VISUALIZATION OF LEARNING IN NEURAL NETWORKS

By

OSMAN ÖZ

Bachelor of Education

Marmara University

Istanbul, Turkey

1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1997

VISUALIZATION OF LEARNING

IN NEURAL NETWORKS

Thesis Approved:

J Chandler

Thesis Adviser

Wm Nick Street

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation and respect to my thesis advisor, Dr. John P. Chandler, for his patient guidance, thoughtful suggestions, and friendship. My sincere appreciation and respect extends to my other committee members Dr. K. M. George and Dr. N. Street, whose guidance, suggestions, and friendship are also invaluable.

Moreover, I wish to express my sincere gratitude to the Simav Technical Education Faculty of Dumlupinar University, Turkey for their financial support during my study at Oklahoma State University.

I would also like to thank my friends, Asli Korkut, Kezban Sahiner and Pyong Wook for their precious suggestions, strong encouragement at times of difficulty, and support.

Finally, I would like thank my parents for their patience, understanding, support, and encouragement.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

I

INTRODUCTION

Introduction of "hidden layers", nonlinear activation functions, and error backpropagation sets a starting point of a new era in the history of artificial neural networks by overcoming the theoretical limitations of perceptrons and earlier linear networks. Representing intermediate processing and computing nonlinear recognition functions by the popular Backpropagation Algorithm [18] triggered neural networks research. Backpropagation applied to the practical applications for training multi-layer feed-forward networks. A neural network is a system of neurons linked to each other by weighted connections. Training a neural network is the process of adjusting the connection weights to minimize the difference between the desired output and the produced output. Whether the neural network is created by hardware or software, it is a mathematical system consisting of information that is represented by a large number of real-valued parameters. This structure creates the fundamental problem of the neural networks: poor understanding of the system and training process. Applying visualization techniques, such as representing the parameters by different graphical objects with different attributes, can improve the understanding of the training process and the system.

This paper discusses scientific visualization and simulation methods applied to neural networks and presents the simulator (VisualNN) designed for visualizing the neural network, its parameters, and training process.

The next chapter provides a brief introduction and information about neural networks, learning in neural networks, and learning algorithms used in the simulator. The third chapter discusses the simulation and visualization methods. Chapter IV introduces the visual neural network simulator VisualNN 1.0 and discusses its design and implementation issues. Chapter V presents how it is used with an example network. The final chapter contains the results and discusses what else can be donc for improving the performance, efficiency, and applicability of the simulator. Appendices contain the forms and source code of the simulator.

II

NEURAL NETWORKS AND LEARNING

Neural Networks Fundamentals

A neural network is a system of highly interconnected neurons. A neuron simply has two parts: an adder function Σ and an activation (transfer) function F (Figure 1). A neuron combines the inputs and sends an output value based on the nature of the activation function [19]. The output of a neuron can be described as follows:

$$O = F(\sum_{i} \mathrm{W_i\,I_i})$$

Selection of an activation function depends on the problem. In this study, the popular sigmoid function which takes any input and squashes the output into the range 0 to 1 is used [2]. This function is differentiable and works according to the following expression:

$$O = \frac{1}{1 + \mathrm{e}^{-i}}$$

where O is the output and i is the input of the function ($\mathrm{W_i\,I_i}$).

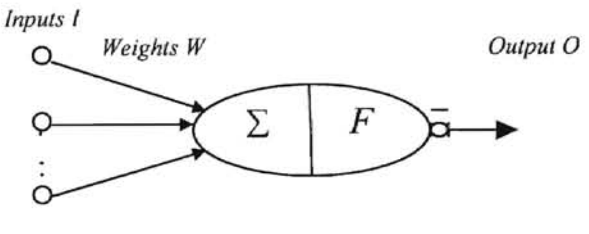


Figure 1 A simple neuron

The output of a neuron can be connected to the input of another neuron via connections weights. Neurons are usually organized into groups called *layers.* Outside connection of a network is provided only by the input and output layers. That is the reason for calling the intermediate layers *hidden layers*. A network with no feedback from one layer to another is called a *feedforward* network. A neuron receives signals from previous layer neurons and sends a signal to the next layer neurons. If each neuron in a layer has a connection to each neuron in the next layer, this type of network is called *fully-connected.* Generally, the behavior of a network is determined by the connection weights. A learning rule modifies the connection weights and determines the network behavior. The learning method we used in this study is called *supervised learning.* It modifies the connection weights by the information gathered from the comparison of the desired output and actual output [6] [19].

Learning in Neural Networks

Regardless of the type, all neural networks learn by example, including unsupervised networks which are not told when they are right or wrong. Since we are interested only in supervised networks, this needs to be explained. Networks with supervision are trained with pairs of information: an input pattern and an output (desired) pattern. Training is a process of presenting each pair, or exemplar, to the network, one at a time. The network produces an output pattern after the presentation of the input pattern, and compares the produced output with the desired output. The learning algorithm adjusts the weights in order to reduce the difference between the desired output and the produced output. This process is repeated for all the training pairs until the network learns all of the pairs as well as it can. Presenting all the training patterns to the network once is called an *epoch.*

Learning Algorithms

Backpropagation Algorithm with Momentum Parameter

Standard backpropagation algorithm iteratively adjusts the connection weights to minimize the difference between the desired output and actual output by propagating the error backward through the network. The algorithm contains two main parts: a forward pass and a backward pass. The forward pass takes the input pattern *I*, presents it to the network, and produces the network output pattern *O* after intermediate processing. The backward pass finds the total error *E* in the output by the following:

$$E^k = (O^k - T)^2$$

where E^k is the error at *k*th representation, *O* is the produced output, and *T* is the target output of one training example.

$$E^k(w) = \sum_{i=1}^{n} (O_i^k - T_i)^2$$

where *n* is the number of output neurons.

The total error over the complete training set is then calculated :

$$E(w) = \frac{1}{m} \sum_{k=1}^{m} E^k(w)$$

where *m* is the number of training examples.

The algorithm just computes the partial first derivative of the overall error function E with respect to each weight w_i in the network. When the derivatives are given, a minimization process (gradient descent) on E can be carried out. In general, the connection weights can be modified at the kth input/output pair by the following:

$$w_{ij}(k) = w_{ij}(k-1) - e(k)\frac{\partial E^k}{\partial w_{ij}}$$

where e is the learning rate (step size).

We need an appropriate learning rate for the modification of the weights. This creates the biggest problem for the algorithm. In order to get stable convergence, a small learning rate has to be chosen, but having a small learning rate leads to very slow learning [9]. On the other hand, too large a learning rate leads to an unstable algorithm. A momentum term was introduced to deal with the problem. The weight at time t-1 is added to the weight at the time t with the momentum parameter [16]. The adjustment of the weights can be described as

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

where Δw_{ij} is a matrix representing the change in matrix w_{ij}.

The matrix Δw_{ij} is computed as follows:

$$\Delta w_{ij}(t) = eE_j I_i + \Theta \Delta w_{ij}(t-1)$$

where Θ is the momentum factor, E_j is the error vector, and I_i is the input vector.

Inclusion of the weights at the time *t-1* does not mean that time is incorporated into the network. It only means that weight modification may depend on the previous modification made.

The Quickprop Algorithm

Fahlman's Quickprop algorithm [7] proceeds like the backpropagation algorithm except it keeps copies of the error derivatives of the previous epoch and the differences between the current and previous values of the weights. Therefore, the weight updates differ from the backpropagation algorithm. The computation of the weights uses only the information local to the weight being updated:

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

$$\Delta w_{ij}(t) = e\frac{S_{ij}(t)}{S_{ij}(t-1) - S_{ij}(t)}\Delta w_{ij}(t-1)$$

where $S_{ij}(t)$ and $S_{ij}(t-1)$ are the current and previous values of the error derivative.

The basis for this weight updating formula lies on two assumptions: first, the error vs. weight curve for each weight can be approximated by a parabola whose arms open upward; second, the change in the slope of the error curve as seen by each weight is not affected by all of the other weights that change at the same time [7].

Delta-Bar-Delta Algorithm

Delta-Bar-Delta is another heuristic approach to speed up the convergence of the training process for neural networks. This method assumes that the learning rate for one weight may not be appropriate for another weight. Thus, every connection weight should have its own learning rate [11] [16]. This is achieved by the following method: when the sign of the weight update changes for several steps, the learning rate for that weight is decreased. If the sign is the same for several steps, the learning rate for that weight is increased [16].

Generally speaking, each weight may be quite different in terms of its effect on the overall error surface. Changing the learning rate continually for each connection over time may speed up the connection weight convergence. This method increments the learning rate linearly to prevent the step taken from being too large. Decrementing is done geometrically to ensure the step size is positive.

Conjugate Gradient Algorithm

This algorithm differs from the three methods explained earlier in several ways. This is a minimization approach to solve the slow convergence problem of the standard backpropagation algorithm. The algorithm requires line search in each iteration [9]. Line search is used to find the minimum point along the search direction. For a gradient descent method, the search direction can be found from the gradient vector.

Given an initial point x, and a search direction d, the line search problem is the minimization of the following:

$$F(\alpha) = F(x + \alpha d)$$

where F is function to be minimized, α is a scalar value which has to be found in order to minimize the function.

A conjugate gradient method can be easily adapted to neural networks since it only involves function evaluations and/or gradient calculations [20]. The function we are going to evaluate is the error function which is gathered from the difference of the produced output and the desired output after the *forward pass* of the backpropagation algorithm. The calculation of the derivative of the error function with respect to the weights is used to find a search direction. Another difference of this method is that the error function is based on the entire training set. This method is called *batching* where weights are updated after an epoch instead of modification after each input/output pattern presentation (incremental updating).

III

NEURAL NETWORK SIMULATION AND VISUALIZATION

Today many neural network simulators are publicly available to researchers, most of them free of charge. However, almost every research group prefers to develop its own simulator. This is because of efficiency, flexibility, correctness, and maintainability concerns [12]. Although it is limited in the maximum size of the network that can be created, we tried to develop an efficient, flexible, maintainable, and correct simulator. VisualNN is designed for flexibility and user-friendliness. For correctness, we tried to minimize the duplication of the code. Visualizing is a key to find obvious problems of the simulations [12]. Thus, it has been helpful to correct the problems as well as to understand the learning process.

Neural Network Visualization Methods

This section describes some of the visualization methods that have been used to understand the learning and structure of neural networks. The methods developed for visualizing neural networks are not limited to those we explain in this chapter. There are other visualization methods such as trajectory diagrams and hyperplane diagrams [5]. They are either out of the scope of this study or developed for different purposes. We will explain the visualization methods used in VisualNN in the next chapter.

Hinton Diagrams

One of the pioneering methods developed to visualize neural networks, almost as old as the Backpropagation algorithm, was the Hinton diagram [10]. It provides a visual display of the connection weights and bias weights related to a particular neuron in a network. Figure 2 shows an example network. The diagram shows the signs and magnitudes of all incoming and outgoing connection weights and bias weights. A box represents each weight on the diagram. The sign of the weight is represented by the color of the box (white-positive, black-negative) while the magnitude of the weight is represented by the area of the box.

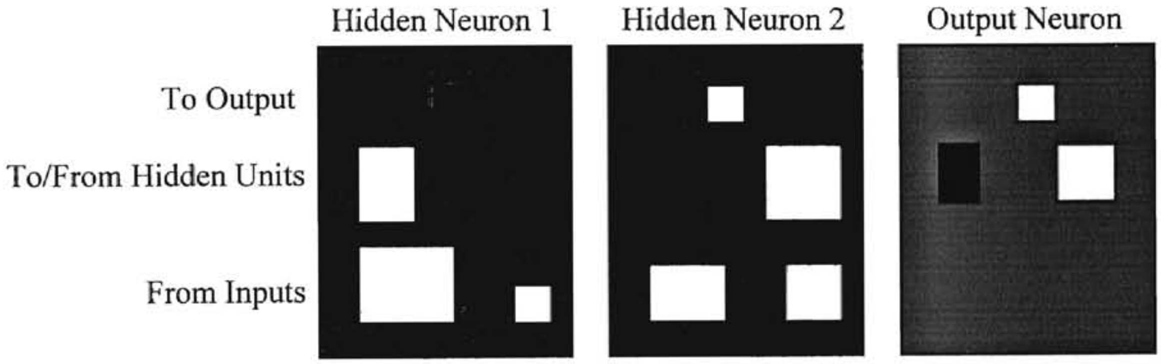


Figure 2 An example network with Hinton diagrams

Even though Hinton diagrams help to understand a neural network by making it easy to see the signs and magnitudes of the connection weights and bias weights that contribute to a neuron's activation, it is a weak method for visualization, because the topology of a network is not clear from a set of Hinton diagrams.

Bond Diagrams

The bond diagram, like the Hinton diagram, provides a visual display for the sign and magnitude of each connection weight and bias weight in the network [22]. Figure 3 shows an example bond diagram. In this method, each neuron represented by a disk. For hidden layer and output layer neurons, the size of the disk shows the magnitude of the neuron's bias weight. The connection weights are represented by bonds linking the disks. The sign of the weight is represented by different colors while the magnitude of the weight is represented by the amount of the bond. The topology of a network is apparent from the bond diagrams but it is not easy to see how the connection weights add up against a bias weight. This is due to the representation of weights and biases with different objects.

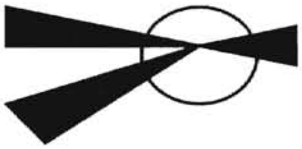


Figure 3 A bond diagram

Lascaux Visualization Method

Craven and Shavlik developed a neural network visualization tool, called Lascaux [5]. We refer to the methods used in the interface provided by Lascaux as Lascaux visualization methods. Visualization is applied to the forward propagation of activations, the backward propagation of error, and changes to the connection weights and bias weight of the network. A neuron is represented by a box (Figure 4). The activation of each neuron is represented by a bar-like display. The level of the display filled with black represents the activation of the unit. The display is completely filled if the neuron is active. The display is not filled at all if the neuron is inactive.

