# Congestion Arbitration And Source Problem Prediction Using ANN In Wireless Networks


*Developed By*
**Saadia Arshad**
**Ambreen Siddiqui**

*Supervised By*
**Dr. Sikander Hayat Khiyal**
**Dr. Muhammad Sher**

**Department of Computer Science**
**Faculty of Applied Science**
**International Islamic University, Islamabad.**
**(2004)**

**In the name of Almighty Almighty Allah,**

**The most gracious,**

**The most Merciful.**

**Department of Computer Science**

**International Islamic University, Islamabad.**

**Date: 14th December 2004**

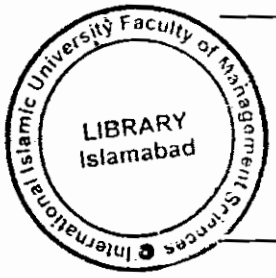
## Final Approval

It is certified that we have completely read the thesis, titled "**Congestion arbitration and source problem in ANN using neural networks**" submitted by Saadia Arshad and Ambreen Siddiqui under university registration number 104-CS/MS/02 and 94-CS/MS/02 respectively. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad, for the degree of **Master of sciences.**

**Committee**

**External Examiner**

Dr. Abdul Sattar
Consultant,
AIOU, Islamabad.




**Internal Examiner**

Mr. Asim Munir,
Assistant Professor,
Department of Computer Science
IIUI

**Supervisors**

Dr. Sikander Hayat Khayal
Head,
Department of Computer Science,
International Islamic University,
Islamabad.

&

Dr . M . Sher
Assistant Professor,
Department of Computer Science,
International Islamic University,
Islamabad.

# Dedication

Dedicated to our parents

A dissertation submitted to the

**Department of computer science,**

International Islamic University, Islamabad

As a partial fulfillment of the requirements

for the award of the degree of

**Master of Science**

# Declaration

We hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software entirely on the basis of our personal efforts made under the sincere guidance of my teachers. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

**Saadia Arshad**
**104-CS/MS/02**

**Ambreen Siddiui**
**94-CS/MS/02**

# Acknowledgements

We would like to express our sincere gratitude and deep appreciation to my advisor, Dr Sikander Hayat Khiyal, and Dr. Muhammad Sher for his guidance, encouragement, and support throughout the project. I also want to thank the other committee members.

Our gratitude also goes to our parents and our brothers for their devotion and encouragement.

Saadia Arshad

Ambreen Siddiqui

# Project In Brief

| | |
|---|---|
| Project Title: | Congestion arbitration and source problem in ANN using neural networks |
| Objective: | To gain knowledge. |
| Undertaken By: | Saadia Arshad<br>104-CS/MS/02<br><br>Ambreen Siddiqui<br>94-CS/MS/02 |
| Supervised By: | Dr. Sikander Hayat Khayal<br>Head,<br>Department of Computer Sciences,<br>International Islamic University,<br>Islamabad.<br><br>Dr. Muhammad Sher<br>Assistant Professor,<br>Department of Computer Sciences,<br>International Islamic University,<br>Islamabad. |
| Technologies Used: | Microsoft®, NS-2, JDK 1.3 |
| System Used: | Pentium® IV |
| Operating System Used: | Microsoft® windows XP |
| Date Started: | 15th September, 2003 |
| Date Completed: | 20th July, 2004 |

# Abstract

Congestion is the problem which occurs when demand for a resource outstrips the capacity. In wireless networks, congestion may occur through antenna, satellite link, routers and switches which are shared by several sources. The congestion control scheme described here employs a neural network to predict the state of congestion in a wireless network over a prediction horizon. We propose using learning techniques to predict the problems before they start impacting the performance of services especially in wireless communication. In this thesis we focus on using a feed forward neural network to predict severe congestion in a wireless network. We also use neural networks to predict the source or sources responsible for the congestion, and we design and apply a control method for limiting the rate of the offending sources so that congestion can be avoided. This thesis introduces an adaptive neuro-control strategy, adaptive neural swarming (ANS). A highly non-linear bioreactor benchmark is used in the control simulation. Based on the neural predictor output, source rate control signals are obtained by minimizing a cost function which represents the cumulative differences between a set-point and the predicted output. An analytical procedure for the source rate control signal computations is given using gradient functions of the neural network predictor by the use of wireless session protocol (WSP), wireless transaction protocol(WTP). Unlike the RED and usual TCP/IP flow control, the proposed method is applied only to selected nodes and converges to the

final rate faster. The described techniques set the stage for a new wave of wireless network managers that are capable of preventing wireless networking problems instead of repairing them.

## List Of Figures:



## List of Tables:

# INTRODUCTION

# 1. Introduction

Neural Networks are originally developed from the inspiration of human brains. In this chapter we will discuss the basics of neural network, its approaches, application in the field and the overwork.

## *1.1. Neural Networks*

Neural networks, more accurately called Artificial Neural Networks (ANN), are computational models that consist of a number of simple processing units that communicate by sending signals to each other over a large number of weighted connections. They were originally developed from the inspiration of human brains. In human brains, a biological neuron collects signals from other neurons through a host of fine structures called *dendrites*. The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches. At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

Like human brains, neural networks also consist of processing units (artificial neurons) and connections (weights) between them. The processing units transport incoming information on their outgoing connections to other units. The "electrical"

information is simulated with specific values stored in those weights that make these networks have the capacity to learn, memorize, and create relationships amongst data.

A very important feature of these networks is their adaptive nature where "learning by example" replaces "programming" in solving problems. This feature renders these computational models very appealing in application domains where one has little or incomplete understanding of the problems to be solved, but where training data are available

## 1.2. *Approaches in Neural Network*

There are many different types of neural networks, and they are being used in many fields. And new uses for neural networks are devised daily by researchers. Some of the most traditional applications include [1][2]:

- Classification – To determine military operations from satellite photographs; to distinguish among different types of radar returns (weather, birds, or aircraft); to identify diseases of the heart from electrocardiograms.
- Noise reduction – To recognize a number of patterns (voice, images, etc.) corrupted by noise.
- Prediction – To predict the value of a variable given historic values. Examples include forecasting of various types of loads, market and stock forecasting, and weather forecasting. The model built in this thesis falls into this category.

## 1.3. Applications

In the world of networking, more emphasis is being placed on speed, connectivity, and reliability. When network problems occur, they often catastrophically break the service for those enterprises or individuals that depend on the network connection. Sometimes, such breaks of service are just annoying, but for companies and commercial users they often mean lost revenues on the order of thousands, or even millions, of dollars. Such breaks have become a significant problem in all forms of electronic commerce.

A network congestion collapse occurs when the network is increasingly busy but little useful work is getting done. Congestion management features allow you to control congestion by determining the order in which packets are transmitted out an interface based on priorities assigned to those packets. Congestion management entails the creation of queues, assignment of packets to those queues based on the packet's classification, and scheduling of the packets in a queue for transmission. The congestion management QOS feature offers four types of queuing protocols, each of which allows you to specify creation of a different number of queues, affording greater or lesser degrees of differentiation of traffic and the order in which that traffic is transmitted. During periods with light traffic, that is, when no congestion exists, packets are transmitted out the interface as soon as they arrive. During periods of transmit congestion at the outgoing interface, packets arrive faster than the interface can transmit them. If you use congestion management features, packets accumulating at an interface are queued until the interface is free to transmit them; they are then

scheduled for transmission according to their assigned priority and the queuing mechanism configured for the interface. The router determines the order of packet transmission by controlling which packets are placed in which queue and how queues are serviced with respect to each other.

## *1.4. Ourwork*

It is important to avoid high packet loss rates in the internet. This problem is further critical in the wireless communication because of the shared transmission medium, dynamic topologies, different protocols (like WSP and WTP) and costly medium. The bandwidth utilization in the wireless medium requires a good congestion control mechanism. To address this difficulty, a system is needed to insure network availability and efficiency by preventing such costly wireless network breakdowns. The first step towards this end is to create a system with the intelligence to recognize, as early as possible, early signs of incoming network service difficulties. If the problem can be recognized in advance, changing network parameters can possibly circumvent the problem.

There are techniques of fancy queuing to avoid congestion but the major problem of knowing about congestion in advance still remains unsolved. In this work we propose a neural network based forecasting technique to estimate and forecast congestion state in advance on the basis of input traffic from various sources arriving at the router [5]. We have used a feed forward neural network to forecast and then apply the results with the simulated traffic generated through network simulator (NS-2). The section-2

describes about some background information on existing congestion management techniques. The next section will describe design of an artificial neural network (ANN) for congestion forecasting. Next in section-4 we will discuss the implementation detail of our experimental setup. Then in section -5 we present the results of our work, the conclusion and the references.

# ANALYSIS

# 2. Analysis

This chapter focuses on the fundamentals of Neural Networks, the processing unit and different functions like combinational, activation and objective function. Network topologies and network learning in neural networks is of great importance, hence dicussed later in the chapter.

## *2.1. Fundamentals of Neural Networks*

Neural networks, sometimes referred to as connectionist models, are parallel-distributed models that have several distinguishing features [3]:

- A set of processing units;
- An activation state for each unit, which is equivalent to the output of the unit;
- Connections between the units. Generally each connection is defined by a weight $w_{jk}$ that determines the effect that the signal of unit $j$ has on unit $k$;
- A propagation rule, which determines the effective input of the unit from its external inputs;
- An activation function, which determines the new level of activation based on the effective input and the current activation;
- An external input (bias, offset) for each unit;
- A method for information gathering (learning rule);
- An environment within which the system can operate, provide input signals and, if necessary, error signals.

## 2.2. Processing Unit

A processing unit (Figure 2-1), also called a neuron or node, performs a relatively simple job; it receives inputs from neighbors or external sources and uses them to compute an output signal that is propagated to other units.

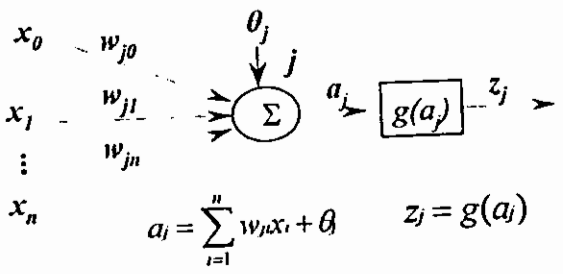



Figure 2-1 Processing unit

Within the neural systems there are three types of units:

1) Input units, which receive data from outside of the network;
2) Output units, which send data out of the network;
3) Hidden units, whose input and output signals remain within the network.

Each unit $j$ can have one or more inputs $x_0, x_1, x_2, \ldots x_n$, but only one output $z_j$. An input to a unit is either the data from outside of the network, or the output of another unit, or its own output.

## 2.3. Combination Function

Each non-input unit in a neural network combines values that are fed into it via synaptic connections from other units, producing a single value called net input. The function that combines values is called the *combination function*, which is defined by

a certain propagation rule. In most neural networks we assume that each unit provides an additive contribution to the input of the unit with which it is connected. The total input to unit $j$ is simply the weighted sum of the separate outputs from the connected units plus a threshold or bias term $\theta_j$ :

$$a_j = \sum_{i=1}^{n} w_{ji} x_i + \theta_j \tag{2.1}$$

The contribution for positive $w_{ji}$ is considered as an excitation and an inhibition for negative $w_{ji}$. We call units with the above propagation rule *sigma units*.

In some cases more complex rules for combining inputs are used. One of the propagation rules known as *sigma-pi* has the following format [3]:

$$a_j = \sum_{i=1}^{n} w_{ji} \prod_{k=1}^{m} x_{ik} + \theta_j \tag{2.2}$$

Lots of combination functions 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. It is needed in much the same way as the constant polynomial '1' is required for approximation by polynomials.

## 2.4. *Activation Function*

Most units in neural network transform their net inputs by using a scalar-to-scalar function called an *activation function*, yielding a value called the unit's activation. Except possibly for output units, the activation value is fed to one or more other units. Activation functions with a bounded range are often called squashing functions. Some of the most commonly used activation functions are [4]:

1) Identity function (Figure 2-1)

$$g(x) = x \tag{2.3}$$

It is obvious that the input units use the identity function. Sometimes a constant is multiplied by the net input to form a linear function.

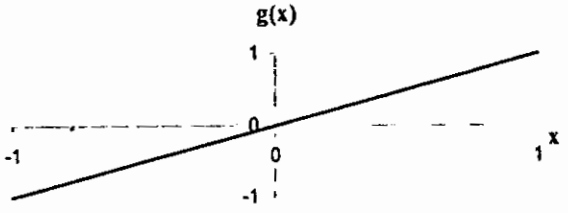



Figure 2-1 Identity function

2) Binary step function (Figure 2-2)

Also known as *threshold function* or *Heaviside function*. The output of this function is limited to one of the two values:

$$g(x) = \begin{cases} 1 & \text{if}(x \geq \theta\ ) \\ 0 & \text{if}(x < \theta) \end{cases} \tag{2.4}$$

This kind of function is often used in single layer networks.


Figure 2-2 Binary step function

3) Sigmoid function (Figure 2-3)

$$g(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

This 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. It applies to applications whose desired output values are between 0 and 1.

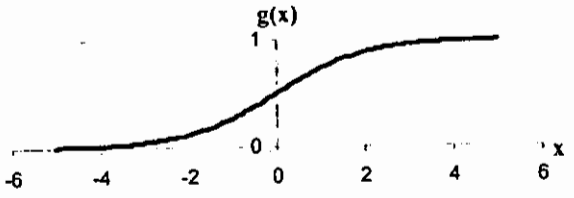



Figure 2-3 Sigmoid function

4) Bipolar sigmoid function (Figure 2-4)

$$g(x) = \frac{1-e^{-x}}{1+e^{-x}} \tag{2.6}$$

This function has similar properties with the *sigmoid function.* It works well for applications that yield output values in the range of [-1,1].

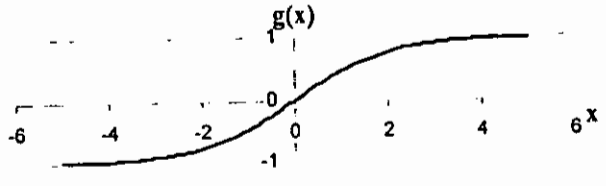



Figure 2-4 Bipolar sigmoid function

Activation functions for the hidden units are needed to introduce non-linearity into the networks. The reason is that a composition of linear functions is again a linear function. However, it is the non-linearity (i.e., the capability to represent nonlinear

functions) that makes multi-layer networks so powerful. Almost any nonlinear function does the job, although for back-propagation learning it must be differentiable and it helps if the function is bounded (see Section 3.6). The sigmoid functions are the most common choices [5].

For the output units, activation functions should be chosen to be suited to the distribution of the target values. We have already seen that for binary [0,1] outputs, the sigmoid function is an excellent choice. For continuous-valued targets with a bounded range, the sigmoid functions are again useful, provided that either the outputs or the targets to be scaled to the range of the output activation function. But if the target values have no known bounded range, it is better to use an unbounded activation function, most often the identity function (which amounts to no activation function). If the target values are positive but have no known upper bound, an exponential output activation function can be used [5].

## 2.5. *Network topologies*

The topology of a network is defined by the number of layers, the number of units per layer, and the interconnection patterns between layers. They are generally divided into two categories based on the pattern of connections:

### 2.5.1. Feed-forward networks

Feed-forward networks (Figure 3-1), where the data flow from input units to output units is strictly feed-forward. The data processing can extend over multiple layers of units, but no feedback connections are present. That is, connections extending from

outputs of units to inputs of units in the same layer or previous layers are not permitted. Feed-forward networks are the main focus of this thesis. Details will be described in Chapter 3.

### 2.5.2. Recurrent networks

Recurrent networks (Figure 2-1), which contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which activation does not change further. In other applications in which the dynamical behavior constitutes the output of the network, the changes of the activation values of the output units are significant.

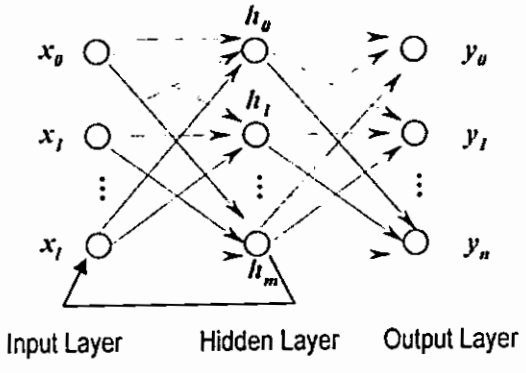



Figure 2-1 Recurrent neural network

## 2.6. *Network Learning*

The functionality of a neural network is determined by the combination of the topology (number of layers, number of units per layer, and the interconnection pattern between the layers) and the weights of the connections within the network. The topology is usually held fixed, and the weights are determined by a certain training algorithm. The process of adjusting the weights to make the network learn the relationship between the inputs and targets is called *learning*, or *training*. Many learning algorithms have been invented to help find an optimum set of weights that results in the solution of the problems. They can roughly be divided into two main groups:

### 2.6.1. Supervised Learning

The network is trained by providing it with inputs and desired outputs (target values). These input-output pairs are provided by an external teacher, or by the system containing the network. The difference between the real outputs and the desired outputs is used by the algorithm to adapt the weights in the network (Figure 2-1). It is often posed as a function approximation problem - given training data consisting of pairs of input patterns $x$, and corresponding target $t$, the goal is to find a function $f(x)$ that matches the desired response for each training input.

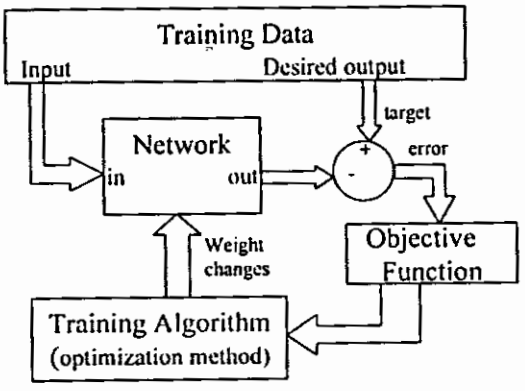



Figure 2-1 Supervised learning model

### 2.6.2. Unsupervised Learning

With unsupervised learning, there is no feedback from the environment to indicate if the outputs of the network are correct. The network must discover features, regulations, correlations, or categories in the input data automatically. In fact, for most varieties of unsupervised learning, the targets are the same as inputs. In other words, unsupervised learning usually performs the same task as an auto-associative network, compressing the information from the inputs.

## 2.7. *Objective Function*

To train a network and measure how well it performs, an *objective function* (or *cost function*) must be defined to provide an unambiguous numerical rating of system performance. Selection of an objective function is very important because the function represents the design goals and decides what training algorithm can be taken. To develop an objective function that measures exactly what we want is not an easy task.

A few basic functions are very commonly used. One of them is the sum of squares *error function*,

$$E = \frac{1}{NP} \sum_{p=1}^{P} \sum_{i=1}^{N} (t_{pi} - y_{pi})^2 \qquad (2.7)$$

where $p$ indexes the patterns in the training set, $i$ indexes the output nodes, and $t_{pi}$ and $y_{pi}$ are, respectively, the target and actual network output for the $i$th output unit on the $p$th pattern. In real world applications, it may be necessary to complicate the function with additional terms to control the complexity of the model.

# DESIGN

# 3. Design

In design section we introduce the basic architecture of Feed Forward Neural Network, its representation Capability and the network structure. Hidden Layers of the network, Back-Propagation and Optimization Algorithms are the key parts of this section.

## 3.1. Basic Architecture Of Feed Forward Neural Networks

A layered feed-forward network consists of a certain number of layers, and each layer contains a certain number of units. There is an input layer, an output layer, and one or more hidden layers between the input and the output layer. Each unit receives its inputs directly from the previous layer (except for input units) and sends its output directly to units in the next layer (except for output units). Unlike the *Recurrent network,* which contains feedback information, there are no connections from any of the units to the inputs of the previous layers nor to other units in the same layer, nor to units more than one layer ahead. Every unit only acts as an input to the immediate next layer. Obviously, this class of networks is easier to analyze theoretically than other general topologies because their outputs can be represented with explicit functions of the inputs and the weights.

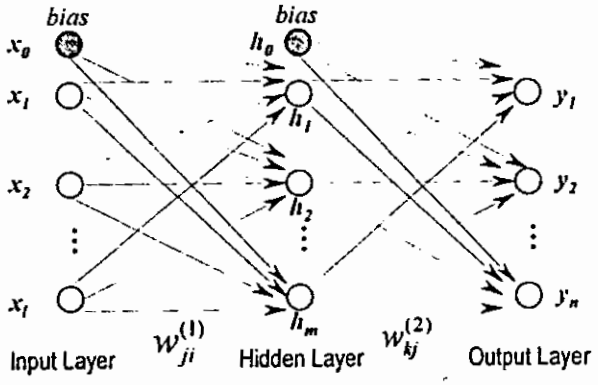



Figure 3-1 Feed-forward neural network

An example of a layered network with one hidden layer is shown in Figure 3-1. In this network there are $l$ inputs, $m$ hidden units, and $n$ output units. The output of the $j$th hidden unit is obtained by first forming a weighted linear combination of the $l$ input values, then adding a bias,

$$a_j = \sum_{i=1}^{l} w_{ji}^{(1)} x_i + w^{(1)}{}_{j0} \tag{3.1}$$

where $w_{ji}^{(1)}$ is the weight from input $i$ to hidden unit $j$ in the first layer and $w_{j0}^{(1)}$ is the bias for hidden unit $j$. If we are considering the bias term as being weights from an extra input $x_0 = 1$, (3.1) can be rewritten to the form of,

$$a_j = \sum_{i=0}^{l} w_{ji}^{(1)} x_i \tag{3.2}$$

The activation of hidden unit $j$ then can be obtained by transforming the linear sum using an activation function $g(x)$:

$$h_j = g(a_j) \tag{3.3}$$

The outputs of the network can be obtained by transforming the activation of the hidden units using a second layer of processing units. For each output unit $k$, first we get the linear combination of the output of the hidden units,

$$a_k = \sum_{j=1}^{m} w_{kj}^{(2)} h_j + w_{k0}^{(2)} \tag{3.4}$$

Again we can absorb the bias and rewrite the above equation to,

$$a_k = \sum_{j=0}^{m} w_{kj}^{(2)} h_j \tag{3.5}$$

Then applying the activation function $g2(x)$ to (3.5) we can get the $k$th output

$$y_k = g2(a_k) \tag{3.6}$$

Combining (3.2), (3.3), (3.5) and (3.6) we get the complete representation of the network as

$$y_k = g2(\sum_{j=0}^{m} w_{kj}^{(2)} g(\sum_{i=0}^{l} w_{ji}^{(1)} x_i)) \quad (3.7)$$

The network of Figure 3-1 is a network with one hidden layer. We can extend it to have two or more hidden layers easily as long as we make the above transformation further.

One thing we need to note is that the input units are very special units. They are hypothetical units that produce outputs equal to their supposed inputs. No processing is done by these input units.

## 3.2. *Representation Capability*

The feed-forward networks provide a general framework for representing non-linear functional mapping between a set of input variables and a set of output variables. The representation capability of a network can be defined as the range of mappings it can implement when the weights are varied. Theories [5] [6] [7] show that:

- Single-layer networks are capable of representing only linearly separable functions or linearly separable decision domains.

- Two hidden layered networks can represent an arbitrary decision boundary to arbitrary accuracy with threshold activation functions and could approximate any smooth mapping to any accuracy with sigmoid activation functions.

- One hidden layered network can approximate arbitrarily well any functional continuous mapping from one finite-dimensional space to another, provided that the number of hidden units is sufficiently large.

To be more precise, feed-forward networks with a single hidden layer and trained by least-squares are statistically consistent estimators of arbitrary square-integral regression functions if assumptions about samples, target noises, number of hidden units, and other factors are all met. Feed-forward networks with a single hidden layer using threshold or sigmoid activation functions are universally consistent estimators of binary classifications under similar assumptions.

## 3.3. *Network Structure Design*

Though theoretically there exists a network that can simulate a problem to any accuracy, there is no easy way to find it. To define an *exact* network architecture such as how many hidden layers should be used, how many units should there be within a hidden layer for a certain problem is always a painful job. Here we will give a brief discussion about the issues to be considered when we design a network.

## 3.4. *Number of Hidden Layers*

Because networks with two hidden layers can represent functions with any kind of shapes, there is no theoretical reason to use networks with more than two hidden layers. It has also been determined that for the vast majority of practical problems, there is no reason to use more than one hidden layer. Problems that require two hidden layers are only rarely encountered in practice. Even for problems requiring more than one hidden layer theoretically, most of the time, using one hidden layer performs much better than using two hidden layers in practice [1]. Training often slows dramatically when more hidden layers are used. There are several reasons why we should use as few layers as possible in practice:

- Most training algorithms for feed-forward network are gradient-based. The additional layer through which errors must be back propagated makes the gradient very unstable. The success of any gradient-directed optimization algorithm is dependent on the degree to which the gradient remains unchanged as the parameters vary.

- The number of local minima increases dramatically with more hidden layers. Most of the gradient-based optimization algorithms can only find local minima, thus they miss the global minima. Even though the training algorithm can find the global

minima, there is a higher probability that after much time-consuming iteration, we will find ourselves stuck in a local minimum and have to escape or start over.

Of course, it is possible that for a certain problem, using more hidden layers of just a few units is better than using fewer hidden layers requiring too many units, especially for networks that need to learn a function with discontinuities. In general, it is strongly recommended that one hidden layer be the first choice for any practical feed-forward network design. If using a single hidden layer with a large number of hidden units does not perform well, then it may be worth trying a second hidden layer with fewer processing units.

## 3.5. *Number of Hidden Units*

Another important issue in designing a network is how many units to place in each layer. Using too few units can fail to detect the signals fully in a complicated data set, leading to *underfitting*. Using too many units will increase the training time, perhaps so much that it becomes impossible to train it adequately in a reasonable period of time. A large number of hidden units might cause *overfitting*, in which case the network has so much information processing capacity, that the limited amount of information contained in the training set is not enough to train the network.

The best number of hidden units depends on many factors – the numbers of input and output units, the number of training cases, the amount of noise in the targets, the

complexity of the error function, the network architecture, and the training algorithm [5].

There are lots of "rules of thumb" for selecting the number of units in the hidden layers [1] [5]:

- $m \in [l, n]$ - between the input layer size and output layer size
- $m = \frac{2(l+n)}{3}$ - two third of the input layer size plus the output layer size
- $m < 2l$ - less than twice the input layer size
- $m = \sqrt{l \cdot n}$ - squared input layer size multiplied by output layer size

Those rules can only be taken as a rough reference when selecting a hidden layer size. They do not reflect the facts well because they only consider the factor of the input layer size and output layer size but ignore other important factors that we previously mentioned.

In most situations, there is no easy way to determine the optimal number of hidden units without training using different numbers of hidden units and estimating the generalization error of each. The best approach to find the optimal number of hidden units is trial and error. In practice, we can use either the forward selection or backward selection to determine the hidden layer size. *Forward* selection starts with choosing an appropriate criterion for evaluating the performance of the network. Then we select a small number of hidden units, like two if it is difficult to guess how small it is; train and test the network; record its performance. Next we slightly increase the number of hidden units; train and test until the error is acceptably small, or no significant

improvement is noted, whichever comes first. *Backward* selection, which is in contrast with forward selection, starts with a large number of hidden units, and then decreases the number gradually [1][8]. This process is time-consuming, but it works well.

## 3.6. *Back-Propagation*

Back-propagation is the most commonly used method for training multi-layer feed-forward networks. It can be applied to any feed-forward network with differentiable activation functions. This technique was popularized by Rumelhart, Hinton and Williams [9].

For most networks, the learning process is based on a suitable error function (Section 2.7), which is then minimized with respect to the weights and bias. If a network has differential activation functions, then the activations of the output units become differentiable functions of input variables, the weights and bias. If we also define a differentiable error function of the network outputs such as the sum-of-square error function, then the error function itself is a differentiable function of the weights. Therefore, we can evaluate the derivative of the error with respect to weights, and either using the popular gradient descent or other optimization methods can then use these derivatives to find the weights that minimize the error function. The algorithm for evaluating the derivative of the error function is known as *back-propagation*, because it propagates the errors backward through the network.

## 3.7. *Error Function Derivative Calculation*

We considers a general feed-forward network with arbitrary differentiable non-linear activation functions and a differential error function.

From Section 3.1, we know that each unit $j$ is obtained by first forming a weighted sum of its inputs of the form,

$$a_j = \sum_i w_{ji} z_i \tag{3.8}$$

where $z_i$ is the activation of an unit, or input. We then apply the activation function

$$z_j = g(a_j) \tag{3.9}$$

Note that one or more of the variables $z_j$ in (3.8) could be an input, in which case we will denote it by $x_i$. Similarly, the unit $j$ in (3.9) could be an output unit, which we will denote by $y_k$.

