

From equation (2.9) we have,

$$\frac{\partial o_k}{\partial r_k} = f'(r_k) \quad \text{.....(2.16)}$$

Now, for $\lambda = 1$,

$$f'(r_k) = \frac{\exp(-r_k)}{[1 + \exp(-r_k)]^2} = \frac{1}{[1 + \exp(-r_k)]} \frac{1 + \exp(-r_k) - 1}{[1 + \exp(-r_k)]}$$

Or,

$$f'(r_k) = o_k(1 - o_k) \quad \dots\dots(2.17)$$

Also, from equation (2.10), we have,

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k) \quad \dots\dots(2.18)$$

Hence, combining equations (2.11), (2.14), (2.17) and (2.18) we have,

$$\Delta v_{kj} = \eta(d_k - o_k)o_k(1 - o_k)p_j \quad \dots\dots(2.19)$$

Hence, the modified weight v'_{jk} becomes

$$v'_{kj} = v_{kj} + \Delta v_{kj} = v_{kj} + \eta(d_k - o_k)o_k(1 - o_k)p_j \quad \dots\dots(2.20)$$

for $j = 1, 2, \dots\dots J$ and $k = 1, 2, \dots\dots K$

For the adjustment of weights between the input layer and hidden layer,

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}; \quad \dots\dots(2.21)$$

for $i = 1, 2, \dots\dots I$ and $j = 1, 2, \dots\dots J$

Now,

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} \quad \dots\dots(2.22)$$

From equation (2.5) we have,

$$\frac{\partial y_j}{\partial w_{ji}} = z_i \quad \dots\dots(2.23)$$

Now,

$$\frac{\partial E}{\partial y_j} = \frac{\partial E}{\partial p_j} \frac{\partial p_j}{\partial y_j} \quad \dots\dots(2.24)$$

Now, proceeding as in equation (2.17) and from equation (2.7) we have,

$$\frac{\partial p_j}{\partial y_j} = f'(y_j) = p_j(1-p_j) \quad \dots\dots\dots(2.25)$$

Again,

$$\frac{\partial E}{\partial p_j} = \frac{\partial}{\partial p_j} \left\{ \frac{1}{2} \sum_{k=1}^K (d_k - o_k)^2 \right\}$$

Or,

$$\frac{\partial E}{\partial p_j} = \frac{\partial}{\partial p_j} \left[\frac{1}{2} \sum_{k=1}^K \{d_k - f(r_k)\}^2 \right]$$

Or,

$$\frac{\partial E}{\partial p_j} = - \sum_{k=1}^K (d_k - o_k) \frac{\partial}{\partial p_j} \{f(r_k)\}$$

Or,

$$\frac{\partial E}{\partial p_j} = - \sum_{k=1}^K (d_k - o_k) f'(r_k) \frac{\partial r_k}{\partial p_j} \quad \dots\dots\dots(2.26)$$

From equation (2.8), we have,

$$\frac{\partial r_k}{\partial p_j} = v_{kj} \quad \dots\dots\dots(2.27)$$

Putting equation (2.27) and (2.17) into equation (2.26) we have,

$$\frac{\partial E}{\partial p_j} = - \sum_{k=1}^K (d_k - o_k) o_k (1 - o_k) v_{kj} \quad \dots\dots\dots(2.28)$$

Hence, from equations (2.24), (2.25) and (2.28), we have,

$$\frac{\partial E}{\partial y_j} = - p_j (1 - p_j) \sum_{k=1}^K (d_k - o_k) o_k (1 - o_k) v_{kj} \quad \dots\dots\dots(2.29)$$

From, equations (2.21), (2.22), (2.23) and (2.29) we have,

$$\Delta w_{ji} = \eta z_i p_j (1 - p_j) \sum_{k=1}^K (d_k - o_k) o_k (1 - o_k) v_{kj} \quad \dots\dots\dots(2.30)$$

Let us define,

$$\delta_{ok} = (d_k - o_k) o_k (1 - o_k)$$

Then from equation (2.20),

$$\begin{aligned} v'_{kj} &= v_{kj} + \eta \delta_{ok} p_j \\ \text{for } j &= 1, 2, \dots, J \text{ and } k = 1, 2, \dots, K \end{aligned} \quad \dots\dots(2.31)$$

And, from equation (2.30),

$$\Delta w_{ji} = \eta z_i p_j (1 - p_j) \sum_{k=1}^K \delta_{ok} v_{kj} \quad \dots\dots(2.32)$$

Hence, the modified weights w'_{ji} is given by,

$$w'_{ji} = w_{ji} + \eta z_i p_j (1 - p_j) \sum_{k=1}^K \delta_{ok} v_{kj} \quad \dots\dots(2.33)$$

Equations (2.31) and (2.33) describe the relationships through which the various weights in the ANN are to be modified. It is to be noted that this modification proceeds backward from the output layer to the input layer via the hidden layers. The modifications are dependent upon the error at the output layer. Hence this training algorithm is known as the “Error-back-propagation training algorithm”. Note that the relationships described in equations (2.31) and (2.33) are for one single input-output pattern. When P patterns are presented, the weights are modified for each pattern according to the equations (2.31) and (2.33) and the cumulative error (summation of the total errors over all the P patterns) is calculated. If the cumulative error is less than a specified tolerance limit, the ANN is said to have converged or “trained” properly. This entire sequence of modification of the weights over all the P patterns is known as one iteration of training of ANN. If the cumulative error is not less than the specified tolerance value, another iteration of training starts. If the training of the ANN is proceeding properly, the cumulative error in the subsequent iterations should reduce. Iterations of training are continued till the cumulative error becomes less than the specified limit.

To accelerate the convergence of the error-back-propagation algorithm, a factor called “momentum factor” is often used. In this method, the weight adjustment in the current step is supplemented with a fraction of the immediate past weight adjustment. Mathematically this is expressed as,

$$\Delta W(t) = \eta \nabla E(t) + \alpha \Delta E(t - 1)$$

Where, α = Momentum factor
 t = For current training step
 $t - 1$ = For immediate past training step

Based on the above discussions, the detailed algorithm for error-back-propagation training procedure for multi-layer feedforward ANN is described below.

ERROR BACK PROPAGATION ALGORITHM (EBPTA) [6]

Given are P training pairs $\{z_1, d_1, z_2, d_2, \dots, z_P, d_P\}$

where z_i is $(I \times 1)$, d_i is $(K \times 1)$ and $i = 1, 2, \dots, P$

The input vectors have been augmented by fixing the I^{th} component of each z_i to a value -1.0 . Size $J-1$ of the hidden layer having outputs \mathbf{p} is selected. As hidden layer output have also been augmented, the J^{th} component of \mathbf{p} is also of value -1.0 . \mathbf{p} is $(J \times 1)$ and \mathbf{o} is $(K \times 1)$, where \mathbf{o} is the output vector.

Step 1: $\eta \leftarrow 0$, ϵ_{max} chosen.

Weight \mathbf{V} and \mathbf{W} are initialized at small random values; \mathbf{V} is $(K \times J)$ and \mathbf{W} is $(J \times I)$.

$$q \leftarrow 1, p \leftarrow 1, E \leftarrow 0$$

Step 2: Training step starts here.

Input is presented and the layers' outputs computed :

$$z \leftarrow z_p, d \leftarrow d_p, p_j \leftarrow f(w_j^T z)$$

where w_j , a column vector, is j -th row of \mathbf{W} , and

$$o_k \leftarrow f(v_k^T p)$$

where v_k , a column vector, is the k -th row of \mathbf{V} .

Step 3: Error value is computed:

$$E \leftarrow \frac{1}{2} [d_k - o_k]^2 + E, \text{ for } k = 1, 2, \dots, K$$

Step 4: Error signal vector δ_o and δ_p of both layers are computed.

Vector δ_o is $(K \times 1)$, δ_p is $(J \times 1)$.

The error signal terms of the output layer in this step are

$$\delta_{ok} = (d_k - o_k)(1 - o_k)o_k$$

for $k = 1, 2, \dots, K$

The error signal terms of the hidden layer in this step are

$$\delta_{pj} = p_j(1 - p_j) \sum_{k=1}^K \delta_{ok} v_{kj}, \text{ for } j = 1, 2, \dots, J$$

Step 5: Output layer weights are adjusted:

$$v_{kj} \leftarrow v_{kj} + \eta \delta_{ok} p_j,$$

for $k = 1, 2, \dots, K$ and $j = 1, 2, \dots, J$

Step 6: Hidden layer output are adjusted:

$$w_{ji} \leftarrow w_{ji} + \eta \delta_{pj} z_i,$$

for $j = 1, 2, \dots, J$ and $i = 1, 2, \dots, I$

Step 7: If $p < P$ then $p \leftarrow p + 1$, $q \leftarrow q + 1$, and go to step 2: otherwise, go to step 8.