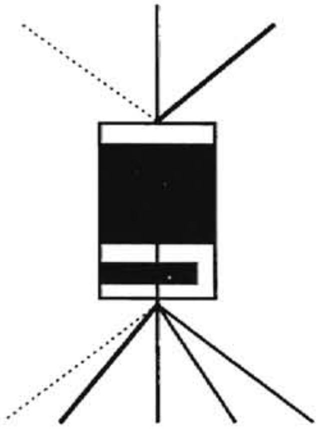


Figure 4 A neuron in Lascaux

Connection weights and bias weights are represented by lines. The sign of the weight is represented by the type of the lines (solid line-positive, dashed line-negative) while the magnitude of the line is represented by the width of the line. Activation display is divided into two and the bar on the left is used for representing the error.

IV

VisualNN NEURAL NETWORK SIMULATOR

As mentioned above, the objective of this study was to develop a user-friendly neural network simulator. VisualNN was developed using Microsoft Visual Basic 4.0 for Windows, and is a 32-bit application that runs under Windows NT and Windows 95. This means that the simulator is an event-driven application. The simulator contains a visual screen that shows the changes in the network graphically, a value screen that shows the changes in numbers, and an error graph screen that shows the change in the error value of the system after each iteration.

The simulator is very flexible in several ways. First, the user is able to select the training algorithm from the four algorithms mentioned above. Second, the user is able to edit the training set, network parameters, and test data at run-time. The user is also able to decide the number of hidden layers, either 1 or 2, the number of neurons in each layer, and whether neurons have bias connections or not.

At the beginning, a network must be created. In order to train the network, a training set must be created. The trained network then can be run either on the training set or on the test set. For further convenience all of the created information, either network or data files, can be saved for later usage. Saving the network would be especially helpful for the user. A network file contains the structure of the network including the number of hidden layers and number of neurons in each layer, bias information, connection weights, and/or bias weights. The training and test files contain sets of input/output patterns.

Design and Implementation of the Simulator

VisualNN consists of three main windows: main window which includes the menus, error graph window, and network values window. Figure 5 shows the main windows of the simulator. There are nine other windows supporting the menu and simulation events.

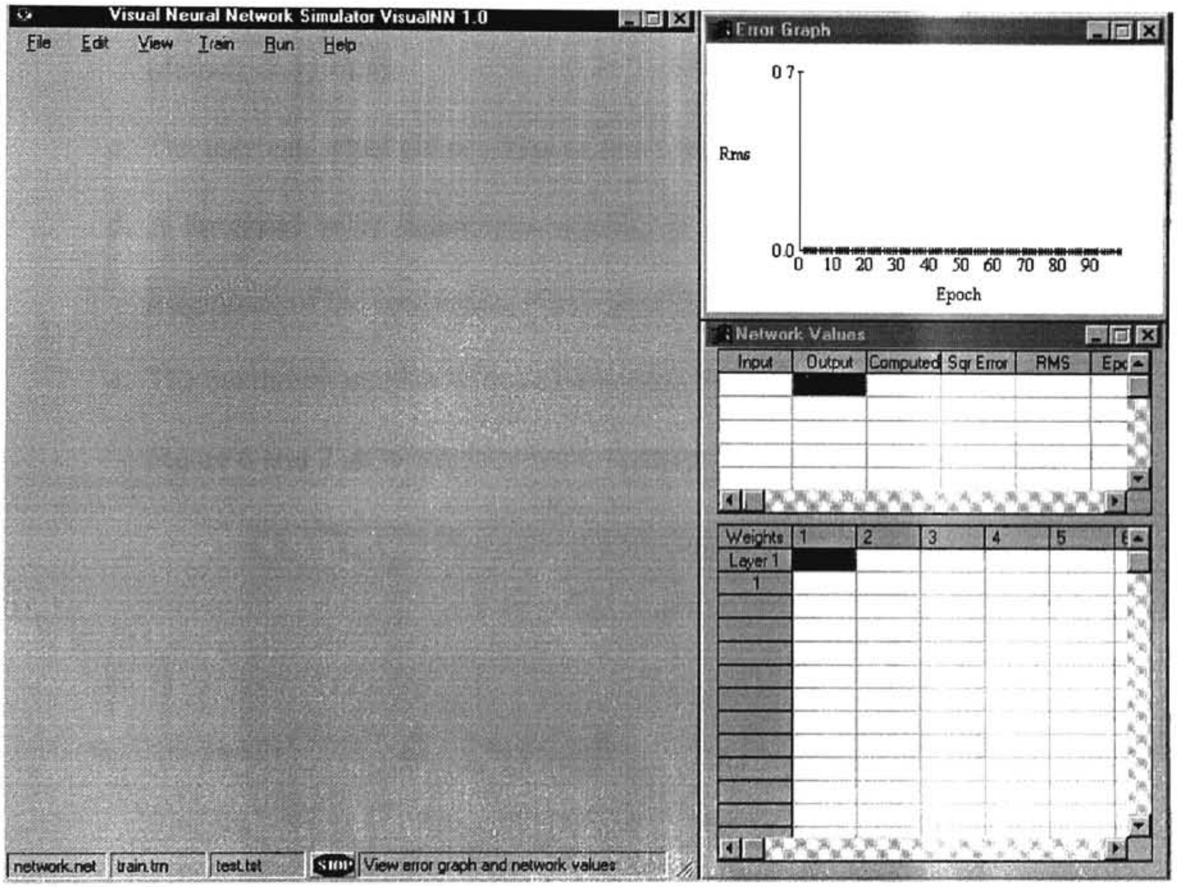

Visual Neural Network Simulator VisualNN 1.0
File Edit View Train Run Help
Error Graph
0.7
Rms
0.0
0 10 20 30 40 50 60 70 80 90
Epoch
Network Values
Input Output Computed Sqr Error RMS Epc
Weights 1 2 3 4 5
Layer 1
1
network.net train.trn test.tst
STOP
View error graph and network values

Figure 5 Main windows of the simulator

The next two sections contain the decisions involved in implementation and design of the simulator.

Design Decisions

a. One hidden layer is always capable of solving any problem for neural network applications [9]. Two-layer networks are rarely used, although a two-layer network can solve a problem more economically. The user can select a one or two-hidden-layer architecture (the default is one hidden layer).

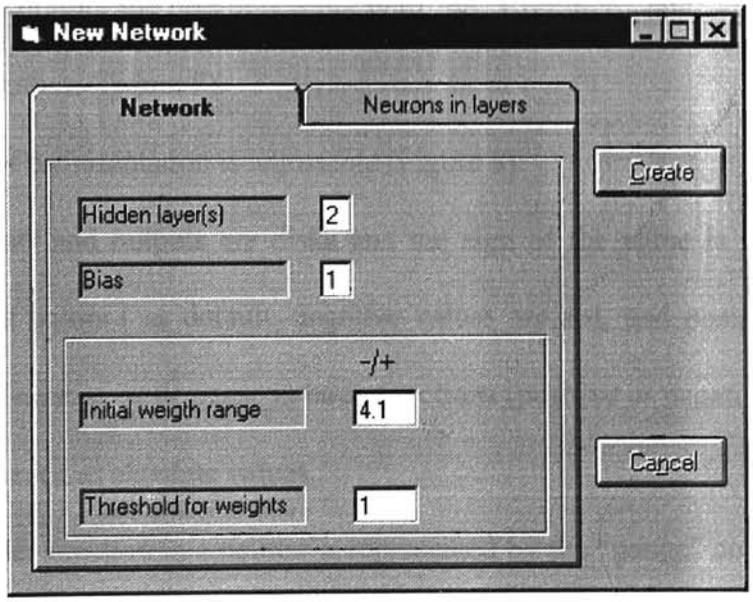
b. The user is able to set the random initialization range of the connection weights (default -0.3, +0.3).

c. The user can select either a bias or non-bias option (the default is non-bias).

d. A threshold value determines whether to show the connection or not. If the magnitude of the connection is less than this value, the line is not drawn.

e. The maximum number of neurons is eight for each layer.

Figure 6 and 7 show the process of creating the maximum possible network.



Hidden layer(s) 2

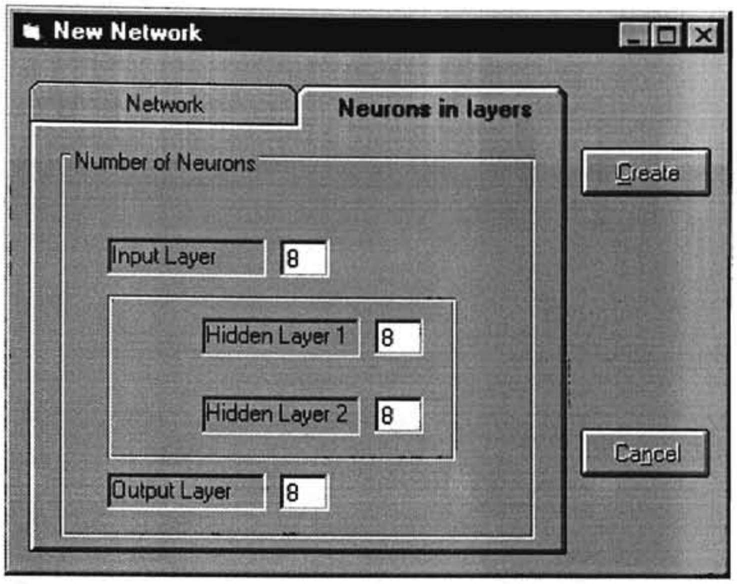
Bias 1

-/+

Initial weigth range 4.1

Threshold for weights 1

(a)



Number of Neurons

Input Layer 8

Hidden Layer 1 8

Hidden Layer 2 8

Output Layer 8

(b)

Figure 6 New network window

f. The user is able to see the error graph and the network values as well as the visual network.

g. The training and running process can be done by a “Step” or a “Full” option. In the step option, the program waits until the “Continue” button is clicked after each iteration (Figure 8).

h. Speed of the simulation is adjustable (Figure 8).

i. All inputs and outputs are disks and the sign of the value is represented by different colors (as default, negative values are red, and positive values are green). Increase or decrease in both directions (positive or negative) is reflected to the screen as absolute values.

j. The adder function in a neuron is represented by the “gauge” control of Visual Basic where we can dynamically change the min and max values. The reason for this approach is that we cannot know in advance the total sum of the incoming signals.

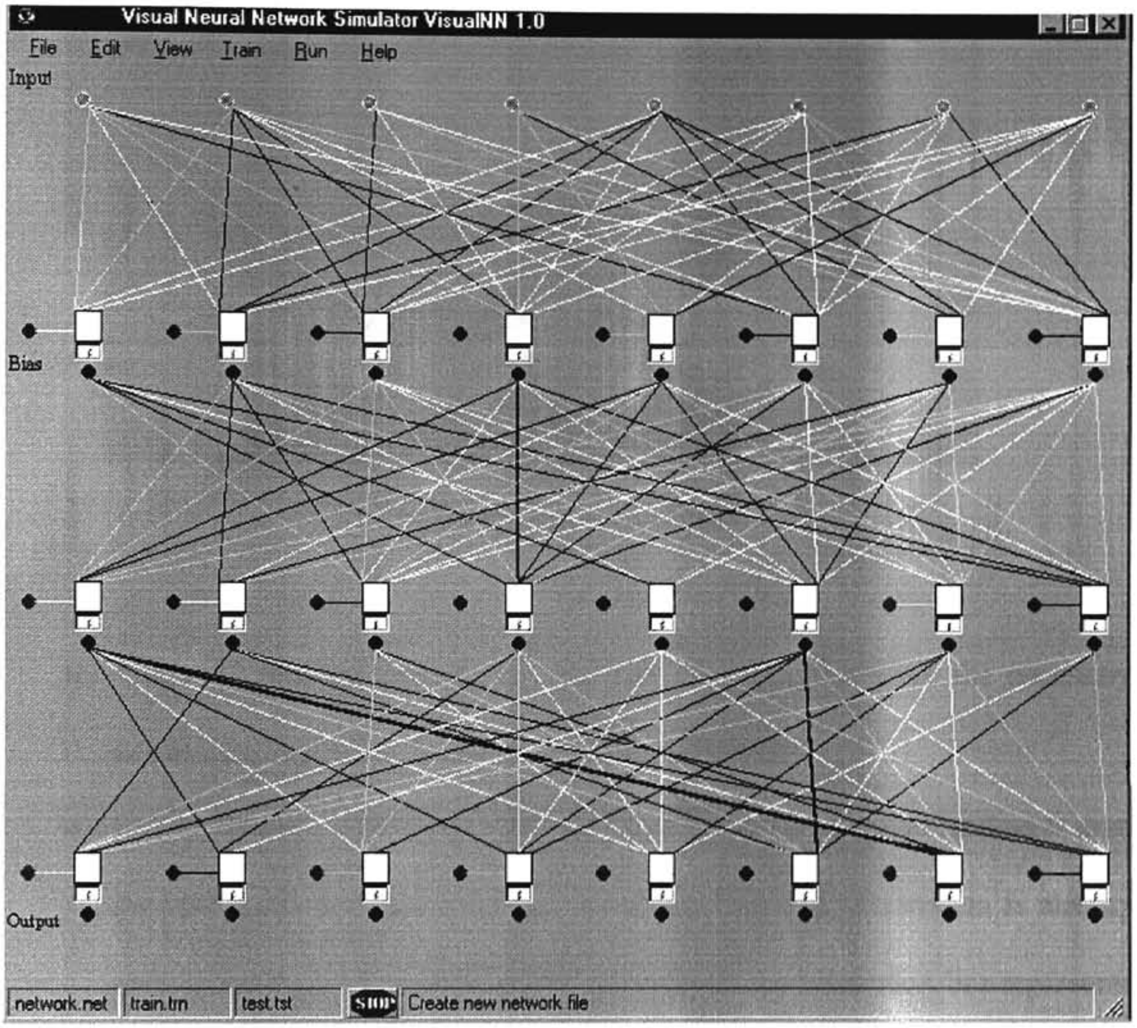
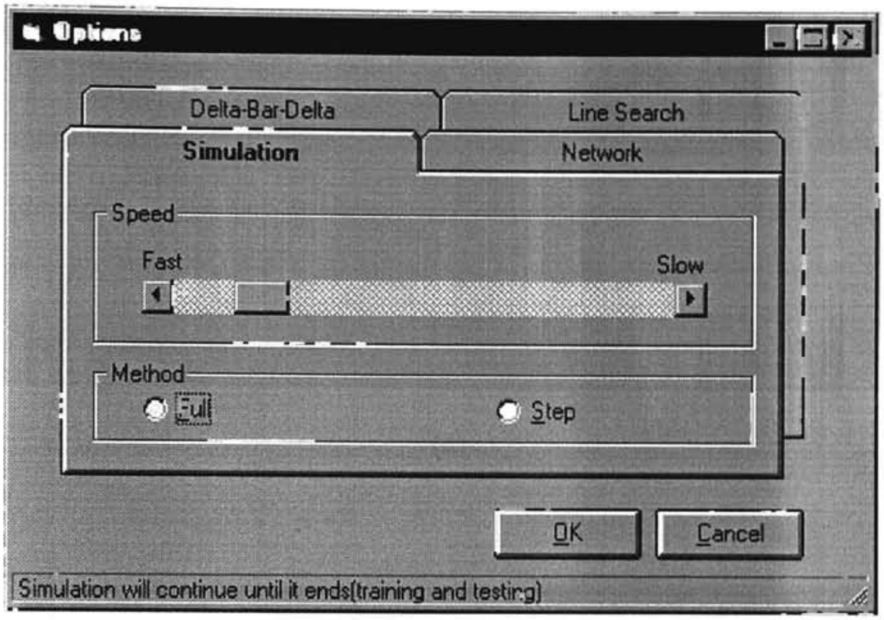