The error function will be written as a sum, over all patterns in the training set, of an error defined for each pattern separately,

$$E = \sum_p E_p, \quad E_p = E(Y;W) \tag{3.10}$$

where $p$ indexes the patterns, $Y$ is the vector of outputs, and $W$ is the vector of all weights. $E_p$ can be expressed as a differentiable function of the output variable $y_k$.

The goal is to find a way to evaluate the derivatives of the error functions $E$ with respect to the weights and bias. Using (3.10) we can express these derivatives as sums over the training set patterns of the derivatives for each pattern separately. Now we will consider one pattern at a time.

For each pattern, with all the inputs, we can get the activations of all hidden and output units in the network by successive application of (3.8) and (3.9). This process is called *forward propagation* or *forward pass*. Once we have the activations of all the outputs, together with the target values, we can get the full expression of the error function $E_p$.

Now consider the evaluation of the derivative of $E_p$ with respect to some weight $w_{ji}$. Applying the chain rule can get the partial derivatives

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \tag{3.11}$$

where we define

$$\delta_j = \frac{\partial E_p}{\partial a_j} \tag{3.12}$$

From equation (3.11) it is easy to see that the derivative can be obtained by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end. Thus the task becomes to find the $\delta_j$ for each hidden and output unit in the network.

For the output unit, $\delta_k$ is very straightforward,

$$\delta_k = \frac{\partial E_p}{\partial a_k} = \frac{\partial E_p}{\partial y_k} g'(a_k) \tag{3.13}$$

For a hidden unit, $\delta_k$ is obtained indirectly. Hidden units can influence the error only through their effects on the unit $k$ to which they send output connections so

$$\delta_j = \frac{\partial E_p}{\partial a_j} = \sum_k \frac{\partial E_p}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{3.14}$$

The first factor is just the $\delta_k$ of unit $k$ so

$$\delta_j = \frac{\partial E_p}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j} \tag{3.15}$$

For the second factor we know that if unit $j$ connects directly to unit $k$ then $\partial a_k / \partial a_j = g'(a_j) w_{kj}$, otherwise it is zero. So we can get the following *back-propagation* formula,

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \tag{3.16}$$

which means that the values of $\delta$ for a particular hidden unit can be obtained by propagating the $\delta$'s backwards from units later in the network, as shown in Figure 3-1. Recursively applying the equation gets the $\delta$'s for all of the hidden units in a feed-forward network, no matter how many layers it has.

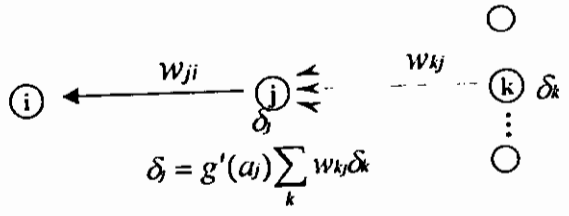



Figure 3-1 Backward propagation

## 3.8. *Weight Adjustment with the Gradient Descent Method*

Once we get the derivatives of the error function with respect to weights, we can use them to update the weights so as to decrease the error. There are many varieties of gradient-based optimization algorithms based on these derivatives. One of the simplest of such algorithms is called *gradient descent* or *steepest descent*. With this

algorithm, the weights are updated in the direction in which the error $E$ decreases most rapidly, i.e., along negative gradient. The weight updating process begins with an initial guess for weights (which might be chosen randomly), and then generates a sequence of weights using the following formula,

$$\Delta w_{ji}^{(r+1)} = -\eta \frac{\partial E}{\partial w_{ji}} \tag{3.17}$$

where $\eta$ is a small positive number called the *learning rate*, which is the step size we need to take for the next step. Gradient descent only tells us the direction we will move to, but the step size or learning rate needs to be decided as well. Too low a learning rate makes the network learn very slowly, while too high a learning rate will lead to oscillation. One way to avoid oscillation for large $\eta$ is to make the weight change dependent on the past weight change by adding a *momentum* term,

$$\Delta w_{ji}^{(r+1)} = -\eta \frac{\partial E}{\partial w_{ji}} + \alpha \Delta w_{ji}^{(r)} \tag{3.18}$$

That is, the weight change is a combination of a step down the negative gradient, plus a fraction α of the previous weight change, where $0 \le \alpha < 1$ and typically

$0 \le \alpha < 0.9$ [6].

The role of the learning rate and the momentum term are shown in Figure 3-1 [3]. When no momentum term is used, it typically takes a long time before the minimum is reached with a low learning rate (a), whereas for large learning rates the minimum may be never reached because of oscillation (b). When adding a momentum term, the minimum will be reached faster (c).

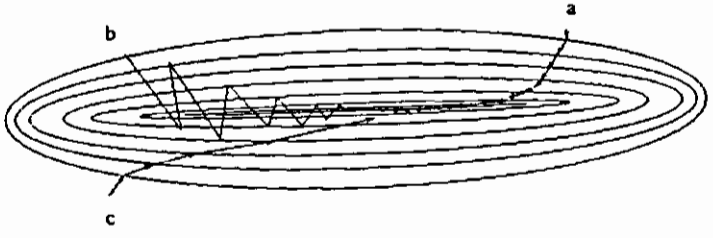



Figure 3-1 The descent vs. learning rate and momentum

There are two basic weight-update variations: batch learning and incremental learning. With *batch learning*, the weights are updated over all the training data. It repeats the following loop: a) Process all the training data; b) Update the weights. Each such loop through the training set is called an *epoch*. While for *incremental learning*, the weights are updated for each sample separately. It repeats the following loop: a) Process one sample from the training data; b) Update the weights.

## 3.9. *Other Optimization Algorithms*

Though the gradient descent optimization method used in the standard back-propagation learning algorithm is widely used and proven very successful in many applications, it does suffer two problems:

- The convergence tends to be extremely slow
- Convergence to the global minimum is not guaranteed

Many researchers [6][7][10][11][12] have devised improvements to the standard gradient descent method such as dynamically modifying learning parameters or adjusting the steepness of the sigmoid function.

In appropriate circumstances, other optimization methods may be better than the gradient descent. Many converge much faster than gradient descent in certain situations while others promise a higher probability of convergence to global minima [6].

One of the most often recommended optimization methods to replace the gradient descent is *conjugate gradient descent* [1][6][7], which is a *direction set* minimization method. Minimization along a direction $d$ brings the function $E$ to a place where its gradient is perpendicular to $d$. Instead of following the gradient at every step, a set of $n$ directions is constructed which are all conjugate to each other, such that

minimization along one of these directions does not spoil the minimization along one of the earlier direction.

Gradient methods using second-derivatives (Hessian matrix), such as *Newton's* method, can be very efficient under certain conditions [6]. Where first-order methods use a local linear approximation of the error surface, second-order methods use a quadratic approximation. Because such methods use all the first and second order derivative information in exact form, local convergence properties are excellent. Unfortunately, they are often impractical because explicit calculations of the full Hessian matrix can be very expensive in large problems.

Some powerful, stochastic optimization methods such as *simulated annealing* [1][6] and *genetic algorithms* [1][6], which can overcome the local minima, have also been used successfully in a number of problems.

# IMPLEMENTATION

# 4. Implementation

Data is being collected and processed after the analysis and design of neural network by determining the types of variables and preliminary data.

## 4.1. Data Collection, Analysis and Processing

One of the most important components in the success of any neural network solution is the data. The quality, availability, reliability, repeatability, and relevance of the data used to develop and run the system is critical to its success. Even a primitive model can perform well if the input data has been processed in such a way that it clearly reveals the important information. On the other hand, even the best model cannot help us much if the necessary input information is presented in a complex and confusing way.

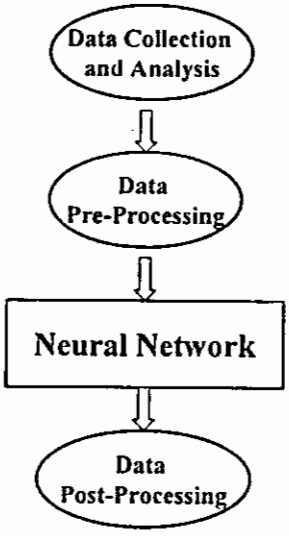



Figure 4-1 Data processing

Data processing starts from the data collection and analysis, followed by pre-processing and then feeds to the neural network. Finally, post-processing is needed to transform the outputs of the network to the required outputs (Figure 4-1), if necessary. This chapter discusses some of the most important considerations involved in processing data for neural networks.

## 4.2. *Types of Variables*

Variables can be roughly divided into two categories based on their properties [1][7]:

### 4.2.1. Categorical Variables

Categorical variables do not have a natural ordering – they do not have relationships like "greater than" or "less than". Some of them come from some input values that do not have numerical values but have to transform to numerical values as input

variables. For example, a variable called "color type", which can take on the value "red", "green", and "yellow" is a categorical variable. Sex is a categorical variable too. Numerical data can also be categorical. Zip codes and telephone area codes are classic examples.

Categorical variables can be presented to the networks with the *1-of-c* encoding scheme, which has as many units as there are values that the variable can take on. Exactly one of the units will be turned on according to the value of the variable, and all the other units will be turned off. In the above "color type" example, it requires three input variables, with the three colors represented by input values of (1,0,0), (0,1,0) and (0,0,1).

Another way to encode categorical variables is to represent all the possible values to one continuous input variable. For example, the "red", "green", and "yellow" could be represented as 0.0, 0.5, and 1.0. The bad news for this method is that it imposes an artificial ordering on the data that does not exist. But for variables with a large number of categories, this can dramatically decrease the number of input units.

### 4.2.2. Ordinal Variables

Ordinal Variables have a natural ordering. Such data can be simply transformed directly into corresponding values of a continuous variable, either with or without scaling.

## 4.3. Data Collection

The data collection plan typically consists of three tasks:

- Identifying the data requirement

The first thing to do when planning data collection is to decide what data we will need to solve the problem. In general, it will be necessary to obtain the assistance of some experts in the field. We need to know: a) What data are definitely relevant to the problem; b) What data may be relevant; c) What data are collateral. Both relevant and possibly relevant data should be considered as inputs to the application.

- Identifying data sources

The next step is to decide from where the data will be obtained. This will allow us to make realistic estimates of the difficulty and expense of obtaining it. If the application demands real time data, these estimates should include an allowance for converting analogue data to digital form. In some cases, it may be desirable to obtain data from a simulation of the real situation. This could be the case if the application is intended to monitor conditions which have health, safety or significant cost implications. Care must be taken to ensure that the simulation is accurate and representative of the real case.

- Determining the data quantity

It is important to make a reasonable estimation of how much data we will need to develop the neural network properly. If too little data is collected, it may not reflect the full range of properties that the network should be learning, and this will limit its performance with unseen data. On the other hand, it is possible to introduce unnecessary expense by collecting too much data. In general, the quantity of data required is governed by the number of training cases that will be needed to ensure the network performs adequately. The intrinsic dimensionality of the data and the required resolution are the main factors determining the number of training cases and, therefore, the quantity of data required.

It is vital to assess correctly the quality of the data that will be presented to the neural network. Often, the data will be less than perfect, and if the network is to perform correctly then it needs to be trained with a greater quantity of data than would be the case if high quality data were available.

## *4.4. Preliminary Data Analysis*

There are two basic techniques which can be used to help us understand the data:

- Statistical analysis

Neural networks can be regarded as extensions of standard statistical techniques, and so such tests can give us an idea of the performance the network is likely to achieve.

In addition, analysis can give useful clues to the defining features - for example, if the data is divided into classes, a statistical test can determine the possibility of distinguishing between the different classes in raw data or pre-processed data.

- Data visualization

Plotting a graph of the data in a suitable format enables us to spot distinguishing features, such as kinks or peaks, which characterize the data. This will enable us to plan and, if practicable, test the pre-processing required to enhance those features.

Preliminary data analysis often combines both visualization and statistical tests in an iterative manner. Visualization gives an appraisal of the data, and ideas about the underlying patterns, while statistical analysis enables us to test those ideas.

## 4.5. *Data Preparation*

When the raw data has been collected, it may need converting into a more suitable format. At this stage, we should do the following:

### 4.5.1. Data Validity Checks

Data validity checks will reveal any patently unacceptable data that, if retained, would produce poor results. A simple data range check is an example of validity checking. For example, if we have collected oven temperature data in degrees centigrade, we would expect values in the range 50 to 400. A value of, say, -10, or 900, is clearly wrong.

If there is a pattern to the distribution of faulty data (for example, if most was collected on a Monday morning), try and diagnose the cause. Depending on the nature of the fault, we may need to discard the data or make allowances for its shortcomings. If there exist undesirable deterministic components such as trends or seasonal variation, they should be removed first [13].

### 4.5.2. Partitioning of Data

Partitioning is the process of dividing the data into validation sets, training sets, and test sets. By definition, *validation sets* are used to decide the architecture of the network; *training sets* are used to actually update the weights in a network; *test sets* are used to examine the final performance of the network. The primary concerns should be to ensure that: a) the training set contains enough data, and suitable data distribution to adequately demonstrate the properties we wish the network to learn; b) there is no unwarranted similarity between data in different data sets.

## *4.6. Data Pre-Processing*

Theoretically, a neural network could be used to map the raw input data directly to required output data. But in practice, it is nearly always beneficial, sometimes critical to apply pre-processing to the input data before they are fed to a network. There are many techniques and considerations relevant to data pre-processing. Pre-processing can vary from simple filtering (as in time-series data), to complex processes for extracting features from image data. Since the choice of pre-processing algorithms

depends on the application and the nature of the data, the range of possibilities is vast. However, the aims of pre-processing algorithms are often very similar, namely [1][5][7]:

1) Transform the data into a form suited to the network inputs - this can often simplify the processing that the network has to perform and lead to faster development times. Such transformations may include:

- Apply a mathematical function ( logarithm or square) to an input;
- Encode textual data from a database;
- Scale data so that it has a zero mean and a standard deviation of one;
- Take the Fourier transform of a time-series.

2) Select the most relevant data - This may include simple operations such as filtering or taking combinations of inputs to optimize the information content of the data. This is particularly important when the data is noisy or contains irrelevant information. Careful selection of relevant data will make networks easier to develop and improve their performance on noisy data.

3) Minimize the number of inputs to the network - Reducing the dimensionality of the input data and minimizing the number of inputs to the network can simplify the problem. In some situations - for example in image processing - it is simply impossible to apply all the inputs to the network. In an application to classify cell types from microscope images, each image may contain a quarter of a million pixels:

clearly, it would not be feasible to use that many inputs. In this case, the pre-processing might compute some simple parameters such as area and length/height ratio, which would then be used as inputs to the network. This process is called *feature extraction* [7].

## 4.7. Data Post-Processing

Post-processing covers any process that is applied to the output of the network. As with pre-processing, it is entirely dependent on the application and may include detecting when a parameter exceeds an acceptable range, or using the output of a network as one input to a rule-based processor. Sometimes it is just the reverse process of data pre-processing.

# THE LOAD FORECAST SYSTEM

# 5. The Load Forecast System

Neural networks are computational models with the capacity to learn, to generalize, or to organize data based on parallel processing. These networks can be trained with a powerful and computationally efficient method called error back-propagation. Forecasting the behavior of complex system has been a broad application domain for neural networks.

Network traffic forecast is a relatively new application of neural networks. We propose a system that uses neural networks to detect network congestion before it results in a breakdown of the network service and which also identifies the source of the congestion. Having the nodes identified, our system applies the flow rate restriction adaptively to the identified sources to avoid congestion overflowing the router's buffers. It should be noted that design and experiments presented in this thesis focus on congestion control; however, the techniques could be applicable to other network problems. It should also be noted that the remedy in the form of flow rate restrictions can be applied directly to the original flow source if it is within the domain controlled by our system, or it could be applied to the edge router to the domain to which our system is applied. In the latter case, the restriction will result in the packets of the restricted flow being dropped at the edge router to the domain [4]. This kind of a solution in which the congestion is decomposed and "moved" from the internal routers to the edge routers is becoming increasingly popular in modern traffic management. Finally, it should be noted that in such edge-control techniques the domains controlled by separate systems will collaborate through the edge routers.

Dropping packets at the entry edge router of one domain will cause the packets to be dropped at the exit router of the neighboring domain which will treat such dropping as congestion and then will identify the source of the flow.

As a result, our techniques can be applied locally at a domain of the decomposed network and their congestion solution will iteratively be mapped to the corresponding edge routers of the intermediate domains until the source of the flow is found and informed of the need to decrease the traffic. Another remark is needed to relate the present work to the differentiated services, methods of creating different levels of services for customers willing to pay higher levels. As more products are created to control networks, differentiated services will become very important for the Internet. Identification of sources that can be forced to limit their flow rate can lead to accounting for different priorities of traffic and offending flows.

For example, if traffic from a certain machine is deemed high priority, the system may restrict other machines, instead of slowing down the high priority machine. Changing the architecture [13] to operate on a flow basis instead of a machine basis could also be easily done to account for a variety of traffic from each machine, each with a different priority. Hence, the techniques that we present in this thesis are directly applicable to the "best effort traffic" over the Internet, but their extensions to Differentiated Services or Quality of Services environments are straightforward.

## 5.1. *Factors Affecting Load*

The most difficult part of building a good model is to choose and collect the training and testing input data. A number of research papers [14][15][16][17][18][19][20] show that the following factors influence the demand of load:

### 5.1.1. Weather Conditions

This includes temperature, wind velocity, cloud cover, dew point, rainfall, and snowfall.

### 5.1.2. Packet Scheduling

The basic function of packet scheduling is to reorder the output queue. There are many papers that have been written on possible ways to manage the output queue, and the resulting behavior. Perhaps the simplest approach is a priority scheme, in which packets are ordered by priority, and highest priority packets always leave first. This has the effect of giving some packets absolute preference over others; if there are enough of the higher priority packets, the lower priority class can be completely prevented from being sent. An alternative scheduling scheme is round-robin or some variant, which gives different classes of packets access to a share of the link. A variant called Weighted Fair Queuing, or WFQ, has been demonstrated to allocate the total bandwidth of a link into specified shares. There are more complex schemes for queue management, most of which involve observing the service objectives of individual packets, such as delivery deadline, and ordering packets based on these criteria.

### 5.1.3. Queue Size

The controlled dropping of packets is as important as their scheduling. Most obviously, a router must drop packets when its buffers are all full. This fact, however, does not determine which packet should be dropped. Dropping the arriving packet, while simple, may cause undesired behavior. In the context of today's Internet, with TCP operating over best effort IP service, dropping a packet is taken by TCP as a signal of congestion and causes it to reduce its load on the network. Thus, picking a packet to drop is the same as picking a source to throttle. Without going into any particular algorithm, this simple relation suggests that some specific dropping controls should be implemented in routers to improve congestion control. In the context of real-time services, dropping more directly relates to achieving the desired quality of service. If a queue builds up, dropping one packet reduces the delay of all the packets behind it in the queue. The loss of one can contribute to the success of many. The problem for the implementor is to determine when the service objective (the delay bound) is in danger of being violated. One cannot look at queue length as an indication of how long packets have sat in a queue. If there is a priority scheme in place, packets of lower priority can be pre-empted indefinitely, so even a short queue may have very old packets in it. While actual time stamps could be used to measure holding time, the complexity may be unacceptable. Some simple dropping schemes, such as combining all the buffers in a single global pool, and dropping the arriving packet if the pool is full, can defeat the service objective of a WFQ scheduling scheme. Thus, dropping and scheduling must be coordinated.

### 5.1.4. Link sharing

This scheme can provide controlled link sharing. The service objective here is not to bound delay, but to limit overload shares on a link, while allowing any mix of traffic to proceed if there is spare capacity. This use of ANN is available in commercial routers today, and is used to segregate traffic into classes based on such things are protocol type or application. For example, one can allocate separate shares to TCP, IPX and SNA, and one can assure that network control traffic gets a guaranteed share of the link.

### 5.1.5. Predictive real-time service

This service is actually more subtle than guaranteed service. Its objective is to give a delay bound which is, on the one hand, as low as possible, and on the other hand, stable enough that the receiver can estimate it. The mechanism leads to a guaranteed bound, but not necessarily a low bound.

### 5.1.6. Bandwidth Utilization

To explore the feasibility of the proposed scheme in ANN networks, we conduct simulation on the NS-2 simulator by considering a multi-node topology (Fig. 1). Many connections are established between a set of senders and receivers through the core ANN network. To focus on the data traffic over the forward path from node 0 to node 3, we assume the reverse path for ACK transfer is loss-free. In addition to explicit TCP traffic, we use self-similar traffic sources to generate background UDP traffic. For nodes 1, 2, 4 and 5, each one connects with four self-similar traffic sources via local add ports to generate UDP traffic with transmission load 0.1. The

generated traffic is equally dispersed to the other core nodes. Note that Fig.1 does not show those UDP sources for the purpose of simplicity.

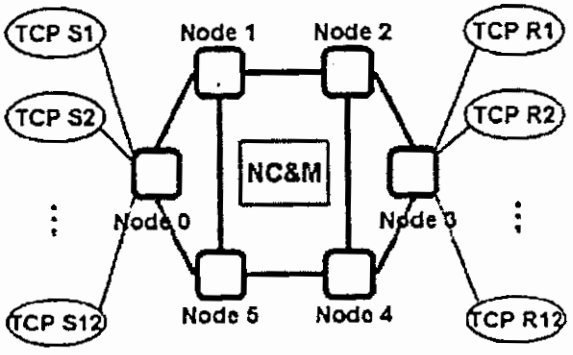



Figure 5-1 Network Topology used in Simulation

Due to the explicit TCP traffic and underground UDP traffic, the link from node 1 to 2 will become

overloaded at some time-points during the simulation period. Fig. 5.2 and 5.3 depict the measured link utilization over the link 1->2 in a network with and without congestion control, respectively. By dynamically adjusting the ACK transmission rate based on network status, the proposed scheme results in a smoother traffic load (Fig.5.2), and prevents the adverse load oscillations that are observed in the network without active congestion control (Fig.5.3).

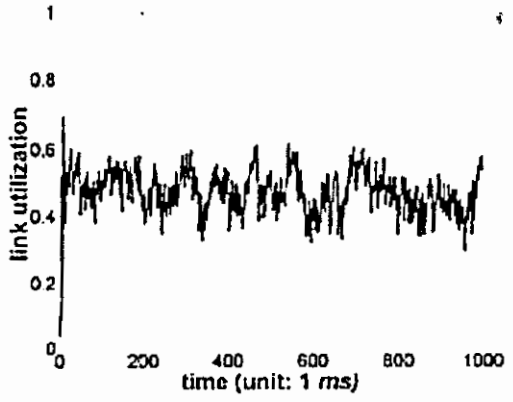



Figure 5-2 Utilization of link 1->2 in Network with Active Congestion Control

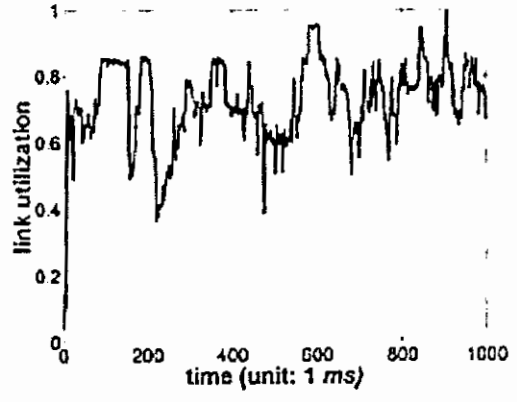



Figure 5-3 Utilization of link 1->2 in Network without Active Congestion Control

| | no congestion control | active congestion control |
|---|---|---|
| average utilization of link 1->2 | 0.719 | 0.4685 |
| packet-loss rate | 0.0442 | 0.00130 |
| average goodput | 25.5 Mbytes/sec | 38.4 Mbytes/sec |
| fairness index | 0.918 | 0.985 |

Table 5-1: Simulation Results of Link Utilization

## 5.2. *The Load Forecast Model*

### 5.2.1. Input Data

This table shows the types of incoming stream and it also specifies the delay, throughput requirements and error free Sequenced delivery.

| Traffic Type | Delay Requirement | Throughput Requirement | Error-free Sequenced Delivery |
|---|---|---|---|
| Interactive Simulation | High | Moderate-High | Yes |
| Network Monitoring | Moderate | High | Yes |
| Virtual Terminal | Low | Low | Yes |
| Bulk Transfer | Moderate | Moderate | No |
| Message | Moderate | Moderate | Yes |
| Voice | Low, constant | Moderate | No |
| Video | high,constant | High | Yes |
| Facsimile | Moderate | High | Yes |
| Image Transfer | Variable | High | Yes |
| Distributed Computing | Low | Variable | Yes |
| Network Control | Moderate | Low | Yes |

Table 5-1: Communication Traffic Requirements

### 5.2.2. Network Architecture

The network consists of one input layer, one output layer and one hidden layer. Obviously, there is only one output unit – the load. The number of input units is also fixed, depends on how many factors are included in the model, and how the factors are

encoded. The number of hidden units needs to be decided by training with some test sets. Figure 5.4 is the architecture of the load forecast model including all of the six effects that we mentioned before.

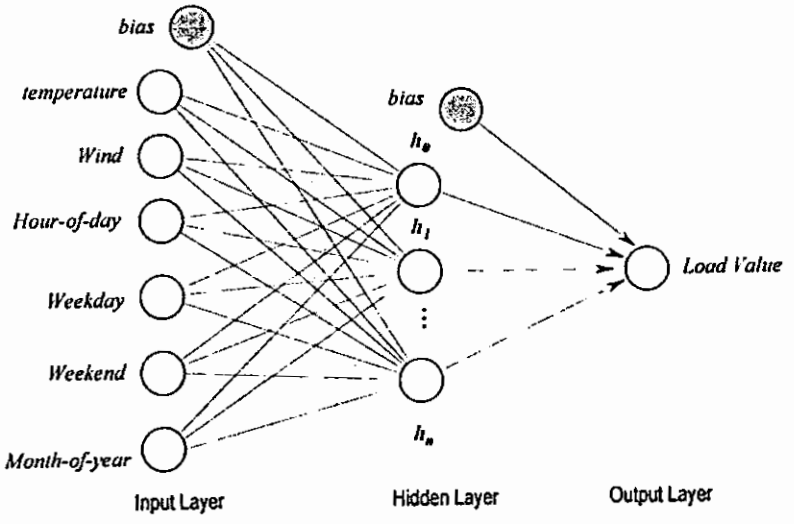



Figure 5-1 Load forecast model

The network requires enough hidden units to learn the general features of the relationship. With too many hidden units, it will cause *overfitting* while too few will lead to *underfitting* (Section 3.5). The goal is to use as few units in the hidden layer as possible while still retaining the network's ability to learn the relationships among the data. As mentioned earlier, including more than a single middle layer does not significantly improve the accuracy of the predictions.

The activation functions of the hidden units are sigmoid functions while the output activation function can be either a sigmoid function or a linear function, which can be selected by the users.