Step 8: The training cycle is completed.

$$\text{rms error, } E \leftarrow \frac{1}{PK} \sqrt{E}$$

For $E < \text{emax}$, terminate the training session. Output weights W , V , q and E .

If $E > \text{emax}$, then $E \leftarrow 0$, $p \leftarrow 1$, and initiate the new training cycle by going to step 2.

Once the neural network is trained, the input pattern for which the output pattern is to be obtained, is presented to the ANN and the output is calculated in the feedforward mode following step 2 of the algorithm.

2.3 CONCLUSION

In this chapter, the detail algorithm of the error back propagation training algorithm for multi-layer feedforward neural network is discussed. For the development of ANN based optimal capacitor switching algorithm, this algorithm has been used for training the neural network.

OPTIMAL CAPACITOR SWITCHING ALGORITHM

One of the most important and desirable control function in a modern distribution automation system is volt-var control (VVC). Primarily, VVC deals with the voltage and reactive power control in the distribution system, although in concrete terms, the main objective of VVC may not be directly minimizing the voltage violations or reactive power violations. One of the typical objectives of VVC is the real power loss minimization subjected to various operating constraints in the system. As discussed in Chapter 1, this objective can be achieved by controlling the reactive power support at different buses in the system. Different constraints in the system may be in the form of voltage constraints at different buses, loading constraints at different lines, power factor and reactive power demand constraints at the substation etc. Additional to the requirement of minimization of power loss, it is also desirable that the solution be obtained in least possible number of steps.

To solve this constrained optimization problem, several approaches have been suggested in the literature. A co-ordination method for switching discrete switchable capacitors and tap changers has been proposed in [7]. This approach is based on a simplifying assumption that all capacitors and tap changers have equal increments. Roytelman et. al. [5] solves this problem by oriented discrete gradient method. In this method, the best direction for search is determined by calculating sensitivity factors.

In this work, the basic approach outlined in [5] has been followed to determine the optimal capacitor switching patterns. Although the basic philosophy remains the same, for this work, it has been slightly modified. In the following sections, the optimal capacitor switching algorithm is described in detail.

3.1 OBJECTIVE FUNCTION AND CONSTRAINTS

From a mathematical point of view, the optimal capacitor switching problem is a constrained minimization problem where the constraints are inequality constraints. In this

work, the objective is to minimize the real-power loss in the distribution system. Consequently, the objective function is

$$P_L = \sum_{i=1}^{nl} |I_i|^2 r_i \quad \dots\dots(3.1)$$

Where, $P_L \rightarrow$ Total real power loss in the system.

$nl \rightarrow$ Number of feeders in the system.

$r_i \rightarrow$ Resistance of the i^{th} feeder.

$I_i \rightarrow$ Magnitude of the current flow through i^{th} feeder.

The value of P_L is computed by using an accurate load-flow program given the load-profile in the system and the setting of other control variables (such as transformer tap settings, settings of any shunt-connected reactors, capacitors etc.). The objective is to minimize the value of P_L subjected to different inequality constraints. Also, it is desirable that the solution of this problem be obtained at minimum number of steps.

The different inequality constraints in the system are :

- (a) Voltage at all the buses should be within some specified minimum and maximum limits.

Mathematically,

$$V_i^{\min} \leq V_i \leq V_i^{\max} \quad \text{for } i = 1, 2, \dots, nb \quad \dots\dots(3.2)$$

Where, $nb \rightarrow$ Number of buses in the system

$V_i \rightarrow$ Voltage at i^{th} bus

$V_i^{\min}, V_i^{\max} \rightarrow$ Minimum and Maximum limits of the voltages of the i^{th} bus

- (b) Current flow through each feeder is less than the maximum specified limit.

$$I_i \leq I_i^{\max} \quad \text{for } i = 1, 2, \dots, nl \quad \dots\dots(3.3)$$

$I_i^{\max} \rightarrow$ Maximum value of I_i

- (c) Current flow through each transformer must be less than the maximum allowed limit.

$$I_i^t \leq I_i^{t\max} \quad \text{for } i = 1, 2, \dots, nt \quad \dots\dots(3.4)$$

Where, $nt \rightarrow$ Number of transformers in the system

$I_i^t \rightarrow$ Current through i^{th} transformer

$I_i^{\text{tmax}} \rightarrow$ Maximum allowable current through i^{th} transformer

- (d) Power factor at the substation must be within certain minimum and maximum limits.

$$PF_{\text{sub}}^{\min} \leq PF_{\text{sub}} \leq PF_{\text{sub}}^{\max} \dots\dots(3.5)$$

Where, $PF_{\text{sub}} \rightarrow$ Power factor at the substation

$PF_{\text{sub}}^{\min} \rightarrow$ Minimum allowable limit for PF_{sub}

$PF_{\text{sub}}^{\max} \rightarrow$ Maximum allowable limit for PF_{sub}

- (e) Reactive power demand at the distribution substation must be within specified limits.

$$Q_{\text{sub}}^{\min} \leq Q_{\text{sub}} \leq Q_{\text{sub}}^{\max} \dots\dots(3.6)$$

Where, $Q_{\text{sub}} \rightarrow$ Reactive power demand at the distribution substation

$Q_{\text{sub}}^{\min} \rightarrow$ Minimum limit of Q_{sub}

$Q_{\text{sub}}^{\max} \rightarrow$ Maximum limit of Q_{sub}

In this work, among all the above constraints, only the constraint described in equation (3.2) is considered. Hence, the statement of the constrained optimization problem is as follows:

“Under the current loading condition, find the optimal switching pattern of the already installed shunt capacitors in the distribution system such that the real power loss in the system is minimum and simultaneously the voltages at all the buses lie within their respective operating limits”.

3.2 SOLUTION METHODOLOGY

As the capacitors to be switched are discrete, not continuous, in nature, the optimal capacitor switching problem is essentially a discrete (integer) programming problem with non-linear objective function and inequality constraints. To solve such kind of discrete optimization problem, combinatorial methods are generally used. In combinatorial method, all possible discrete solutions are checked and finally the most optimal solution is

selected. Depending upon the size and nature of the optimization problem, the number of possible solutions may be very large and hence, time taken to find the most optimum solution may be very large. To overcome this problem, combinatorial methods use special strategies such that only effective subsets of all possible solutions are searched.

One of the most commonly used, simple and reliable combinatorial strategies is gradient descent method. This method can be used for any type of variables and objective function. In this method, the control variables are moved by a reasonable step in that direction in which objective function decreases the most. For discrete control variables, the chosen step size is generally equal to the discrete increments of the control variables. This direction (which is generally termed ‘the descent direction’) can be chosen by a number of methods, such as (a) by chance (Monte Carlo method), (b) by any evaluation strategy or (c) by the largest negative partial derivative of the objective function with respect to the control variables. In the last case, the method is known as the gradient or oriented discrete co-ordinate descent method.

In this work, the oriented discrete co-ordinate descent method has been used to determine the optimal capacitor switching pattern for a given loading condition. In this method, as discussed earlier, the partial derivative of the objective function F is calculated with respect to the discrete control variables X_i at the current operating point. The partial derivative is computed as the ratio of the differences in the objective function to the corresponding increments in the discrete control variables. Mathematically,

$$\frac{\partial F^k}{\partial X_i^k} = \frac{F^{k+1} - F^k}{X_i^{k+1} - X_i^k} \dots\dots(3.7)$$

Where k, k+1 are the current and the next position of the control variables. The corresponding values of the objective function are denoted by F^k and F^{k+1} respectively.

In this work, the objective function is the real power loss in the distribution system, which is calculated by a load-flow program given the system loading pattern and the size and switching status of the discrete capacitor bank. The switching status of the shunt capacitor bank represents the control variables. The rated sizes of the capacitor bank are the increments for the capacitors.

The variables (switching pattern) which gives rise to the largest negative partial derivative is the direction at which the capacitors should be switched. Starting from the initial operating condition, this switching process is repeated till there is no further decrease in the real power loss in the distribution system.

Based on the above discussion, the algorithm for finding the solution of optimal capacitor switching problem is as follows:

Step 1. Read the input data.

Step 2. From the input data, identify the buses at which the discrete capacitor banks are connected, the size of the connected capacitor banks and the initial switching status of the capacitor banks (i.e. which capacitor is 'ON' and which capacitor is 'OFF'). The input operating condition is termed as the 'base operating condition'.