Figure 7 Maximum possible network

k. Scaling for the neuron inputs and outputs is the same.

l. The user is able to stop the simulation at any time and to see everything in the simulation until the "Close" option is selected in the "File" menu or the "Stop" icon is clicked.

m. The user is able to save the network, training set, and test set, and to open previously saved files (Figure 9).



Speed

Fast Slow

Method

● Full ○ Step

Simulation will continue until it ends(training and testing)

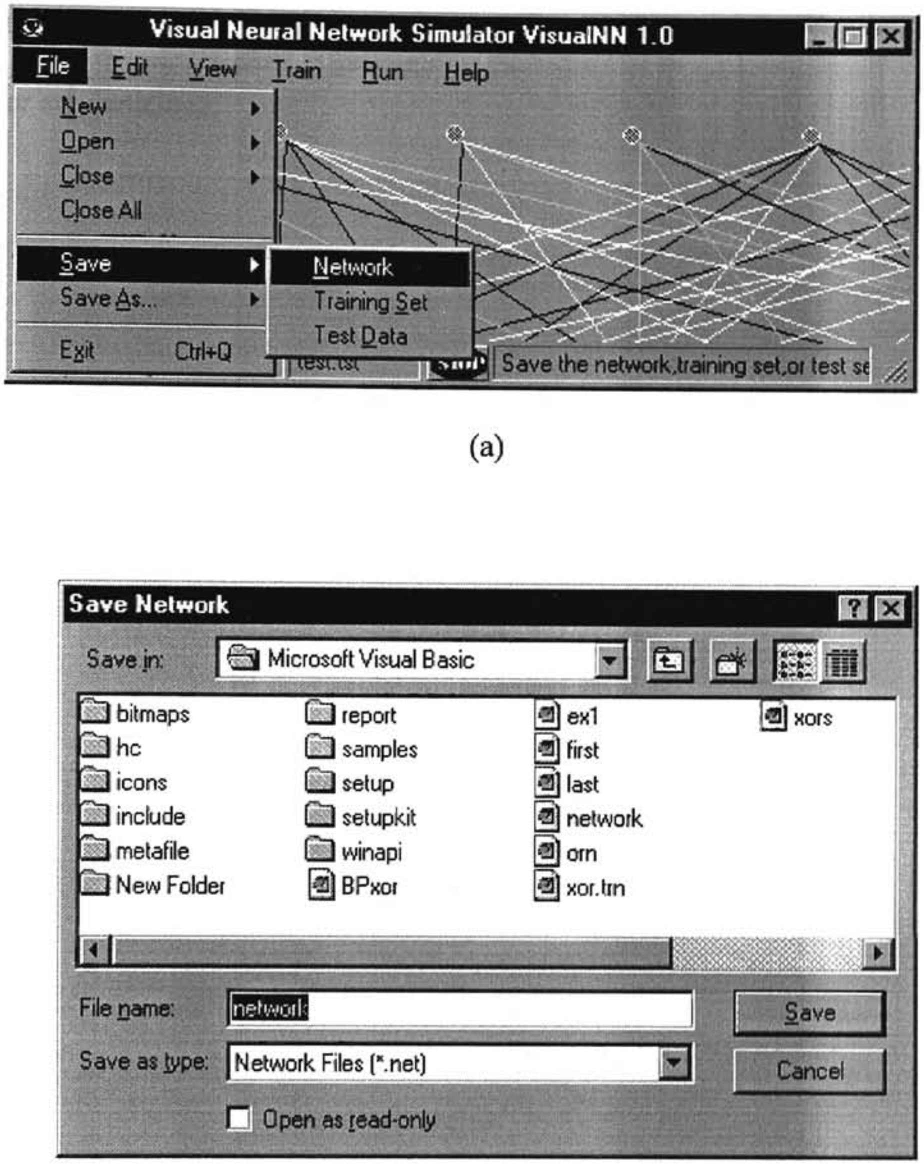
Figure 8 Options window

n. When the training algorithm is selected, the user must enter the necessary parameters, depending on the algorithm (Figure 10).

o. Unlike the other parts of the network, a non-linear enlargement method is used for connection and bias weights. As seen in Figure 7, the system is already complicated. For the sake of clarity, the color of the connection line represents the value of that connection (Figure 11).

p. The user is able to create the training and test set as well as to edit the network, training, and test files (Figure 12).

q. A window in the Help menu shows the instructions related to the use of the simulator (Figure 13).



(a)

File name: network

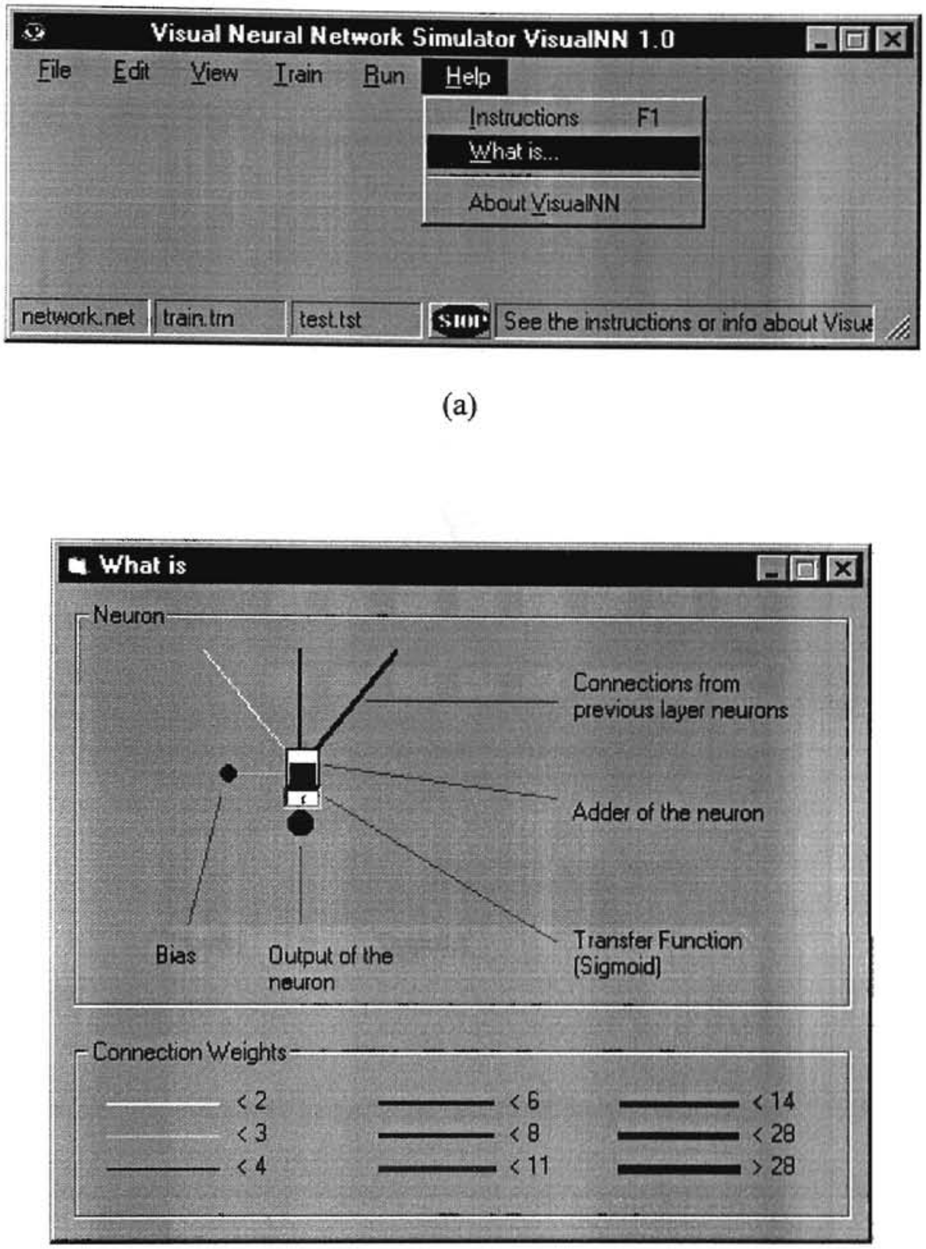
Save as type: Network Files (*.net)

☐ Open as read-only

(b)

Figure 9 File saving process

Figure 10 shows the parameters window for the Backprop algorithm. Figure 11 shows the Whatis form of the simulator where the visualization methods of the simulator are explained. Figure 12 shows the training or test set creation window. Figure 13 shows

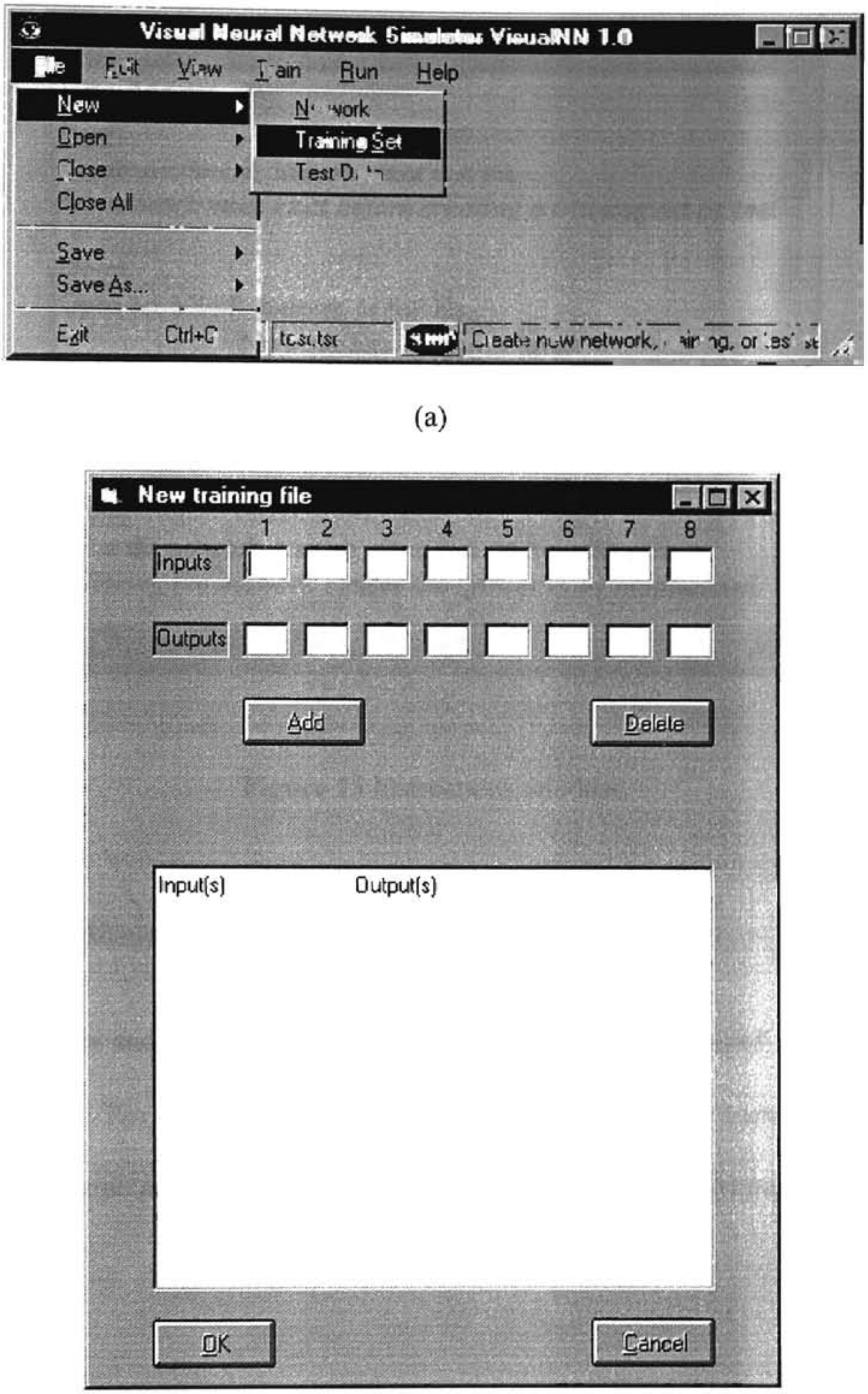


See the instructions or info about Visu

(a)