## 5.3. *Implementation of The Back-Propagation Algorithm*

The network is trained using the back-propagation algorithm. The standard sum-of-squares error function is used

$$E = \frac{1}{2}\sum_{k=1}^{n}(t_k - y_k)^2 \tag{5.1}$$

Here is the Java code for the error function, which is one of the methods within the *NeuralNetwork* class:

```
public double errorFunction (double[] x, double[] y) {
    double sum = 0.0;
    for (int i=0; i<x.length; i++)  sum += (x[i] - y[i])*(x[i] - y[i]);
    return 0.5 * sum;
}
```

As mentioned above, the activation function for the hidden units is the sigmoid function:

$$g(x) = \frac{1}{1+e^{-x}} \tag{5.2}$$

This function has a very useful feature – its derivative can be expressed in the following form:

$$g'(x) = g(x)(1-g(x)) \tag{5.3}$$

The above two equations can be easily coded:

```
public double sigmoid(double x) {
  if (x >  50.) return 1.0;
```

```
    if (x < -50.) return 0.0;
    return 1.0 /(1.0+Math.exp(-x));
  }
  public double sigmoidDerivative(double x) {
    return x*(1.0-x);
  }
```

The first step for the back-propagation is *forward propagation*

```
void feedForward() {
  //For hidden units
  for (int i = 0; i<numberOfHiddenUnit; i++) {
      double sum = 0.0;
      for (int j=0; j<numberOfInputUnit+1; j++) {
        if (j==numberOfInputUnit)
          sum += weightLayer1[j][i]; // Include the Bias term
        else  sum += weightLayer1[j][i]*inputs[j];
      }
      hiddens[i] = sigmoid(sum);
  }
 //For output units
  for (int i = 0; i<numberOfOutputUnit; i++) {
    double sum =0.0;
```

```
        for (int j=0; j<numberOfHiddenUnit; j++) {
            sum += weightLayer2[j][i]*hiddens[j];
        }
        outputs[i] = sigmoid(sum);
    }
}
```

The second step is *error back-propagation*. Using the expression derived from (3.13) and (3.16), we obtain the following results. For the output units, the δ's are given by

$$\delta_k = y_k - t_k \tag{5.4}$$

while for units in the hidden layer the δ's are found using

$$\delta_j = z_j(1 - z_j)\sum_{k=1}^{c} w_{kj}\delta_k \tag{5.5}$$

The derivatives with respect to the first layer and second layer weights are then given by

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i \text{ and } \frac{\partial E}{\partial w_{kj}} = \delta_k z_j \tag{5.6}$$

We use the gradient descent algorithm with momentum (3.18) to update the weights:

```
void backpropagation(double rate, double alpha) {
    double[] delta1 = new double[numberOfHiddenUnit];
    double[] delta2 = new double[numberOfOutputUnit];
    //Delta  for second layer
```

```
for (int j=0; j<numberOfOutputUnit; j++) {
    delta2[j] = targets[j] - outputs[j];
}
//Delta  for first layer
for (int j=0; j<numberOfHiddenUnit; j++) {
    double sum = 0.0;
    for (int k=0; k<numberOfOutputUnit; k++) {
        double term = delta2[k] * weightLayer2[j][k];
        if (outputActivationType==1) term *=sigmoidDerivative(outputs[k]);
        sum += term;
    }
    delta1[j] = sum;
}
//Update the second layer weights
for (int i=0; i<numberOfHiddenUnit; i++) {
    for (int j=0; j<numberOfOutputUnit; j++) {
        double delta = delta2[j]*hiddens[i];
        if (outputActivationType==1)  delta *= sigmoidDerivative(outputs[j]);
        double weightChange = rate * delta +alpha*momentum2[i][j];
        weightLayer2[i][j] += weightChange;
        momentum2[i][j] = weightChange;
```

```
        }
      }
      //Update the first layer weights
      for (int i=0; i<numberOfInputUnit+1; i++) {
        for (int j=0; j<numberOfHiddenUnit; j++) {
          if (i!=numberOfInputUnit && inputs[i]==0) {
            momentum1[i][j] = 0.;
          }
          else {
            double delta = delta1[j]*sigmoidDerivative(hiddens[j]);
            if (i!=numberOfInputUnit) delta *= inputs[i];
            double weightChange = rate * delta +alpha*momentum1[i][j];
            weightLayer1[i][j] += weightChange;
            momentum1[i][j] = weightChange;
          }
        }
      }
    }
```

Batch learning method was adopted to train the networks.

## 5.4. *Network Generalization*

The *Split-sample* (or *hold-out validation*) method [5] is used to estimate generalization error. With this method, part of the data is reserved as a test set that will not be used in the training. After training, run the network on the test set. The error on the test set provides us an unbiased estimate of the generalization error, with which we can decide whether the model is sufficiently general.

## 5.5. *Features of The System*

The load forecast system has several useful features:

- It checks the importance of each effect
- It helps find optimal number of hidden units
- It checks the influence of the learning rate and momentum
- It displays the training and forecasting result in graphics and tabular form
- It displays training errors

**Error! Reference source not found.** is the main screen of the Load Forecast system.

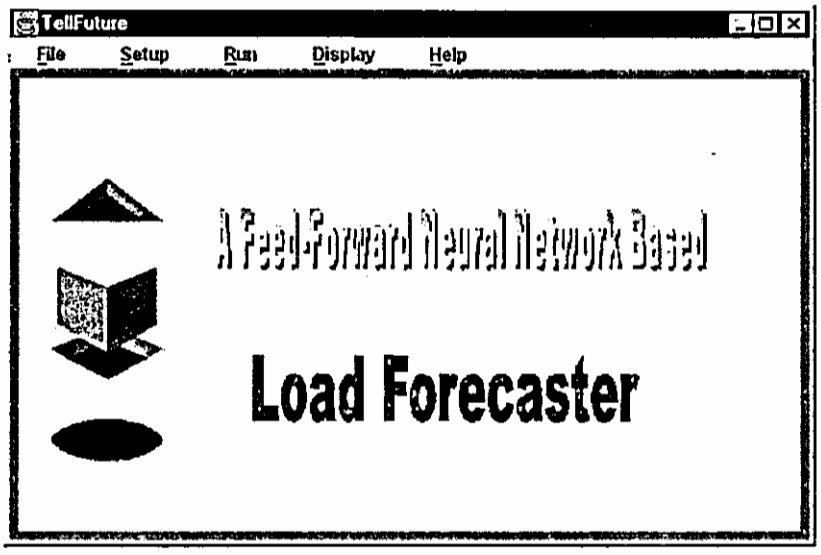



Figure 5-1 load forecast system main screen

## 5.6. *Effects Setup*

We can specify which effects can be included in the model by selecting **Setup->Effect Setup** menu, a dialog as show in Figure 5.5 will appear. Selecting or deselecting each factor can help us to evaluate the importance of this factor. This provides us an efficient way to decide whether the effect should be included in the model or not if we are not quite sure. For this model, the default values including all the effects are the best.

## 5.7. *Network Parameters Setup*

Once the input units and the output units are known, we can use the "Network Setup" (Figure 5.6) to find the optimal number of hidden units. This is a trial and error process. We can start with a large number of epochs and a small number of hidden units. Next, we train the network and write down the minimum training error that the

network can reach. Gradually we increase the number of hidden units and repeat the above process until the training error has no big changes.

We can also use this setup to study the influence of the learning rate and momentum to the training time. We can also choose different activation functions for the output units to see whether they have similar performances or not.

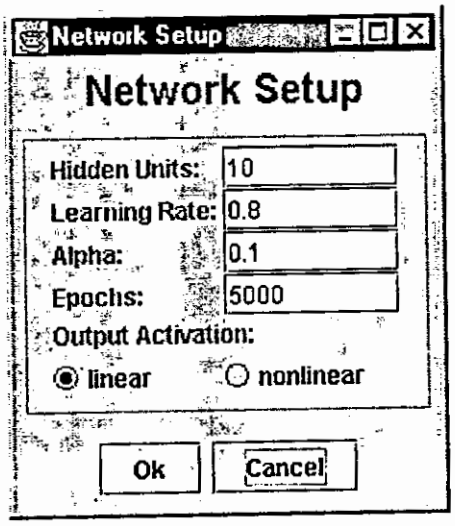



Figure 5-1 Network setup

## 5.8. *Training the Network*

From the main screen menu **File->Load Training Set**, training data which includes both input values and target values can be loaded to the system. The data are stored in the format as described previously. After setting up the effects and network parameters, we can click the **Run-> Training** menu to start the training process. The process stops upon hitting the error tolerance, or reaching the epoch limits. The training error for each epoch is shown in the prompt as the training process is going on. Once the training process finishes, we can click the **Display->Display Training**

**Results** menu to show the training results in graphic form (Figure 5.7) or tabular form (Figure 5-2). The hourly training errors are shown in column 4 (Figure 5-2).

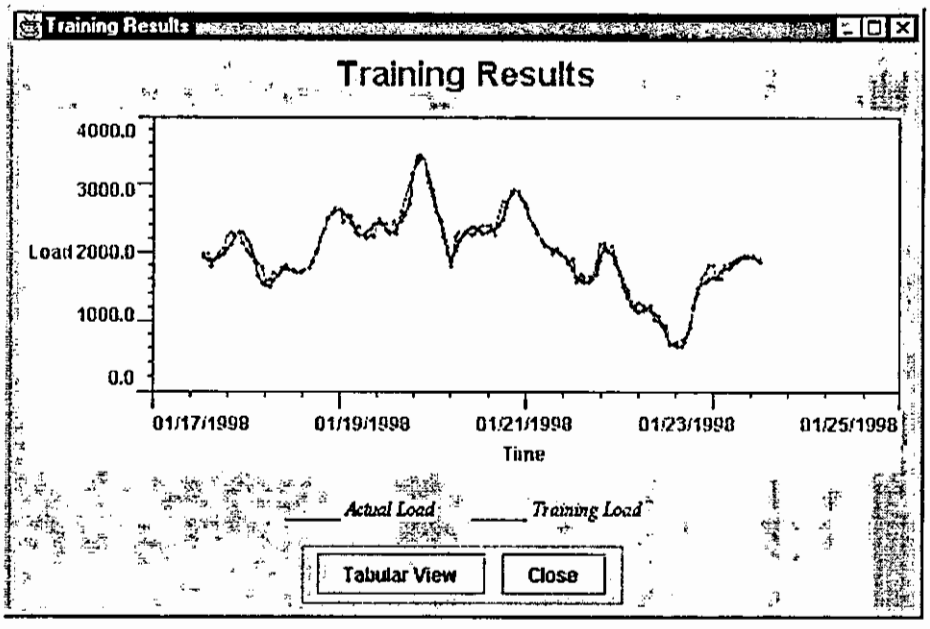



Figure 5-1 Training result graphic display

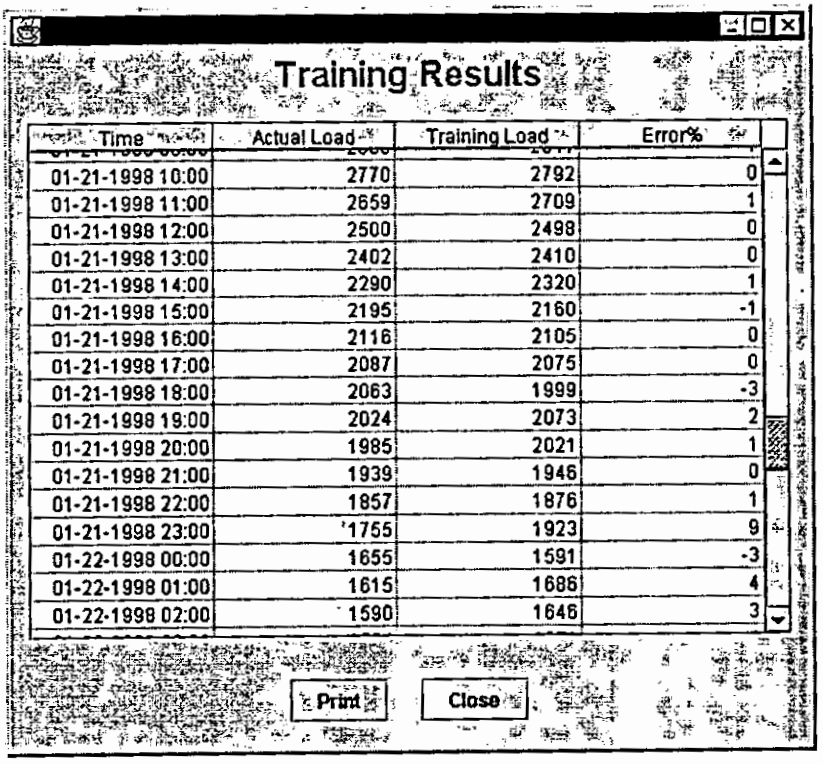



Figure 5-2 Training result tabular display

## 5.9. *Testing the Network*

Once we finish training the network, we need to do a generalization test to see whether this model is sufficiently general. This can be done by first loading the test set from menu **File->Load Test Set**; then clicking the **Run-> Testing** menu to run the system; and finally clicking the **Display->Display Testing Results** menu to show the testing results.

## 5.10. *Future Load forecasting*

After the network has been well trained and passed the generalization test, we can load the future predicted input data from the menu **File->Load Predicted Inputs,** and then

click the **Run-> Predict** menu to get the forecasted loads. The results can be displayed in graphic form Figure 5.10 or tabular form, which is similar to Figure 5.9 except without "Actual Load" and "Error%" columns.

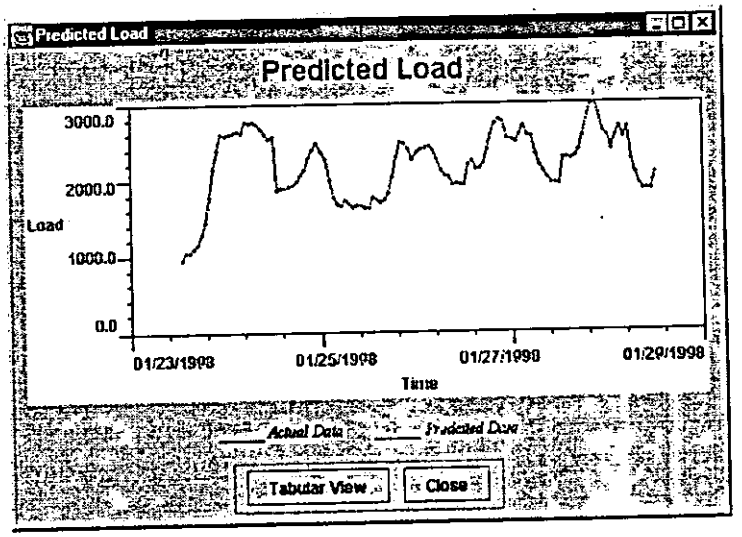



Figure 5-1 Load forecast graphic display

# RESULTS

# 6. Results

In this section we present the general analysis of results. We also discuss the future work by the end of the topic.

## 6.1. *Analysis of Results*

- The multi-layered feed-forward networks perform very well for short-term data load forecasting. The forecast accuracy has been in excess of 90% for this model. The weather and the calendar information have great impact on the load, especially the temperature.

- A reasonable number of hidden units for this model is between 7 and 12, and we recommend 10. Performances are not good if the numbers are less than 5 while larger than 12 has no significant decrease in the training error (Figure 6-1).

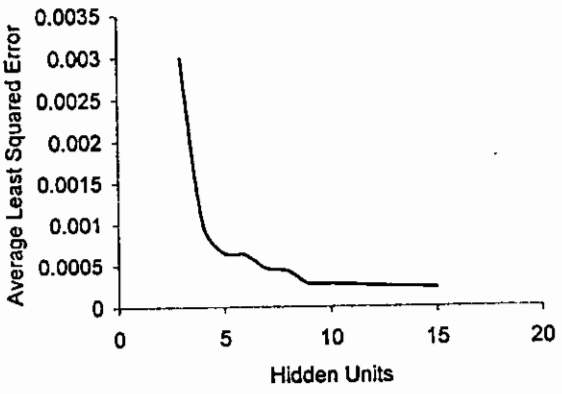



Figure 6-1 Average squared error vs. hidden units

- Learning rate and momentum have significant influence on the training process with the gradient descent method. Table 5.3 shows the epochs that the training processes taken to meet the error tolerant (average square error is 0.0005) or reach the epoch limit (9999) with different values of learning rate and momentum, where each pair had 10 tests. It is easy to see that too large and too small learning rates converge slowly, while high momentum helps small learning rate to converge fast. The best learning rate and momentum term are 0.8 and 0.1 respectively for this model.

- There are no big differences between using a sigmoid activation function and a linear activation function for the output unit.

Table 6-1 Training epochs vs. learning rate and momentum

| Learning Rate | Test | Momentum | | | | |
|---|---|---|---|---|---|---|
| | | 0 | 0.1 | 0.2 | 0.4 | 0.6 |
| 1.2 | 1 | 1493 | 1867 | 1933 | 3030 | 4806 |
| | 2 | 2014 | 1233 | 1207 | 2683 | 4728 |
| | 3 | 961 | 1649 | 3260 | 3898 | 5547 |
| | 4 | 2099 | 1642 | 2127 | 4230 | 2841 |
| | 5 | 1538 | 3158 | 2061 | 1967 | 9999 |
| | 6 | 2789 | 1834 | 3498 | 1804 | 9999 |
| | 7 | 1135 | 1650 | 1215 | 5708 | 2091 |
| | 8 | 2271 | 4508 | 1668 | 8503 | 9999 |
| | 9 | 1257 | 1936 | 2127 | 3454 | 3238 |
| | 10 | 4696 | 1253 | 1267 | 1625 | 9999 |
| | **Average** | **2025** | **1728** | **2036** | **3690** | **6325** |
| 1.0 | 1 | 2914 | 2813 | 4211 | 3421 | 9999 |
| | 2 | 3469 | 1106 | 2496 | 1701 | 4419 |
| | 3 | 1108 | 1770 | 2472 | 1898 | 9999 |
| | 4 | 2179 | 3019 | 1448 | 2568 | 9999 |
| | 5 | 1957 | 1468 | 2045 | 1567 | 9999 |
| | 6 | 1354 | 2086 | 961 | 2101 | 5333 |
| | 7 | 887 | 1796 | 2393 | 3286 | 9999 |
| | 8 | 2315 | 1442 | 1027 | 1268 | 1496 |
| | 9 | 1058 | 2143 | 1462 | 3698 | 3109 |
| | 10 | 1838 | 1626 | 1248 | 1324 | 3504 |
| | **Average** | **1908** | **1927** | **1976** | **2283** | **6786** |
| 0.8 | 1 | 1414 | 1689 | 2027 | 2022 | 1888 |
| | 2 | 4411 | 916 | 2374 | 6034 | 3147 |
| | 3 | 1600 | 1010 | 1960 | 1271 | 3100 |
| | 4 | 1532 | 1093 | 3634 | 1952 | 2992 |
| | 5 | 1237 | 1519 | 1603 | 1775 | 1782 |
| | 6 | 1011 | 1037 | 1448 | 3415 | 3256 |
| | 7 | 2622 | 1829 | 7767 | 1096 | 1002 |
| | 8 | 1509 | 1073 | 1293 | 1058 | 7131 |
| | 9 | 1565 | 2288 | 879 | 3006 | 2614 |
| | 10 | 1425 | 1141 | 2590 | 3372 | 2193 |
| | **Average** | **1833** | **1360** | **2558** | **2500** | **2911** |

(Continued)

| Learning Rate | Test | Momentum | | | | |
|---|---|---|---|---|---|---|
| | | 0 | 0.1 | 0.2 | 0.4 | 0.6 |
| 0.6 | 1 | 1493 | 1129 | 1000 | 630 | 3945 |
| | 2 | 5471 | 2582 | 845 | 1981 | 2091 |
| | 3 | 5461 | 1031 | 3003 | 1226 | 767 |
| | 4 | 1972 | 2734 | 2952 | 640 | 913 |
| | 5 | 2364 | 1796 | 1119 | 2405 | 1358 |
| | 6 | 1943 | 870 | 7932 | 1377 | 1099 |
| | 7 | 2088 | 5425 | 999 | 2209 | 953 |
| | 8 | 1753 | 2490 | 3348 | 3547 | 1517 |
| | 9 | 1887 | 2077 | 5242 | 1040 | 2705 |
| | 10 | 2728 | 2753 | 2968 | 2395 | 2205 |
| | **Average** | **2716** | **2289** | **2941** | **1745** | **1755** |
| 0.4 | 1 | 3861 | 2272 | 5690 | 1467 | 962 |
| | 2 | 5253 | 6670 | 1925 | 5917 | 1671 |
| | 3 | 2622 | 1506 | 1324 | 1302 | 765 |
| | 4 | 1762 | 5158 | 1489 | 2203 | 3333 |
| | 5 | 1554 | 3009 | 1138 | 1620 | 1370 |
| | 6 | 1722 | 1727 | 3962 | 1084 | 1340 |
| | 7 | 2507 | 6240 | 1948 | 1253 | 1351 |
| | 8 | 2222 | 4112 | 2035 | 814 | 3877 |
| | 9 | 1966 | 2170 | 1761 | 2560 | 2094 |
| | 10 | 2367 | 955 | 1667 | 1794 | 1207 |
| | **Average** | **2584** | **3382** | **2294** | **2001** | **1797** |
| 0.2 | 1 | 9999 | 2684 | 3611 | 4148 | 2639 |
| | 2 | 4065 | 4114 | 4564 | 2219 | 1441 |
| | 3 | 4187 | 3325 | 2874 | 3783 | 2594 |
| | 4 | 4821 | 4627 | 8726 | 5053 | 2388 |
| | 5 | 2713 | 2401 | 2090 | 1718 | 2297 |
| | 6 | 7866 | 3322 | 4891 | 7496 | 2481 |
| | 7 | 3908 | 8884 | 9999 | 6844 | 2681 |
| | 8 | 1619 | 1767 | 3095 | 3529 | 2623 |
| | 9 | 4740 | 6457 | 2221 | 2218 | 2282 |
| | 10 | 9999 | 3613 | 1483 | 1907 | 1563 |
| | **Average** | **5392** | **4119** | **4355** | **3892** | **2299** |

## 6.2. *Future Work*

We hope that we will continue this research and will Insha-Allah be able to present its extended architecture and the broader vision of our project.

### 6.2.1. Extending Architecture

An extended architecture can be formed to suit the biggest communication networks by decomposing these large networks into domains small enough to influence easily. Most large networks are already composed hierarchically of domains that are collections of local networks. When the larger networks have been divided, a separate protocol or addition to current protocols can be used to determine domain-to-domain agreements. As illustrated in Figure 5, the learning Control Agent would be somewhere within each domain defined in the network. Using the forecasting and detective powers of the agent, each domain would be regulated and operating at safe levels. To negotiate any problems between domains, the control agents communicate with each other as to the effects of one domain on another. The control agents take this information into account just as they take the information from local nodes into account. The inter-domain negotiation will promote the same safe state of operations between domains, as it will inside each domain. In particular, a border router between two domains is a source in one and the destination in the other. If the flow creates the congestion in the second domain, the agent in control of the second domain will apply corrections to the source border router, causing the router to drop packets from the flow to enforce the lower rate allocated to the flow. The dropping of packets at the border router, which is a destination for the first domain, will be perceived as congestion by the agent controlling the first domain. As a result, the agent of the first domain will apply correction to the source of the traffic, thereby eliminating congestion. An important feature of this architecture is that if the agent has no control over the source, it will apply rate correction to the first router under its control

causing it to drop packets from the offending flow. As a result, our technique will remedy malicious attacks by the "unruly" tcp sources by dropping the packets not responding to the request for the lower rate of traffic from the source.

## Broader Detection

At present, our system only predicts and detects traffic congestion. It would be useful to create a system that can do the same for other problems. It would also be of great value to train the network over many different types of traffic. Due to the changing nature of networking, it is not unreasonable for the network, at some point, not to fit the same mold our neural network learned. For this reason, it could prove beneficial to train the network with a real time recurrent learning algorithm to allow the neural network to continue learning after the training period [19]. Alternatively, we can envision an architecture in which a current generation of the neural network is used for control while simultaneously a new generation is being trained on the current traffic patterns. We believe that the period of run/training could be quite infrequent, something like few weeks at a time.

Different control neural networks could be trained for different times, special events, and other cases where specialized traffic needs are noticed.

# CONCLUSIONS

# 7. Conclusions

Neural networks can learn to approximate any function and behave like associative memories by using just example data that is representative of the desired task. They are model free estimators and are capable of solving complex problems based on the presentation of a large number of training data. This gives them a key advantage over traditional approaches to function estimation such as the statistical methods. Neural networks estimate a function without a mathematical description of how the outputs functionally depend on the inputs - they represent a good approach that is potentially robust and fault tolerant.

In this thesis, we examine the properties of the feed-forward neural networks and the process of determining the appropriate network inputs and architecture, and built up a short-term load forecast system. This system performs very well for short-term load forecasting. The forecast accuracy has been in excess of 90%.

In order to forecast the future load from the trained networks, we need to use the history loads, Queue size, packet scheduling, and delay information in addition to the predicted future number of packets and Queue size. Compared to other regression methods, the neural networks allow more flexible relationships between Queue size, number of packets, buffer information and load pattern. It has also been shown by other researchers that multi-layer feed-forward neural network performs best for short-term load forecasting [14][21].

We utilize only Queue size, number of packets and buffer information since they are the only information available to us. Since the neural networks simply interpolate

among the training data, it will give high errors with the test data that is not close enough to any one of the training data.

Feed-forward neural networks can be used in many kinds of forecasting in different industrial areas. Similar models can be built to make electric load forecasting, daily water consumption forecasting, stock and markets forecasting, traffic flow and product sale forecasting [22][23] as long as correct relationships between the inputs and the outputs can be captured and put in the models. But there is no universal network paradigm suitable for all kinds of forecasting problems. For each problem, a detailed analysis of domain data and the acquisition of prior knowledge are necessary to find a suitable model. The introduction of prior knowledge in input selection, input encoding, or architecture determination is often very useful, especially when the available domain data is limited.

The standard back-propagation algorithm for training feed-forward neural networks has proven robust even for difficult problems. However, its high performance results are attained at the expense of a long training time to adjust the network parameters, which can be discouraging in many real-world applications. Even on relatively simple problems, it often requires a lengthy training process in which the complete set of training examples is processed hundreds or thousands of times. Thus some accelerating techniques or advanced training algorithms (Section 3.8) can be applied to improve the performance of the networks.

We have seen that neural networks produced fairly accurate forecasts. In this work, we have tried to demonstrate that a neural network is a viable method of implementing a realistic forecasting application for data communication networks.

We have illustrated, through the use of a network simulator, that a neural network can be used to achieve great accuracy in predicting network congestion problem. Wireless networks need a special concern for congestion management. This work is applicable to both wired and wireless environment. We realize many more problems exist that for which neural network forecasting approach is applicable, but predicting congestion is just the initial step towards our research goals. When structural information of an actual data network is used to form the connections between layers of the neural network, this special design forces the neural network to consider the relationships only of those nodes that we think are important. A learning mechanism can be of great value for a network manager. The generalization power of a neural network particularly is appropriate because of the unpredicted variance of parameters that the network manager encounters. Neural networks are an appropriate mechanism for decision making in pro-active network management and should be the subject of more research.