Step 3. Let n_c denotes the number of capacitors connected in the system. Set $MODE_i =$ "SWITCHABLE". Where $MODE_i =$ mode of the capacitor i , either "SWITCHABLE" or "FIXED", for $i = 1, 2, \dots, n_c$.

Step 4. Calculate the real power loss at the base operating condition. Let this be denoted as PL_B .

Step 5. For each capacitor $i = 1, 2, \dots, n_c$, perform the following steps, if $MODE_i$ is not equal to "FIXED".

(a) Change the status of the capacitor. That is, if the capacitor is already 'ON', then make it 'OFF' or if the capacitor is already 'OFF', make it 'ON'. The status of the remaining capacitors remains unchanged.

(b) Calculate the power loss in the system at this new configuration by running load-flow. Let this loss be denoted as PL_i .

(c) Calculate the partial derivative PD_i by the following relationship:

$$PD_i = \frac{PL_i - PL_B}{C_i}$$

where C_i is the capacitance value of the i^{th} capacitor.

(d) After the load-flow solution is obtained, check whether the voltage at any bus violates the minimum and maximum limits. If there is no violation, store the value of PD_i . If there is violation, do not store the value of PD_i .

(e) Restore the original status of the capacitor i .

Step 6. From the stored values of PD_i , find the index j , $1 \leq j \leq nc$, For which PD_j is negative maximum. If all the stored partial derivatives (PD_i) are positive, optimal switching pattern has been achieved and go to step 11. Otherwise, go to step 7.

Step 7. Set PD_B equal to PD_j .

Step 8. Change the status of the capacitor j .

Step 9. Set $MODE_j = \text{"FIXED"}$.

Step 10. Go back to Step 5.

Step 11. Print the total system loss and the switching status of the capacitors.

3.3 CONCLUSION

In this chapter, an oriented discrete co-ordination descent method is described for finding the optimal capacitor switching pattern under a given loading condition. In this work, only the constraint of bus-voltage violation has been taken into consideration. By the use of this algorithm, a large number of optimal switching patterns under varying load conditions can be generated for training and testing of ANN.

RESULTS AND DISCUSSION

It has been already discussed in Chapter 1 that the main objective of developing an ANN based optimal capacitor switching algorithm is to reduce the time taken to arrive at the optimal capacitor switching pattern for a given loading pattern. To achieve this objective, some researchers have reported ANN based capacitor switching technique in the literature [8]. However, in [8], multi-stage ANN architecture has been used, which enhances the complexity of implementation of the proposed ANN technique. On the other hand, if the same objective can be achieved with a single-stage ANN architecture, the implementation of the ANN algorithm would be simpler. In this work, a three-layer, single stage, feedforward neural network has been used for developing the ANN based optimal capacitor switching algorithm.

As has been already discussed in Chapter 1, development of any ANN based algorithm essentially consists of two steps. Firstly, the ANN should be trained properly and secondly, once the ANN is trained, the accuracy of the result predicted by the ANN must be tested. To train the ANN, a number of input-output pairs (input : loading pattern of the distribution system and output : optimal capacitor switching pattern) must be submitted to the ANN. These input-output pairs are obtained by a conventional optimal capacitor switching algorithm at different loading patterns in the system. During training, the weights of the interconnections in the ANN architecture are suitably modified according to the training algorithm. Once the ANN is trained, the performance of the ANN is tested by comparing the output predicted by the ANN with that obtained by the conventional algorithm.

To illustrate the development of an ANN based optimal switching algorithm, a sample 30-bus system [8] has been chosen. The one-line diagram of the network is shown in Fig. 4.1. The data for this test system are given in Tables A.1 and A.2 in Appendix A. The loading pattern given in Table A.1 is termed as "base operating condition". In this system, there are total 22 load points. At each load point, both real (KW) and reactive (KVAR) loads are specified. Hence, total number of real and reactive loads in the system

is 44. There are also 17 positions of switchable capacitors installed in the system. Hence, the ANN architecture used in this work has 44 input nodes and 17 output nodes. Only one hidden layer of the ANN has been chosen and the number of nodes in the hidden layer has been chosen as 15.

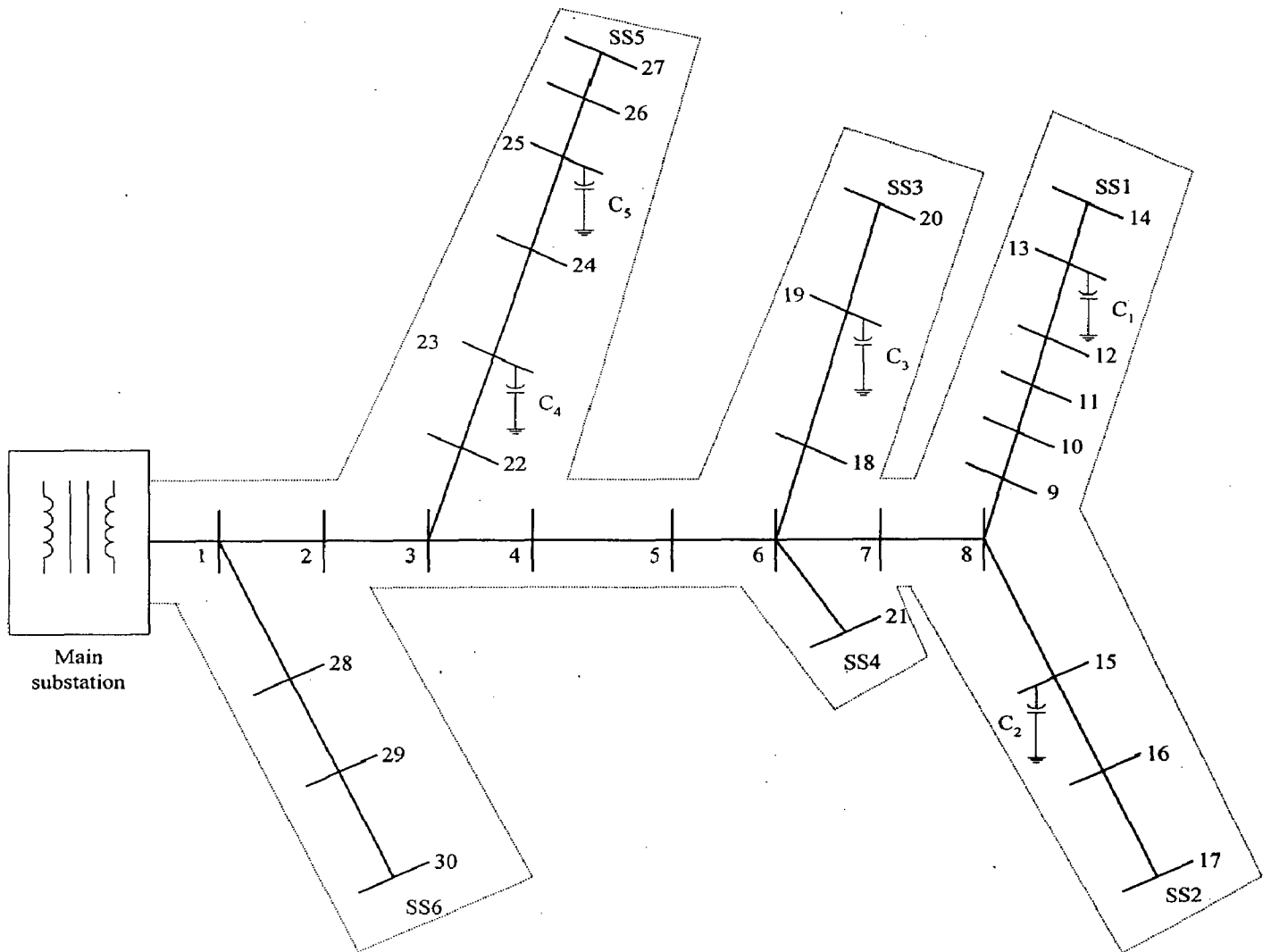


Figure 4.1 : One-line Diagram of the Sample Distribution System

To train the ANN, it is necessary to generate a number of input-output patterns at different loading conditions. The different loading conditions in the system are achieved by varying the KW and KVAR loads in the system within a certain range with respect to the “base operating condition”. For example, the KW and KVAR loads can be varied in

such a way that the new loading condition always remains within a range of 90% - 110% of the “base operating condition”. Similarly, for any other specified ranges, the KW and KVAR loads can be varied accordingly. In this work, four different ranges of loading have been considered. These are, a) 90% - 110%, b) 80% - 120%, c) 70% - 130% and d) 60% - 140%, of the “base operating condition”. To vary the loading within any specified range (e.g. 90% - 110%), a large quantity of random numbers within a range of 0.9 – 1.1 have been generated and subsequently, the KW and KVAR loads at “base operating condition” have been multiplied by these random numbers. Consequently, a large number of loading conditions within a range of 90% - 110% of the “base operating condition” are generated. Once these new loading conditions are generated, the oriented discrete descent method described in Chapter 3 have been used to find out the optimal capacitor switching patterns at each of these new loading conditions. Thus, a number of input-output patterns are generated. Out of these generated input-output patterns, some of the patterns have been used to train the network and some of remaining patterns have been used to test the performance of the trained ANN. To train the ANN, error-back-propagation training algorithm as described in Chapter 2 has been used.