(b)

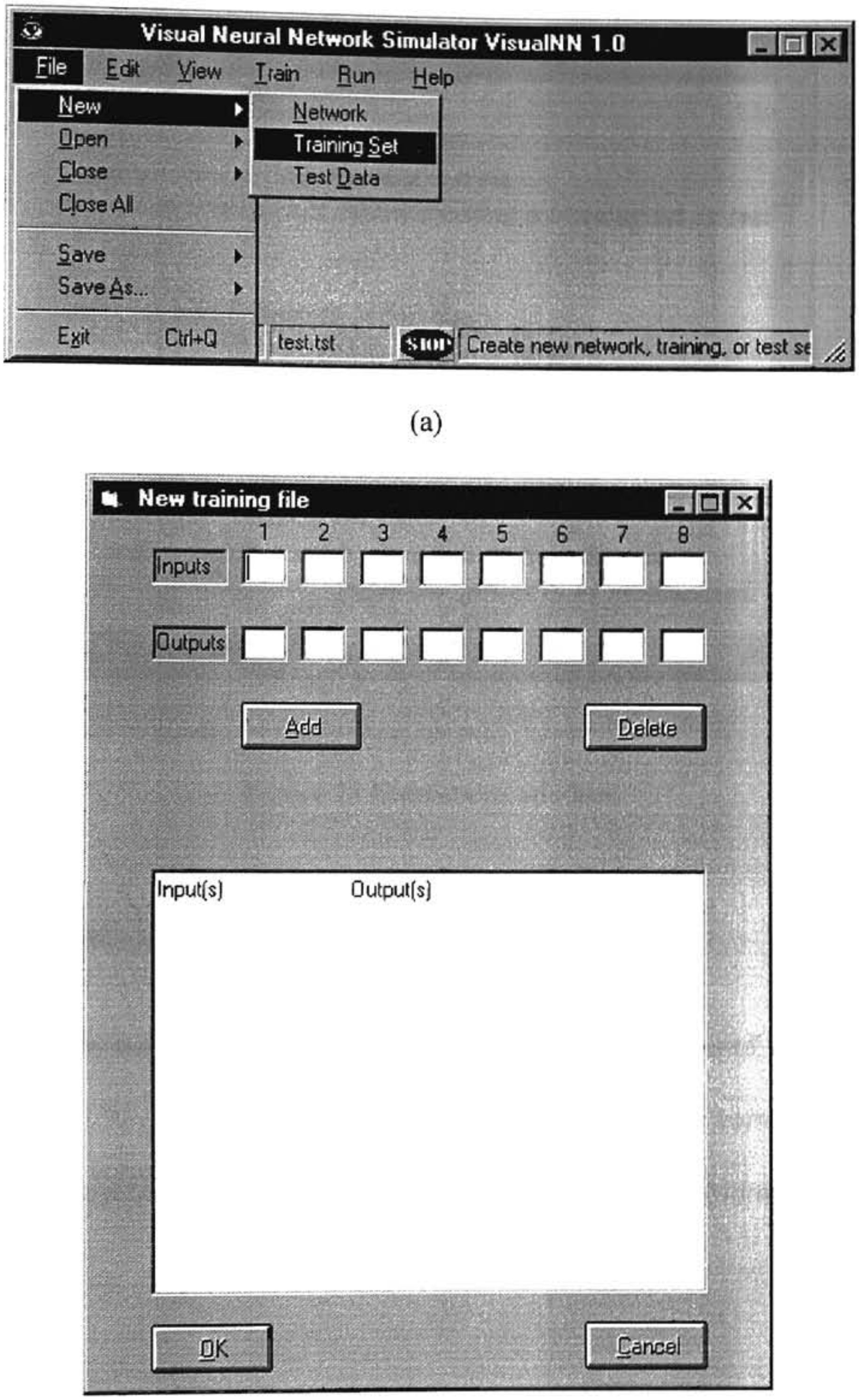
Figure 11 Whatis window of the simulator



(a)

(b)

Figure 12 New training set creation window

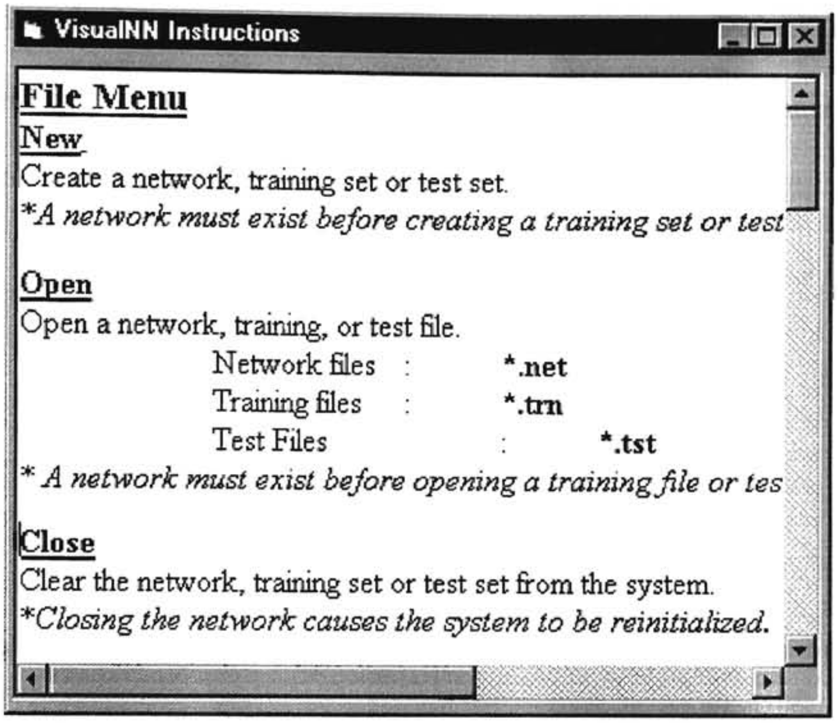


Create new network, training, or test se

(a)

(b)

Figure 12 New training set creation window



File Menu

New

Create a network, training set or test set.

**A network must exist before creating a training set or test*

Open

Open a network, training, or test file.

Network files	:	***.net**
Training files	:	***.trn**
Test Files	:	***.tst**

* *A network must exist before opening a training file or tes*

Close

Clear the network, training set or test set from the system.

**Closing the network causes the system to be reinitialized.*

Figure 13 Instructions window

Implementation Decisions

Efficiency and correctness were the most important concerns of the development of the simulator. The following decisions were made to ensure the correctness, memory efficiency (dynamic arrays), and efficient use of the storage space (minimum number of parameters):

a. An object pointer array is assigned to each object type group (i.e. lines) for fastest access to each object, e.g.,

```
Set LinesPointer2(1, i) = mainFrm.Line1(16 - i)
Set CurrentAdder = AddGauge2(i - 1)
```

b. Instead of having only one matrix for all the connection weights and bias weights, we use different matrix arrays for each layer's connection weights and different vector arrays for each layer's bias weights.

```
' Weight array pointers for each layer
Public W1(), W2(), W3() As Single

' Bias weight array pointers for each layer
Public bW1(), bW2(), bW3() As Single
```

c. Instead of having fixed size arrays, we use dynamic arrays to avoid unnecessary allocation of memory.

```
ReDim W1(1 To inputNeurons, 1 To h1Neurons)
If bias = 1 Then
    ReDim bW1(1 To h1Neurons)
End If
If hLayers = 2 Then
    ReDim W2(1 To h1Neurons, 1 To h2Neurons)
    ReDim W3(1 To h2Neurons, 1 To outputNeurons)
    If bias = 1 Then
            ReDim bW2(1 To h2Neurons)
            ReDim bW3(1 To outputNeurons)
    End If
Else
    ReDim W2(1 To h1Neurons, 1 To outputNeurons)
    If bias = 1 Then
            ReDim bW2(1 To outputNeurons)
    End If
End If
```

d. For the efficient use of the storage space, the minimum number of parameters are saved to a file to represent a network. There are three different types of

A STATE UNIVERSITY

files: network files, training set files, and test set files, where file extensions are ***.net***, ***.trn***, and ***.tst***, respectively.

e. Even though they have different extensions, training and test files can replace each other.

The following are the file formats used in the simulator:

Network File Format

```
<Number of hidden layers>          ' 1 or 2
<Bias>                             ' 0 or 1
' Neurons in layers
<Input> <First Hidden> [<Second Hidden>] <Output>
' Weights
<W11> <> <> ... <W1,first hidden neurons>
<> ....................................................
<>
<W input neurons,1> <> ... <W inputneurons, first hidden neurons>
[<Bias 1> <> <> ... <Bias first hidden neurons>]
...
...
[<Bias 1> <> <> ... <Bias output neurons>]
```

Training and Test File Format

Although they have different extensions, training and test files have the same format.

```
<Number of I/O pairs (K)> <Number of Inputs (N)> <Number of Outputs (M)>
<Input 1,1> <> <> ... <Input 1,N> <Output 1,1> <> <> ... <Output 1,M>
...
...
...
<Input  K,1> <> <> ... <Input K,N> <Output K,1> <> <> ... <Output K,M>
```

Representing the Weights

Unlike the inputs, the outputs, and the adders, the color and thickness of the lines cannot be changed by the user. The following coloring and non-linear sizing method is used for the lines representing connection and bias weights:

Color	Size	Magnitude of the weight
Bright White	1	< 2
Light Yellow	1	< 3
Light Blue	1	< 4
Blue	2	< 6
Black	2	< 8
Light Magenta	3	< 11
Light Red	3	< 14
Light Red	4	< 28
Light Red	5	> 28

Visual Basic Controls

The following Visual Basic controls are used in the design of the window interfaces of the simulator:

a. *Line control*: A Line control is a graphical control displayed as a horizontal, vertical, or diagonal line. This control is used to draw lines on forms at design time.

A STATE UNIVERSITY

b. *Shape control*: This control is a graphical control displayed as a rectangle, square, oval, circle, rounded rectangle, or rounded square.

c. *TextBox control*: This control, sometimes called an edit field or edit control, displays information entered at design time, entered by the user, or assigned to the control in code at run time.

d. *Label control*: This is a graphical control used to display text that a user cannot change directly.

e. *Frame control*: A Frame control provides an identifiable grouping for controls. It can also be used to subdivide a form functionally.

f. *Gauge control*: This control creates user-defined gauges with a choice of linear (filled) or needle styles. This is used for the adder function in a neuron.

g. *ComboBox* control: This control is the combination of a TextBox control and a ListBox control that users can enter information in the text box portion or select an item from the list box portion of the control.

h. *Grid control*: This control displays a series of rows and columns. The intersection of a row and column is a cell. The contents of each cell can be read and set in code. This is used in the network values window.

i. *SSTab control*: This control provides a group of tabs, each of which acts as a container for other controls. Only one tab is active in the control at a time.

j. *SSCommand* control: This control is the 3D emulation of the standard Visual Basic command button control, which performs a task when the user either clicks the button or presses a key.

A STATE UNIVERSITY

k. *Graph control*: This control allows the user to design graphs interactively on the forms. At run time, the new data can be sent to the graph. This control can draw them, print them, copy them onto the Clipboard, or change their styles and shapes. This control is used to create the error graph of the simulator.

l. *RichTextBox control*: This control is similar to Textbox control but different text fonts and colors can be used.

m. *Common Dialog control*: This control allows the user to access Windows' Open, Save, Save As, and Color dialog boxes.

In addition to these controls, *Option button* and *ScrollBars* controls are also used.

A STATE UNIVERSITY

V

RUNNING THE SIMULATOR WITH AN EXAMPLE NETWORK

This chapter introduces VisualNN and its use by using the XOR network that learns the exclusive-or problem.