# Appendix A(user Manual)

## //Program 1

## //DataSet.Java

```
import java.io.*;
import java.util.*;
import java.util.Date;
import java.text.*;

/** The DataSet class Uses the following variables:<ul>
 * <li>values[row][column]:      Storing the data for the dataset
 * <li>maxm[column]:             The maximum value in that column
 * <li>minm[column]:             The minimum value in that column
 * <li>scaleMaxm:                The maxm value to which the dataset is scaled
 * <li>scaleMinm:                The minm value to which the dataset is scaled
 * <li>order[row]:               The random shuffling order
 * <li>type                      Type of data Input or Output
 * </ul>

*/

public class DataSet implements Serializable {
   public String name;
   public int fromRow;
   public int toRow;

   public int inputs, outputs, rows;

   public double[] indexValues = null;
   public double[] dateValues = null;
   public double[][] inputValues = null;
   public double[][] scaledInputValues = null;
   public double maxInput[]   = null;
   public double minInput[]   = null;
   public double inputScaleMax=1;
   public double inputScaleMin=0;

   public double[][] outputValues = null;
   public double[][] scaledOutputValues = null;
   public double maxOutput[]= null;
   public double minOutput[]= null;
   public double outputScaleMax=1;
   public double outputScaleMin=0;

   public double[][] predictValues = null;
   public double[][] scaledPredictValues = null;

   public double[] count = null;
   public int[] order   = null;
```

```
public DataSet() {
}

 public DataSet(double index[],double date[],double input[][]){
          this.indexValues=index;
          this.dateValues=date;
          this.inputValues=input;
 try {
                    rows = inputValues.length;
   inputs  = inputValues[0].length;
           fromRow = 1;
           toRow = rows + 1;

   predictValues        = new double[rows][outputs];
   scaledInputValues    = new double[rows][inputs];
   count              = new double[rows];
   minInput             = new double[inputs];
   maxInput             = new double[inputs];
   order              = new int[rows];

   for (int kcol=0; kcol<inputs;kcol++) {
     minInput[kcol]=Double.POSITIVE_INFINITY;
     maxInput[kcol]=Double.NEGATIVE_INFINITY;
   }


                    for(int krow=0;krow<rows;krow++){
       for (int kcol=0; kcol<inputs;kcol++) {
        maxInput[kcol] = Math.max(maxInput[kcol],inputValues[krow][kcol]);
        minInput[kcol] = Math.min(minInput[kcol],inputValues[krow][kcol]);
     }

     count[krow] = (double)krow;
     order[krow] = krow;
   }
    randomize();
 }
 catch(Exception ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
 }
}


 public DataSet(double index[],double date[],double input[][],double output[][]){
          this.indexValues=index;
          this.dateValues=date;
          this.inputValues=input;
          this.outputValues=output;
 try {
                    rows = inputValues.length;
   inputs  = inputValues[0].length;
   outputs = outputValues[0].length;
           fromRow = 1;
           toRow = rows + 1;
```

```
    predictValues        = new double[rows][outputs];
    scaledInputValues    = new double[rows][inputs];
    scaledOutputValues   = new double[rows][outputs];
    scaledPredictValues  = new double[rows][outputs];
    count                = new double[rows];
    minInput             = new double[inputs];
    minOutput            = new double[outputs];
    maxInput             = new double[inputs];
    maxOutput            = new double[outputs];
    order                = new int[rows];

    for (int kcol=0; kcol<inputs;kcol++) {
      minInput[kcol]=Double.POSITIVE_INFINITY;
      maxInput[kcol]=Double.NEGATIVE_INFINITY;
    }

     for (int kcol=0; kcol<outputs;kcol++) {
      minOutput[kcol]=Double.POSITIVE_INFINITY;
      maxOutput[kcol]=Double.NEGATIVE_INFINITY;
    }

                    for(int krow=0;krow<rows;krow++){
        for (int kcol=0; kcol<inputs;kcol++) {
         maxInput[kcol] = Math.max(maxInput[kcol],inputValues[krow][kcol]);
         minInput[kcol] = Math.min(minInput[kcol],inputValues[krow][kcol]);
      }

      for (int kcol=0; kcol<outputs;kcol++) {
        maxOutput[kcol] = Math.max(maxOutput[kcol],outputValues[krow][kcol]);
        minOutput[kcol] = Math.min(minOutput[kcol],outputValues[krow][kcol]);
      }

       count[krow] = (double)krow;
       order[krow] = krow;
    }
      randomize();
    }
    catch(Exception ex) {
      System.out.println(ex.getMessage());
      ex.printStackTrace();
    }
  }

public DataSet(String name, int fromRow, int toRow, int inputs, int outputs) {
    this.name      = name;
    this.fromRow   = fromRow;
    this.toRow     = toRow;
    this.inputs    = inputs;
    this.outputs   = outputs;
}


public DataSet(String name, int inputs, int outputs) {
    this.name      = name;
    this.fromRow   = 0;
```

```
    this.toRow      = 0;
    this.inputs     = inputs;
    this.outputs    = outputs;
}


public DataSet(String name) {
    this.name       = name;
    this.fromRow    = 0;
    this.toRow      = 0;
}

public void setName(String s) {name = s;}
public void setFromRow(int i) {fromRow = i;}
public void setToRow  (int i) {toRow   = i;}
  public void setInputValues  (double[][]input) {inputValues=input;}
  public void setDateValues (double []date) {dateValues = date;}
  public void setIndexValues  (double []index) {indexValues = index;}
public String getName() {return name;}
public int getFromRow() {return fromRow;}
public int getToRow  () {return toRow;}
public int getRows   () {return rows;}
public int getInputs () {return inputs;}
public int getOutputs() {return outputs;}
public double getInputValues  (int i, int j) {return inputValues[i][j];}
public double getOutputValues (int i, int j) {return outputValues[i][j];}


public void CopyDataSet(DataSet ds) {
    name       = ds.name;
    fromRow = ds.fromRow;
    toRow   = ds.toRow;
    inputs  = ds.inputs;
    outputs  = ds.outputs;
  rows    = ds.rows;

  indexValues        = ds.indexValues;
  dateValues          = ds.dateValues;

    inputValues         = ds.inputValues;
    scaledInputValues   = ds.scaledInputValues;
    maxInput             = ds.maxInput;
    minInput             = ds.minInput;
    inputScaleMax        = ds.inputScaleMax;
    inputScaleMin        = ds.inputScaleMin;

    outputValues        = ds.outputValues;
    scaledOutputValues  = ds.scaledOutputValues;
    maxOutput           = ds.maxOutput;
    minOutput           = ds.minOutput;
    outputScaleMax       = ds.outputScaleMax;
    outputScaleMin       = ds.outputScaleMin;

    predictValues       = ds.predictValues;
    scaledPredictValues    = ds.scaledPredictValues;
```

```
    count           = ds.count;
    order           = ds.order;
}



public void setVariables() {
    try {
                        rows = inputValues.length;
      inputs  = inputValues[0].length;
      outputs = outputValues[0].length;
                        fromRow = 1;
                        toRow = rows + 1;

      predictValues       = new double[rows][outputs];
      scaledInputValues   = new double[rows][inputs];
      scaledOutputValues  = new double[rows][outputs];
      scaledPredictValues = new double[rows][outputs];
      count           = new double[rows];
      minInput          = new double[inputs];
      minOutput         = new double[outputs];
      maxInput          = new double[inputs];
      maxOutput         = new double[outputs];
      order           = new int[rows];

      for (int kcol=0; kcol<inputs;kcol++) {
        minInput[kcol]=Double.POSITIVE_INFINITY;
        maxInput[kcol]=Double.NEGATIVE_INFINITY;
      }

      for (int kcol=0; kcol<outputs;kcol++) {
        minOutput[kcol]=Double.POSITIVE_INFINITY;
        maxOutput[kcol]=Double.NEGATIVE_INFINITY;
      }
                        for(int krow=0;krow<rows;krow++){
        for (int kcol=0; kcol<inputs;kcol++) {
          maxInput[kcol] = Math.max(maxInput[kcol],inputValues[krow][kcol]);
          minInput[kcol] = Math.min(minInput[kcol],inputValues[krow][kcol]);
        }

        for (int kcol=0; kcol<outputs;kcol++) {
          maxOutput[kcol] = Math.max(maxOutput[kcol],outputValues[krow][kcol]);
          minOutput[kcol] = Math.min(minOutput[kcol],outputValues[krow][kcol]);
        }

        count[krow] = (double)krow;
        order[krow] = krow;
      }
      randomize();
    }
    catch(Exception ex) {
      System.out.println(ex.getMessage());
      ex.printStackTrace();
    }
```

```
}


//Normalizes the database given just the scale

  public void normalize (double inputScaleMin,  double inputScaleMax,
                         double outputScaleMin, double outputScaleMax) {
    this.inputScaleMax  = inputScaleMax;
    this.inputScaleMin  = inputScaleMin;
    this.outputScaleMax = outputScaleMax;
    this.outputScaleMin = outputScaleMin;

  //Apply the scale such that the minm and maxm values become equal to scaleMinm and
ScaleMaxm.
    for (int i=0; i<rows; i++)  {

     for (int j=0; j<inputs; j++) {
       if (maxInput[j] > minInput[j]) {
         scaledInputValues[i][j] = inputScaleMin + (inputScaleMax-inputScaleMin)
                                 * (inputValues[i][j] - minInput[j]) / (maxInput[j] - minInput[j]);
       }
       else {
         scaledInputValues[i][j] = (inputScaleMax + inputScaleMin)/2.0;
       }
     }

     for (int j=0; j<outputs; j++) {
       if (maxOutput[j] > minOutput[j]) {
         scaledOutputValues[i][j]   = outputScaleMin + (outputScaleMax-outputScaleMin)
                                    *  (outputValues[i][j]  -  minOutput[j])  /  (maxOutput[j]  -
minOutput[j]);
         scaledPredictValues[i][j]= outputScaleMin + (outputScaleMax-outputScaleMin)
                                    *  (predictValues[i][j]  -  minOutput[j])  /  (maxOutput[j]  -
minOutput[j]);
       }
       else {
         scaledOutputValues[i][j]  = (outputScaleMax + outputScaleMin)/2.0;
         scaledPredictValues[i][j] = (outputScaleMax + outputScaleMin)/2.0;
       }
     }
    }
  }


  public void normalize (double inputScaleMin,  double inputScaleMax) {
    this.inputScaleMax  = inputScaleMax;
    this.inputScaleMin  = inputScaleMin;

  //Apply the scale such that the minm and maxm values become equal to scaleMinm and
ScaleMaxm.
    for (int i=0; i<rows; i++)  {

     for (int j=0; j<inputs; j++) {
       if (maxInput[j] > minInput[j]) {
         scaledInputValues[i][j] = inputScaleMin + (inputScaleMax-inputScaleMin)
```

```
                                * (inputValues[i][j] - minInput[j]) / (maxInput[j] - minInput[j]);
        }
        else {
          scaledInputValues[i][j] = (inputScaleMax + inputScaleMin)/2.0;
        }
      }
    }
  }

// Shuffles the values of a dataset and places the shuffled indices in the array mixCheck
  public void randomize() {
    sequential();

    for (int i =0; i<rows ; i++) {
      int j   = (int)(Math.random()*rows);
      int k   = order[i];
      order[i] = order[j];
      order[j] = k;
    }
  }

  public void randomize(int partial)  {
    for (int i =0; i<partial ; i++) {
      int j   = (int)(Math.random()*partial);
      int k   = order[i];
      order[i] = order[j];
      order[j] = k;
    }
  }

// Orders the data sequentially
  public void sequential() {
    for (int i =0; i<rows ; i++) {
      order[i] = i;
    }
  }

// This overloaded version of the randomize method shuffles a dataset with a given shuffling order

  public void randomize(int[] givenOrder) {

    for (int i =0; i<rows ; i++) {
      order[i] = givenOrder[i];
    }
  }

  public double[] getInputRow  (int i) {return inputValues  [i];}
  public double[] getOutputRow (int i) {return outputValues [i];}
  public double[] getPredictRow(int i) {return predictValues[i];}

  public double[] getRandomInputRow  (int i) {return inputValues  [order[i]];}
```

```
public double[] getRandomOutputRow (int i) {return outputValues [order[i]];}
public double[] getRandomPredictRow(int i) {return predictValues[order[i]];}

public double[] getScaledInputRow  (int i) {return scaledInputValues  [i];}
public double[] getScaledOutputRow (int i) {return scaledOutputValues [i];}
public double[] getScaledPredictRow(int i) {return scaledPredictValues[i];}

public double[] getScaledRandomInputRow  (int i) {return scaledInputValues  [order[i]];}
public double[] getScaledRandomOutputRow (int i) {return scaledOutputValues [order[i]];}
public double[] getScaledRandomPredictRow(int i) {return scaledPredictValues[order[i]];}

public double[] getInputCol  (int i) {
   double[] d = new double[rows];
   for (int j=0; j<rows; j++) {
      d[j] = inputValues  [j][i];
   }
   return d;
}

public double[] getOutputCol (int i) {
   double[] d = new double[rows];
   for (int j=0; j<rows; j++) {
      d[j] = outputValues  [j][i];
   }
   return d;
}

public double[] getPredictCol(int i) {
 double[] d = new double[rows];
   for (int j=0; j<rows; j++) {
      d[j] = predictValues  [j][i];
   }
   return d;
}

// Print debug of the data
public void print() {

   System.out.println("=== DataSet: "+name+" ===");
   System.out.println("Rows = "+rows+" Inputs = "+inputs+" Outputs = "+outputs);
   if (rows > 0 ) {
      for (int i =0; i<rows ; i++) {

         for (int j =0; j<inputs ; j++) {
            if (j==0) System.out.print("Row ["+i+"] ");
            System.out.print(","+inputValues[i][j]);
         }

         for (int j =0; j<outputs ; j++) {
            System.out.print(","+outputValues[i][j]);
         }

         System.out.println("");
      }
   }
}
```

```
public void printScales() {
   System.out.println("=== DataSet: "+name+" Scales===");
   System.out.println("Rows = "+rows+" Inputs = "+inputs+" Outputs = "+outputs);

   System.out.println("Input Scale Maximum, Minimum "+inputScaleMax+","+inputScaleMin);
   for (int i =0; i<inputs ; i++) {
      System.out.println("Input "+i+" Maximum, Minimum = "+maxInput[i]+","+minInput[i]);
   }

   System.out.println("Output Scale Maximum, Minimum
"+outputScaleMax+","+outputScaleMin);
   for (int i =0; i<outputs ; i++) {
      System.out.println("Output "+i+" Maximum, Minimum = "+maxOutput[i]+","+minOutput[i]);
   }

}

// Print randomized data

   public void printRandom() {
      System.out.println("=== Randomized DataSet: "+name+" ===");
      System.out.println("Rows = "+rows+" Inputs = "+inputs+" Outputs = "+outputs);
      if (rows > 0 ) {
         for (int i =0; i<rows ; i++) {

            for (int j =0; j<inputs ; j++) {
               if (j==0) System.out.print("Row ["+order[i]+"] ");
               System.out.print(","+inputValues[order[i]][j]);
            }

            for (int j =0; j<outputs ; j++) {
               System.out.print(","+outputValues[order[i]][j]);
            }
            System.out.println("");
         }
      }
   }

// Print scaled data

   public void printScaled() {
      System.out.println("=== Scaled DataSet: "+name+" ===");
      System.out.println("Rows = "+rows+" Inputs = "+inputs+" Outputs = "+outputs);
      if (rows > 0 ) {
         for (int i =0; i<rows ; i++) {

            for (int j =0; j<inputs ; j++) {
               if (j==0) System.out.print("Row ["+i+"] ");
               System.out.print(","+scaledInputValues[i][j]);
            }

            for (int j =0; j<outputs ; j++) {
```

```
            System.out.print(","+scaledOutputValues[i][j]);
          }

          System.out.println("");
        }
      }
    }


    public void unscalePrediction () {
      for (int i=0; i<rows; i++)  {
      for (int j=0; j<outputs; j++) {
          if (maxOutput[j] > minOutput[j]) {
            predictValues[i][j] =   minOutput[j]  + (scaledPredictValues[i][j] - outputScaleMin)
                  / (outputScaleMax-outputScaleMin)
                  * (maxOutput[j] - minOutput[j]) ;
        }
        else {
          predictValues[i][j] = (maxOutput[j] + minOutput[j])/2.0;
        }
       }
      }
    }


}
```

**//Program # 2**

**//DataSetFactory.Java**

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.sql.*;
import java.util.Date;

/** The DataSetFactory class is the factory to create
 * different type of dataset based on the input data

**/

public class DataSetFactory{

    //Construtor
    public DataSetFactory(){}


    public static double[][] createInputSet(Effect effect,String[][] data) {
            int monthEffect=0;
            int weekdayEffect=0;
            int weekendEffect=0;
            int hourEffect=0;
            int tempratureEffect=0;
            int windEffect=0;

            if(effect.getMonthEffect()) monthEffect=1;
            if(effect.getWeekdayEffect()) weekdayEffect=1;
            if(effect.getWeekendEffect()) weekendEffect=1 ;
            if(effect.getHourEffect()) hourEffect=1;
            if(effect.getTempratureEffect()) tempratureEffect=1;
            if(effect.getWindEffect()) windEffect=1;

            int                                          columns
=
monthEffect+weekdayEffect+weekendEffect+hourEffect+tempratureEffect+windEffect;
            double[][] input=new double[data.length][columns];
            Calendar calendar = new GregorianCalendar();
            calendar.setTimeZone(TimeZone.getDefault());
            SimpleDateFormat formatter = new SimpleDateFormat ("MM-dd-yyyy",Locale.US);
            formatter.setTimeZone(TimeZone.getDefault());


            for(int i=0;i<data.length;i++){
                    ParsePosition pos = new ParsePosition(0);
        Date startDate = formatter.parse(data[i][0], pos);
                    calendar.setTime(startDate);
                    if(monthEffect==1)
                            input[i][0]=calendar.get(Calendar.MONTH);
                    if(weekdayEffect==1)
                            input[i][monthEffect]=calendar.get(Calendar.DAY_OF_WEEK);
                     if(weekendEffect==1){
```

```
    if(calendar.get(Calendar.DAY_OF_WEEK)==1||calendar.get(Calendar.DAY_OF_WEEK)==7)
                                        input[i][monthEffect+weekdayEffect] = 1;
                                else
                                        input[i][monthEffect+weekdayEffect] = 0;
                        }
                        if(hourEffect==1)
                                input[i][monthEffect+weekdayEffect+weekendEffect]                    =
Integer.valueOf(data[i][1]).intValue();

                        if(tempratureEffect==1){
                                input[i][monthEffect+weekdayEffect+weekendEffect+hourEffect]
=Double.valueOf(data[i][2]).doubleValue();
                        }
                        if(windEffect==1){

    input[i][monthEffect+weekdayEffect+weekendEffect+hourEffect+tempratureEffect]                    =
Double.valueOf(data[i][3]).doubleValue();
                        }
                }
                return input;
        }



    public static double[][] createOutputSet(String[][] data) {
                double[][] output=new double[data.length][1];
                for(int i=0;i<data.length;i++){
                        output[i][0] = Double.valueOf(data[i][4]).doubleValue();
                }
                return output;
        }

    public static double[] createIndexSet(String[][] data) {
                double[] index=new double[data.length];
                for(int i=0;i<data.length;i++){
                        index[i] = i;
                }
                return index;
        }


    public static double[] createTimeSet(String[][] data) {
                double[] timeSet = new double[data.length];
                Calendar calendar = new GregorianCalendar();
                calendar.setTimeZone(TimeZone.getDefault());
                SimpleDateFormat formatter = new SimpleDateFormat ("MM-dd-yyyy HH",Locale.US);
                formatter.setTimeZone(TimeZone.getDefault());

                for(int i=0;i<data.length;i++){
                        ParsePosition pos = new ParsePosition(0);
            Date startDate = formatter.parse(data[i][0]+" "+data[i][1], pos);
                        timeSet[i]=startDate.getTime()/1000;
                }
                return timeSet;
        }
```

```
    public static String[] getTimeString(String[][] data) {
            String[]  timeString = new String[data.length];
            for(int i=0;i<data.length;i++){
        timeString[i] = data[i][0]+" "+data[i][1]+":00";
            }
            return timeString;
    }


}
```

**// Program # 3**

**//Effect.Java**

```
/** Tthe class defines the Effects and Network paramaters

**/

public class Effect {

    //Effects
    boolean weekdayEffect=true;
    boolean hourEffect=true;
    boolean tempratureEffect=true;
    boolean windEffect=true;
    boolean monthEffect=true;
    boolean weekendEffect=true;

    //Network Paramaters
    double rate       = 0.8;
    double alpha      = 0.1;
    double sample     = 1.;
    int   epochs      = 5000;
    int   nodes       = 10;
    int   type        = 0;

    //Constructor
    public Effect(){
    }

    public void setWeekdayEffect(boolean effect){      weekdayEffect=effect;     }
    public boolean getWeekdayEffect()                  {       return weekdayEffect;     }

    public void setWeekendEffect(boolean effect){      weekendEffect=effect;     }
    public boolean getWeekendEffect()                  {       return weekendEffect;     }

    public void setHourEffect(boolean effect)    {     hourEffect=effect;        }
    public boolean getHourEffect()                             {       return hourEffect;}

    public void setMonthEffect(boolean effect)  {      monthEffect=effect;       }
    public boolean getMonthEffect()                            {       return monthEffect;
    }

    public void setTempratureEffect(boolean effect){tempratureEffect=effect;       }
    public boolean getTempratureEffect()               {       return tempratureEffect;  }

    public void setWindEffect(boolean effect)    {     windEffect=effect;        }
    public boolean getWindEffect()                             {       return windEffect;
    }


    public double getRate()       {return rate;}
    public void setRate(double value){ rate=value;}
```

```
    public double getAlpha(){return alpha;        }
    public void setAlpha (double value){ alpha=value;}

    public double getSample() {return sample;}
    public void setSample(double value){ sample=value;}

    public int    getEpochs(){ return epochs;}
    public void setEpochs(int value){ epochs=value;}

    public int    getNodes(){ return nodes;}
    public void setNodes(int value){ nodes=value;}

    public int    getType() { return type;}
    public void setType(int value){ type=value;}



    public static void main(String[] args)
    {
            new Effect();
    }
}
```

**// Program # 4**

**//EffectSetup.Java**

```
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.text.*;

/** The EffectSetup class is the GUI to set up the Effect

*/

public class EffectSetup extends JFrame implements ActionListener{

    Effect effect;
    static EffectSetup instance=null;

    private JCheckBox monthEffectCheckBox          = new JCheckBox("Month-Of-Year");
    private JCheckBox weekdayEffectCheckBox        = new JCheckBox("Weekday");
    private JCheckBox hourEffectCheckBox           = new JCheckBox("Hour-Of-Day");
    private JCheckBox weekendEffectCheckBox        = new JCheckBox("Weekend");
    private JCheckBox tempratureEffectCheckBox     = new JCheckBox("Temperature");
    private JCheckBox windEffectCheckBox= new JCheckBox("Wind Velocity");

    private JButton okButton     = new JButton("Ok");
    private JButton cancelButton = new JButton("Cancel");


    /**
            * Constructor
    */
  EffectSetup( Effect effect){
            this.effect=effect;
            Insets insl = new Insets(5,5,5,5);
            Insets ins = new Insets(0,0,0,0);

            setTitle("Effects Setup");
            Font font = new Font ("Dialog", Font.BOLD, 12);

            weekdayEffectCheckBox.setSelected(effect.getWeekdayEffect());
            weekendEffectCheckBox.setSelected(effect.getWeekendEffect());
            monthEffectCheckBox.setSelected(effect.getMonthEffect());
            tempratureEffectCheckBox.setSelected(effect.getTempratureEffect());
            hourEffectCheckBox.setSelected(effect.getHourEffect());
            windEffectCheckBox.setSelected(effect.getWindEffect());

            JPanel effectPane = new JPanel(new GridBagLayout());
            effectPane.setBorder(new TitledBorder(""));
```

```
        MyUtility.makeGbComp(effectPane,tempratureEffectCheckBox
,MyUtility.WEST,MyUtility.NONE,0,0,1,1,1.,0.,ins);

    MyUtility.makeGbComp(effectPane,windEffectCheckBox,MyUtility.WEST,MyUtility.NONE,0,1
,1,1,1.,0.,ins);

    MyUtility.makeGbComp(effectPane,hourEffectCheckBox,MyUtility.WEST,MyUtility.NONE,0,2
,1,1,1.,0.,ins);

    MyUtility.makeGbComp(effectPane,weekdayEffectCheckBox,MyUtility.WEST,MyUtility.NON
E,0,3,1,1,1.,0.,ins);

    MyUtility.makeGbComp(effectPane,weekendEffectCheckBox,MyUtility.WEST,MyUtility.NON
E,0,4,1,1,1.,0.,ins);
        MyUtility.makeGbComp(effectPane,monthEffectCheckBox,
MyUtility.WEST,MyUtility.NONE,0,5,1,1,1.,0.,ins);


        JPanel buttonPane = new JPanel();
        buttonPane.add(okButton);
        buttonPane.add(cancelButton);

        JLabel titleLabel=new JLabel("Effects Setup");
        titleLabel.setFont(new Font ("SansSerif", Font.BOLD, 20));
        JPanel wholePane = new JPanel(new GridBagLayout());


    MyUtility.makeGbComp(wholePane,titleLabel,MyUtility.C,MyUtility.NONE,0,0,1,1,0.,0.,ins1);

    MyUtility.makeGbComp(wholePane,effectPane,MyUtility.C,MyUtility.BOTH,0,1,1,1,1.,0.,ins1);

    MyUtility.makeGbComp(wholePane,buttonPane,MyUtility.C,MyUtility.BOTH,0,2,1,1,1.,0.,ins1);
        okButton.addActionListener(this);
        cancelButton.addActionListener(this);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
            }
        });
        getContentPane().add(wholePane);
        setSize(300,300);
        MyUtility.centerWindow(this);
        setVisible(true);
}


 /**
 * Invoked when an action occurs
 */


 public void actionPerformed(ActionEvent evt){
        Object obj = evt.getSource();
        if (obj == cancelButton){   //close the window
            dispose();
        }
```

```
            if (obj == okButton){
                    if(setParamater()) dispose();
            }
    }


    public boolean setParamater(){
            int nodes=0;
            int epochs=0;
            double rate=0;
            double alpha=0;
            int type=1;

            effect.setWeekdayEffect(weekdayEffectCheckBox.isSelected());
            effect.setWeekendEffect(weekendEffectCheckBox.isSelected());
            effect.setMonthEffect(monthEffectCheckBox.isSelected());
            effect.setTempratureEffect(tempratureEffectCheckBox.isSelected());
            effect.setHourEffect(hourEffectCheckBox.isSelected());
            effect.setWindEffect(windEffectCheckBox.isSelected());
            return true;
    }

    /**
            *Stand alone application entry point
    */
   public static void main(String [] arg){
     new EffectSetup(new Effect());
   }


}
```

**//Program # 5**

**//Forcaster.Java**

```
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.text.*;
import java.util.Date;
import java.math.*;


/** The Forecaster class is the forecast engine

**/

public class Forecaster {

    Effect effect;
    NeuralNetwork nn;

    String[][] trainingSet;
    double[] trainingActual;
    double[] trainingResult;

    String[][] testSet;
    double[] testActual;
    double[] testResult;

    String[][] predictSet;
    double[] predictResult;

    double[] trainingTime;
    double[] testTime;
    double[] predictTime;

    String [] trainingTimeString;
    String [] testTimeString;
    String [] predictTimeString;

    double tolerant = 0.0005;

    public void setTrainingSet(String[][] dataSet){
            trainingSet=dataSet;
    }

    public void setTestSet(String[][] dataSet ) {
            testSet=dataSet;
    }

    public void setPredictSet(String[][] dataSet ) {
            predictSet=dataSet;
```

```
}

public Forecaster(Effect effect){
        this.effect=effect;
}

public Forecaster(Effect effect, String[][] dataSet) {
        this.effect=effect;
        this.trainingSet=dataSet;
}


void training() throws Exception{
                double      rate = effect.getRate();
                double     alpha = effect.getAlpha();
                double    sample = effect.getSample();
                int         epochs = effect.getEpochs();
                int          nodes = effect.getNodes();
                int                type=effect.getType();

                double[][] inputValues =DataSetFactory.createInputSet(effect,trainingSet);
                double[][] outputValues=DataSetFactory.createOutputSet(trainingSet);
                double[] indexValues =DataSetFactory.createIndexSet(trainingSet);
                trainingTime  =DataSetFactory.createTimeSet(trainingSet);


                trainingTimeString  =DataSetFactory.getTimeString(trainingSet);


                DataSet ds=new DataSet(indexValues,trainingTime,inputValues,outputValues);
                ds.normalize(0,1,0,1);

                nn=new NeuralNetwork(ds,type,nodes,epochs,rate,alpha,tolerant);

                nn.train();
                trainingResult=ds.getPredictCol(0);

                trainingActual =new double[outputValues.length];
                for(int i=0;i<outputValues.length;i++)
                        trainingActual[i]=outputValues[i][0];

}

 void testing() throws Exception{
                double[][] inputValues =DataSetFactory.createInputSet(effect,testSet);
                double[][] outputValues=DataSetFactory.createOutputSet(testSet);
                double[] indexValues =DataSetFactory.createIndexSet(testSet);
                testTime  =DataSetFactory.createTimeSet(testSet);
                testTimeString  =DataSetFactory.getTimeString(testSet);


                DataSet ds=new DataSet(indexValues,testTime,inputValues,outputValues);

                ds.normalize(0,1,0,1);
                nn.setDataSet(ds);
                nn.predict();
```

```
                testResult=ds.getPredictCol(0);

                testActual =new double[outputValues.length];
                for(int i=0;i<outputValues.length;i++)
                        testActual[i]=outputValues[i][0];
    }

    void predict() throws Exception{
                double[][] inputValues =DataSetFactory.createInputSet(effect,predictSet);
                double[] indexValues =DataSetFactory.createIndexSet(predictSet);
                predictTime  =DataSetFactory.createTimeSet(predictSet);
                predictTimeString  =DataSetFactory.getTimeString(predictSet);


                double[][] outputValues= new double[inputValues.length][1];

                DataSet ds=new DataSet(indexValues,predictTime,inputValues,outputValues);

                ds.normalize(0,1,0,1);
                nn.setDataSet(ds);
                nn.predict();
                predictResult=ds.getPredictCol(0);
                for (int i=0;i<predictResult.length;i++ )
                {
                        System.out.println("i="+predictResult[i]);
                }
    }


    void displayTraining() throws Exception{
            if(trainingResult ==null) throw new Exception("Network not trained");
            String title="Training Results";
            String label="Training Load";

            new
GraphicDisplay(title,label,trainingTimeString,trainingTime,trainingActual,trainingResult);
    }

    void displayTesting() throws Exception{
            if(testResult ==null) throw new Exception("Network not predicted");
            String title="Testing Results";
            String label="Predicted Load";
            new GraphicDisplay(title,label,testTimeString,testTime,testActual,testResult);
    }


    void displayPredicted() throws Exception{
            if(predictResult ==null) throw new Exception("Network not predicted");
            String title="Predicted Load";
            String label="Predcited Data";
            new GraphicDisplay(title,label,predictTimeString,predictTime,predictResult);
    }

}
```

**//Program # 6**

**//GraphicDisplay.Java**

```
import java.awt.*;
import java.awt.Color;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.table.*;
import java.text.*;
import java.sql.*;
import java.util.Date;
import java.math.*;

/** The GraphicDisplay class is the GUI to display result

*/

public class GraphicDisplay extends JFrame implements ActionListener{
    private String appID = "TddReporter";
    final double DISTANCE=60*60*24./5;                    // For

    private JButton viewButton          = new JButton("Tabular View");
    private JButton cancelButton        = new JButton("Close");

    String   start =null;
    String   end = null;
   String  xlabel = "Time";
   String  ylabel = "Load";