The results of ANN training and testing, when the loading has been varied within a range of 90% - 110% of the “base operating point”, are tabulated in Table 4.1. As observed from this table, for this operating range, 10000 input-output patterns have been used to train the ANN and after the ANN is trained, 1000 input-output patterns, which had not been used during training, have been used to test the performance of the ANN. The performance of the ANN is quantified by “test error (percentage mismatch)”. It is shown in Table 4.1 that for learning constant = 0.01 and momentum factor = 0.8, the “test error (percentage mismatch)” is 3.57. The meaning of the “test error” is as follows. As discussed earlier, each pattern has 17 output values. Hence, for total 1000 test patterns, there are total 17000 output values. Upon testing, the ANN predicts wrong output values for 3.57% of 17000 output values, i.e, for $3.57 \times 170 = 607$ output values, the predictions of the ANN do not match with the output values obtained from oriented discrete descent method. For the rest $17000 - 607 = 16393$ output values, predictions of ANN exactly match with those obtained from the oriented discrete descent method. Table 4.1 also shows the rms error during training after 50 cycles of ANN training for this combination

of learning constant and momentum factor. It may be argued that the rms error during training would probably reduce had the training of the ANN been continued for more number of cycles and consequently, the test error would also reduce. To verify this, a graph of rms error during training with respect to the number of training cycles has been plotted as shown in Fig. 4.2. From this figure, it is observed that the training error remains practically constant after 5 cycles of training. Hence, the error would not further reduce even with more number of training cycles.

Table 4.1

I = 45, J = 15, K = 17, $\lambda = 1.0$, Number of training patterns = 10000			Range of loading = 90% - 110% Number of test patterns = 1000	
Value of learning constant (η)	Value of momentum factor (mf)	No of cycles taken at convergence	rms error at convergence	Test error (percentage mismatch)
0.01	0.8	50	0.000208	3.57
0.6	0.9	30	0.000209	6.21
0.05	0.6	39	0.000209	5.02

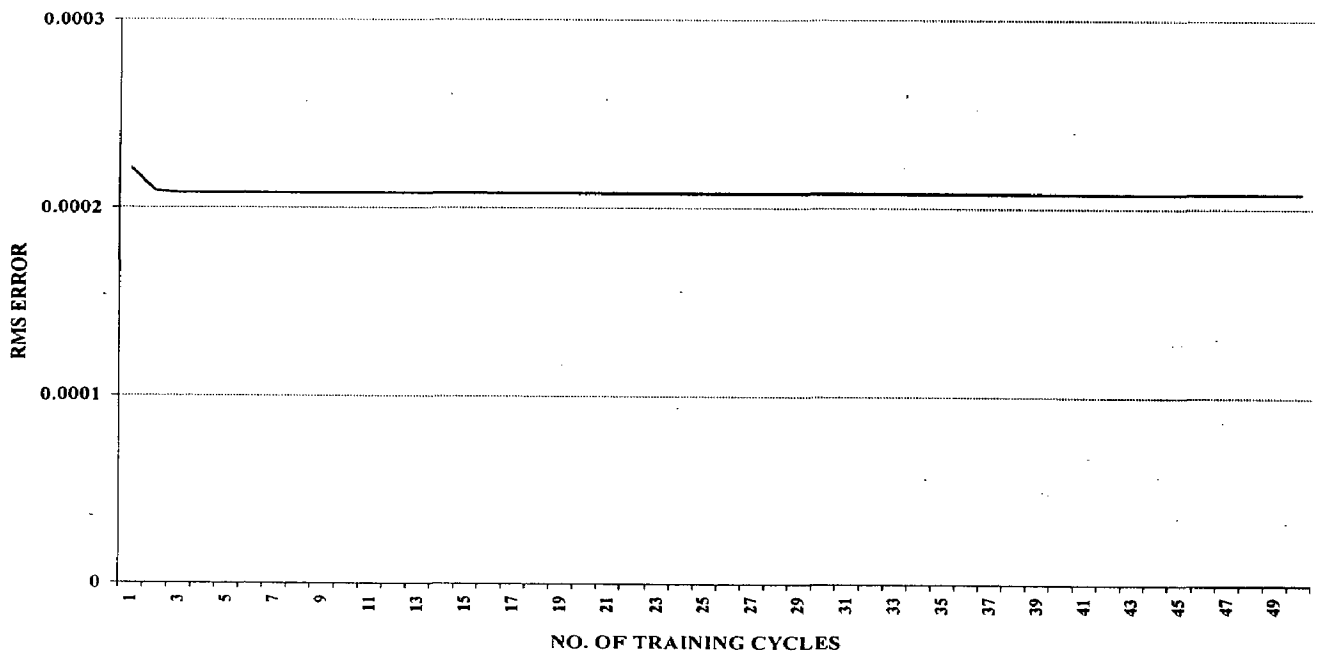


Figure 4.2 : Operating range 90% - 110%

Table 4.1 also shows the training and testing results for other combinations of learning constant and momentum factor. It is observed that for all the other combinations, the performance of the ANN is worse than that obtained with the first combination (learning constant = 0.01 and momentum factor = 0.8). Different other combinations of learning constant and momentum factor have also used. In every case, the test error has been found to be greater than 3.57%. Hence, this combination of learning constant and momentum factor is the most optimum combination and to denote this, the results corresponding to this combination in Table 4.1 have been given in boldface letters.

Results corresponding to the operating range 80% - 120% are given in Table 4.2. As in the case of Table 4.1, the results for most optimum combination are shown in boldface letters and the plot of rms training error Vs. number of training cycles for the optimum combination is shown in Fig. 4.3. Similarly, corresponding results and plot for the operating range 70% - 130% are given in Table 4.3 and Fig. 4.4 respectively and those for the operating range 60% - 140% are shown in Table 4.4 and Fig. 4.5 respectively.

Table 4.2

I = 45, J = 15, K = 17, $\lambda = 1.0$, Number of training patterns = 20000		Range of loading = 80% - 120% Number of test patterns = 1000		
Value of learning constant (η)	Value of momentum factor (mf)	No of cycles taken at convergence	rms error at convergence	Test error (percentage mismatch)
0.05	0.6	220	0.000189	6.58
0.001	0.9	75	0.000191	12.572
0.6	0.9	70	0.000192	16.783
0.67	0.6	50	0.000125	31.727

From tables 4.1 – 4.4 it is found that best performance of the ANN is achieved for the lowest range (90% - 110%) and the highest range (60% - 140%). For the lowest range, the outputs of various patterns were quite close to each other. Hence, after training, the ANN was able to predict the correct output patterns for most of the input patterns as it was able to learn the implicit relation between the input and output patterns relatively easily. On the other hand, for the highest range, the outputs of various patterns were diverse in nature. Hence, to learn the implicit relation properly, number of training patterns required

by the ANN was double (20,000) the number that was required for the lowest range (10,000). For the other two ranges, although 20000 training patterns were used, the test error of the ANN was more. It may be possible that if more number of training patterns were used or some other combination of learning constant and momentum factor were used, possibly the performance of the ANN would have been better for the other two ranges. But as the highest range also covers the loading patterns in the other ranges and the performance of the ANN in the highest range is almost the best, this exercise was not felt necessary. It is to be noted that the values of the learning constant and the momentum factor were found by trial and error method. From the tables, it can also be observed that for best performance, the value of the learning constant preferably be in the order of 0.01 and the value of the momentum factor preferably be in the range of 0.6 – 0.8. It is to be noted that these values are not universal in nature, they are only valid for the test system under consideration. For any other system, the appropriate values of these two constants must be found out by trial and error method. It may be argued that it is possible to reduce the test errors of the ANN further by other combinations of number of training patterns, learning constant and the momentum factor. However, by carrying out large number of training sessions of the ANN by various combination of the learning constant and momentum factor, it was found that to improve the performance of the ANN slightly better, the time taken for the training becomes unacceptably large. Hence, those results are not given in this thesis report.