Creating the Network

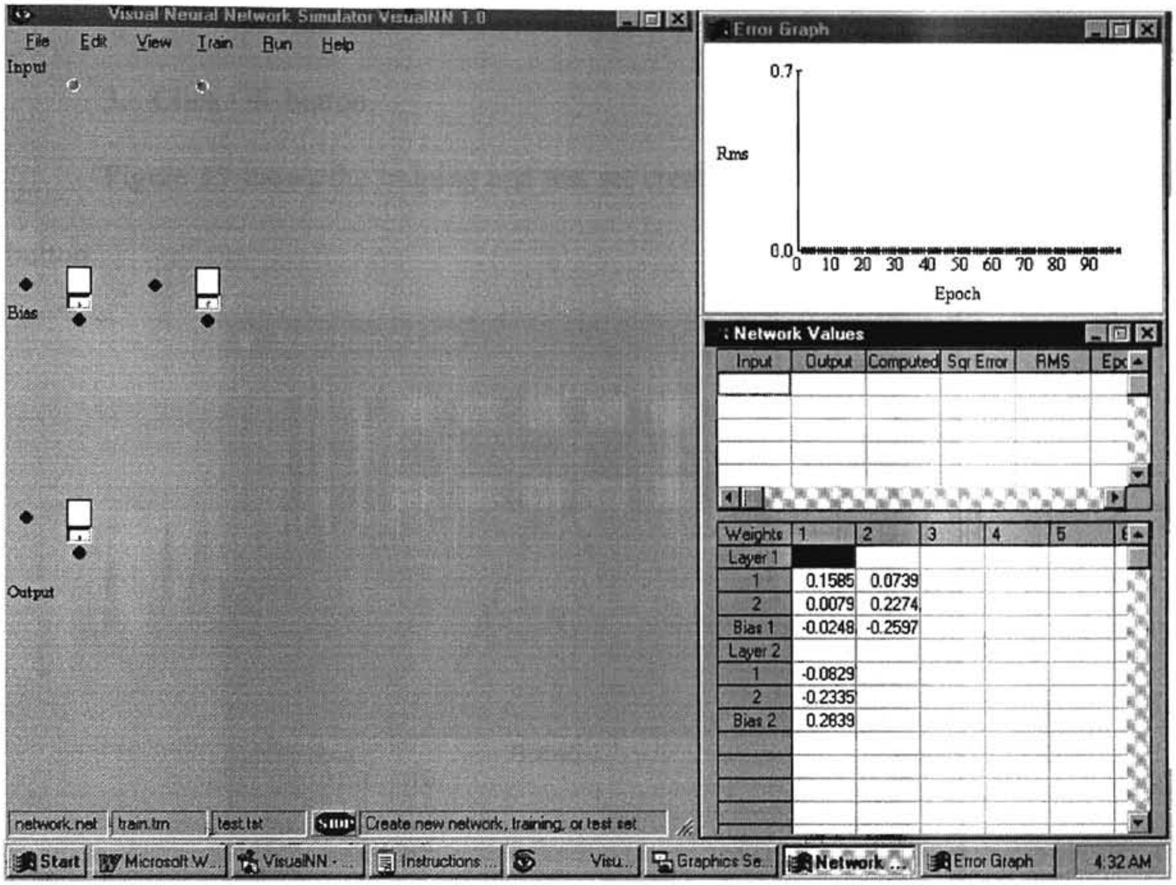
Before anything else is done, a network must be created. To create the XOR network;

1. Select New-Network from File menu.
2. Enter the following:

Hidden layers	: 1
Bias	: 1
Initial Weight Range	: 0.3
Threshold for Weights	: 0.31
Input Layer	: 2
Hidden Layer 1	: 2
Output Layer	: 1

3. Click Create button.

Figure 14 shows the resulting network, initial connection weights, and bias weights. As we see from the result, there is no connection between any neuron or bias. This is because of setting the initial weight range less than the threshold value. Since all the values between -0.3 and 0.3 are less than 0.31, all the connection lines representing weights are invisible. We now have a network to train and need a training set to train the XOR network.



Create new network, training, or test set

Weights	1	2	3	4	5
Layer 1					
1	0.1585	0.0739			
2	0.0079	0.2274			
Bias 1	-0.0248	-0.2597			
Layer 2					
1	-0.0829				
2	-0.2335				
Bias 2	0.2839				

Figure 14 Created XOR network with initial weights

Creating the Training Set

Now that we have created the XOR network, we must create some input/output pairs to train it on. To create the training set:

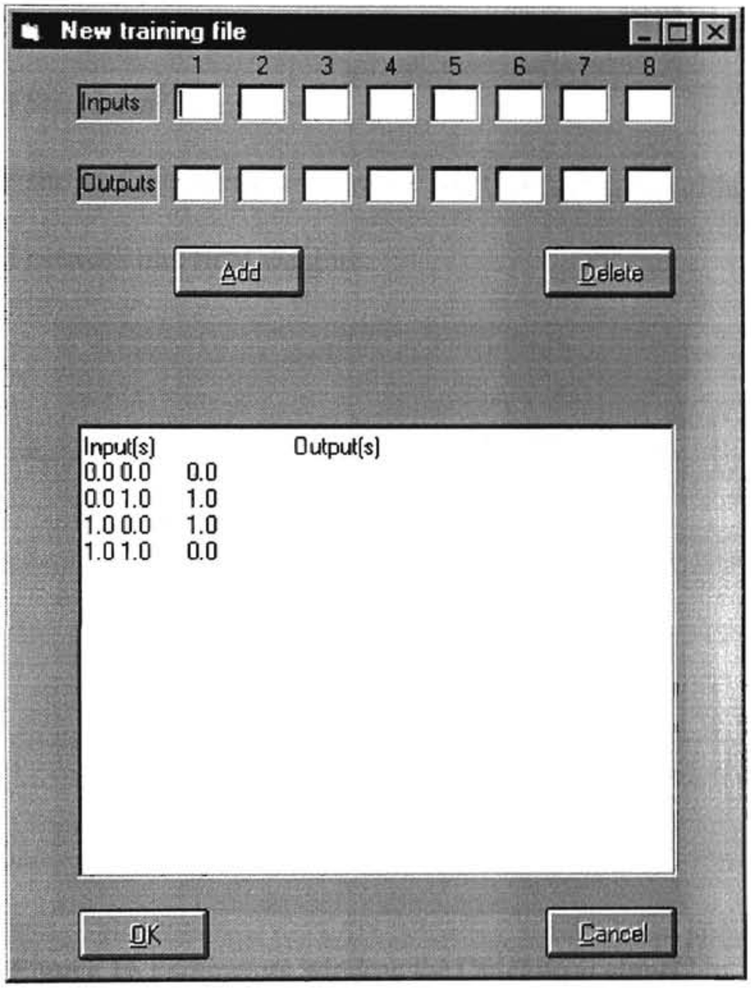
1. Select New-Training Set from the File menu.
2. Enter the following input/output pairs:

Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	0

Click Add button after entering an input/output pair.

3. Click OK button.

Figure 15 shows the training and test set creation window before clicking the OK button.



```
Input(s)            Output(s)
0.0 0.0   0.0
0.0 1.0   1.0
1.0 0.0   1.0
1.0 1.0   0.0
```

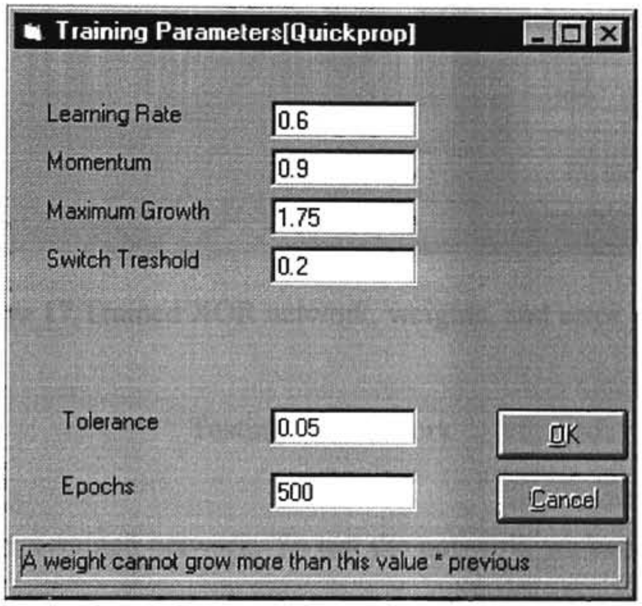
Figure 15 Training set creation window before OK button clicked.

Training the Network

Since we have a network and a training set, we can train the network. To train the network:

1. Select Training Algorithms-Quickprop (or any other algorithm) from Train menu.
2. Enter the necessary parameters for Quickprop algorithm or accept the default values.
3. Click OK button.
4. Select Start from Train menu.

Figure 16 shows the parameters window for Quickprop algorithm and Figure 17 shows the trained network and final weights.



Learning Rate 0.6

Momentum 0.9

Maximum Growth 1.75

Switch Treshold 0.2

Tolerance 0.05

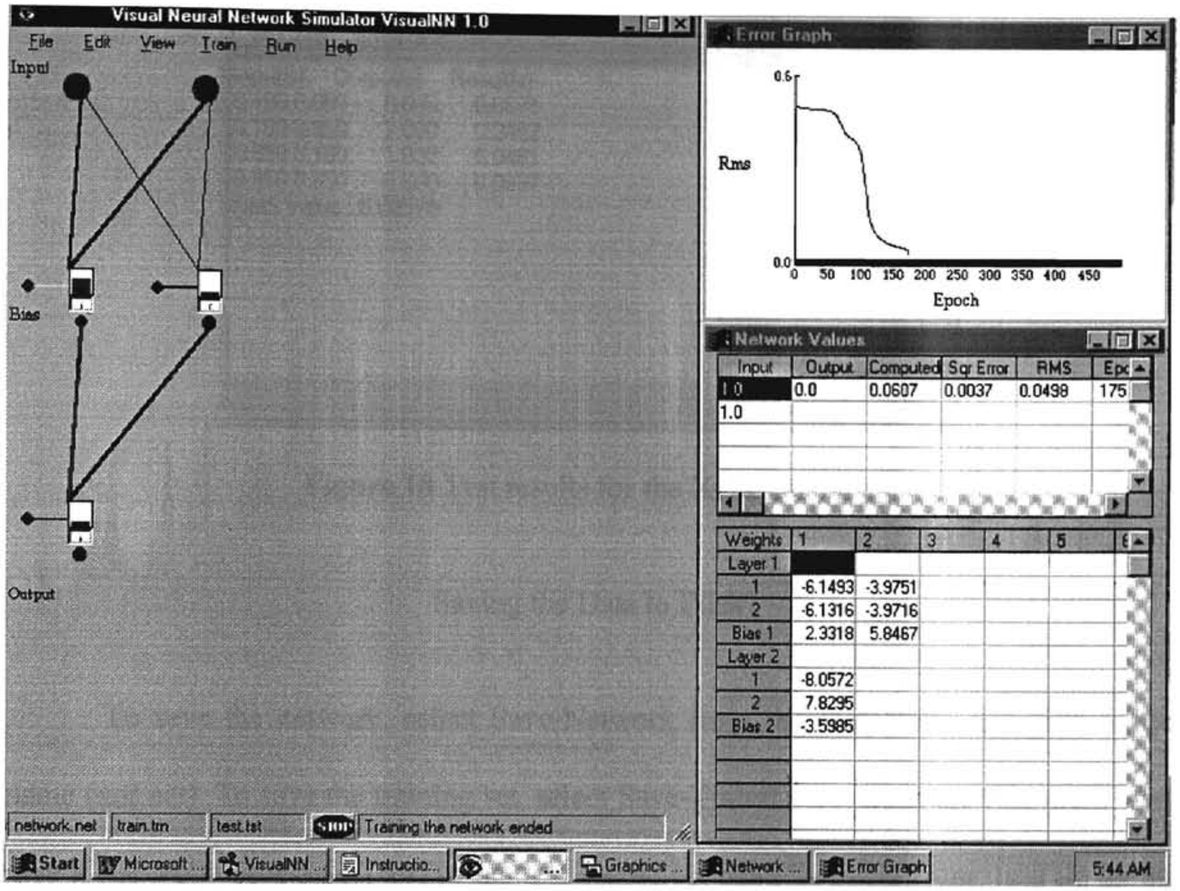
Epochs 500

A weight cannot grow more than this value * previous

Figure 16 Parameters window for Quickprop algorithm

Training will continue until one of the following happens:

1. From Train menu, End is selected.
2. STOP icon is clicked.
3. Error of the network is less than equal to Tolerance value.
4. Maximum number of epochs is reached.



Training the network ended

Input	Output	Computed	Sqr Error	RMS	Epc
1.0	0.0	0.0607	0.0037	0.0498	175
1.0					

Weights	1	2	3	4	5
Layer 1					
1	-6.1493	-3.9751			
2	-6.1316	-3.9716			
Bias 1	2.3318	5.8467			
Layer 2					
1	-8.0572				
2	7.8295				
Bias 2	-3.5985				

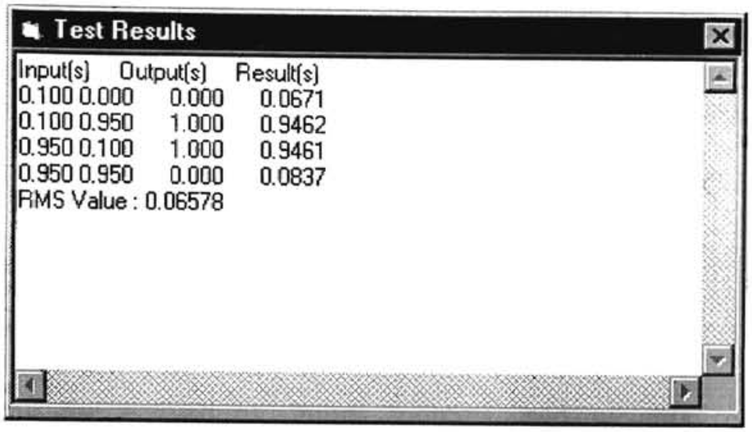
Figure 17 Trained XOR network, weights, and error graph

Testing the Network

We now have a trained network. To test the network, we need to create a test set. Follow the training set creation steps with the following input/output pairs:

Inputs	Output
0.1 0	0
0.1 0.95	1
0.95 0.1	1
0.95 0.95	0

To test the network select Start from the Run menu. Figure 18 shows the test results.



```
Input(s)    Output(s)   Result(s)
0.100 0.000   0.000    0.0671
0.100 0.950   1.000    0.9462
0.950 0.100   1.000    0.9461
0.950 0.950   0.000    0.0837
RMS Value : 0.06578
```

Figure 18 Test results for the XOR network

Saving the Data to Files

To save the network, select Save-Network from the File menu and give a file name (xor.net). To save the training set, select Save-Training Set from the File menu and give a file name (xor.trn). To save the test set, select Save-Test Set from the File menu and give a file name (xor.tst).

Figure 19 shows the contents of the files *xor.net*, *xor.trn*, and *xor.tst*.

1	4 2 1	4 2 1
1	0 0 0	0.1 0 0
2 2 1	0 1 1	0.1 0.95 1
-6.1493 -3.9751	1 0 1	0.95 0.1 1
-6.1316 -3.9716	1 1 0	0.95 0.95 0
2.3318 5.8467		
-8.0572		
7.8295		
-3.5985		
xor.net	xor.trn	xor.tst

Figure 19 Contents of the XOR network data files

VI

RESULTS, CONCLUSION AND FUTURE WORK

Results

Visualizing the activations, inputs, outputs, and weights resulted in better understanding of the things happening inside the network during the learning process. As seen in Figure 20, understanding a neuron's behavior is rather easy. If the adder display is completely filled with red, the output is 0 and the neuron is totally inactive. If the adder display is completely filled with green, the output is 1 and the neuron is totally active.