    Object[][] data=null;
    double   [] x=null;
    double  [] x1=null;
    double  [] x2=null;
    double  [] y1=null;
    double  [] y2=null;

    String  title=null;;
    String label=null;
    Vector display=new Vector();
    JPanel plotPane;
    Object[] columnNames=null;
    PlotAxis plotAxis;
    String timeString[];
     /** Construct a TddReporter
     */
      public GraphicDisplay (String title, String label, String [] timeString, double x[],double
y1[],double y2[] ) {
            this.timeString=timeString;
            this.x=x;
            this.y1=y1;
```

```
            this.y2=y2;
            this.title = title;
            this.label = label;

            setTitle(title);
            Font font = new Font ("Dialog", Font.BOLD, 12);

            plotPane =new JPanel(new BorderLayout());
            JPanel root = new JPanel(new GridBagLayout());
            Insets ins = new Insets(0,0,0,0);
            Insets ins1 = new Insets(10,0,0,0);
            Insets ins2 = new Insets(5,10,5,10);

            JLabel subTitleLabel = new JLabel(title);
            subTitleLabel.setFont(new Font("SansSerif", Font.BOLD, 22));
            MyUtility.makeGbComp(root,
subTitleLabel,MyUtility.C,MyUtility.NONE,0,1,1,1,0.,0.,ins2);


            JPanel control = new JPanel(new GridBagLayout());
            control.setBorder(new TitledBorder(""));

            plotPane.setBackground(Color.white);

            JPanel command = new JPanel(new GridBagLayout());
            command.setBorder(new TitledBorder(""));


   MyUtility.makeGbComp(command,viewButton,MyUtility.WEST,MyUtility.NONE,0,0,1,1,.0,.0,
new Insets(0,5,0,5));

   MyUtility.makeGbComp(command,cancelButton,MyUtility.WEST,MyUtility.NONE,3,0,1,1,.0,.0
,new Insets(0,5,0,5));


            JPanel legendPane=new JPanel(new GridBagLayout());

            JLabel legend1 = new JLabel("_____ Actual Load ");
            legend1.setFont(new Font("Serif", Font.ITALIC, 12));
            legend1.setForeground(Color.black);
            JLabel legend2 = new JLabel("_____ "+label);
            legend2.setFont(new Font("Serif", Font.ITALIC, 12));
            legend2.setForeground(Color.red);
            MyUtility.makeGbComp(legendPane,                    legend1,MyUtility.CENTER,
MyUtility.NONE,0,0,1,1,0.,.0,ins2);
            MyUtility.makeGbComp(legendPane,        legend2,        MyUtility.CENTER,
MyUtility.NONE,1,0,1,1,0.,.0,ins2);


            MyUtility.makeGbComp(root,          plotPane,           MyUtility.CENTER,
MyUtility.BOTH,0,3,1,1,1.,1.0,ins2);

            MyUtility.makeGbComp(root,          legendPane,         MyUtility.CENTER,
MyUtility.NONE,0,4,1,1,0.,.0,ins2);
            MyUtility.makeGbComp(root,                      command,MyUtility.CENTER,
MyUtility.NONE,0,5,1,1,0.,.0,ins2);
```

```
            plotChart("",x,y1,y2);
            cancelButton.addActionListener(this);
            viewButton.addActionListener(this);
            getContentPane().add(root);
            pack();
            setVisible(true);
            MyUtility.centerWindow(this);
    }

    public GraphicDisplay (String title, String label, String [] timeString, double x[],double y1[] ) {
            this.timeString=timeString;
            this.x=x;
            this.y1=y1;
            this.title = title;
            this.label = label;

            setTitle(title);
            Font font = new Font ("Dialog", Font.BOLD, 12);

            plotPane =new JPanel(new BorderLayout());
            JPanel root = new JPanel(new GridBagLayout());
            Insets ins = new Insets(0,0,0,0);
            Insets ins1 = new Insets(10,0,0,0);
            Insets ins2 = new Insets(5,10,5,10);

            JLabel subTitleLabel = new JLabel(title);
            subTitleLabel.setFont(new Font("SansSerif", Font.BOLD, 22));
            MyUtility.makeGbComp(root,
subTitleLabel,MyUtility.C,MyUtility.NONE,0,1,1,1,0.,0.,ins2);


            JPanel control = new JPanel(new GridBagLayout());
            control.setBorder(new TitledBorder(""));

            plotPane.setBackground(Color.white);

            JPanel command = new JPanel(new GridBagLayout());
            command.setBorder(new TitledBorder(""));


    MyUtility.makeGbComp(command,viewButton,MyUtility.WEST,MyUtility.NONE,0,0,1,1,.0,.0,
new Insets(0,5,0,5));

    MyUtility.makeGbComp(command,cancelButton,MyUtility.WEST,MyUtility.NONE,3,0,1,1,.0,.0
,new Insets(0,5,0,5));


            JPanel legendPane=new JPanel(new GridBagLayout());

            JLabel legend1 = new JLabel("_____ Actual Data ");
            legend1.setFont(new Font("Serif", Font.ITALIC, 12));
            legend1.setForeground(Color.black);
            JLabel legend2 = new JLabel("_____ "+label);
            legend2.setFont(new Font("Serif", Font.ITALIC, 12));
            legend2.setForeground(Color.red);
```

```
        MyUtility.makeGbComp(legendPane, legend1,MyUtility.CENTER,
MyUtility.NONE,0,0,1,1,0.,.0,ins2);
        MyUtility.makeGbComp(legendPane, legend2, MyUtility.CENTER,
MyUtility.NONE,1,0,1,1,0.,.0,ins2);


        MyUtility.makeGbComp(root, plotPane, MyUtility.CENTER,
MyUtility.BOTH,0,3,1,1,1.,1.0,ins2);

        MyUtility.makeGbComp(root, legendPane, MyUtility.CENTER,
MyUtility.NONE,0,4,1,1,0.,.0,ins2);
        MyUtility.makeGbComp(root, command,MyUtility.CENTER,
MyUtility.NONE,0,5,1,1,0.,.0,ins2);
        plotChart("",x,y1);
        cancelButton.addActionListener(this);
        viewButton.addActionListener(this);
        getContentPane().add(root);
        pack();
        setVisible(true);
        MyUtility.centerWindow(this);
    }


/**
        Implement the ActionLister interface
    */

    public void actionPerformed(ActionEvent e) {
        String s = e.getActionCommand();
        if (s.equals("Close")){
            try{
                if(display!=null)
                    for(int i=0;i<display.size();i++)
                        if (display.elementAt(i)!=null)
                            ((JFrame)display.elementAt(i)).dispose();
                dispose();

            }
            catch(Exception ex){}
         }
         if (s.equals("Tabular View")){
            setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
            Object[][] data =new Object[x.length][4];

            double totalError=0;
            for( int i=0;i<data.length;i++){
                data[i][0]=timeString[i];
                data[i][1]=""+(int)y1[i];
                data[i][2]=""+(int)y2[i];
                double error=(y2[i]-y1[i])*100/y1[i];
                data[i][3]=""+(int)error;
                totalError+=Math.abs(error);
            }
```

```
                System.out.println("Average error= "+totalError/data.length);

                Object[] columnNames={"Time","Actual Load",label," Error%"};
                new TableView(title,data,columnNames);
                setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
        }

}



public void plotCurveChart(String title) {
try {
                Plot plot  = new Plot(x1, true);
                PlotSeries ps = plot.addSeries(y1);
                ps.setSymbol(false);
                ps.setColor(Color.black);
                plotAxis = new PlotAxis(plot,title ,xlabel ,ylabel);

                        plotPane.removeAll();
                        plotPane.add(plotAxis);
                        plotPane.validate();
  }
  catch(Exception ex){
    System.out.println(ex.getMessage());
    ex.printStackTrace();
  }
 }


public void plotChart(String title,double[]x,double[] y1,double[]y2) {
try {
                    Plot2 p  = new Plot2(x,true);
                    if(x!=null && x.length>0){
       PlotSeries ps1 = p.addSeries(x,y1);
       ps1.setSymbolSize(2);
       ps1.setColor(Color.black);
                    }
                        if(x!=null && x.length>0){
      PlotSeries ps2 = p.addSeries(x,y2);
      ps2.setSymbolSize(2);
      ps2.setColor(Color.red);
                        }

                plotAxis = new PlotAxis(p,title ,xlabel ,ylabel);

                        plotPane.removeAll();
                        plotPane.add(plotAxis);
                        plotPane.validate();
   }
   catch(Exception ex){
     System.out.println(ex.getMessage());
```

```
    ex.printStackTrace();
  }
 }

public void plotChart(String title,double[]x,double[] y1) {
try {
                    Plot2 p  = new Plot2(x,true);
                            if(x!=null && x.length>0){
      PlotSeries ps2 = p.addSeries(x,y1);
      ps2.setSymbolSize(2);
      ps2.setColor(Color.red);
                            }

                plotAxis = new PlotAxis(p,title ,xlabel ,ylabel);

                            plotPane.removeAll();
                            plotPane.add(plotAxis);
                            plotPane.validate();
  }
  catch(Exception ex){
    System.out.println(ex.getMessage());
    ex.printStackTrace();
  }
 }


public void plotBarChart(String title) {
try {
                            Plot2 p  = new Plot2(x,true);
                        if(x!=null && x.length>0){
      PlotSeries ps1 = p.addSeries(x,y1);
      ps1.setSymbolSize(2);
                            ps1.setBarChart(true);
      ps1.setColor(Color.black);
                    }
                            if(x!=null && x.length>0){
      PlotSeries ps2 = p.addSeries(x,y2);
                                    ps2.setBarChart(true);
      ps2.setSymbolSize(2);
      ps2.setColor(Color.red);
                            }

                plotAxis = new PlotAxis(p,title ,xlabel ,ylabel);

                            plotPane.removeAll();
                            plotPane.add(plotAxis);
                            plotPane.validate();
  }
  catch(Exception ex){
    System.out.println(ex.getMessage());
    ex.printStackTrace();
  }
 }
```

```
    public void nodata(){
                    plotPane.removeAll();
                plotPane.setBackground(Color.white);
                    plotPane.validate();
                    JOptionPane.showMessageDialog(this, "There is no data available for this time
period.",
                    "Warning Message", JOptionPane.WARNING_MESSAGE);
                    repaint();
    }



    /*Stand alone application entry point
            */
    public static void main(String [] arg){
            String host="localhost";
            if(arg.length>0) host= arg[0];
     // new GraphicDisplay();
    }

}
```

**//Program # 7**

**//MyUtility.Java**

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.table.*;
import javax.swing.text.*;
import java.text.*;

/** The MyUtility is utility class
*/

public class MyUtility{
    public static final int FONTSIZE = 10;        //these const could be put into a interface and let this
class
    public static final int PAGEWIDTH = 1024;//implement it and share it with other classes
    public static final int PAGEROWS = 35;
    public static final int XINITIAL = 30;
    public static final int YINITIAL = 30;


    public static final int CENTER              = GridBagConstraints.CENTER;
    public static final int WEST                = GridBagConstraints.WEST;
    public static final int SOUTHWEST           = GridBagConstraints.SOUTHWEST;
    public static final int SOUTH               = GridBagConstraints.SOUTH;
    public static final int SOUTHEAST= GridBagConstraints.SOUTHEAST;
    public static final int EAST                = GridBagConstraints.EAST;
    public static final int NORTHEAST           = GridBagConstraints.NORTHEAST;
    public static final int NORTH               = GridBagConstraints.NORTH;
    public static final int NORTHWEST           = GridBagConstraints.NORTHWEST;
    public static final int BOTH                = GridBagConstraints.BOTH;
    public static final int NONE                = GridBagConstraints.NONE;
    public static final int HORIZONTAL          = GridBagConstraints.HORIZONTAL;
    public static final int VERTICAL   = GridBagConstraints.VERTICAL;
    public static final int REMAINDER           = GridBagConstraints.REMAINDER;
    public static final int RELATIVE   = GridBagConstraints.RELATIVE;

    public static final int C = GridBagConstraints.CENTER;
    public static final int W = GridBagConstraints.WEST;
    public static final int SW= GridBagConstraints.SOUTHWEST;
    public static final int S = GridBagConstraints.SOUTH;
    public static final int SE= GridBagConstraints.SOUTHEAST;
    public static final int E = GridBagConstraints.EAST;
    public static final int NE= GridBagConstraints.NORTHEAST;
    public static final int N = GridBagConstraints.NORTH;
    public static final int NW= GridBagConstraints.NORTHWEST;
    public static final int B= GridBagConstraints.BOTH;
    public static final int H= GridBagConstraints.HORIZONTAL;
    public static final int V     = GridBagConstraints.VERTICAL;

    public static Color[] colors = {
```

```
            Color.red, Color.black, Color.blue, Color.cyan, Color.gray,
            Color.orange, Color.pink, Color.yellow, Color.white, Color.lightGray,
            Color.darkGray, Color.green, Color.magenta};

    /** an array of string to represent the color
    */
    public static String[] colorStrings = {
                        "red", "black", "blue", "cyan", "gray",
                        "orange", "pink", "yellow", "white", "lightgray",
                        "darkgray", "green", "magenta"};


    /** convenient method to put component into gridbag layout
    */
    public static void makeGbComp(Container cont, Component comp, int x, int y, int w, int h,
            double weightx, double weighty, Insets ins){

            GridBagLayout gbl = (GridBagLayout) cont.getLayout();
            GridBagConstraints c = new GridBagConstraints();

            c.fill = GridBagConstraints.BOTH;
            c.gridx = x;
            c.gridy = y;
            c.gridwidth = w;
            c.gridheight = h;
            c.weightx = weightx;
            c.weighty = weighty;
            c.insets = ins;

            cont.add(comp);
            gbl.setConstraints(comp, c);
    }

    /** convenient method to put component into gridbag layout
    */
    public static void makeGbComp(Container cont, Component comp, int anchor, int fill, int x, int y,
int w, int h,
            double weightx, double weighty, Insets ins){

            GridBagLayout gbl = (GridBagLayout) cont.getLayout();
            GridBagConstraints c = new GridBagConstraints();

            c.fill = GridBagConstraints.BOTH;
            c.gridx = x;
            c.gridy = y;
            c.gridwidth = w;
            c.gridheight = h;
            c.weightx = weightx;
            c.weighty = weighty;
            c.insets = ins;
            c.anchor = anchor;
            c.fill = fill;

            cont.add(comp);
            gbl.setConstraints(comp, c);
    }
```

```
    /** convenient method to put component into gridbag layout
    */
    public static void makeGbComp(Container cont, Component comp, int anchor, int fill, int x, int y,
int w, int h,
            double weightx, double weighty, Insets ins, int ipadx, int ipady){

            GridBagLayout gbl = (GridBagLayout) cont.getLayout();
            GridBagConstraints c = new GridBagConstraints();

            c.fill = GridBagConstraints.BOTH;
            c.gridx = x;
            c.gridy = y;
            c.gridwidth = w;
            c.gridheight = h;
            c.weightx = weightx;
            c.weighty = weighty;
            c.insets = ins;
            c.anchor = anchor;
            c.fill = fill;
            c.ipadx = ipadx;
            c.ipady = ipady;

            cont.add(comp);
            gbl.setConstraints(comp, c);
    }

    /** convenient method to put component into gridbag layout
    */
    public static void makeGbComp(Container cont, Component comp, int x, int y, int w, int h,
            double weightx, double weighty){

            GridBagLayout gbl = (GridBagLayout) cont.getLayout();
            GridBagConstraints c = new GridBagConstraints();

            c.fill = GridBagConstraints.BOTH;
            c.gridx = x;
            c.gridy = y;
            c.gridwidth = w;
            c.gridheight = h;
            c.weightx = weightx;
            c.weighty = weighty;

            cont.add(comp);
            gbl.setConstraints(comp, c);
    }

    /** center the window on the screen
    */
    public static void centerWindow(Window window){
            int sW = Toolkit.getDefaultToolkit().getScreenSize().width;
            int sH= Toolkit.getDefaultToolkit().getScreenSize().height;
            int h = window.getSize().height;
            int w = window.getSize().width;
            window.setLocation(((int)sW/2)-((int)w/2),((int)sH/2)-((int)h/2));
```

```
}

public static String toDateString(java.sql.Timestamp ts){
        String s = ts.toString();
        String y = s.substring(0,4);
        String m = s.substring(5,7);
        String d = s.substring(8,10);
        return m+"/"+d+"/"+y;
}

public static String toTimeString(java.sql.Timestamp ts){
        String s = ts.toString();
        String h = s.substring(11,13);
        String n = s.substring(14,16);
        String sr = h+":"+n;
        if (sr.equals("00:00")) return "";
        else return sr;

}


public static Color getColor(String s){
        for (int i=0; i<colors.length; i++) {
                if (s.toLowerCase().equals(colorStrings[i])) return colors[i];
        }
        return Color.black;
}


/** get Color object based on index
*/
public static Color getColor(int i){
        if (i>=0 && i<colors.length) return colors[i];
                return Color.black;
}

/** a method return a string from a Timestamp based on a certain format
*/
public static String toString(java.sql.Timestamp ts){
        String s = ts.toString();
        String y = s.substring(0,4);
        String m = s.substring(5,7);
        String d = s.substring(8,10);
        String h = s.substring(11,13);
        String n = s.substring(14,16);
        return m+"/"+d+"/"+y+" "+h+":"+n;
}

}
```

**//Program # 8**

**//NetworkSetup.Java**

```
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.text.*;


/** The class NetworkSetup is the GUI to set up the network
paramaters

*/

public class NetworkSetup extends JFrame implements ActionListener{

   Effect effect;
   static NetworkSetup instance=null;
   private JTextField nodeText  = new JTextField(8);
   private JTextField rateText  = new JTextField(8);
   private JTextField alphaText = new JTextField(8);
   private JTextField epochsText= new JTextField(8);
   private JRadioButton linearRadio=new JRadioButton("linear");
   private JRadioButton nonlinearRadio=new
JRadioButton("nonlinear");
   private JButton okButton     = new JButton("Ok");
   private JButton cancelButton = new JButton("Cancel");

   /**
         * Constructor
   */
    NetworkSetup( Effect effect){
         this.effect=effect;
         Insets ins1 = new Insets(5,5,5,5);
         Insets ins = new Insets(0,0,0,0);

         setTitle("Network Setup");
         Font font = new Font ("Dialog", Font.BOLD, 12);

         ButtonGroup typeGroup = new ButtonGroup();
         typeGroup.add(linearRadio);
         typeGroup.add(nonlinearRadio);

         if(effect.getType()==0) linearRadio.setSelected(true);
         else nonlinearRadio.setSelected(true);

         nodeText.setText(""+effect.getNodes());
         epochsText.setText(""+effect.getEpochs());
```

```
        alphaText.setText(""+effect.getAlpha());
        rateText.setText(""+effect.getRate());


        JPanel parameterPane = new JPanel(new GridBagLayout());
        parameterPane.setBorder(new TitledBorder(""));
        MyUtility.makeGbComp(parameterPane,new JLabel("Hidden
Units: ") ,MyUtility.WEST,MyUtility.NONE,0,0,1,1,0.,0.,ins);

  MyUtility.makeGbComp(parameterPane,nodeText,MyUtility.WEST,MyUtil
ity.NONE,1,0,1,1,0.,0.,ins);
        MyUtility.makeGbComp(parameterPane,new JLabel("Learning
Rate: ") ,MyUtility.WEST,MyUtility.NONE,0,1,1,1,0.,0.,ins);

  MyUtility.makeGbComp(parameterPane,rateText,MyUtility.WEST,MyUtil
ity.NONE,1,1,1,1,0.,0.,ins);
        MyUtility.makeGbComp(parameterPane,new JLabel("Alpha:
"),MyUtility.WEST,MyUtility.NONE,0,2,1,1,0.,0.,ins);

  MyUtility.makeGbComp(parameterPane,alphaText,MyUtility.WEST,MyUti
lity.NONE,1,2,1,1,0.,0.,ins);
        MyUtility.makeGbComp(parameterPane,new JLabel("Epochs: ")
,MyUtility.WEST,MyUtility.NONE,0,3,1,1,0.,0.,ins);

  MyUtility.makeGbComp(parameterPane,epochsText,MyUtility.WEST,MyUt
ility.NONE,1,3,1,1,0.,0.,ins);
        MyUtility.makeGbComp(parameterPane,new JLabel("Output
Activation: ") ,MyUtility.WEST,MyUtility.NONE,0,4,2,1,0.,0.,ins);


  MyUtility.makeGbComp(parameterPane,linearRadio,MyUtility.WEST,MyU
tility.NONE,0,5,1,1,0.,0.,ins);

  MyUtility.makeGbComp(parameterPane,nonlinearRadio,MyUtility.WEST,
MyUtility.NONE,1,5,1,1,0.,0.,ins);

        JPanel buttonPane = new JPanel();
        buttonPane.add(okButton);
        buttonPane.add(cancelButton);

        JLabel titleLabel=new JLabel("Network Setup");
        titleLabel.setFont(new Font ("SansSerif", Font.BOLD, 20));
        JPanel wholePane = new JPanel(new GridBagLayout());


  MyUtility.makeGbComp(wholePane,titleLabel,MyUtility.C,MyUtility.N
ONE,0,0,1,1,0.,0.,ins1);

  MyUtility.makeGbComp(wholePane,parameterPane,MyUtility.C,MyUtilit
y.BOTH,0,1,1,1,1.,0.,ins1);

  MyUtility.makeGbComp(wholePane,buttonPane,MyUtility.C,MyUtility.B
OTH,0,2,1,1,1.,0.,ins1);
        okButton.addActionListener(this);
        cancelButton.addActionListener(this);
        addWindowListener(new WindowAdapter() {
              public void windowClosing(WindowEvent e) {
```

```
                dispose();
            }
        });
        getContentPane().add(wholePane);
        setSize(300,300);
        MyUtility.centerWindow(this);
        setVisible(true);
    }


/**
* Invoked when an action occurs
*/

public void actionPerformed(ActionEvent evt){
            Object obj = evt.getSource();
      if (obj == cancelButton){    //close the window
            dispose();
       }
       if (obj == okButton){
            if(setParamater()) dispose();
       }
}


 public boolean setParamater(){

       int nodes=0;
       int epochs=0;
       double rate=0;
       double alpha=0;

       try{
             nodes   =
Integer.valueOf(nodeText.getText()).intValue();
             epochs =
Integer.valueOf(epochsText.getText()).intValue();

  rate=Double.valueOf(rateText.getText()).doubleValue();

  alpha=Double.valueOf(alphaText.getText()).doubleValue();
             if(rate<0 ||alpha>1 ||alpha<0){
                  JOptionPane.showMessageDialog(this, "Rate or
alpha format error ");
                  return false;
             }
        }
        catch( NumberFormatException e){
                  JOptionPane.showMessageDialog(this, "Invalide
input data ");
             return false;
        }

        int type=1;
        if(linearRadio.isSelected()) type=0;
```

```
        else {
                MyUtility.makeGbComp(this,new JLabel(""+p.xMin),
                        MyUtility.NW,MyUtility.NONE,3,p.nyTicks+3,1,1,.5,0.,ins);
                MyUtility.makeGbComp(this,new JLabel(""+p.xMax),

MyUtility.NE,MyUtility.NONE,3+p.nxTicks,p.nyTicks+3,1,1,.5,0.,ins);
        }


        for (int i=1;i<p.nxTicks;i++) {
                double d = p.xMin+(double)i*p.xTick;

                if (p.xIsTime) {
                        java.sql.Timestamp ts1 = new java.sql.Timestamp ((long)d*1000);

                        String xs = MyUtility.toDateString(ts1);
                        MyUtility.makeGbComp(this,new JLabel(xs),
                        MyUtility.CENTER,MyUtility.NONE,3+i,p.nyTicks+3,1,1,1.,0.,ins);

                //      String xst = MyUtility.toTimeString(ts1);
                //      MyUtility.makeGbComp(this,new JLabel(xst),
                //      MyUtility.CENTER,MyUtility.NONE,3+i,p.nyTicks+4,1,1,1.,0.,ins);
                }
                else {
                        MyUtility.makeGbComp(this,new JLabel(""+d),

  MyUtility.CENTER,MyUtility.NONE,3+i,p.nyTicks+3,1,1,1.,0.,ins);
                }
        }

  }



  public PlotAxis(Plot2 p, String title,String xlabel, String ylabel) {

        setLayout(new GridBagLayout());
// add
        setBackground(Color.white);
        Insets ins  = new Insets(1,1,1,1);
        Insets ins0 = new Insets(0,0,0,0);

        MyUtility.makeGbComp(this,new PlotTick(p.nxTicks,true),

    MyUtility.CENTER,MyUtility.BOTH,3,p.nyTicks+2,p.nxTicks+1,1,.0,.0,ins0,0,0);

        MyUtility.makeGbComp(this,new PlotTick(p.nyTicks,false),
                MyUtility.CENTER,MyUtility.BOTH,2,1,1,p.nyTicks+1,.0,.0,ins0,0,0);


        MyUtility.makeGbComp(this,new JLabel(title),
                MyUtility.CENTER,MyUtility.NONE,0,0,p.nxTicks+4,1,0.,0.,ins);

        MyUtility.makeGbComp(this,p,
                MyUtility.CENTER,MyUtility.BOTH,3,1,p.nxTicks+1,p.nyTicks+1,0.,0.,ins0);
```

```
MyUtility.makeGbComp(this,new JLabel(ylabel),
        MyUtility.EAST,MyUtility.NONE,0,1,1,p.nyTicks+1,.0,.0,ins);

MyUtility.makeGbComp(this,new JLabel(""+p.yMax),
        MyUtility.NORTHEAST,MyUtility.NONE,1,1,1,1,0.,.5,ins);
MyUtility.makeGbComp(this,new JLabel(""+p.yMin),
        MyUtility.SOUTHEAST,MyUtility.NONE,1,p.nyTicks+1,1,1,0.,.5,ins);
for (int i=1;i<p.nyTicks;i++) {
        double d = p.yMax-(double)i*p.yTick;
        MyUtility.makeGbComp(this,new JLabel(""+d),
        MyUtility.EAST,MyUtility.NONE,1,i+1,1,1,0.,1.,ins);
}

MyUtility.makeGbComp(this,new JLabel(xlabel),
        MyUtility.CENTER,MyUtility.NONE,3,p.nyTicks+5,p.nxTicks+1,1,.1,0.,ins);

if (p.xIsTime) {
        java.sql.Timestamp ts1 = new java.sql.Timestamp ((long)p.xMin*1000);
        String xs  = MyUtility.toDateString(ts1);
        String xst = MyUtility.toTimeString(ts1);
        MyUtility.makeGbComp(this,new JLabel(xs),
                MyUtility.NW,MyUtility.NONE,3,p.nyTicks+3,1,1,.5,0.,ins);

        //comment out By Zhanshou May 19, we do not need hours, date will be enough
//      MyUtility.makeGbComp(this,new JLabel(xst),
//              MyUtility.NW,MyUtility.NONE,3,p.nyTicks+4,1,1,.5,0.,ins);

        ts1 = new java.sql.Timestamp ((long)p.xMax*1000);
        xs  = MyUtility.toDateString(ts1);
        xst = MyUtility.toTimeString(ts1);
        MyUtility.makeGbComp(this,new JLabel(xs),

MyUtility.NE,MyUtility.NONE,3+p.nxTicks,p.nyTicks+3,1,1,.5,0.,ins);

        //comment out By Zhanshou May 19, we do not need hours, date will be enough
        //MyUtility.makeGbComp(this,new JLabel(xst),
        //      MyUtility.NE,MyUtility.NONE,3+p.nxTicks,p.nyTicks+4,1,1,.5,0.,ins);
      }
      else {
        MyUtility.makeGbComp(this,new JLabel(""+p.xMin),
                MyUtility.NW,MyUtility.NONE,3,p.nyTicks+3,1,1,.5,0.,ins);
        MyUtility.makeGbComp(this,new JLabel(""+p.xMax),

MyUtility.NE,MyUtility.NONE,3+p.nxTicks,p.nyTicks+3,1,1,.5,0.,ins);
      }


      for (int i=1;i<p.nxTicks;i++) {
        double d = p.xMin+(double)i*p.xTick;

        if (p.xIsTime) {
                java.sql.Timestamp ts1 = new java.sql.Timestamp ((long)d*1000);

                String xs = MyUtility.toDateString(ts1);
                MyUtility.makeGbComp(this,new JLabel(xs),
```

```
                        MyUtility.CENTER,MyUtility.NONE,3+i,p.nyTicks+3,1,1,1.,0.,ins);
                    //comment out By Zhanshou May 19, we do not need hours, date will be
enough
                    //      String xst = MyUtility.toTimeString(ts1);
                    //      MyUtility.makeGbComp(this,new JLabel(xst),
                    //      MyUtility.CENTER,MyUtility.NONE,3+i,p.nyTicks+4,1,1,1.,0.,ins);
                    }
                    else {
                        MyUtility.makeGbComp(this,new JLabel(""+d),

    MyUtility.CENTER,MyUtility.NONE,3+i,p.nyTicks+3,1,1,1.,0.,ins);
                    }
            }

    }


}
```