It has been already discussed in Chapter 1 that the prime motive behind the development of ANN based algorithm is to reduce the time taken for deciding the optimal capacitor switching pattern for a given loading condition. It has been found that for 1000 loading patterns, the CPU time taken by the oriented discrete descent method for determining corresponding optimal switching patterns is 380.439545 secs, whereas, the CPU time taken by the ANN is 1.648351 secs. Hence, the ANN based method is almost 230 times faster. Obviously, for larger systems, the difference between the time taken by the combinatorial method and the ANN would be more and consequently, for a practical, real life large system, the ANN based method would be much faster than the traditional methods and hence, would be more suitable for on-line application.

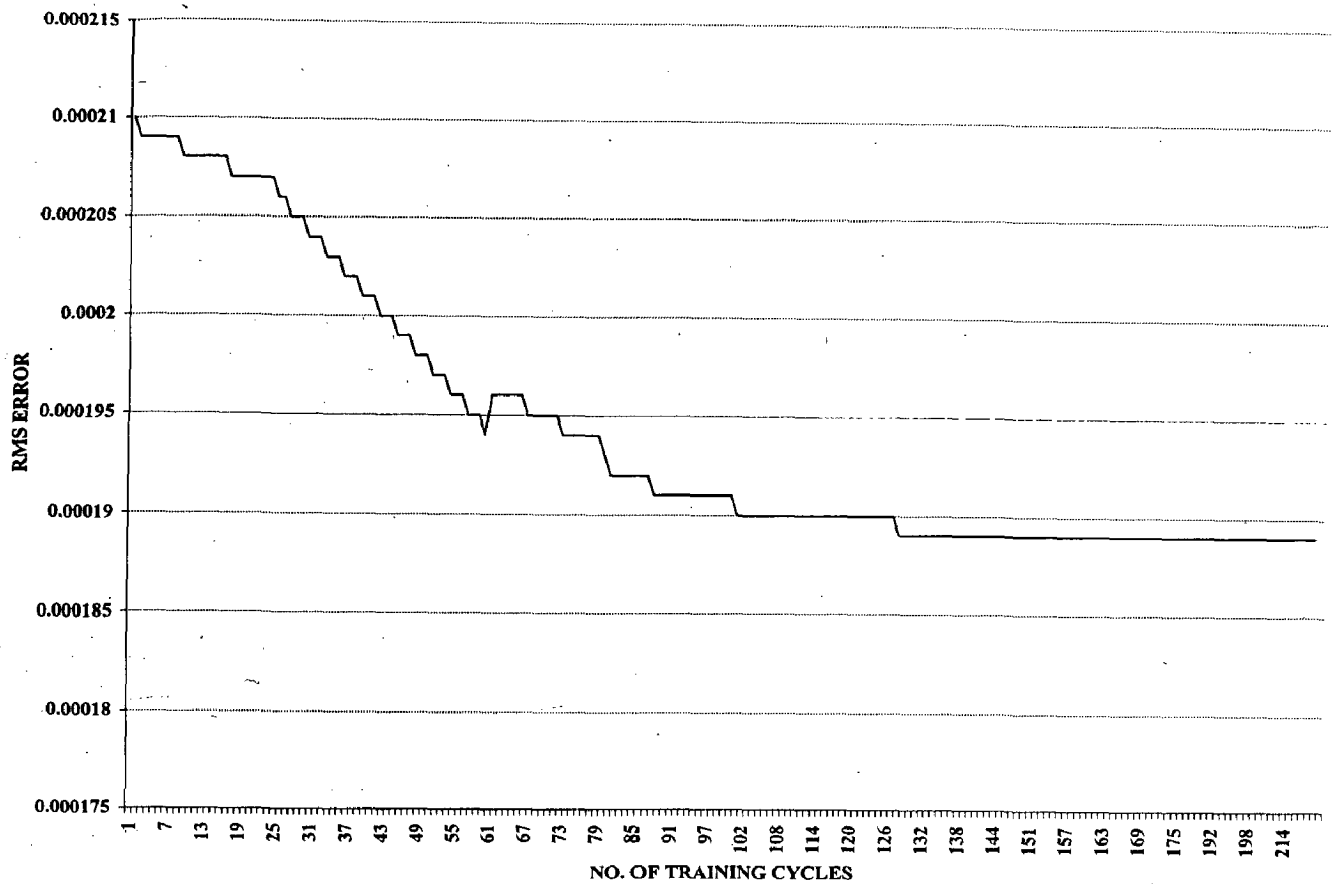


Figure 4.3: Operating range 80% - 120%

Table 4.3

I = 45, J = 15, K = 17, $\lambda = 1.0$, Number of training patterns = 20000		Range of loading = 70% - 130% Number of test patterns = 1000		
Value of learning constant (η)	Value of momentum factor (mf)	No of cycles taken at convergence	rms error at convergence	Test error (percentage mismatch)
0.01	0.6	200	0.000176	3.8
0.08	0.6	90	0.000198	4.94
0.2	0.9	Error was randomly increasing and decreasing		
0.001	0.8	40	0.00041	Convergence rate was very poor

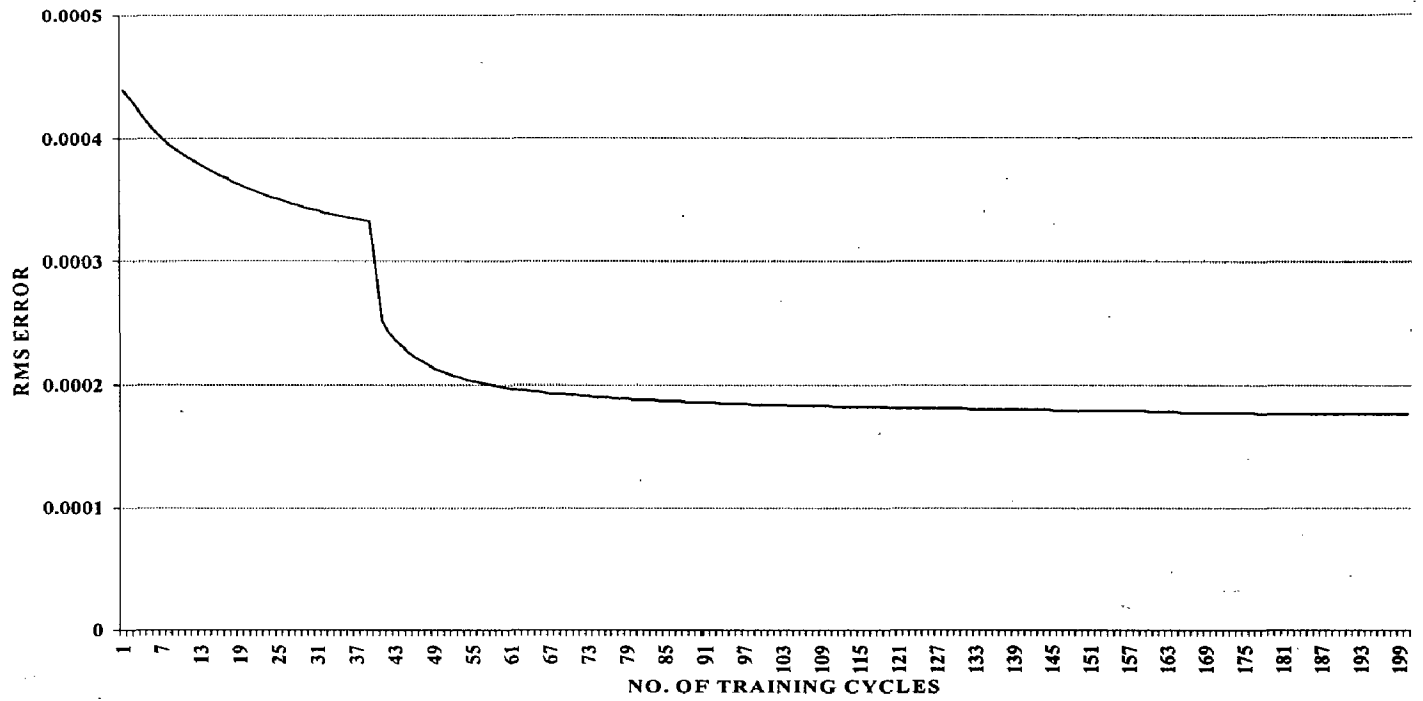
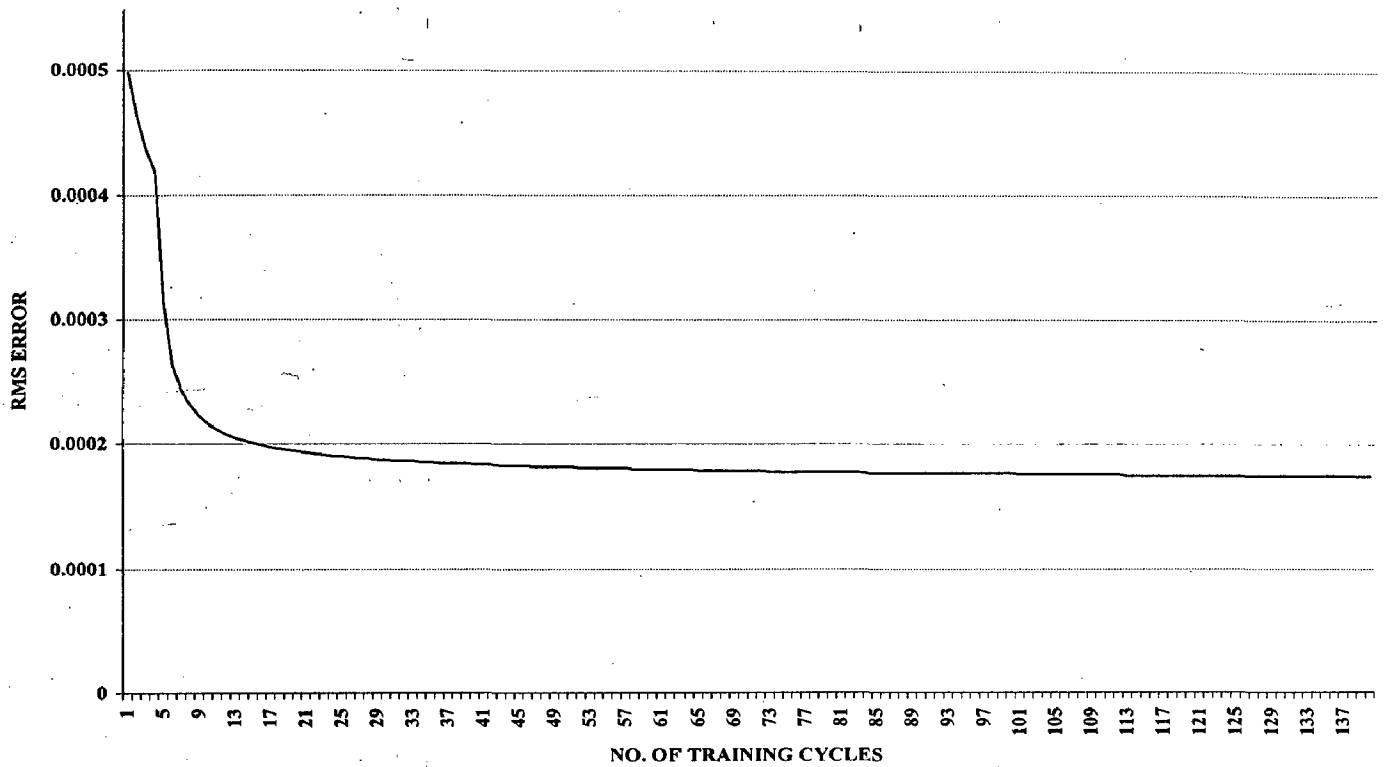