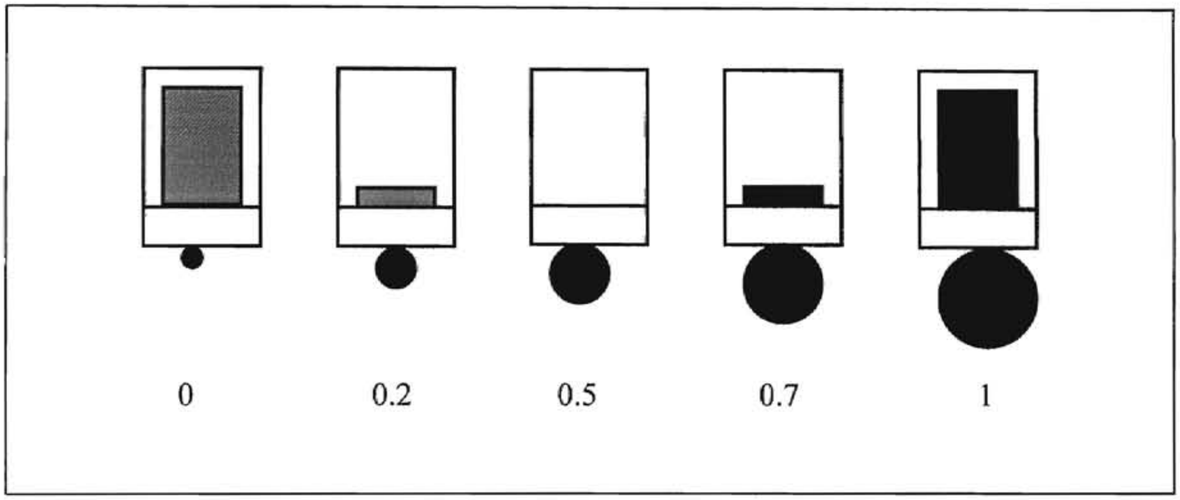
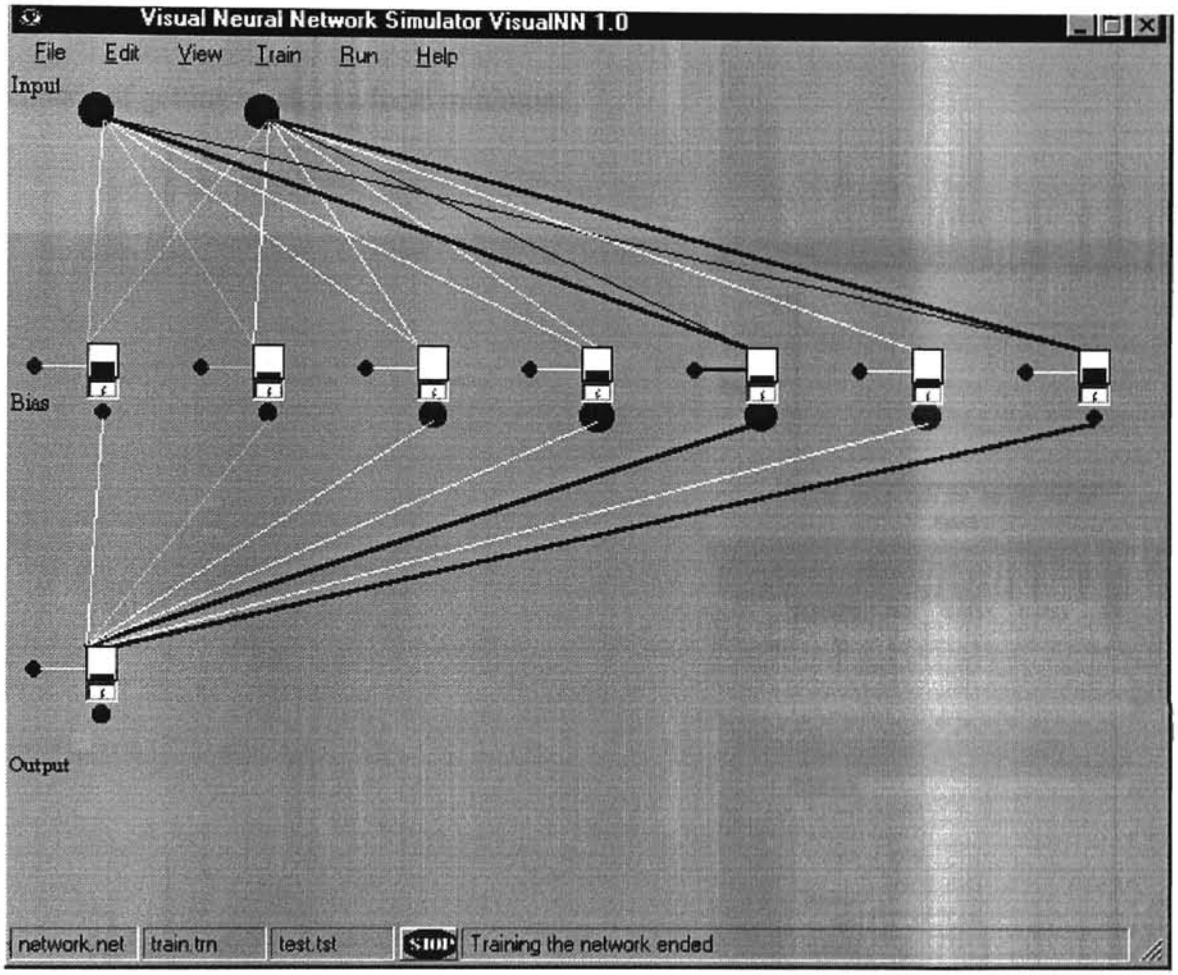


Figure 20 A neuron with different activations

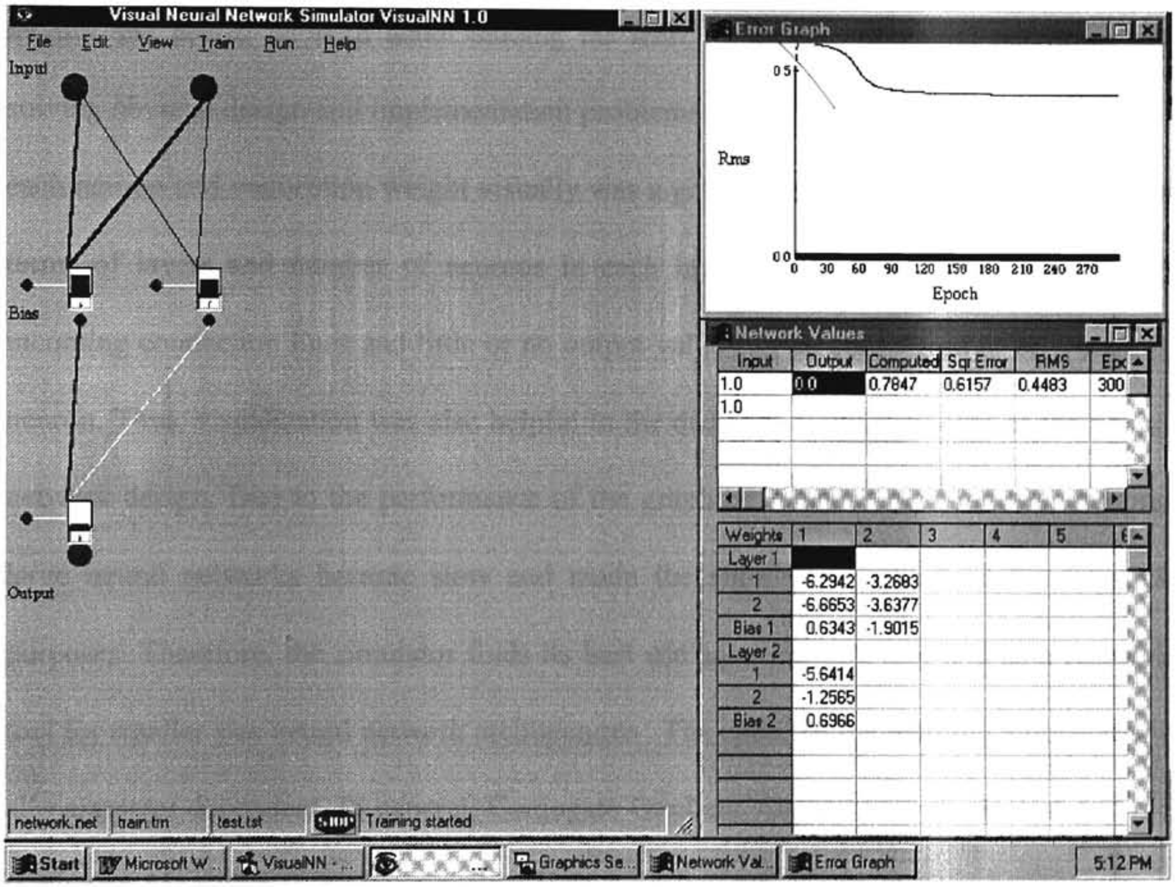
Representing the weights with different size and color lines resulted in solving two important issues. First, being able to see the connection and bias lines with no effects (or with little effects) helped to minimize the number of neurons necessary for solving the problem by discarding unnecessary neurons. Figure 21 shows the XOR network with unnecessary neurons. Neuron 5 and neuron 7 are the effective ones. Second , when the algorithm reaches a local minimum instead of the global minimum, it was easy to find the reason for that. It was also possible to see whether a neuron or a group of connection and bias weights causes the network to get stuck in a local minimum. Figure 22 shows the XOR network stuck in a local minimum.



Training the network ended

Figure 21 The XOR network with unnecessary neurons

Another result we gathered from the simulator is a dilemma about random initial weight range. Choosing a small initial weight range caused long training times but avoided getting stuck in a local minimum. On the other hand, choosing a large initial weight range decreased the number of presentations of training pairs but increased the chance of getting stuck in a local minimum.



Input	Output	Computed	Sqr Error	RMS	Epc
1.0	0.0	0.7847	0.6157	0.4483	300
1.0					

Weights	1	2	3	4	5
Layer 1					
1	-6.2942	-3.2683			
2	-6.6653	-3.6377			
Bias 1	0.6343	-1.9015			
Layer 2					
1	-5.6414				
2	-1.2565				
Bias 2	0.6966				

Figure 22 The XOR network that is stuck in local minimum

Conclusion

The purpose of this study was to develop an efficient, flexible, and user-friendly neural network simulator as a teaching and research tool. For better understanding of the simulation, scientific visualization techniques, such as representing the parameters by different graphical objects with different attributes, were used to represent the numerical data. The simulator was developed on the Microsoft Windows Operating System with the Visual Basic programming language as a 32-bit application for maintainability, software reuse, and greater accessibility.

Applying scientific visualization techniques to the simulator was found to be helpful and useful for both understanding the learning process of neural networks and solving obvious design and implementation problems. Being able to inspect the effects of each neuron and connection weight visually was a great help in optimizing the system in terms of layers and number of neurons in each layer. A neuron with small (lighter) incoming connection lines and little or no output value can be considered to be a useless neuron. Thus, visualization was also helpful in the decision-making process of the neural network design. Due to the performance of the graphics interface of Windows, training large neural networks became slow and made the simulator impractical for research purposes. Therefore, the simulator finds its best use as a teaching tool and as a research tool for smaller size neural network architectures . The speed of the learning algorithms is all parameter dependent. In general, Conjugate Gradient Algorithm is the fastest and the Backprop Algorithm is the slowest algorithm.

Future Work

There are two main deficiencies of VisualNN. The first one is that the size of the network that can be created is limited and the second one is the slow speed of the visual simulation. The reason for the first one is to try to fit the network into one screen. In order to get rid of this deficiency, the size can be made unlimited and only the parts of the network that are chosen by the user can be shown. The reason for the second one is the slow performance of the Windows Graphics Interface. In order to get rid of this deficiency, animation and the WinG method of Visual Basic can be used, but it will still be slower than with the DOS Graphics Interface.

For further improvement and using all the capabilities of Windows, a toolbox can be created. The toolbox may contain the neuron object, the connection line object, etc.

Finally, the simulator can have different neural network architectures such as Recurrent Networks with feedback.

REFERENCES

[1] C. M. Arnaldo, W. D. Miller and L. P. Gonzalez, "The geometry of backpropagation training: visualization, heuristics, and theory", *Technical Report*, Computer Science Department, Oklahoma State University, Stillwater, OK, 1991.

[2] A. Blum, *Neural Networks in C++; an Object-Oriented Framework for Building Connectionist Systems*, New York, John Wiley & Sons Ltd., 1992.

[3] E. Boonin, *User Interface Design*, Visual Basic 4.0 Expert Solutions, Que Corporation, 1996.

[4] D. V. Camp, "A User's Guide for The Xerion Neural Network Simulator Version 4.1", Department of Computer Science, University of Toronto, Canada, 1995.

[5] M. W. Craven and J. W. Shavlik, "Visualizing Learning and Computation in Artificial Neural Networks", *Machine Learning Research Group Working Paper 91-5*, Computer Sciences Department, University of Wisconsin, Madison, WI, 1991.

[6] M. Chester, *Neural Networks: A Tutorial*, Englewood Cliffs, NJ, PTR Prentice Hall, Inc., 1993.

[7] S. E. Fahlman, "An empirical study of learning speed in Back-Propagation Networks", *CMU Technical Report*, CMU-CS-88-162, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, September 1988.

[8] N. Gurewich and O. Gurewich, *Visual Basic 4 — Teach Yourself in 21 Days*, Indianapolis, IN, Sams Publishing, 1995.

[9] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*, Boston, MA, Prindle, Weber, Schmidt Publishing Company, 1996.