**//Program # 13**

**//PlotSeries.java**

```
import java.awt.*;

/** the class is a line drawing tool

*/

public class PlotSeries {
    double[] x=null;
    double[] y                    = null;
    Color   color  = Color.blue;
    boolean  connect= true;
    boolean  symbol = true;
    int     size   = 6;
    boolean  bar    = false;

   public PlotSeries(double[] y, boolean connect, boolean symbol) {
     this.y = y;
             this.connect = connect;
             this.symbol  = symbol;
     }


   public PlotSeries(double[] y) {
      this.y = y;
             this.connect = true;
             this.symbol  = true;
     }

     public PlotSeries(double x[],double[] y) {
     this.x = x;
             this.y = y;
             this.connect = true;
             this.symbol  = true;
     }


    public void setConnect(boolean connect) {
             this.connect = connect;
      }


    public void setBarChart(boolean bar) {
             this.bar = bar;
      }


    public void setColor(Color color) {
             this.color = color;
      }
```

```
    public void setSymbol(boolean symbol) {
            this.symbol = symbol;
      }

    public void setSymbolSize(int size) {
            this.symbol = true;
            this.size = size;
      }

}
```

**//Program # 14**

**//PlotTick.java**

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
/** the class is a line drawing tool

*/

public class PlotTick extends JPanel {

    public boolean horizontal = true;
    public int nticks = 10;
    public Color color = Color.black;

    public static void main(String[] args){

            JFrame jf = new JFrame("Horizontal");
            jf.getContentPane().add(new PlotTick(10,true));
            jf.setSize(400,400);
            MyUtility.centerWindow(jf);
            jf.setVisible(true);

            JFrame jf2 = new JFrame("Vertical");
            jf2.getContentPane().add(new PlotTick(10,false));
            jf2.setSize(400,400);
            MyUtility.centerWindow(jf2);
            jf2.setVisible(true);

    }

    public PlotTick() {
      this.horizontal = true;
    }

    public PlotTick(int nticks) {
      this.nticks = nticks;
      this.horizontal = true;
    }

    public PlotTick(int nticks, boolean horizontal) {
      this.nticks = nticks;
      this.horizontal = horizontal;
    }

    public PlotTick(boolean horizontal) {
      this.horizontal = horizontal;
     }

    public void paint (Graphics g) {
            if (nticks==0) return;
```

```
//    g.setColor(MyUtility.getColor("white"));
//    g.fillRect(0, 0, getSize().width, getSize().height);

      g.setColor(color);
      if (horizontal) {
            int tick = getSize().width/nticks;
            for (int i=0; i<nticks; i++) {
                  g.drawLine(i*tick,0,i*tick,getSize().height);
                  for (int j=0; j<4; j++) {
                        int minor = i*tick+(j+1)*tick/5;
                        g.drawLine(minor,0,minor,getSize().height/4);
                  }
            }
            g.drawLine(getSize().width-1,0,getSize().width-1,getSize().height);
      }
      else {
            int tick = (getSize().height-1)/nticks;
            for (int i=0; i<nticks; i++) {
                  g.drawLine(0,i*tick,getSize().width,i*tick);
                  for (int j=0; j<4; j++) {
                        int minor = i*tick+(j+1)*tick/5;
                        g.drawLine(getSize().width*3/4,minor,getSize().width,minor);
                  }
            }
            g.drawLine(0,getSize().height-1,getSize().width,getSize().height-1);
      }
    }

}
```

**//Program # 15**

**//TableView.java**

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.text.*;
import java.sql.*;
import java.util.Date;

/** The class TableView is a basic class that can display a two dimension data in a table.

*/

public  class TableView extends JFrame implements ActionListener{
    DefaultTableModel dtm;
  JButton print = new JButton("Print");
  JButton close = new JButton("Close");
 // TableSorter  sorter;
  JTable table;
    String title;
    Font font[];
    private static int MAX_NUM=5;
  public TableView(String title,Object[][] data, Object[] columnNames){
    Insets ins    new Insets(10,10,10,10);
    Insets ins1 = new Insets(5,10,5,10);
     JLabel titleLabel=new JLabel(title);
     titleLabel.setFont(new Font("SanSerif", Font.BOLD, 22));

    dtm = new DefaultTableModel();
    JPanel whole = new JPanel(new GridBagLayout());


    dtm.setDataVector(data, columnNames);

     table = new JTable(dtm);
     setAlignment(table);

    JScrollPane scrollpane = new JScrollPane(table);

    JPanel bottom = new JPanel(new GridBagLayout());
    MyUtility.makeGbComp(bottom,                                                    print,
GridBagConstraints.CENTER,GridBagConstraints.NONE, 0, 0, 1, 1, .0, .0, ins1);
    MyUtility.makeGbComp(bottom,                                                    close,
GridBagConstraints.CENTER,GridBagConstraints.NONE, 1, 0, 1, 1, .0, .0, ins1);

     MyUtility.makeGbComp(whole, titleLabel, MyUtility.C,MyUtility.NONE,0,1,1,1,0.,0.,ins1);
    MyUtility.makeGbComp(whole,                                                     scrollpane,
GridBagConstraints.CENTER,GridBagConstraints.BOTH, 0, 2, 1, 1, 1.0, 1.0, ins);
```

```
    MyUtility.makeGbComp(whole,          bottom,          GridBagConstraints.CENTER,
GridBagConstraints.NONE, 0, 3, 1, 1, .0, .0, ins);

    print.addActionListener(this);
    close.addActionListener(this);
    getContentPane().add(whole);
    pack();
    MyUtility.centerWindow(this);
    setVisible(true);
  }


/**
    * Invoked when an action occurs
    */

  public void actionPerformed(ActionEvent e) {
    String s = e.getActionCommand();
    if (s.equals("Close"))
            this.setVisible(false);
  }


    private void setAlignment(JTable table){
            DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
            DefaultTableCellRenderer rightRenderer = new DefaultTableCellRenderer();
            centerRenderer.setHorizontalAlignment(JLabel.CENTER);
            rightRenderer.setHorizontalAlignment(JLabel.RIGHT);

            for (int c = 0; c < table.getColumnCount(); c++) {
                    table.getColumnModel().getColumn(c).setCellRenderer(rightRenderer);
            }
    }


}
```

**//Program # 16**

**//TellFuture.java**

```
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import javax.swing.filechooser.*;
import javax.accessibility.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.applet.*;
import java.net.*;

/** The TellFuture class is the main class of the system

*/

public class TellFuture extends JPanel implements ActionListener{
   TellFuture tf;
    Effect effect=new Effect();;
    EffectSetup effectSetup;
    NetworkSetup networkSetup;
    Vector inputVector;

    String[][] trainingSet;
    String[][] testSet;
    String[][] predictSet;

   Forecaster forecaster;
   // The Frame
  public static JFrame frame;

  // Current ui
  public String currentUI = "Metal";

  // The width and height of the frame
  public static int WIDTH = 790;
  public static int HEIGHT = 550;
  public static int INITIAL_WIDTH = 400;
  public static int INITIAL_HEIGHT = 200;

   static TellFuture instance;

   java.applet.Applet applet;

   String dataDir = "data";
  String weightsDir = "weights";


  public TellFuture() {
```

```
        this(null);
    }

    /***************************************
                    Constructor
    ***************************************/
    public TellFuture(java.applet.Applet anApplet) {
        super(true);
        instance = this;
        applet = anApplet;
        tf = this;
        setLayout(new BorderLayout());
        // Add a MenuBar
        add(createMenuBar(), BorderLayout.NORTH);
        JPanel logoPanel = createLogo();
        logoPanel.setBackground(Color.white);
        add(logoPanel, BorderLayout.CENTER);
    }


    /**
     * create Logo
     */
    JPanel createLogo() {
        JPanel p = new JPanel();
        p.setLayout(new BorderLayout());
        ImageIcon logo = loadImageIcon("images/forecastorLog.gif","Swing!");
        JLabel logoLabel = new JLabel(logo);
        logoLabel.getAccessibleContext().setAccessibleName("Swing!");
        p.add(logoLabel, BorderLayout.CENTER);
        p.setBorder(new                                    MatteBorder(6,6,6,6,
TellFuture.sharedInstance().loadImageIcon("images/AboutBorder.gif","About Box Border")));
        return p;
    }


    /***************************************/
    /************ create components ************/
    /***************************************/

    /**
     * MenuBar
     */
    JDialog aboutBox;
    JCheckBoxMenuItem cb;
    JRadioButtonMenuItem rb;


    JMenuBar createMenuBar() {
        // MenuBar
        JMenuBar menuBar = new JMenuBar();
        menuBar.getAccessibleContext().setAccessibleName("TellFuture Menus");
        JMenuItem mi;
```

```
                // "File" Menu
                JMenu file = (JMenu) menuBar.add(new JMenu("File"));
                file.setMnemonic('F');
                file.getAccessibleContext().setAccessibleDescription("The standard 'File' application
menu");

                //"Load Training Set" Menu
                mi = (JMenuItem) file.add(new JMenuItem("Load Training Set"));
                mi.setMnemonic('T');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("Load the Training Set from file");
                mi.addActionListener(this);


                //"Load Weights" Menu
                mi = (JMenuItem) file.add(new JMenuItem("Load Test Set"));
                mi.setMnemonic('E');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("Load the Test Set from file");
                mi.addActionListener(this);


                //"Load Weights" Menu
                mi = (JMenuItem) file.add(new JMenuItem("Load Predicted Inputs"));
                mi.setMnemonic('P');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("Load the Predicted Input from
file");
                mi.addActionListener(this);


                file.addSeparator();

                //"Exit" Menu
                mi = (JMenuItem) file.add(new JMenuItem("Exit"));
                mi.setMnemonic('x');
                mi.getAccessibleContext().setAccessibleDescription("Exit the TellFuture application");
                mi.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                System.exit(0);
        }
    });

            // Setup Menu
            JMenu setup = (JMenu) menuBar.add(new JMenu("Setup"));
            setup.setMnemonic('S');
            setup.getAccessibleContext().setAccessibleDescription("Set up");

            mi = (JMenuItem) setup.add(new JMenuItem("Network Setup"));
            mi.setMnemonic('N');
            mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
ActionEvent.ALT_MASK));
```

```
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                mi = (JMenuItem) setup.add(new JMenuItem("Effect Setup"));
                mi.setMnemonic('E');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                // Run Menu
                JMenu run = (JMenu) menuBar.add(new JMenu("Run"));
                run.setMnemonic('R');
                run.getAccessibleContext().setAccessibleDescription("");

                mi = (JMenuItem) run.add(new JMenuItem("Training"));
                mi.setMnemonic('T');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_T,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                mi = (JMenuItem) run.add(new JMenuItem("Testing"));
                mi.setMnemonic('E');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                mi = (JMenuItem) run.add(new JMenuItem("Predict"));
                mi.setMnemonic('P');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                // Display Menu
                JMenu display = (JMenu) menuBar.add(new JMenu("Display"));
                display.setMnemonic('D');
                display.getAccessibleContext().setAccessibleDescription("");

                mi = (JMenuItem) display.add(new JMenuItem("Display Training Results"));
                mi.setMnemonic('T');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_T,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                mi = (JMenuItem) display.add(new JMenuItem("Display Testing Results"));
                mi.setMnemonic('P');
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
ActionEvent.ALT_MASK));
                mi.getAccessibleContext().setAccessibleDescription("");
                mi.addActionListener(this);

                mi = (JMenuItem) display.add(new JMenuItem("Display Predicted Results"));
```

```
            mi.setMnemonic('P');
            mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,
ActionEvent.ALT_MASK));
            mi.getAccessibleContext().setAccessibleDescription("");
            mi.addActionListener(this);

            // Help Menu
            JMenu help = (JMenu) menuBar.add(new JMenu("Help"));
            help.setMnemonic('H');
            help.getAccessibleContext().setAccessibleDescription("");

      //"Help" Menu
            mi = (JMenuItem) help.add(new JMenuItem("About TellFuture"));
            mi.setMnemonic('A');
            mi.getAccessibleContext().setAccessibleDescription("Find out about the TellFuture
application");
            mi.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        if(aboutBox == null) {
                              aboutBox = new
JDialog(TellFuture.sharedInstance().getFrame(), "About TellFuture", false);
                              JPanel groupPanel = new JPanel(new BorderLayout());
                              ImageIcon groupPicture =
loadImageIcon("images/aboutTellFuture.gif",
            "TellFuture");
                              aboutBox.getContentPane().add(groupPanel,
BorderLayout.CENTER);
                              JLabel groupLabel = (new JLabel(groupPicture));
   groupLabel.getAccessibleContext().setAccessibleName("TellFuture Copyright");
   groupLabel.getAccessibleContext().setAccessibleDescription("TellFuture ");
                              groupPanel.add(groupLabel, BorderLayout.CENTER);
                              JPanel buttonPanel = new JPanel(true);
                              groupPanel.add(buttonPanel, BorderLayout.SOUTH);
                              JButton button = (JButton) buttonPanel.add(new
JButton("OK"));
                              button.addActionListener(new ActionListener() {
                                    public void actionPerformed(ActionEvent e) {
            aboutBox.setVisible(false);
                                    }
                              });
                        }
                        aboutBox.pack();
                        aboutBox.show();
                  }
            });
            return menuBar;
      }



public ImageIcon loadImageIcon(String filename, String description) {
      if(applet == null) {
            return new ImageIcon(filename, description);
```

```
                }
                else {
                        URL url;
                        try {
                                url = new URL(applet.getCodeBase(),filename);
                        }
                        catch(MalformedURLException e) {
                                System.err.println("Error trying to load image " + filename);
                                return null;
                        }
                        return new ImageIcon(url, description);
                }
        }


        public static TellFuture sharedInstance() {
                return instance;
        }


        public java.applet.Applet getApplet() {
                return applet;
        }

        public boolean isApplet() {
                return (applet != null);
        }

        public Container getRootComponent() {
                if(isApplet())
                        return applet;
                else
                        return frame;
        }


        public Frame getFrame() {
                if(isApplet()) {
                        Container parent;
                        for(parent = getApplet(); parent != null && !(parent instanceof Frame) ; parent
= parent.getParent());
                                if(parent != null)
                                        return (Frame)parent;
                                else
                                        return null;
                }
                else
                        return frame;
        }


        public void actionPerformed(ActionEvent evt){
                setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
                try{
```

```
                String s = evt.getActionCommand();
                //if (name.equals("File")){
                if (s.equals("Load Training Set")){
                        trainingSet=loadDataSet();
                }
                if (s.equals("Load Test Set")){
                        testSet=loadDataSet();
                }

                if (s.equals("Load Predicted Inputs")){
                        predictSet=loadPredictSet();
                }

                else if (s.equals("Save Weights As...")){
                        saveWeightsAs();
                }
                else if (s.equals("Effect Setup")){
                        showEffectSetup();
                }
                else if (s.equals("Network Setup")){
                        showNetworkSetup();
                }

                else if (s.equals("Training")){
                        training();
                }
                else if (s.equals("Testing")){
                        testing();
                }

                else if (s.equals("Predict")){
                        predict();
                }

                else if (s.equals("Display Training Results")){
                        displayTraining();
                }
                else if (s.equals("Display Testing Results")){
                        displayTesting();
                }
                else if (s.equals("Display Predicted Results")){
                        displayPredicted();
                }

                setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
        }
        catch(Exception e){
                setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
                e.printStackTrace();
        }
}


public String[][] loadDataSet() throws Exception{
        JFileChooser chooser = new JFileChooser();
  File dataParent = new File("./"+dataDir);
```

```
                chooser.setCurrentDirectory(dataParent);
                int retval = chooser.showOpenDialog(this);
                if(retval == 0) {
                        File theFile = chooser.getSelectedFile();
                        if(theFile != null) {
                                FileInputStream inputStream =new FileInputStream(theFile);
                                BufferedReader d = new BufferedReader(new
InputStreamReader(inputStream));
                                inputVector =new Vector();
                                boolean stop= true;
                                while (true)
                                {
                                        String line=d.readLine();
                                        if(line==null) break;
                                        if(line.indexOf("-999") >1) continue;
                                        StringTokenizer st = new StringTokenizer(line);
                                        String[] data=new String[5];
                                        int index=0;
                                        while (st.hasMoreTokens()) {
                                                data[index++]=st.nextToken();
                                        }
                                        inputVector.addElement(data);
                                }
                        }
                }
                String[][] temp =new String[inputVector.size()][5];
                for(int i=0;i<temp.length;i++){
                        String data[] =(String[]) inputVector.elementAt(i);
                        for( int j=0;j<5;j++)
                                temp[i][j]=(String) data[j];
                }
                return temp;
        }


        public String[][] loadPredictSet() throws Exception{
                JFileChooser chooser = new JFileChooser();
         File dataParent = new File("./"+dataDir);
                chooser.setCurrentDirectory(dataParent);
                int retval = chooser.showOpenDialog(this);
                if(retval == 0) {
                        File theFile = chooser.getSelectedFile();
                        if(theFile != null) {
                                FileInputStream inputStream =new FileInputStream(theFile);
                                BufferedReader d = new BufferedReader(new
InputStreamReader(inputStream));
                                inputVector =new Vector();
                                boolean stop= true;
                                while (true)
                                {
                                        String line=d.readLine();
                                        if(line==null) break;
                                        if(line.indexOf("-999") >1) continue;
                                        StringTokenizer st = new StringTokenizer(line);
                                        String[] data=new String[4];
                                        int index=0;
```

```
                                        while (st.hasMoreTokens()) {
                                                data[index++]=st.nextToken();
                                                if(index==4) break;
                                        }
                                        inputVector.addElement(data);
                                }
                        }
                }
                String[][] temp =new String[inputVector.size()][4];
                for(int i=0;i<temp.length;i++){
                        String data[] =(String[]) inputVector.elementAt(i);
                        for( int j=0;j<4;j++)
                                temp[i][j]=(String) data[j];
                }
                return temp;
        }



        public void saveWeightsAs() throws Exception{
                        JFileChooser chooser = new JFileChooser();
                        File weightsParent = new File("./"+weightsDir);
                        chooser.setCurrentDirectory(weightsParent);

                        int retval = chooser.showSaveDialog(this);
                        if(retval == 0) {
                                File theFile = chooser.getSelectedFile();
                                if(theFile != null) {
                                        JOptionPane.showMessageDialog(this, "You chose
this file: " +

chooser.getSelectedFile().getAbsolutePath());
                                                return;
                                        }
                                }

        }


        public void showNetworkSetup(){
                if(effect==null) effect=new Effect();
                if(networkSetup==null){
                        networkSetup=new NetworkSetup(effect);
                }
                else networkSetup.setVisible(true);
        }


        public void showEffectSetup(){
                if(effect==null) effect=new Effect();
                if(effectSetup==null){
                        effectSetup=new EffectSetup(effect);
                }
```

```
            else effectSetup.setVisible(true);
    }


    public void training() throws Exception{
            forecaster =new Forecaster(effect,trainingSet);
            forecaster.training();
    }


    public void testing() throws Exception{
            forecaster.setTestSet(testSet);
            forecaster.testing();
    }

    public void predict() throws Exception{
            forecaster.setPredictSet(predictSet);
            forecaster.predict();
    }


    public void displayTraining() throws Exception{
            forecaster.displayTraining();
    }


    public void displayTesting() throws Exception{
            forecaster.displayTesting();
    }

    public void displayPredicted() throws Exception{
            forecaster.displayPredicted();
    }


    public static void main(String[] args) {
            String vers = System.getProperty("java.version");
     if (vers.compareTo("1.1.2") < 0) {
       System.out.println("!!!WARNING: TellFuture must be run with a " +
                    "1.1.2 or higher version VM!!!");
    }

            // Force TellFuture to come up in the Cross Platform L&F
            try {

    UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
                    // If you want the System L&F instead, comment out the above line and
                    // uncomment the following:
                    //
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception exc) {
                    System.out.println("Error loading L&F: " + exc);
            }

            WindowListener l = new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {System.exit(0);}
```

```
        };

        frame = new JFrame("TellFuture");
        frame.addWindowListener(l);

        //      JOptionPane.setRootFrame(frame);
        // show the frame
        //      frame.setSize(INITIAL_WIDTH, INITIAL_HEIGHT);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        ///     frame.setLocation(screenSize.width/2 - INITIAL_WIDTH/2,
        //      screenSize.height/2 - INITIAL_HEIGHT/2);

        frame.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

        TellFuture tf = new TellFuture();
        frame.getContentPane().removeAll();
        frame.getContentPane().setBackground(Color.white);
        frame.getContentPane().setLayout(new BorderLayout());
        frame.getContentPane().add(tf, BorderLayout.CENTER);
        frame.setLocation(screenSize.width/2 - WIDTH/2,
                screenSize.height/2 - HEIGHT/2);

        frame.setSize(WIDTH, HEIGHT);
        frame.pack();
        frame.setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
        frame.show();

        frame.validate();
        frame.repaint();
        tf.requestDefaultFocus();
    }

}
```

# Appendix B (Figures)

Fig 1: This figure shows the main script of the program

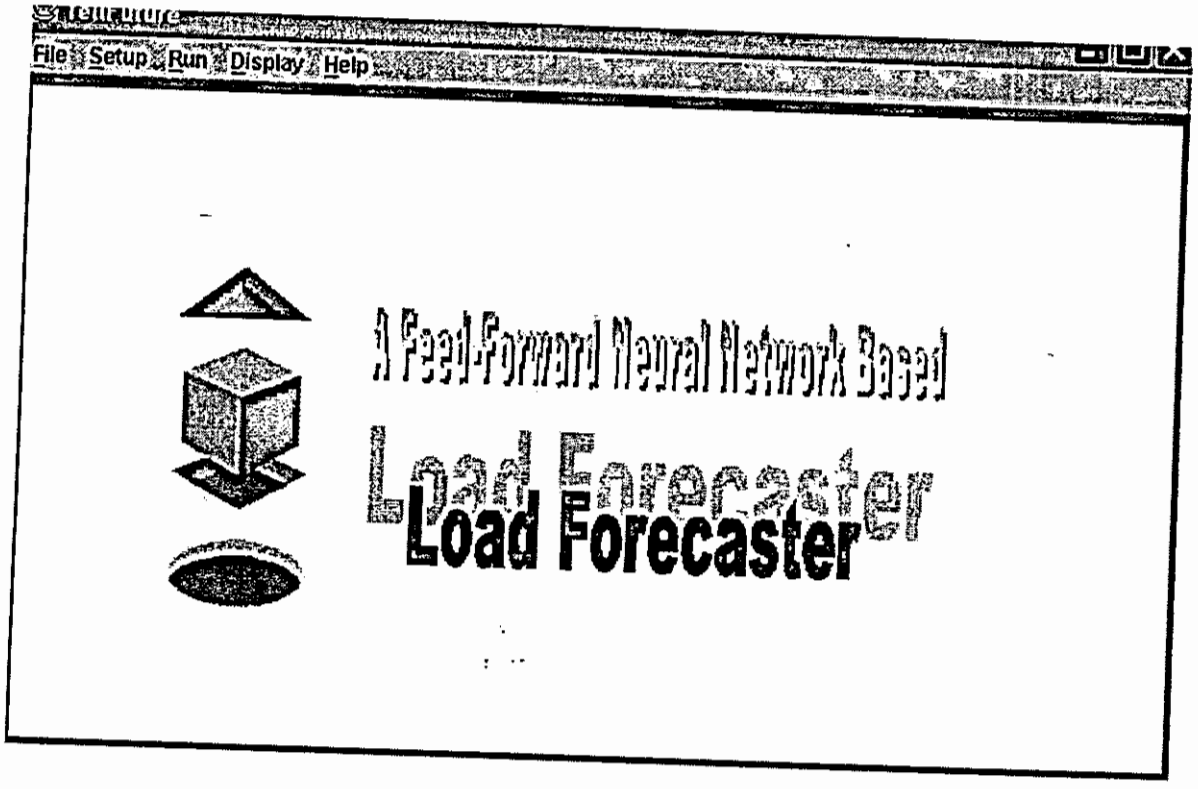


Fig 2: This Figure shows the contents of menu "File".

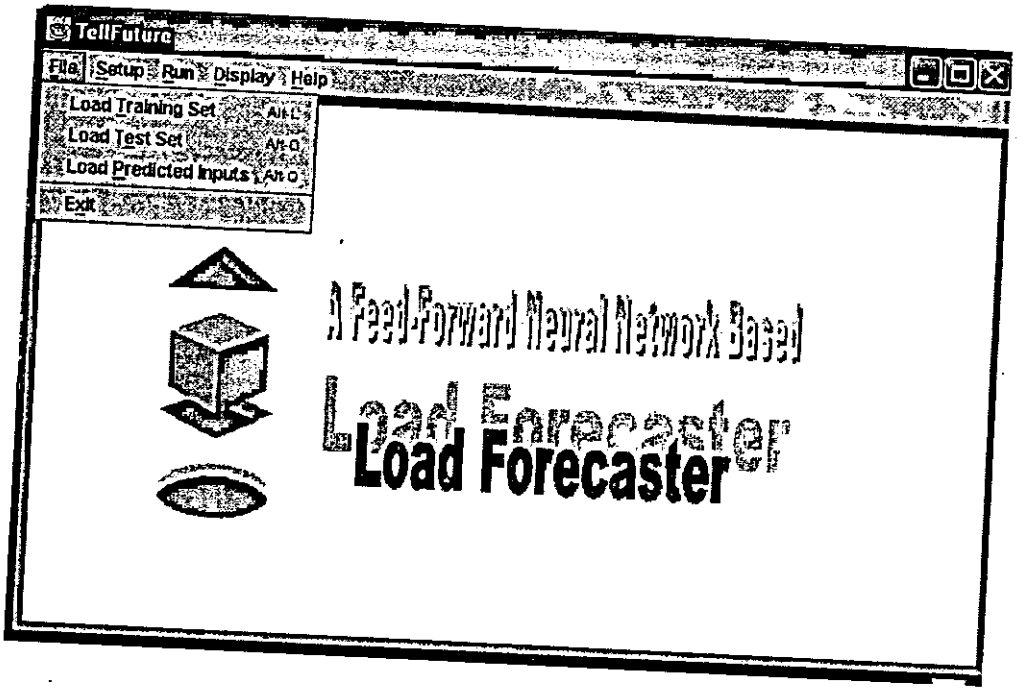


Fig 3: The figure is about the "Setup" menu.

Fig 5: Display menu contains the following.