Figure 4.4: Operating range 70% - 130%

Table 4.4

I = 45, J = 15, K = 17, $\lambda = 1.0$, Number of training patterns = 20000		Range of loading = 60% 140% Number of test patterns = 1000		
Value of learning constant (η)	Value of momentum factor (mf)	No of cycles taken at convergence	rms error at convergence	Test error (percentage mismatch)
0.01	0.8	140	0.000174	3.58
0.9	0.9	130	0.000182	5.26
0.007	0.6	100	0.000203	12.62



NO. OF TRAINING CYCLES
 Figure 4.5: Operating range 60% - 140%



CONCLUSION

In this thesis, a neural network based optimal capacitor switching algorithm has been developed. The main conclusions of this work are :

- The ANN based method is much faster than the traditional combinatorial methods for solving the optimal capacitor switching problem.
- As the range of loading pattern increases, number of training patterns required to train the network adequately also increases.
- The performance of the ANN reaches a plateau when EBPTA is used, after which the performance does not improve or the improvement can be achieved at a cost of unacceptably large training time.
- There is no proven method to choose the various parameters of the ANN such as learning constant and momentum factor. Their values for best performance of the ANN can only be decided by trial and error method.

FUTURE SCOPE OF WORK

As already discussed in the main conclusion, the performance of the ANN reaches a plateau when it is trained by EBPTA. Hence, it is necessary to investigate the performance of the multi-layer feedforward ANN by training it with different other training algorithms such as, over-relaxation error back propagation training algorithm. Moreover, different other type of ANNs such as, functional link network, counter-propagation network, Hopfield network etc. can also be investigated for this application.

REFERENCES

1. Kazuaki Kato, Hiromi Nagasaka, Akimichi-Okimato, Toshihito Kunieda and Toshihiko Nakamura, "*Distribution Automation Systems for High Quality Power Supply*", IEEE Transaction on Power Delivery, Vol. 6, No. 3, July 1991, pp : 1196 - 1203.
2. Hsiao-Dong Chiang, Jin-Cheng Wang, Orville Cockings and Hyoun-Duckshin, "*Optimal Capacitor Placement in Distribution Systems: Part 1: A New Formulation and the Overall Problem*", IEEE Transaction on Power Delivery, Vol. 5, No. 2, April 1990, pp : 634 - 642.
3. Hsiao-Dong Chiang, Jin-Cheng Wang, Orville Cockings and Hyoun-Duckshin, "*Optimal Capacitor Placement in Distribution Systems: Part 2: Solution Algorithms and Numerical Results*", IEEE Transaction on Power Delivery, Vol. 5, No. 2, April 1990, pp : 643 - 649.
4. Y. Baghzoug, "*Effects of Nonlinear Loads on Optimal Capacitor Placement in Radial Feeders*", IEEE Transaction on Power Delivery, Vol. 6, No. 1, January 1991, pp : 245 - 251.
5. I. Roytelman, B. K. Wee and R. L. Lugtu, "*Volt/Var Control Algorithm for Modern Distribution Management System*", IEEE Transactions on Power Systems, Vol.-10, No.-3, August 1995, pp : 1454-1460.
6. Jack M. Jurada, "*Introduction to Artificial Neural Systems*", Jaico Publishing House, 1997.
7. R. Baldic and F. F. Wu, "*Efficient Integer Optimization Algorithm for Optimal Coordination of Capacitor and Regulators*", IEEE Transaction on Power Delivery, Vol. 5, No. 3, 1990, pp : 805 - 812.
8. N. Iwan Santoso and Owen T. Tan, "*Neural-Net Based Real-Time Control of Capacitors Installed on Distribution System*", IEEE Transaction on Power Delivery, Vol. 5, No. 1, January 1991, pp : 262-272.

APPENDIX A

Table A.1

[System Network and Load Data]

Bus i	Bus j	Branch Impedance		Max. Load at Bus j	
		$r_{ij}(\Omega)$	$x_{ij}(\Omega)$	P(kW)	Q(kVar)
0	1	0.5096	1.7030	-	-
1	2	0.2191	0.0118	522	174
2	3	0.3485	0.3446	-	-
3	4	1.1750	1.0214	936	312
4	5	0.5530	0.4806	-	-
5	6	1.6625	0.9365	-	-
6	7	1.3506	0.7608	-	-
7	8	1.3506	0.7608	-	-
8	9	1.3259	0.7469	189	63
9	10	1.3259	0.7469	-	-
10	11	3.9709	2.2369	336	112
11	12	1.8549	1.0449	657	219
12	13	0.7557	0.4257	783	261
13	14	1.5389	0.8669	729	243
8	15	0.4752	0.4131	477	159
15	16	0.7282	0.4102	549	183
16	17	1.3053	0.7353	477	159
6	18	0.4838	0.4206	432	144
18	19	1.5898	1.3818	672	224
19	20	1.5389	0.8669	495	165
6	21	0.6048	0.5257	207	69
3	22	0.5639	0.5575	522	174
22	23	0.3432	0.3393	1917	63

Contd...