[10] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed representations" ,*Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pp. 77-109, Cambridge, MA, MIT Press, 1986.

[11] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation", *Neural Networks*, Vol. 1, pp. 295-307, 1988.

[12] S. Lawrence, A. C. Tsoi, and C. L. Giles, "Correctness, Efficiency, Extendibility and Maintainability in Neural Network Simulation", *International Conference on Neural Networks*, pp. 474-479, Washington, D. C., IEEE Press, 1996.

[13] X. Liu, " A comparison study of feedforward fully-connected neural networks vs. cascade correlation networks for prediction of soil moisture content", *MS Thesis*, Computer Science Department, Oklahoma State University, Stillwater, OK, 1994.

[14] *Microsoft Visual Basic Programmer's Guide*, Microsoft Corporation, 1996.

[15] *Microsoft Visual Basic Language Reference,* Microsoft Corporation, 1996.

[16] A. A. Minai and R. D. Williams, "Acceleration of Back-Propagation through learning rate and momentum adaptation", *International Joint Conference on Neural Networks*, Vol. 1, pp. 676-679, January 1990.

[17] M. D. Moller, "A scaled conjugate gradient algorithm for fast supervised learning", *Neural Networks*, Vol. 6, pp. 525-533, 1993.

[18] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, Vol. 323, pp. 533-536, 1986.

[19] J. Stanley, *Introduction to Neural Networks*, Sierra Madre, CA, California Scientific Software, 1989.

[20] A. Sperduti and A. Starita, "Speed up learning and network optimization with extended back propagation", *Neural Networks*, Vol. 6, 365-383, 1993.

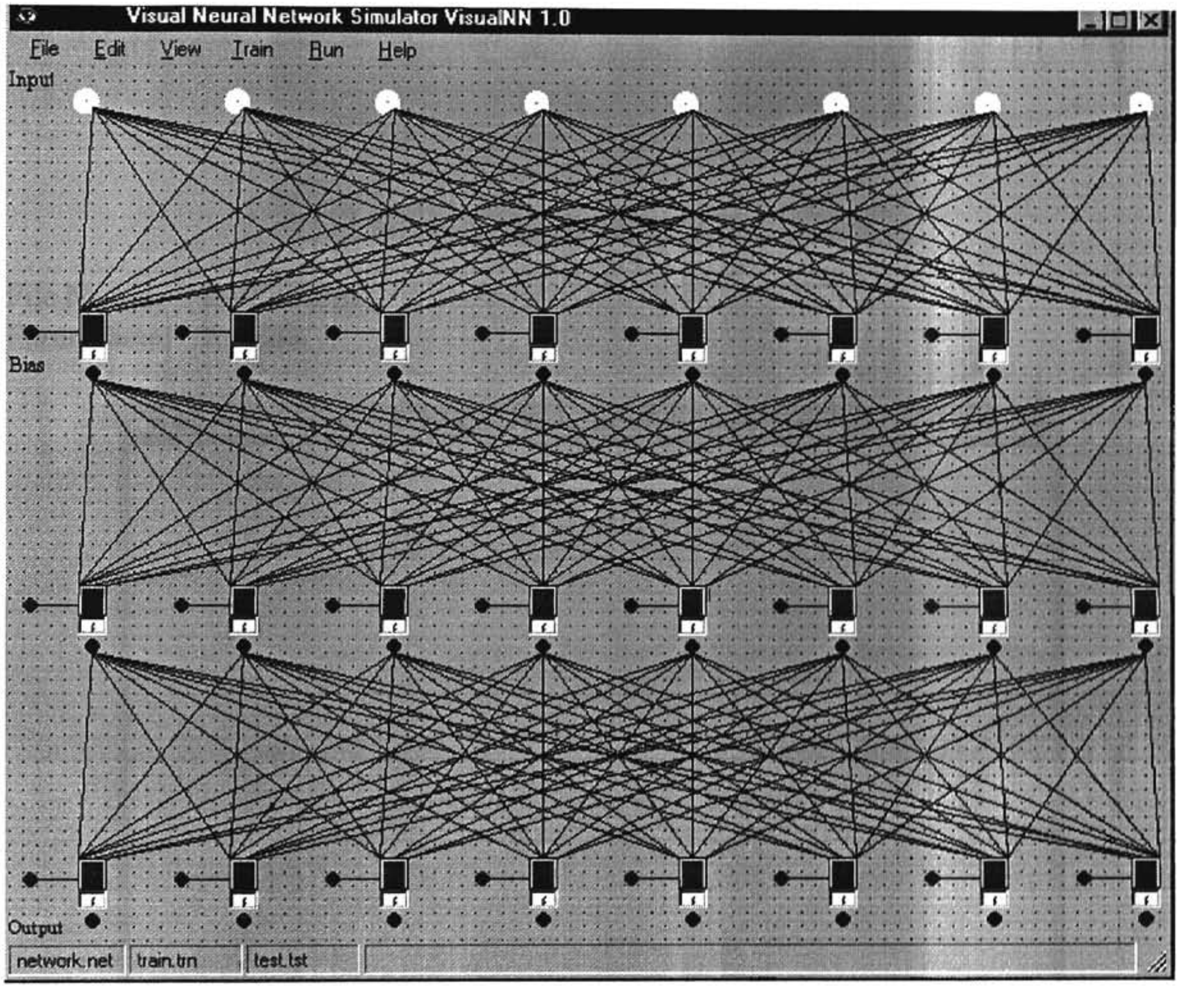
[21] D. Thalmann, editor, *Scientific Visualization and Graphics Simulation*, England, John Wiley & Sons Ltd., 1990.

[22] J. Wejchert and G. Tesauro, "Neural network visualization", *Advances in Neural Networks*, Vol. 2, pp. 465-472, San Mateo, CA, Morgan Kaufmann, 1990.

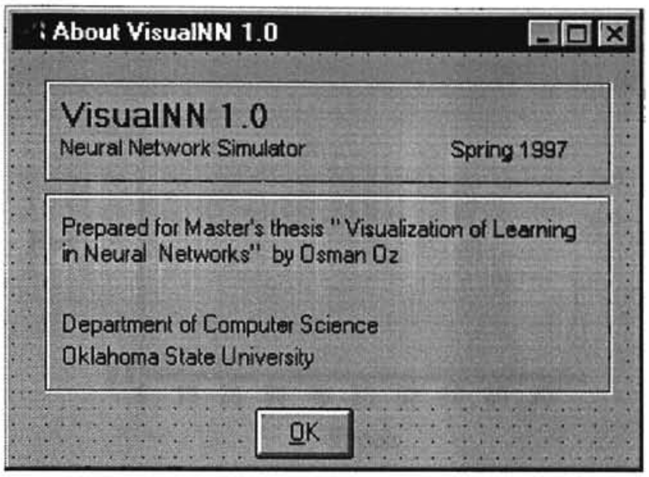
APPENDIX A

WINDOWS FORMS

File Name: Main.frm
Form Name: mainFrm



File Name: frmAbout.frm
Form Name: aboutFrm



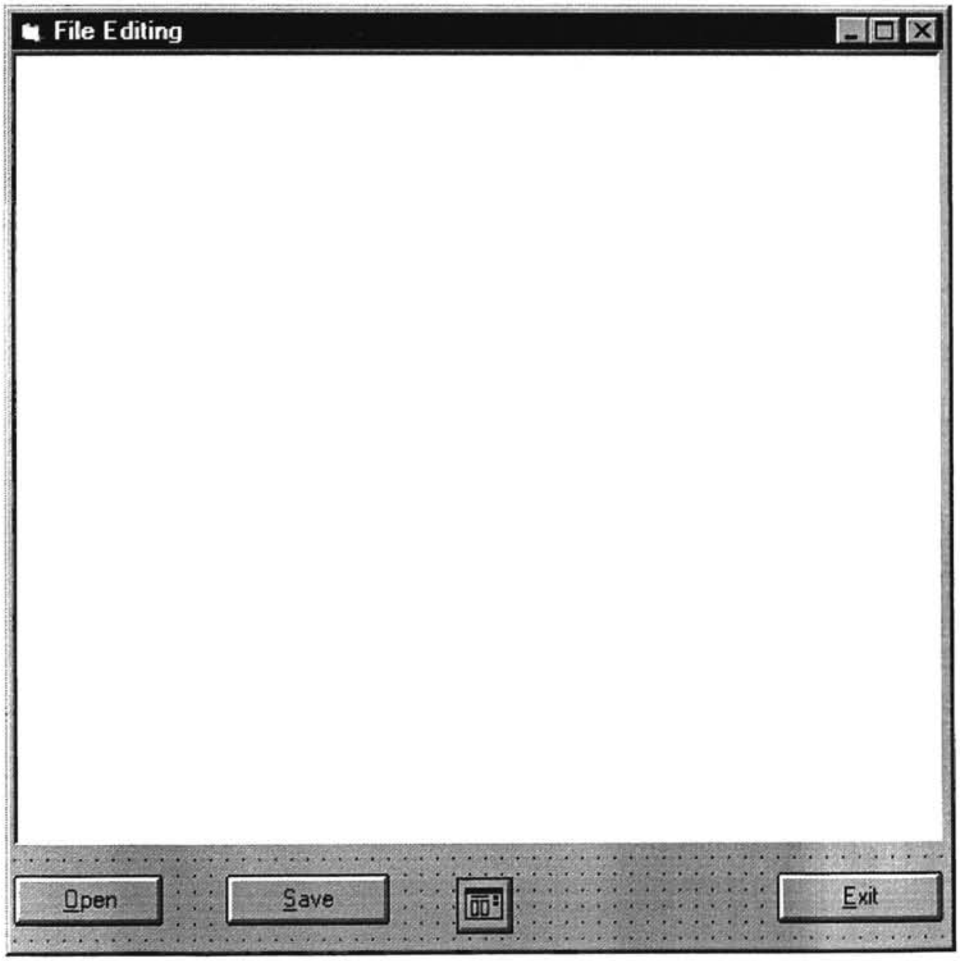
About VisualNN 1.0

VisualNN 1.0
Neural Network Simulator Spring 1997

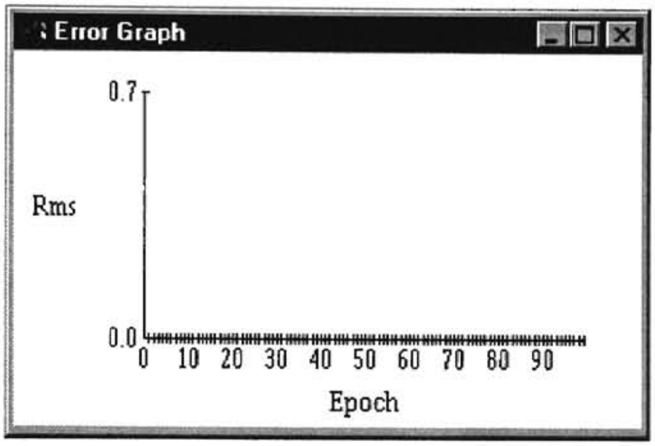
Prepared for Master's thesis "Visualization of Learning in Neural Networks" by Osman Oz

Department of Computer Science
Oklahoma State University

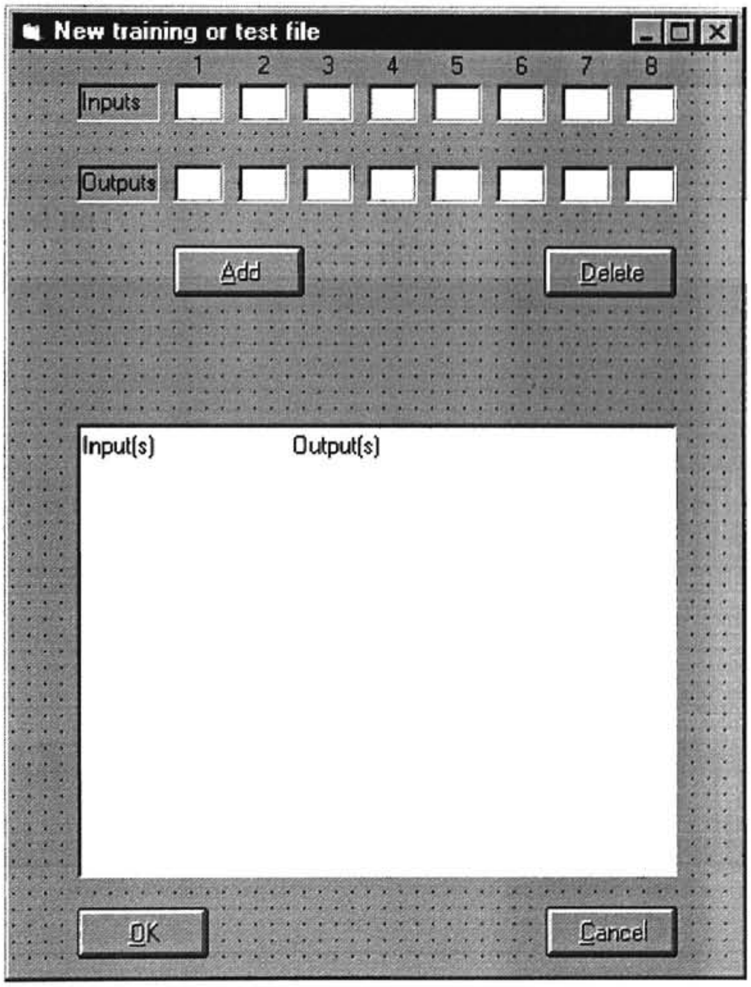
File Name: frmEdit.frm
Form Name: frmEdit



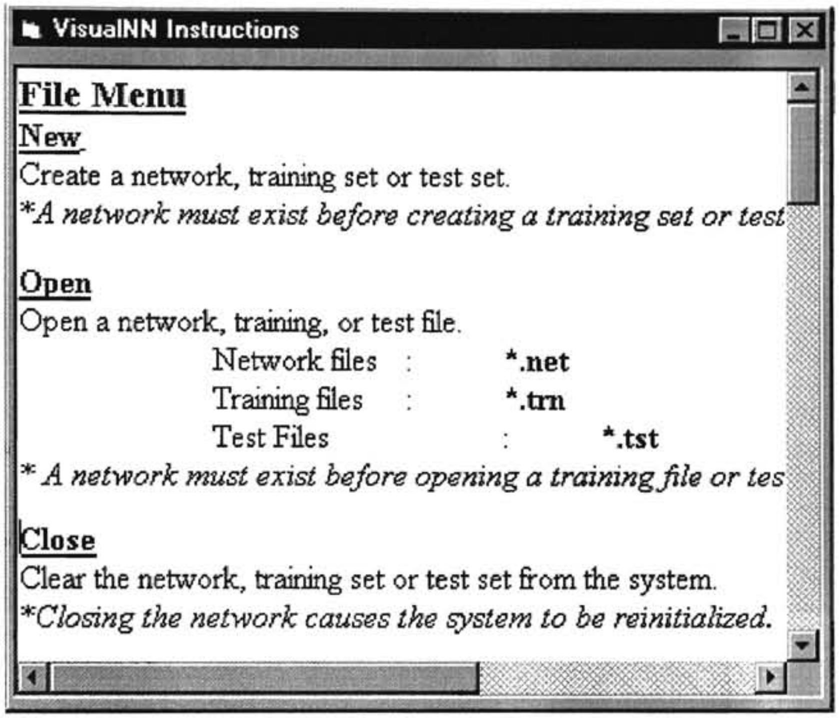
File Name: frmError.frm
Form Name: frmError



File Name: frmInOut.frm
Form Name: frmInOut



File Name: frmInstructions.frm
Form Name: frmInstructions



File Menu

New

Create a network, training set or test set.