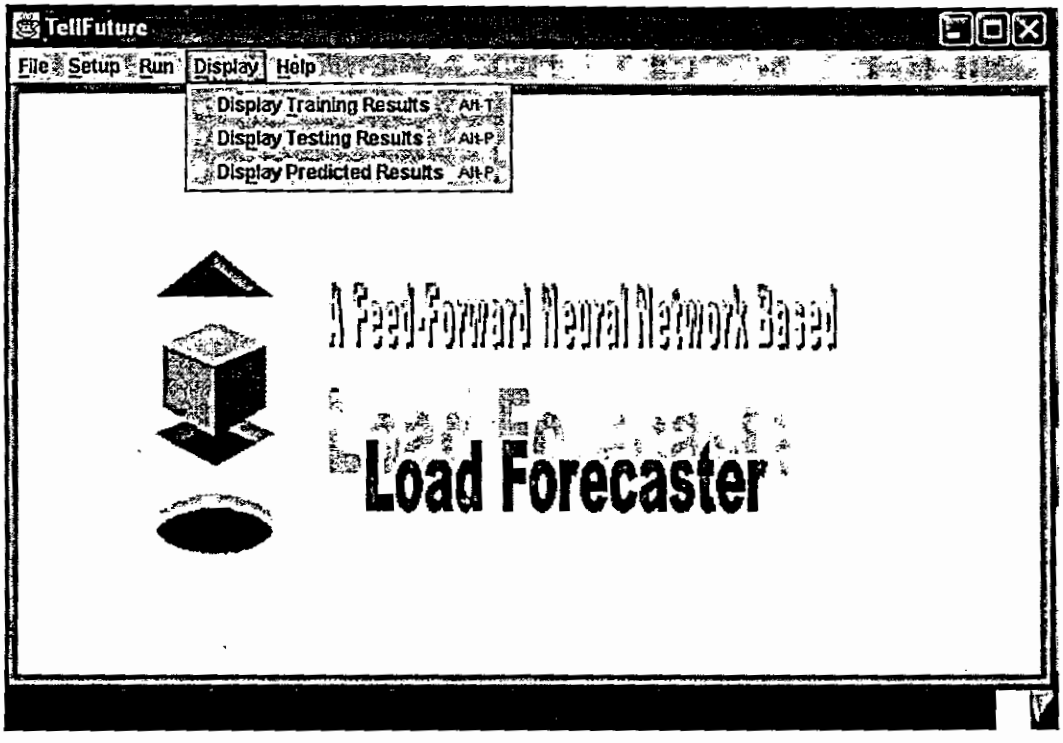


Fig 6: Help menu

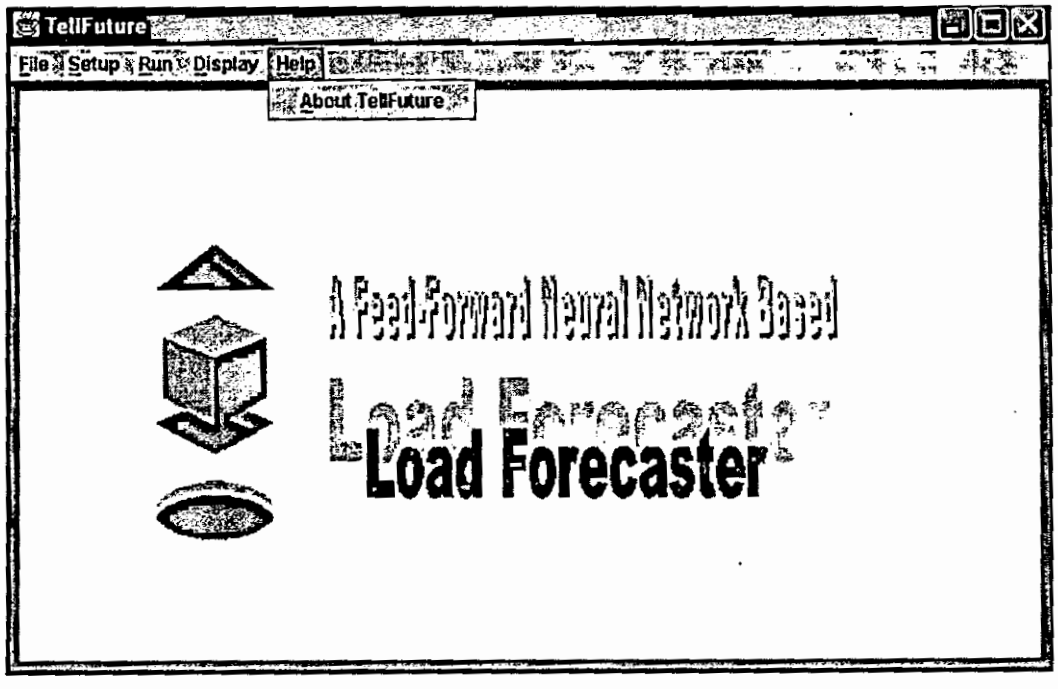


Fig 7: Demonstrating the function of "Open" in the main menu "File"

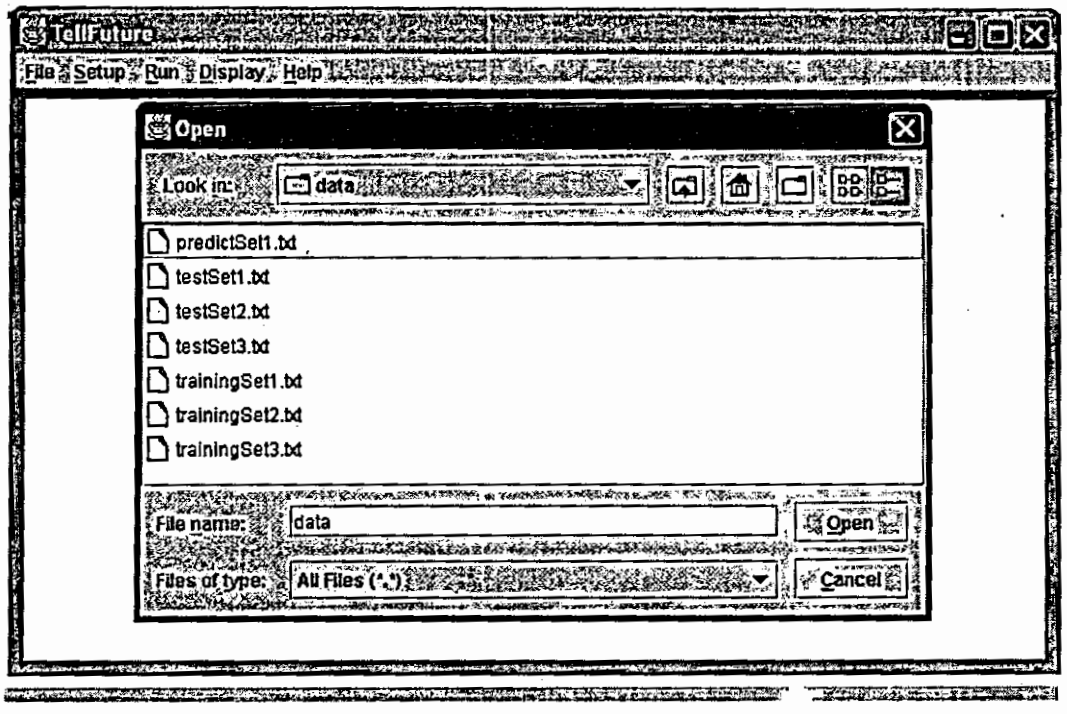


Fig 8: Training results are showed in the figure

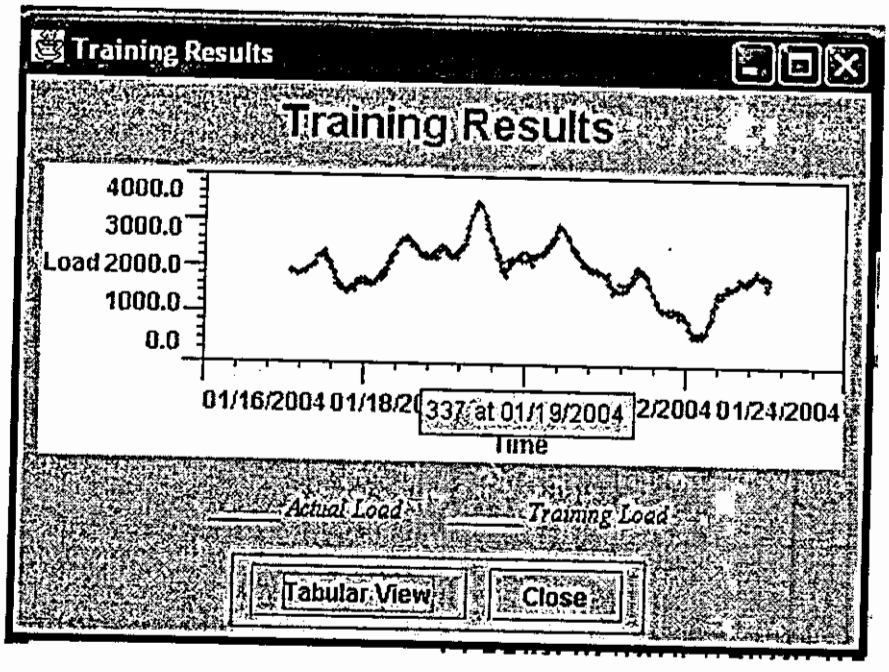


Fig 9: This figure shows the training results with other data, while taking the values of actual and training load.

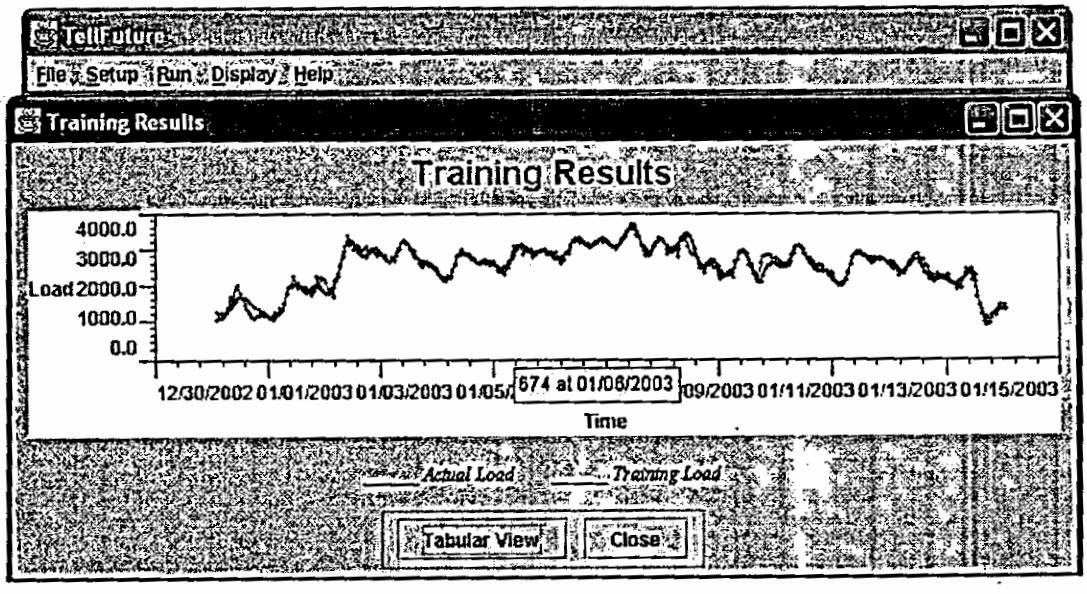


Fig 10: Tabular View of the results showed in graphic form.

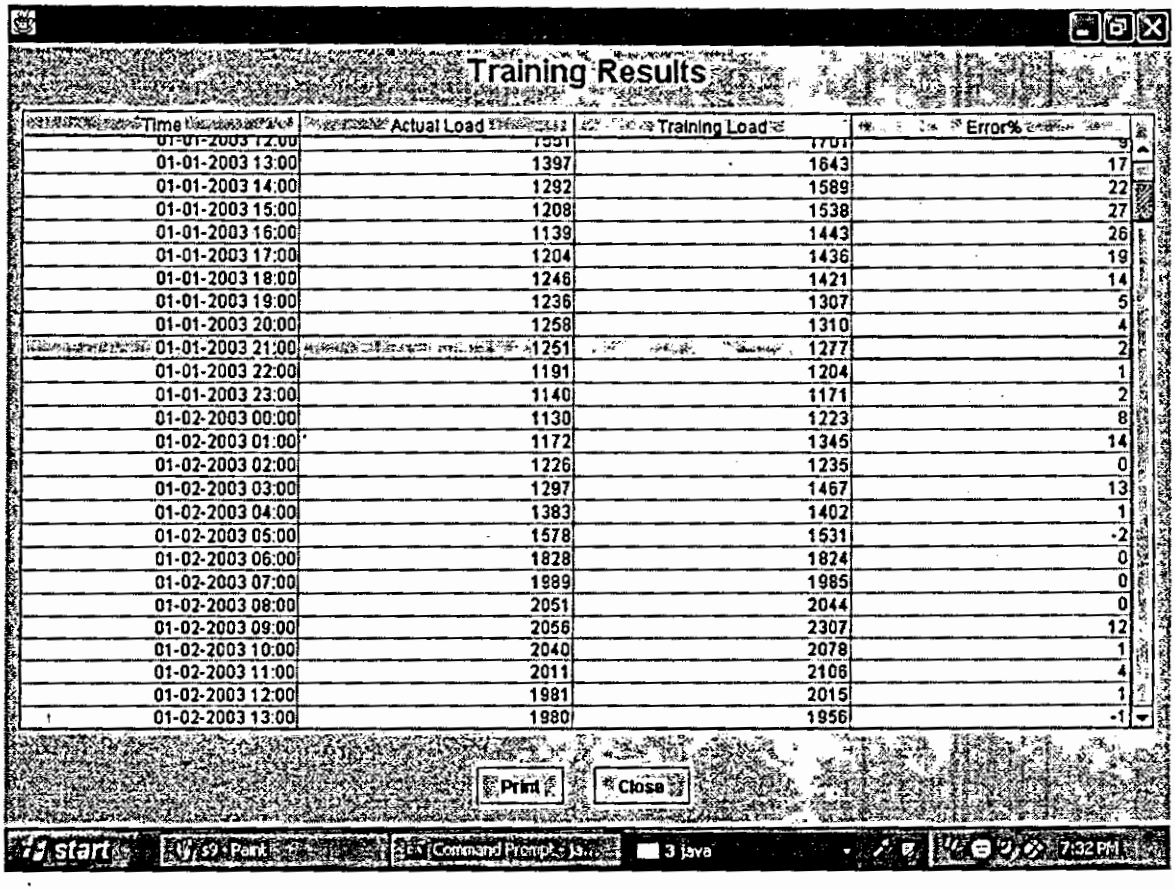


Fig 11: Testing results of the program are showed in the figure.

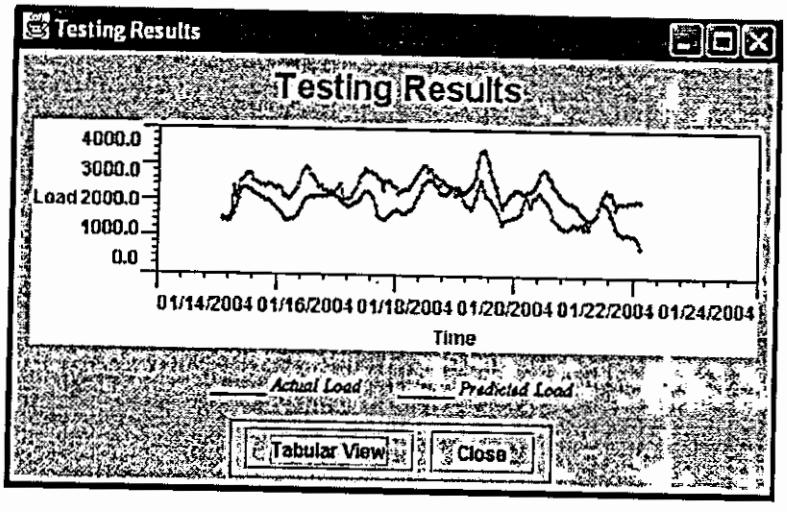
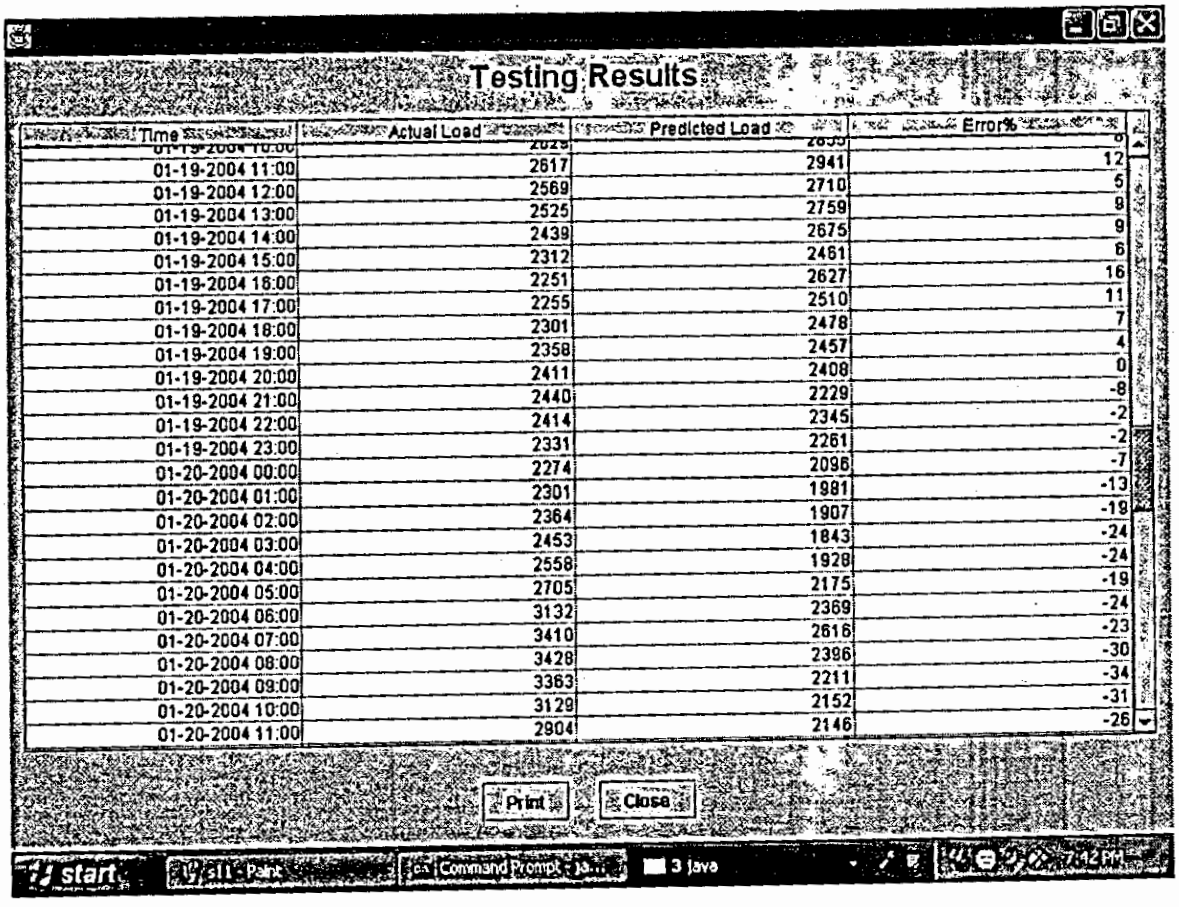


Fig 12: Tabular forms of the testing results.



# REFERENCES

## References

[1] T. Masters, *Practical Neural Network Recipes in C++*. Academic Press, Inc., 1993

[2] S. T. Welstead, *Neural Network and Fuzzy Logic Applications in C++*. John Wiley & Sons, Inc., 1994

[3] Ben Kröse, Patrick V.D. Smagt, *An Introduction to Neural Network*. The University of Amsterdam, 1996

[4] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., 1994

[5] W.S. Sarles, *Neural Network FAQ*, periodic posting to the Usenet newsgroup comp.ai.neural-net, URL: ftp://ftp.sas.com/pub/neural/FAQ.html, 1997

[6] R.D. Reed and Robert J. Mark, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. The MIT Press, 1999

[7] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995

[8] B.D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996

[9] Rumelhart, D.E., Hinton, G.E., and Williams, R.J., *Learning internal representations by error propagation*, Nature, p533-536, Vol. 323, 1986

[10] Y. Fukuoka, H. Matsuki, *A Modified Back-propagation Method to Avoid Local Minima*, Neural Networks, p1059-1072, Vol. 11, 1998

[11] R. Salomon, J. L. Hemmen, *Accelerating Backpropagation through Dynamic Self-Adaptation*, Neural Networks, p589-601, Vol. 9, 1996

[12] S. V. Kamarthi, S. Pittner, *Accelerating Neural Network Training using Weight Extrapolations*, Neural Networks, p1285-1299, Vol. 12, 1999

[13] T. Masters, *Signal and Image Processing with Neural Networks: a C++ Sourcebook.* John Wiley & Sons, Inc., 1994

[14] S. Subramanian, T. Davis, *Neural Net Based Load Forecasting – A Practical Tool for Prediction Future Gas Loads.* PSIG 30th Annual Conference, Denver, Colorado, 1998

[15] D.C. Park, M. A. El-Sharkawi, R. J. Marks II, L. E. Atlas and M. J. Damborg, *Electric Load Forecasting Using An Artificial Neural Network*, IEEE Transaction on Power Systems, p442-449, Vol. 6, 1991

[16] J. S. McMenamin, F. A. Monforte, *Using Neural Networks for Day-Ahead Forecasting.* Western Economic Association 74th Annual Conference, San Diego, 1999

[17] W. P. Wagner, *Daily Peak Load Electricity Forecasting using Artificial Neural Networks.* URL: http://hsb.baylor.edu/ramsower/acis/papers/wagnerw.htm

[18] A. Khotanzad. M. H. Davis, A. Abaye, D. J. Maratukulam, *An Artificial Neural Network Hourly Temperature Forecaster with Application in Load Forecasting.* IEEE Transaction on Power Systems, p870-876, Vol. 11, 1996

[19] S. T. Chen, D. C. Yu, A. R. Moghaddamjo, *Weather Sensitive Short-Term Load Forecasting using Nonfully Connected Artificial Neural Network.* IEEE Transaction on Power Systems, p1098-1104, Vol. 7, 1992

[20] A.G. Baklrtzls, V. Petrldls, S. J. Klartzls, *A Neural Network Short Term Load Forecasting Model for the Greek Power System.* IEEE Transaction on Power Systems, p858-862, Vol. 11, 1995

[21] Peng, T.M., Hubele, N.F., and Karady, *An Adaptive Neural Network Approach to One-Week Ahead Load Forecasting*, IEEE Transactions on Power Systems, p1195-2003, Vol. 8, 1993

[22] J. Angstenberger, *Prediction of the S&P 500 Index with Neural Networks,* p143-152, Neural Networks and their Applications, edited by J. G. Taylor, John Wiley & Sons, Inc., 1996

[23] X. Ding, S. Canu, *Neural Network Based Model for Forecasting,* p153-165, Neural Networks and their Applications, edited by J. G. Taylor, John Wiley & Sons, Inc., 1996

[24] H. C. A. M. Withagen, *Neural Networks: Analog VLSI Implementation and Learning Algorithms.* Technische Universiteit, Eindhoven, 1997

[25] T. Masters, *Advanced Algorithms for Neural Networks: a C++ Sourcebook.* John Wiley & Sons, Inc., 1995

[26] T. Masters, *Neural, Novel & Hybrid Algorithms for Time Series Prediction.* John Wiley & Sons, Inc., 1995

[27] V. Petridis, A. Kehagias, *Predictive Modular Neural Networks: Applications to Time Series.* P123-133, Kluwer Academic Publishers, 1998.


IBCAST
Third International
Bhurban Conference
on Applied Sciences
& Technology
07-12 June, 2004
at Bhurban-Pakistan

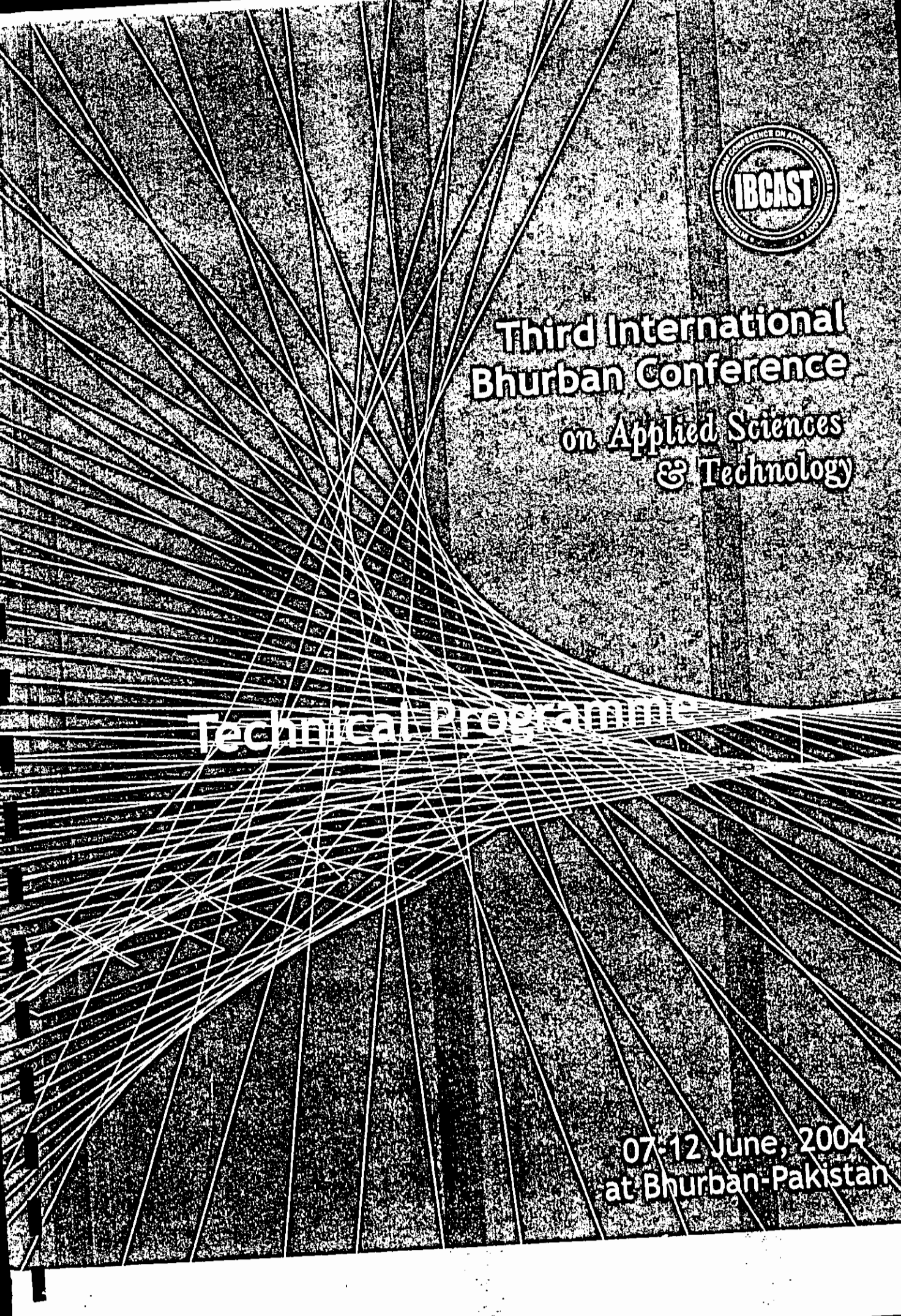
Third International Bhurban Conference on Applied Sciences & Technology



# THIRD
# INTERNATIONAL BHURBAN CONFERENCE
# ON
# APPLIED SCIENCES & TECHNOLOGY

07-12 June, 2004
at Bhurban-Pakistan

# Table of Contents


IBCAST
Third International
Bhurban Conference
on Applied Sciences
& Technology
Technical Programme
07-12 June, 2004
at Bhurban-Pakistan

# WIRELESS COMMUNICATION AND RADAR
# WCR-2004

## Session Program
## Program Coordinator: Dr. Jahangir Kayani
## Thursday, 10th June, 2004

| Timings | Title of paper | Name |
|---|---|---|
| 14:00 - 14:40 | Guest Lecture (Wireless Networking & Software Radio) | *Dr Salahuddin Qazi* |
| 14:40 - 15:00 | SCAPA: An automated approach to scan & Pattern Analysis | *Amir Khurram Rashid Masood Akhtar* |
| 15:00 - 15:20 | Design of a Low Phase Noise Voltage Controlled Oscillator with On-Chip Vs Off-Chip Passive Components" | *Saeed Zafar* |
| 15:20 - 15.50 | TEA BREAK | |
| 15:50 - 16:10 | High Resolution Beamforming Using Wigner-Ville Distribution | *Muhammad Hussain Sial* |
| 16:10 - 16:30 | Novel Fabrication Technique for Flexible Silicon Cables | *Arfan Ghani* |
| 16:30 - 16:50 | Design of Coupled-Line Coupled Hairpin-Line Filter Using EM Analysis for Parameter Extraction | *Tahir Abbas Dr Mojeeb Bin Ihsan* |
| 16:50 - 17:10 | Congestion Arbitration And Source Problem Prediction Using Ann In Wireless Networks | *Saadia Arshad, Ambreen Siddique* |
| 17:10 - 17:30 | Working MANET of Location Aware Nodes & Frame Work to Monitor Nodes in an Interactive Manner | *Asim Munawar* |

Third International Conference on Applied Sciences & Technology

## DIRECTOR

Prof. Riazuddin

## SCIENTIFIC SECRETARY

Prof. Hafeez R. Hoorani

## PROGRAMME COORDINATORS

Dr. Arshad Munir
Dr. Raza Samar
Dr. Mehmood Ahmad Khan
Dr. Jahangir Kayani

## EVALUATION COMMITTEE

**Advanced Materials**
Dr. Arshad Munir
Dr. Abdul Ghaffar
Dr. Shagufta Zulfiqar

**Control and Simulation**
Dr. Nisar Ahmad
Dr. Mian Awais
Dr. M. Shahzad
Dr. Amer Iqbal
Mr. Ayaz Aziz

**Wireless Communication and Radars**
Dr. Jahangir Kayani
Dr. Asif Raza
Dr. Mojeeb Bin Ehsan

**Computational Fluid Dynamics**
Dr. Ruben Avila
Dr. Hossein Hamdani
Dr. Mansoor
Dr. Jahanzeb

## TECHNICAL COMMITTEE

**Advanced Materials**
Dr. Arshad Munir
Dr. Zafar-uz-Zaman
Dr. Rizwan Hussain

**Control and Simulation**
Dr. Anwar Mughal
Dr. M. Nazar
Dr. M. Shahzad

**Wireless Communication and Radars**
Dr. Jahangir Kayani
Dr. Asif Raza
Mr. Naveed Ahsan

**Computational Fluid Dynamics**
Mr. Syed Bilal Hussain Shah
Mr. Ajmal Baig
Mr M. Nasir Kamran

## SESSION –II WIRELESS COMMUNICATION & RADAR 10th June, 2004 (Thursday Afternoon)

**SESSION CHAIR: DR TANVEER UL HAQ**

| | | | | |
|---|---|---|---|---|
| 8 | **Dr Salahuddin Qazi** | **Guest Lecture (Wireless Networks & Enabling Technologies)** | SUNY Institute, NY USA | 14:00-14:40 |
| 9 | Amir Khurram Rashid | SCAPA: An automated approach to scan & Pattern Analysis | NESCOM | 14:40-15:00 |
| 10 | Saeed Zafar | Design of a Low Phase Noise Voltage Controlled Oscillator with On-Chip Vs Off-Chip Passive Components" | Comsats Institute of Information Technology | 15:00-15:20 |
| | | **TEA BREAK** | | 15:20-15:50 |
| 11 | Muhammad Hussain Sial | High Resolution Beamforming Using Wigner-Ville Distribution | NESCOM | 15:50-16:10 |
| 12 | Arfan Ghani | Novel Fabrication Technique for Flexible Silicon Cables | The Balochistan University of IT and Management Sciences | 16:10-16:30 |
| 13 | Tahir Abbas | Design of Coupled-Line Coupled Hairpin-Line Filter Using EM Analysis for Parameter Extraction | NUST | 16:30-16:50 |
| 14 ✓ | Saadia Arshad | Congestion Arbitration And Source Problem Prediction Using Ann In Wireless Networks | International Islamic University Islamabad | 16:50-17:10 |
| 15 | Syed Usman Haider | Design & Implementation of High Resolution Bearing Measurement System using Amplitude Comparision Technique | NESCOM | 17:10-17:30 |

# CONGESTION ARBITRATION AND SOURCE PROBLEM PREDICTION USING ANN IN WIRELESS NETWORKS

Saadia Arshad, Ambreen Siddique, Dr Muhammad Sher, Dr Sikandar Hayat Khiyal
msher313@iiu.edu.pk
Department of Computer Science,
Faculty of Applied Sciences,
International Islamic University,
Islamabad

**Abstract:**

Congestion is the problem which occurs when demand for a resource outstrips the capacity. In wireless networks, congestion may occur through antenna, satellite link, routers and switches which are shared by several sources. The congestion control scheme described here employs a neural network to predict the state of congestion in a wireless network over a prediction horizon. We propose using learning techniques to predict the problems before they start impacting the performance of services especially in wireless communication. In this paper we focus on using a feed forward neural network to predict severe congestion in a wireless network. We also use neural networks to predict the source or sources responsible for the congestion, and we design and apply a control method for limiting the rate of the offending sources so that congestion can be avoided. This paper introduces an adaptive neuro-control strategy, adaptive neural swarming(ANS). A highly non-linear bioreactor benchmark is used in the control simulation. Based on the neural predictor output, source rate control signals are obtained by minimizing a cost function which represents the cumulative differences between a set-point and the predicted output. An analytical procedure for the source rate control signal computations is given using gradient functions of the neural network predictor by the use of wireless session protocol(WSP),wireless transaction protocol(WTP). Unlike the RED and usual TCP/IP flow control, the proposed method is applied only to selected nodes and converges to the final rate faster. The described techniques set the stage for a new wave of wireless network managers that are capable of preventing wireless networking problems instead of repairing them.