23	24	0.5728	0.4979	-	-
24	25	1.4602	1.2692	1116	372
25	26	1.0627	0.9237	549	183
26	27	1.5114	0.8514	792	264
1	28	0.4659	0.0251	882	294
28	29	1.6351	0.9211	882	294
29	30	1.1143	0.6277	882	294
$V_{\text{rated}} = 23 \text{ kV}$					

Table A.2

[Capacitor KVARs at Different Tap Positions]

Tap position	Capacitor kVar				
	Cap. #1 At bus 13	Cap. #2 At bus 15	Cap. #3 At bus 19	Cap. #4 At bus 23	Cap. #5 At bus 25
1	875	875	500	Fixed at 750	600
2	700	700	425		525
3	525	525	350		450
4	350	350	275		375

APPENDIX B

SOFTWARE FOR TRAINING OF ANN

THIS PROGRAM TRAINS A FEED-FORWARD NEURAL NETWORK WITH ERROR-BACK PROPAGATION TRAINING WITH ONE HIDDEN LAYER. THE TRAINING INPUT DATA

IS TAKEN FROM THE FILE 'DATA.DAT' AND THE TRAINING OUTPUT DATA IS TAKEN FROM THE 'OUTPT_VO.DAT' FILE. THE PARAMETERS OF THE ANN IS DEFINED IN THE FILE 'INP_VOLT.DAT' FILE

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#define a 50

void netf(int J,int I,float U[a][a],float x[a],float
net3[a]) ;
void sigm(int J,float net2[a],float lam,float
fnet2[a],float y[a],float f_net2[a]);
void adj(int K,int J,float eta,float mf,float
delo[a],float cw[a][a],float W[a][a],float y[a]);

void main()
{
    FILE *fp, *fq, *fr, *fs, *fw, *fpt_s ;
    float net1[a], net2[a], net3[a] ;
    float U[a][a], V[a][a], W[a][a], x[a], y[a], z[a], o[a]
;
    float fnet1[a],fnet2[a],fnet3[a],d[a],f_net3[a] ;
    float
cu[a][a],cv[a][a],cw[a][a],sum,temp1[a],t[a][a],b[a],f_net1[
a],f_net2[a] ;
    float delo[a],dely[a],delz[a] ;
    int i,j,k,l,I,J,K,n,p,pt,m,q ;
    float
lam,eta,emax,e,erms,mf,r[a],inc,e_1,time1,time2,time ;
    char ans ;

    time1 = clock()/CLK_TCK ;

    /* opening the 'INP_VOLT.DAT' file, which stores the initial
data for the training */
    if ((fr = fopen("inp_volt.dat","r")) == NULL)
    {
```

```

    printf("\nERROR !! the file `INP_VOLT.DAT' does not
exist !\n") ;
    exit(1) ;
}
/* opening the `DATA.DAT' file, which stores the training
input patterns */
if ((fq = fopen("data.dat","r")) == NULL)

    {
    printf("\nERROR !! the file `DATA.DAT' does not exist
!\n") ;
    exit(1) ;
    }

/* opening the `OUTPT_VO.DAT' file, which contains the
training output patterns */
if ((fs = fopen("outpt_vo.dat","r")) == NULL)
    {
    printf("\nERROR !! the file `OUTPT_VO.DAT' does not
exist !\n") ;
    exit(1) ;
    }

/* opening the `SUMMARY.DAT' file, which contains the
summary of training */
if ((fpt_s = fopen("summary.dat","a")) == NULL)
    {
printf("\nERROR !! the file `SUMMARY.DAT' can not open !\n")
;
    exit(1) ;
    }

/* reading from the `INP_VOLT.DAT' file */

fscanf(fr, "%d\t%d\t%d\t%f\t%f\t%f\t%f\t%d\n%d", &I, &J, &K, &lam
, &eta, &mf, &emax, &n, &pt);
fclose(fr) ;
I = I + 1 ;

printf("\nNO. OF INPUTS                :   I = %d", I) ;
printf("\nNO. OF NEURONS IN HIDDEN LAYER:   J = %d", J) ;
printf("\nNO. OF OUTPUTS                 :   K = %d", K) ;
printf("\nNON LINEARITY FACTOR             :   lam =%f", lam) ;
printf("\nLEARNING CONSTANT                  :   eta =%f", eta) ;

```

```

printf("\nMOMENTUM FACTOR           :   mf = %f",mf) ;
printf("\nMIN. ERROR                 :   emax=%f",emax) ;
printf("\nNO. OF CYCLE YOU WANT TO TRAIN ANN :n=%d",n) ;
printf("\nNO. OF PATTERN                 :   pt = %d",pt) ;

printf("\ndo YOU WANT TO START WITH PREVIOUSWEIGHTS(Y/N=) " ;

ans = getch();

/* set the initial weights of the ANN randomly */
if(ans == 'N' || ans == 'n')
{
printf("NO") ;

/* ----- WEIGHTS OF FIRST STAGE ----- */

for(j = 0; j < J; j++)
{
for(i = 0; i < I; i++)
{
inc = rand() / 900000.0 ;
v[j][i] = inc ;
cv[j][i] = 0.0 ;
}
}

/* ----- WEIGHTS OF SECOND STAGE ----- */

for(k = 0; k < K; k++)
{
for(j = 0; j < J; j++)
{
inc = rand() / 900000.0 ;
W[k][j] = inc ;
cw[k][j] = 0.0 ;
}
}

/* get the initial weights from 'W1_VOLT.DAT' file */

else
{

```

```

printf("YES") ;

/* opening the `W1_VOLT.DAT' file, which stores the weight
vectors from previous training */

    if ((fw = fopen("w1_volt.dat","r")) == NULL)
    {
        printf("\nERROR !! the file `W1_VOLT.DAT' does not
exist !\n") ;
        exit(1) ;
    }

/* ----- WEIGHTS OF FIRST STAGE ----- */

    for(j = 0; j < J; j++)
    {
        for(i = 0; i < I; i++)
        {
            fscanf(fw,"%f ",&V[j][i]) ;
            cv[j][i] = 0.0 ;
        }
    }

/* ----- WEIGHTS OF SECOND STAGE ----- */

    for(k = 0; k < K; k++)
    {
        for(j = 0; j < J; j++)
        {
            fscanf(fw,"%f ",&W[k][j]) ;
            cw[k][j] = 0.0 ;
        }
    }

/* ----- CHANGE OF WEIGHT OF THE FIRST STAGE -----
--- */

    for(j = 0; j < J; j++)
    {
        for(i = 0; i < I; i++)
        {
            fscanf(fw,"%f ",&cv[j][i]) ;
        }
    }

/* ----- CHANGE OF WEIGHTS OF SECOND STAGE -----
- */

```

```

    for(k = 0; k < K; k++)
    {
        for(j = 0; j < J; j++)
        {
            fscanf(fw,"%f ",&cw[k][j]) ;
        }
    }

/* close the 'W1_VOLT.DAT' file */
fclose(fw) ;

p = 0 ;

do
{
    /* start of 'do' loop */
    p = p + 1 ;
    e = 0.0 ;
    e_1 = 0.0 ;

/* ----- GET THE INPUT PATTERNS ----- */

    for (m = 0; m < pt; m++)
    {
        for (l = 0; l < (I-1); l++)
        {
            fscanf(fq,"%f ",&z[l]) ;
            z[I-1] = -1.0 ;
        }
    }

/* ----- GET THE DESIRED OUTPUT PATTERNS ----- */

    for (k = 0; k < K; k++)
    {
        fscanf(fs,"%f",&d[k]) ;
    }

/* ----- CALCULATE INPUT TO THE NODES OF THE HIDDEN LAYER -----
--- */
    netf(J, I, V, z, net2);

/* ----- CALCULATE OUTPUT OF THE NODES OF THE HIDDEN LAYER -----
--- */

    sigm(J, net2, lam, fnet2, y, f_net2) ;

```

```

/* ----- CALCULATE INPUT TO THE NODES OF THE OUTPUT LAYER --
---*/

    netf(K, J, W, y, net3) ;

/* ----- CALCULATE OUTPUT AT THE OUTPUT NODES -----*/

    sigm(K, net3, lam, fnet3, o, f_net3) ;

/* calculate error and the derivatives of the error */

    for (k = 0; k < K; k++)
    {
        r[k] = d[k] - o[k] ;
        e_1 = e_1 + r[k] * r[k] ;
        r[k] = r[k] * r[k] * 0.5 ;
        e += r[k] ;
        delo[k] = (d[k] - o[k]) * f_net3[k] ;
    }

    for (j = 0; j < J; j++)
    {
        sum = 0.0 ;
        for (k = 0; k < K; k++)
        {
            sum += delo[k] * W[k][j] ;
        }
        temp1[j] = sum ;
        dely[j] = temp1[j] * f_net2[j] ;
    }

/* ----- MODIFY THE WEIGHTS OF THE SECOND STAGE -----
- */

    adj(K, J, eta, mf, delo, cw, W, y) ;

/* ----- MODIFY THE WEIGHTS OF THE FIRST STAGE -----
- */

    adj(J, I, eta, mf, dely, cv, V, z) ;