**Keywords**: congestion control, neural networks, weight pruning, Meta neural, network problem Prediction, Fair Queuing, flow rate control, WFQ.

## 1. Introduction

In the world of networking, more emphasis is being placed on speed, connectivity, and reliability. When network problems occur, they often catastrophically break the service for those enterprises or individuals that depend on the network connection. Sometimes,

such breaks of service are just annoying, but for companies and commercial users they often mean lost revenues on the order of thousands, or even millions, of dollars. Such breaks have become a significant problem in all forms of electronic commerce.

A network congestion collapse occurs when the network is increasingly busy but little useful work is getting done. Congestion management features allow you to control congestion by determining the order in which packets are transmitted out an interface based on priorities assigned to those packets. Congestion management entails the creation of queues, assignment of packets to those queues based on the packet's classification, and scheduling of the packets in a queue for transmission. The congestion management QoS feature offers four types of queuing protocols, each of which allows you to specify creation of a different number of queues, affording greater or lesser degrees of differentiation of traffic and the order in which that traffic is transmitted. During periods with light traffic, that is, when no congestion exists, packets are transmitted out the interface as soon as they arrive. During periods of transmit congestion at the outgoing interface, packets arrive faster than the interface can transmit them. If you use congestion management features, packets accumulating at an interface are queued until the interface is free to transmit them; they are then scheduled for transmission according to their assigned priority and the queuing mechanism configured for the interface. The router determines the order of packet transmission by controlling which packets are placed in which queue and how queues are serviced with respect to each other.

It is important to avoid high packet loss rates in the internet. This problem is further critical in the wireless communication because of the shared transmission medium, dynamic topologies, different protocols (like WSP and WTP) and costly medium. The bandwidth utilization in the wireless medium requires a good congestion control mechanism. To address this difficulty, a system is needed to insure network availability and efficiency by preventing such costly wireless network breakdowns. The first step towards this end is to create a system with the intelligence to recognize, as early as possible, early signs of incoming network service difficulties. If the problem can be recognized in advance, changing network parameters can possibly circumvent the problem.

There are techniques of fancy queuing to avoid congestion but the major problem of knowing about congestion in advance still remains unsolved. In this work we propose a neural network based forecasting technique to estimate and forecast congestion state in advance on the basis of input traffic from various sources arriving at the router [5]. We have used a feed forward neural network to forecast and then apply the results with the simulated traffic generated through network simulator (NS-2). The section-2 describes about some background information on existing congestion management techniques. The next section will describe design of an artificial neural network (ANN) for congestion forecasting. Next in section-4 we will discuss the implementation detail of our experimental setup. Then in section -5 we present the results of our work, the conclusion and the references.

**Background**

One of the biggest problems with TCP's [5] congestion control algorithm over drop-tail queues is that sources reduce their transmission rates only after detecting packet loss due to queue overflow. Since a considerable amount of time may elapse between the packet drop at the router and its detection at the source, a large number of packets may be dropped as the senders continue transmission at a rate that the network cannot support[10]. Random early detection (RED) queuing alleviates this problem by detecting incipient congestion early and delivering congestion notification to the end-hosts, allowing them to reduce their transmission rates before queue over-flow occurs. In order to be effective, a RED queue must be configured with a sufficient amount of buffer space to accommodate an applied load greater than the link capacity from the instant in time that congestion is detected using the queue length trigger, to the instant in time that the applied load decreases at the bottleneck link in response to congestion notification. RED must also ensure that congestion notification is given at a rate which sufficiently suppresses the transmitting sources without underutilizing the link. Unfortunately, when a large number of TCP [5] sources are active, the aggregate traffic generated is extremely bursty[2]. Bursty traffic often defeats the active queue management techniques used by RED since queue lengths grow and shrink rapidly, well before RED can react.

One way to solve this problem is to use a large amount of buffer space at the RED gateways. For example, it has been suggested that in order for RED to work well, an intermediate router requires buffer space that amounts to twice the bandwidth-delay product. This approach, in fact, has been taken by an increasingly large number of router vendors. Unfortunately, in networks with large bandwidth-delay products, the use of large amounts of buffer adds considerable end-to-end delay and delay jitter. This severely impairs the ability to run interactive applications. In addition, the abundance of deployed routers which have limited memory resources makes this solution undesirable While RED can achieve this ideal operating point, it can do so only when it has a sufficiently large amount of buffer space and is correctly parameterized.

Weighted fair queue (WFQ) is an automated scheduling method that provides fair bandwidth allocation to all network traffic. WFQ applies priority, or weights, to identified traffic to classify traffic into conversations and determine how much bandwidth each conversation is allowed relative to other conversations. WFQ is a flow-based algorithm that simultaneously schedules interactive traffic to the front of a queue to reduce response time and fairly shares the remaining bandwidth among high-bandwidth flows. In other words, WFQ allows you to give low-volume traffic, such as Telnet sessions, priority over high-volume traffic, such as FTP sessions. WFQ gives concurrent file transfers balanced use of link capacity; that is, when multiple file transfers occur, the transfers are given comparable bandwidth.

WFQ provides traffic priority management that dynamically sorts traffic into messages that make up a conversation. WFQ breaks up the train of packets within a conversation to ensure that bandwidth is shared fairly between individual conversations and that low-

volume traffic is transferred in a timely fashion. WFQ classifies traffic into different flows based on packet header addressing, including such characteristics as source and destination network or MAC address, protocol, source and destination port and socket numbers of the session, Frame Relay data-link connection identifier (DLCI) value, and type of service (ToS) value. There are two categories of flows: high-bandwidth sessions and low-bandwidth sessions. Low-bandwidth traffic has effective priority over high-bandwidth traffic, and high-bandwidth traffic shares the transmission service proportionally according to assigned weights. Low-bandwidth traffic streams, which comprise the majority of traffic, receive preferential service, transmitting their entire offered loads in a timely fashion. High-volume traffic streams share the remaining capacity proportionally among themselves. WFQ places packets of the various conversations in the fair queues before transmission. The order of removal from the fair queues is determined by the virtual time of the delivery of the last bit of each arriving packet.

These techniques work well in the common congestion scenarios but are restricted by a lack of early warning system about expected congestion state. For this purpose a forecasting model is important to have some kind of early warning system.

## 2. Experimental Setup:

### Design of ANN

Neural networks are computational models with the capacity to learn, to generalize, or to organize data based on parallel processing. These networks can be trained with a powerful and computationally efficient method called error back-propagation. Forecasting the behavior of complex system has been a broad application domain for neural networks. Network traffic forecast is a relatively new application of neural networks. We propose a system that uses neural networks to detect network congestion before it results in a breakdown of the network service and which also identifies the source of the congestion. Having the nodes identified, our system applies the flow rate restriction adaptively to the identified sources to avoid congestion overflowing the router's buffers. It should be noted that design and experiments presented in this paper focus on congestion control; however, the techniques could be applicable to other network problems. It should also be noted that the remedy in the form of flow rate restrictions can be applied directly to the original flow source if it is within the domain controlled by our system, or it could be applied to the edge router to the domain to which our system is applied. In the latter case, the restriction will result in the packets of the restricted flow being dropped at the edge router to the domain [4]. This kind of a solution in which the congestion is decomposed and "moved" from the internal routers to the edge routers is becoming increasingly popular in modern traffic management. Finally, it should be noted that in such edge-control techniques the domains controlled by separate systems will collaborate through the edge routers. Dropping packets at the entry edge router of one domain will cause the packets to be dropped at the exit router of the neighboring domain which will treat such dropping as congestion and then will identify the source of the flow. As a result, our techniques can be applied locally at a domain of

the decomposed network and their congestion solution will iteratively be mapped to the corresponding edge routers of the intermediate domains until the source of the flow is found and informed of the need to decrease the traffic. Another remark is needed to relate the present work to the differentiated services, methods of creating different levels of services for customers willing to pay higher levels. As more products are created to control networks, differentiated services will become very important for the Internet. Identification of sources that can be forced to limit their flow rate can lead to accounting for different priorities of traffic and offending flows. For example, if traffic from a certain machine is deemed high priority, the system may restrict other machines, instead of slowing down the high priority machine. Changing the architecture [13] to operate on a flow basis instead of a machine basis could also be easily done to account for a variety of traffic from each machine, each with a different priority. Hence, the techniques that we present in this paper are directly applicable to the "best effort traffic" over the Internet, but their extensions to Differentiated Services or Quality of Services environments are straightforward.

**Basic Architecture of Feed Forward ANN**

A layered feed-forward network consists of a certain number of layers, and each layer contains a certain number of units. There is an input layer, an output layer, and one or more hidden layers between the input and the output layer. Each unit receives its inputs directly from the previous layer (except for input units) and sends its output directly to units in the next layer (except for output units). Figure-1 presents a basic architecture of a feed forward ANN [1]. Unlike the Recurrent network, which contains feedback information, there are no connections from any of the units to the inputs of the previous layers nor to other units in the same layer, nor to units more than one layer ahead. Every unit only acts as an input to the immediate next layer. Obviously, this class of networks is easier to analyze theoretically than other general topologies because their outputs can be represented with explicit functions of the inputs and the weights.

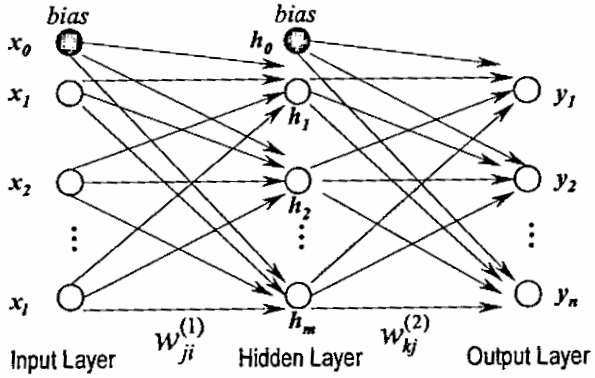



**Figure-1: Feed-forward neural network**

An example of a layered network with one hidden layer is shown in Figure-1. In this network there are l inputs, m hidden units, and n output units. The output of the jth

hidden unit is obtained by first forming a weighted linear combination of the l input values, then adding a bias,

$$a_j = \sum_{i=1}^{l} w_{ji}^{(1)} x_i + w^{(1)}{}_{j0} \tag{3.1}$$

where $w_{ji}^{(1)}$ is the weight from input i to hidden unit j in the first layer and $w_{j0}^{(1)}$ is the bias for hidden unit j. If we are considering the bias term as being weights from an extra input $x_0 = 1$, (3.1) can be rewritten to the form of,

$$a_j = \sum_{i=0}^{l} w_{ji}^{(1)} x_i \tag{3.2}$$

The activation of hidden unit j then can be obtained by transforming the linear sum using an activation function $g(x)$:

$$h_j = g(a_j) \tag{3.3}$$

The outputs of the network can be obtained by transforming the activation of the hidden units using a second layer of processing units. For each output unit k, first we get the linear combination of the output of the hidden units,

$$a_k = \sum_{j=1}^{m} w_{kj}^{(2)} h_j + w_{k0}^{(2)} \tag{3.4}$$

Again we can absorb the bias and rewrite the above equation to,

$$a_k = \sum_{j=0}^{m} w_{kj}^{(2)} h_j \tag{3.5}$$

Then applying the activation function $g2(x)$ to (3.5) we can get the kth output

$$y_k = g2(a_k) \tag{3.6}$$

Combining (3.2), (3.3), (3.5) and (3.6) we get the complete representation of the network as

$$y_k = g2\left(\sum_{j=0}^{m} w_{kj}^{(2)} g\left(\sum_{i=0}^{l} w_{ji}^{(1)} x_i\right)\right) \tag{3.7}$$

The network of Figure-1 is a network with one hidden layer. We can extend it to have two or more hidden layers easily as long as we make the above transformation further. One thing we need to note is that the input units are very special units. They are hypothetical units that produce outputs equal to their supposed inputs. No processing is done by these input units.

## Implementation

A high level view of our architecture reveals a network with a congestion management agent existing somewhere on a node in that network. This congestion management agent has both the power to read from and to influence network nodes. The nodes involved would either report the necessary statistics to the congestion management agent or the congestion management agent would poll these nodes.

## Simulated Network Topology

In the absence of the needed test-bed, NS-2, a discrete-event network simulator targeted at networking research, was used to model the network and different scenarios of network traffic. NS simulates network architectures on a packet by packet basis, giving the user the ability to monitor very specific as well as aggregate statistics about all facets of the network. This, of course, made the integration of a congestion management agent easier, but a similar design could be implemented on a real network. In our example, the network consisted of several nodes in a configuration where all of the network nodes were attempting to send data to one node. Each node attempts to send at a random bit rate. A random amount of variance is given to each node's rate to better represent traffic in a real network and possible traffic coming in from other nodes outside of our simulation. Link capacities between sending nodes were given arbitrary values (described in a later section) for testing purposes. Some links were able to handle much more traffic than other links.

## Congestion Management Agent

We create a congestion management agent containing a neural network that is trained prior to being placed in production. In our simulation, the congestion management agent is called at a regular interval in part of the simulation code. This enables the agent to easily monitor and influence traffic statistics from each node. The congestion management agent gathers information from each managed node, performs several mathematical functions normalizing the values, and makes a decision about where, if anywhere, network problems will occur. With the predicted problem in our grasp, we can take steps to stop or prevent it.

## The Simulation Network

The simulated network is arranged such that six sending nodes are connected to one receiving node through several links which direct the packets to the destination. The sending nodes produce data in a way similar to Universal Datagram Protocol (UDP) agents, sending constant bit rate (CBR) traffic with a randomized parameter to add variance to the traffic. In NS, each connection is explicitly stated and each sending agent in each node is configured to send to a particular receiving agent. To determine how fast the sender sends data [9], the packet size and a packet interval are given in the simulation script that defines the simulation run. The sender sends a packet of the designated size at

the designated interval. The receiver simply has a null agent that receives the data and sends no responses. NS Queue Monitors are attached to the queues to keep track of the status of each queue. We gather statistics such as packets received and the size of the queue during the simulation. During the simulation, the congestion management agent executes at a polling interval, monitoring the traffic and making decisions. Files are created for each node to keep track of that node's data. During each run of the network simulator, the files are extended with the new data from the latest interval. The most important part of the congestion management agent is the neural network prediction module.

**Simulation Interface**

For our agent implementation, we used a single hidden layer, feed-forward neural network. This was a compile application, so wrappers (simulation interface) were needed to control the input and output dealing with the neural network. The wrapper program was written in C and is called after the data files are updated by the simulator. The simulator halts until the C program finishes. To execute the congestion management agent, a C wrapper is first called. This is where the bulk of the calculations for the neural network program are done. It first opens the files written by the simulator which contain historical and current values for the number of packets. The program uses these values and computes the average number of packets, the variance of packets, and the third momentum given the appropriate polling interval. In the first iteration, the average is the current number of packets and the variance and third moment are zero. These values are then normalized for the neural network using a basic normalization function. The normalized values are then combined into one input file to the neural network package for a decision. The neural network program is executed using new input files and the output is rendered in yet another file. This file is read by the C wrapper and converted into a readable format for the simulator to process.

**ANN as a Control Management Agent**

The neural network used by the congestion management agent has 4*n input nodes, 1 hidden layer containing n nodes, and n nodes in the output layer. The 4*n input nodes correspond to the n traffic generating nodes in the network simulation; there are four input nodes for each node in the network simulation corresponding to the average number of packets, variance, and third moment for each monitored node. To stress the importance of adjacency relationships between nodes in the data network, we placed an additional optimization of the structure of the neural network. The weights were pruned to the point in which the neural network reflected the connectivity of the actual network. The n nodes in the hidden layer also represent active nodes in the data network. Instead of providing a fully connected environment between the input layer and the first hidden layer, we only allowed connectivity from input neurons that represent nodes adjacent to represented nodes in the hidden layer. The hidden layer is representative of the participating nodes in the data network. The statistical data regarding each node is provided to the hidden node representing the actual node as well as to the hidden nodes representing the actual node's neighbors. This is continued for all first layer nodes of the neural network. As a result,

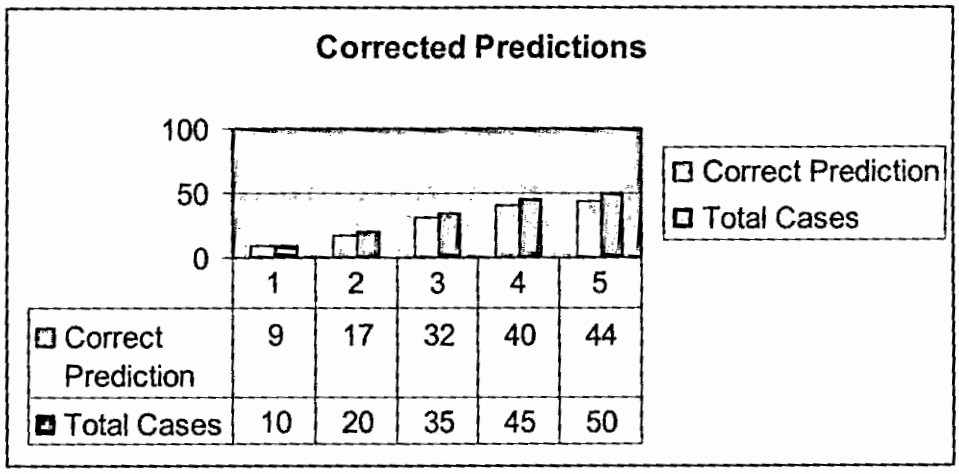
the statistical information from node 1 is given to both the hidden node representing node 1, and the hidden node representing node 5 (1's neighbor.) This process is important in realizing the relationships between adjacent nodes in a data communications network. The output of the neural network is a mask representation of which nodes have been suspected of causing the problem.

**Patching the NS-2**

In NS-2, both an interval and a packet size are provided for agents sitting at the sending nodes to determine bandwidth used. The agent will send one packet at every interval, therefore the smaller the interval, the higher the bit rate. If our neural network predicts that a particular node will be responsible for congestion, we conclude that the predicted problem source is using too many resources. To correct the predicted cause node traffic rate, we add to its sending interval a small ▲t, thus reducing its bit rate. This delta was chosen to be small with respect to the simulation time scale, because we do not want to take the chance of over-correcting or even worse, applying a large correction to the wrong node if our prediction was wrong. To take into account the small ▲t, the interval in which our congestion management agent executes also is relatively small. Therefore many of these small corrections can be applied which corrects a problem slowly without drastically changing any one node's level of service.

**Results**

A general breakdown of the results can be found in the graph of Figure which shows our current application detects and corrects congestion in about 90% of the cases. Our tests include cases in which corrections to one node are required, corrections to multiple nodes are required, and some where no correction is required. Failing includes either missing congestion or predicting congestion when there is none. We ran thirty-one network simulations. Roughly 33% of the cases were simulations of a network without congestion problems. In these cases we would want our control agent to realize that it does not have to do anything. The detector realized that there was no correction needed in all but 1 case. In this isolated case our agent unnecessarily applied a single small correction to a single node. The correction that the control agent applied was with a single ▲t, and therefore was minimal. About 66% of the total cases had various levels of congestion in various locations in the network of the congested cases, we were able to predict the cause and fix the problems 89.99% of the time.




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Correct Prediction | 9 | 17 | 32 | 40 | 44 |
| Total Cases | 10 | 20 | 35 | 45 | 50 |

Figure 2: The bar chart shows the neural network prediction of congestion and the total number of cases

The results shows that 75% of the time we detected congestion, we were able to fix the problem before packets were dropped in the network. These were truly remarkable results, because the congestion was completely eliminated before it occurred. In the cases that we could not stop packets from dropping, we were able to return the network to a stable state within few seconds after packets began to drop.

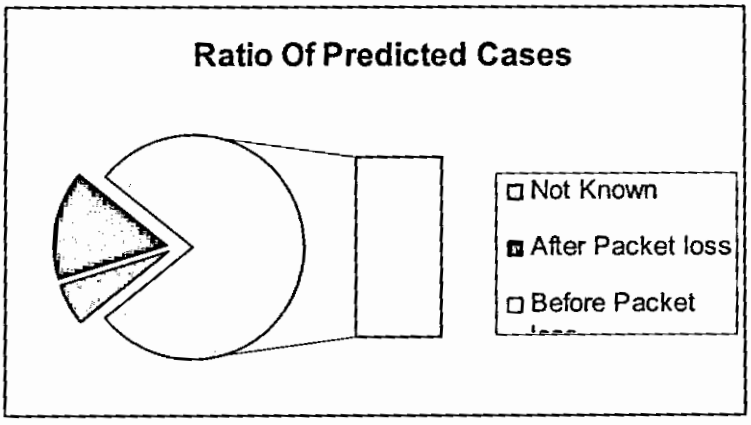



Figure 3: The pie chart describes that around 75% of cases are predicted before any packet loss.

Finally, in the cases that our detector missed the threat of congestion there was a common characteristic. The neural network had trouble detecting congestion when a single node in a particular part of the data network caused a problem. This probably can be improved upon close examination of the training patterns and structure of the neural network.

| Training Errors | | Testing Errors | |
|---|---|---|---|
| Input Combination | Training NRMSE | Input Combination | Training NRMSE |
| 1 | 0.211 | 1 | 0.255 |
| 2 | 0.227 | 2 | 0.239 |
| 3 | 0.217 | 3 | 0.254 |
| 4 | 0.202 | 4 | 0.229 |
| 5 | 0.198 | 5 | 0.225 |
| Average | 0.211 | | 0.240 |
| standard Dev | 0.010 | | 0.014 |

**Figure 4: This table shows the summary of training and testing results of ANN**

## 3.Conclusion

The goal of this paper has been to apply neural networks to the problem of forecasting the congestion state in a packet switching network. We have seen that neural networks produced fairly accurate forecasts. In this work, we have tried to demonstrate that a neural network is a viable method of implementing a realistic forecasting application for data communication networks. We have illustrated, through the use of a network simulator, that a neural network can be used to achieve great accuracy in predicting network congestion problem. Wireless networks need a special concern for congestion management. This work is applicable to both wired and wireless environment. We realize many more problems exist that for which neural network forecasting approach is applicable, but predicting congestion is just the initial step towards our research goals. When structural information of an actual data network is used to form the connections between layers of the neural network, this special design forces the neural network to consider the relationships only of those nodes that we think are important. A learning mechanism can be of great value for a network manager. The generalization power of a neural network particularly is appropriate because of the unpredicted variance of parameters that the network manager encounters. Neural networks are an appropriate mechanism for decision making in pro-active network management and should be the subject of more research.

## 4. References


[1]. Feng, Wu Chang. (1999) Improving Internet Congestion Control and Queue Management Algorithms. PhD thesis, The University of Michigan, Ann Harbor, Michigan. Computer Science and Engineering.

[2]. R.E.Blahut, "Thory and practice od Error Control Codes" Addison Wesley, MA, 1984.

[3]. S.Floyd, K.Fall, "Router Mechanisms to support End-to-End Congestion Control", Technical report, ftp:\\ftp.ee.lbl.gov/papers/collapse.ps

[4]. V.Jacobson,"Congestion Avoidence Control", ACM SIGCOMM'88,August 1988,Stanford, CA, pp.314-329.

[5]. W.Stevens., "TCP Slow Start, Congestion Avoidence, Fast Retransmit, and Fast Recovery Algorithms.",RFC2001, January 1997.
[6]. Murtaza, Mirza and Shah, Jayneen. (1995) Reducing data movement in client/server systems using neural networks. http://www.informs.org/Conf/NO95/TALKS/MA19.3.html.
[7]. Ogier, Richard, Plotkin, Nina T., and Khan, Irfan. (1996) Neural network methods with traffic descriptor compression for call admission control. In Proceedings of the Conference on Computer Communications. IEEE Infocom.
[8]. Ramakrishnan, K and Floyd, S. (1999) "A Proposal to add Explicit Congestion Notification
[9]. Alexander, D. Scott. (1998) The switchware active network architecture. IEEE Network Spe cial Issue on Active and Controllable Networks, 12(3):29-36.
[10]. Allman, M., Paxson, V., and Stevens, W. (1999) TCP Congestion Control. Request for Comments (RFC): 2581. April 1999.
[11]. Bagel, Prasad et al. (2000) "TCP Rate Control," Computer Communication Review, Jan, vol. 30, num=1.
[12]. Bajaj, Sandeep et al. (1999) "Improving Simulation for Network Research," Technical Report 99-702, University of Southern California.
[13]. K. L. Calvert. (1999) Architectural framework for active networks. RFC Draft.
[14]. Campbell, Peter K., Christiansen, Alan, et al. (1997) Experiments with simple neural networks for real-time control. IEEE Journal on Selected Areas in Communications,15(2):165-178.
[15]. Clark, Dave. (1996) Computer Networking: A Systems Approach. Morgan Kaufmann Publishers. (ECN) to IP," Request for Comments: 2481.
[16]. Stallings, William. (1999) "SNMP, SNMPv2, SNMPv3, and RMON1 and 2," 3 ed. Addison Wesley Longman, Inc., Reading, Massachusetts.