}

erms = sqrt(e) ;
erms = erms/pt ;

```

```

    erms = erms/K ;
    printf("\n\nCOUNT NO.=%d  ERMS=%f\nPLEASE WAIT FOR NEXT
CYCLE\n",p,erms);
    fprintf(fpt_s,"%d  %f\n",p,erms) ; /* writing in
summary.dat*/

    if (erms > emax)
    {
        rewind(fq) ;
        rewind(fs) ;
    }

/* OPEN THE `WAT_VOLT.DAT' FILE, WHICH CONTAINS THE WEIGHTS
AFTER TRAINING */

    if ((fp = fopen("wat_volt.dat","w")) == NULL)
    {
        printf("\nERROR !! the file `WAT_VOLT.DAT' can not
open !\n") ;
        exit(1) ;
    }

/* WRITE THE WEIGHTS INTO THE `WAT_VOLT.DAT' FILE */

    for (j = 0; j < J; j++)
    {
        for(i = 0; i < I; i++)
        {
            fprintf(fp,"%f ",V[j][i]) ;
        }
        fprintf(fp,"\n") ;
    }
    for (k = 0; k < K; k++)
    {
        for (j = 0; j < J; j++)
        {
            fprintf(fp,"%f ",W[k][j]) ;
        }
        fprintf(fp,"\n") ;
    }
    for (j = 0; j < J; j++)
    {
        for(i = 0; i < I; i++)
        {
            fprintf(fp,"%f ",cv[j][i]) ;

```



```

        for (j = 0; j < J; j++)
        {
            c_w[k][j] = cw[k][j] ;
            cw[k][j] = 0.0;
            cw[k][j] = eta * delo[k] * y[j] + mf *
c_w[k][j] ;
            W[k][j] = W[k][j] + cw[k][j] ;
        }
    }

} /* end of `adj' function */

/* -- FUNCTION TO CALCULATE OUTPUT OF A NODE THROUGH SIGMOID
FUNCTION -- */

void sigm(int J, float net2[a], float lam, float
fnet2[a], float y[a], float f_net2[a])
{
    int j ;
    float b[a] ;

    for (j = 0; j < J; j++)
    {
        b[j] = - (net2[j] * lam) ;
        b[j] = exp(b[j]) ;
        fnet2[j] = 1.0 / (1.0 + b[j]) ;
        y[j] = fnet2[j] ;
        f_net2[j] = y[j] * (1 - y[j]) ;
    }
    return ;

} /* END OF `sigm' FUNCTION */

/* ----- FUNCTION TO COMPUTE OUTPUT OF EACH STAGE -----
*/

void netf(int J, int I, float U[a][a], float x[a], float
net3[a])
{
    int i, j ;
    float sum ;

    for (j = 0; j < J; j++)
    {
        sum = 0.0 ;

```

```
    for (i = 0; i < I; i++)
        {
            sum = sum + U[j][i] * x[i] ;
        }
    net3[j] = sum ;
}
return ;

} /* end of `netf' function */
```

SOFTWARE FOR TESTING OF ANN

```
#include<stdio.h>
#include<math.h>
#include<time.h>
#define a 50

void netf(int J,int I,float U[a][a],float x[a],float
net3[a]) ;
void sigm(int J,float net2[a],float lam,float
fnet2[a],float y[a],float f_net2[a]);
void adj(int K,int J,float eta,float mf,float
delo[a],float cw[a][a],float W[a][a],float y[a]);

void main()
{
FILE *fp,*fq,*fr,*fs,*fw ;
float net1[a], net2[a], net3[a] ;
float U[a][a], V[a][a], W[a][a], x[a], y[a], z[a], o[a]
;
float fnet1[a],fnet2[a],fnet3[a],d[a],f_net3[a];
float
cu[a][a],cv[a][a],cw[a][a],sum,temp1[a],t[a][a],b[a],f_net1[
a],f_net2[a];
float delo[a],dely[a],delz[a];
int i,j,k,l,I,J,K,n,p,pt,m,q,diff,t_diff;
float lam,e,erms,r[a],inc,time,time1,time2;
char ans;

/* opening the `TIP_CAP.DAT' file, which stores the initial
data for the testing */
if ((fr = fopen("tip_cap.dat","r")) == NULL)
{
printf("\nERROR !! the file `TIP_CAP.DAT' does not
exist !\n") ;
exit(1) ;
}

/* opening the `PAT_CAP.DAT' file, which stores the testing
input patterns */
if ((fq = fopen("pat_cap.dat","r")) == NULL)
```

```

    {
        printf("\nERROR !! the file `PAT_CAP.DAT' does not
exist !\n") ;
        exit(1) ;
    }

/* opening the `TOUT_CAP.DAT' file, which contains the
training output patterns */
    if ((fs = fopen("tout_cap.dat","r")) == NULL)
    {
        printf("\nERROR !! the file `TOUT_CAP.DAT' does not
exist !\n") ;
        exit(1) ;
    }

/* reading from the `TIP_CAP.DAT' file */
    fscanf(fr,"%d\t%d\t%d\t%f\n%d",&I,&J,&K,&lam,&pt) ;
    fclose(fr) ;
    I = I + 1 ;

    printf("\nNO. OF INPUTS                :   I = %d",I) ;
    printf("\nNO. OF NEURONS IN HIDDEN LAYER:   J = %d",J) ;
    printf("\nNO. OF OUTPUTS                :   K = %d",K) ;
    printf("\nNON LINEARITY FACTOR                :   lam= %f",lam) ;
    printf("\nNO. OF PATTERN                :   pt = %d",pt) ;

/* get the initial weights from `W1_CAP.DAT' file */
    if ((fw = fopen("w1_cap.dat","r")) == NULL)
    {
        printf("\nERROR !! the file `W1_CAP.DAT' does not
exist !\n") ;
        exit(1) ;
    }

/* ----- WEIGHTS OF FIRST STAGE ----- */
    for(j = 0; j < J; j++)
    {
        for(i = 0; i < I; i++)
        {
            fscanf(fw,"%f",&V[j][i]) ;
        }
    }

/* ----- WEIGHTS OF SECOND STAGE ----- */
    for(k = 0; k < K; k++)
    {
        for(j = 0; j < J; j++)

```

```

        {
            fscanf(fw,"%f ",&W[k][j]) ;
        }
    }

/* close the `W1_CAP.DAT' file */
    fclose(fw) ;
    fp=fopen("relt_cap.dat","w");

    time1=clock()/CLK_TCK;

/* ----- GET THE INPUT PATTERNS ----- */
    t_diff = 0;
    for (m = 0; m < pt; m++)
    {
        for (l = 0; l < (I-1); l++)
        {
            fscanf(fq,"%f ",&z[l]) ;
            z[I-1] = -1.0 ;
        }
    }
/*printf("\ncheck_1\n") ;*/

/* ----- GET THE DESIRED OUTPUT PATTERNS ----- */
    for (k = 0; k < K; k++)
    {
        fscanf(fs,"%f",&d[k]) ;
    }
/*printf("check_2\n") ;    */

/* ----- CALCULATE OUTPUT OF FIRST STAGE ----- */
    netf(J, I, V, z, net2);
/*printf("check_3\n") ;*/

/* ----- INCLUDE SIGMOIDE FUNCTION IN FIRST STAGE ----- */
    sigm(J, net2, lam, fnet2, y, f_net2) ;
/*printf("check_4\n") ;*/

/* ----- CALCULATE OUTPUT OF SECOND STAGE ----- */
    netf(K, J, W, y, net3) ;
/*printf("check_5\n");*/

/* -----INCLUDE SIGMOIDE FUNCTION IN SECOND STAGE -----
*/
    sigm(K, net3, lam, fnet3, o, f_net3) ;
/*printf("check_6\n");*/

```



```

    b[j] = - (net2[j] * lam) ;
    b[j] = exp(b[j]) ;
    fnet2[j] = 1.0 / (1.0 + b[j]) ;
    y[j] = fnet2[j] ;
    f_net2[j] = y[j] * (1 - y[j]);
}
return ;
}

/* ----- FUNCTION TO COMPUTE OUTPUT OF EACH STAGE -----
*/
void netf(int J,int I,float U[a][a],float x[a],float
net3[a])
{
    int i, j ;
    float sum ;

    for (j = 0; j < J; j++)
    {
        sum = 0.0 ;
        for (i = 0; i < I; i++)
        {
            sum = sum + U[j][i] * x[i] ;
        }
        net3[j] = sum ;
    }
    return;

} /* end of `netf' function */

```

Dec. 10.
5 10,069