**A network must exist before creating a training set or test*

Open

Open a network, training, or test file.

Network files	:	***.net**
Training files	:	***.trn**
Test Files	:	***.tst**

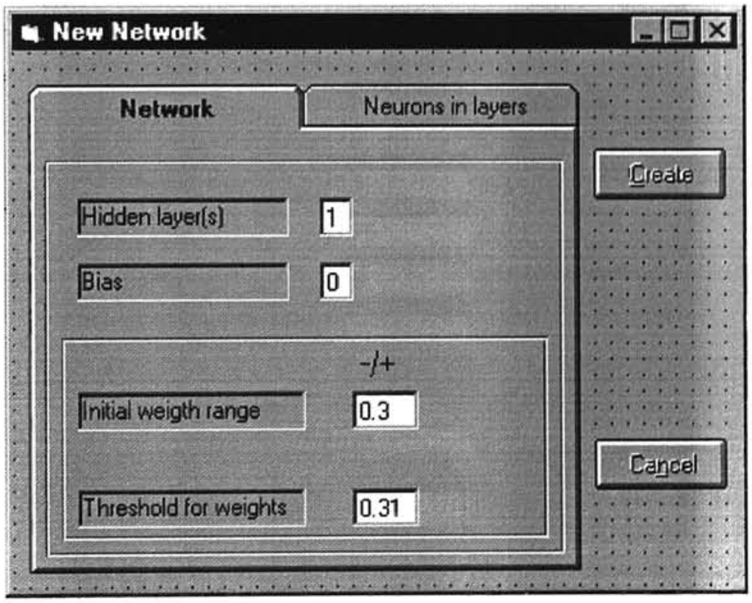
** A network must exist before opening a training file or tes*

Close

Clear the network, training set or test set from the system.

**Closing the network causes the system to be reinitialized.*

File Name: frmNewNet.frm
Form Name: newFrm



Hidden layer(s) 1

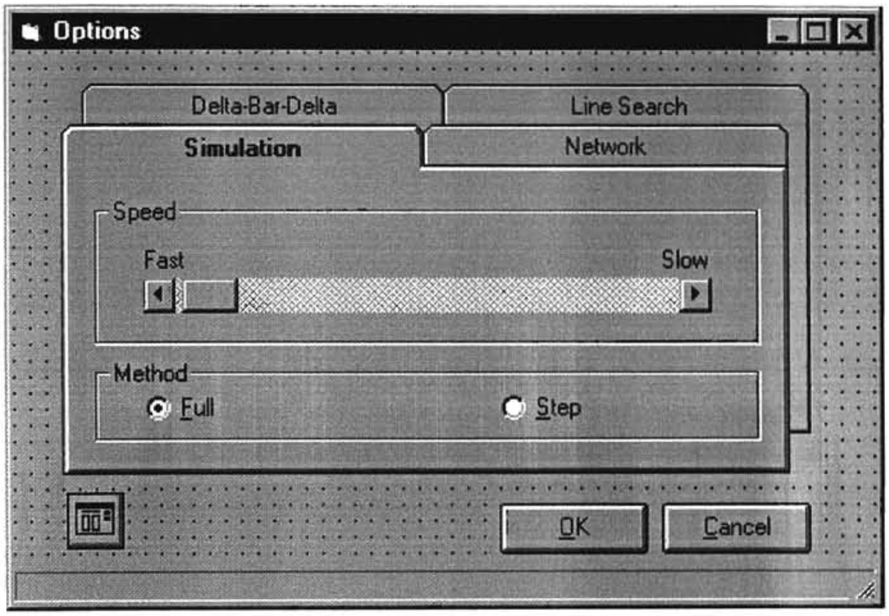
Bias 0

-/+

Initial weigth range 0.3

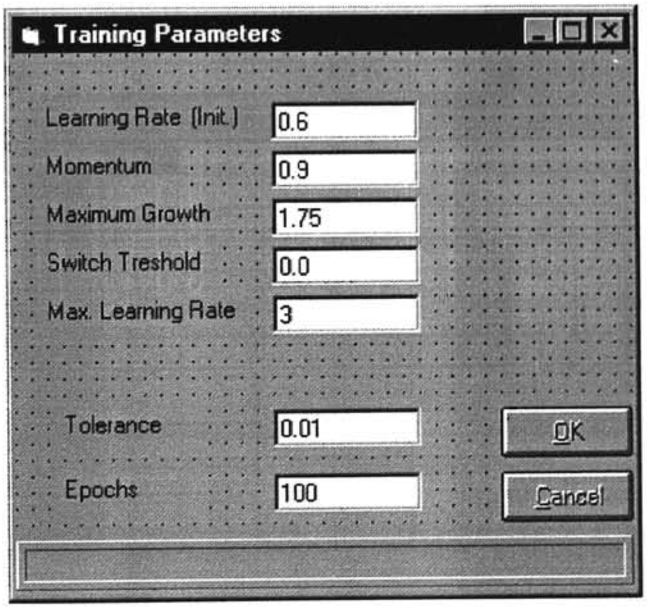
Threshold for weights 0.31

File Name: frmOptions.frm
Form Name: frmOptions



Method
● Full ○ Step

File Name: frmParams.frm
Form Name: paramFrm



Learning Rate (Init.) 0.6

Momentum 0.9

Maximum Growth 1.75

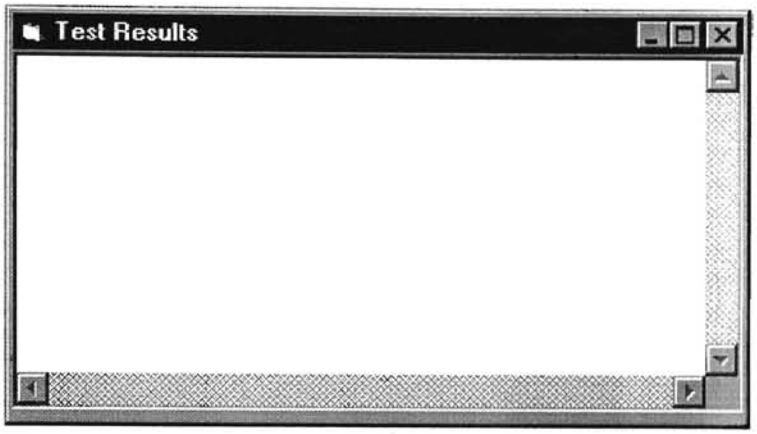
Switch Treshold 0.0

Max. Learning Rate 3

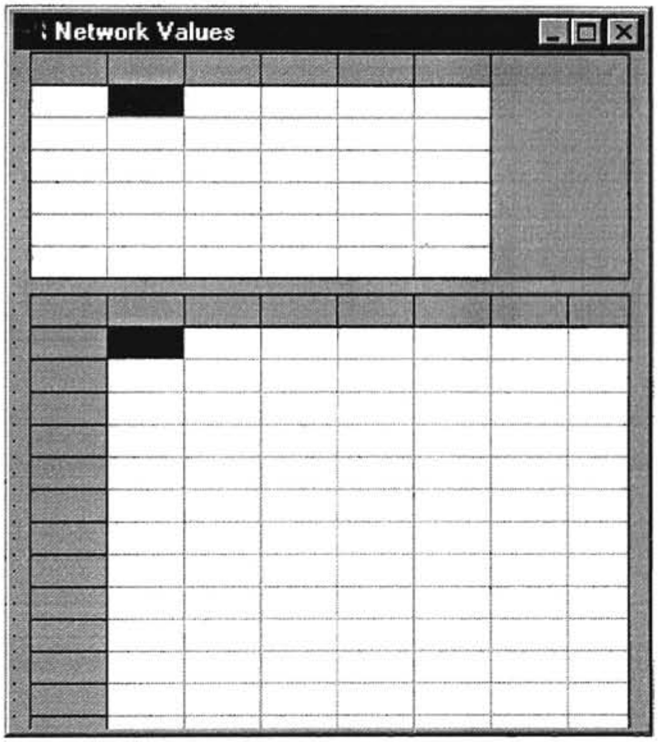
Tolerance 0.01

Epochs 100

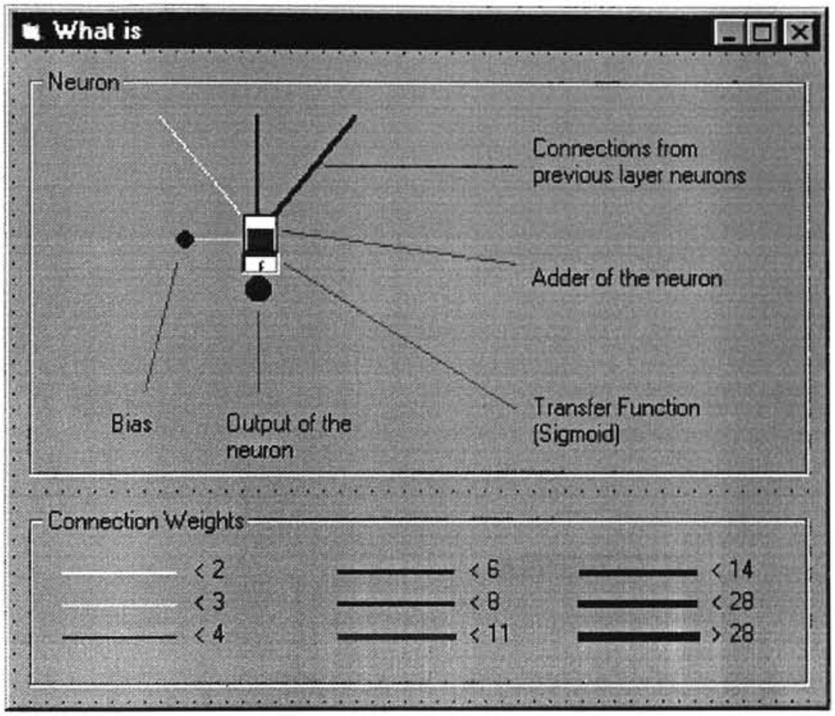
File Name:	frmResult.frm
Form Name:	frmResult



File Name:	frmValues.frm
Form Name:	frmValues



File Name: frmWhatis.frm
Form Name: frmWhatis



APPENDIX B

SOURCE CODE

```
Attribute VB_Name = "GlobalDefs"
'*******************************************************************
***
'  VisualNN 1.0
'  by Osman OZ        Spring 1997
'
'  Computer Science Department,  Oklahoma State University, Stillwater, OK
'*******************************************************************
***
' Global Variables
Public Algorithm As String            'Active training algorithm

Public hLayers As Byte               '# of hidden layers
Public bias As Byte                 'Bias connection flag

Public inputNeurons As Byte           '# of input layer neurons
Public outputNeurons As Byte           '# of output layer neurons
Public h1Neurons As Byte             '# of first hidden layer neurons
Public h2Neurons As Byte             '# of second hidden layer neurons

' Pointers to the lines(weights) in each layer
Public LinesPointer1()
Public LinesPointer2()
Public LinesPointer3()

' Simulation colors
Public NegativeColor
Public PositiveColor

Public GaugeMax As Integer          'Adder max value

Public DummyLine                'Current line pointer
Public LineValue As Single          'Current line(weight) value

Public CurrentAdder              'Current neuron adder pointer
Public CurrentAddValue As Single   'Current neuron adder value

Public CurrentDisc               'Current I/O pointer
Public CurrentDiscValue As Single   'Current I/O value

' Weight array pointers for each layer
Public W1(), W2(), W3() As Single
' Bias weight array pointers for each layer
Public bW1(), bW2(), bW3() As Single

' Error/weight derivative array pointers
Public FirstDer1(), FirstDer2(), FirstDer3() As Single

' Previous derivatives
Public SecondDer1(), SecondDer2(), SecondDer3() As Single

' Bias derivatives
Public FirstDerb1(), FirstDerb2(), FirstDerb3() As Single
Public SecondDerb1(), SecondDerb2(), SecondDerb3() As Single

' Delta weight values
Public dbW1(), dbW2(), dbW3() As Single
Public dW1(), dW2(), dW3() As Single

' Delta learning rate and learning rate for each weight(DBD alg)
Public alpha1(), alpha2(), alpha3() As Single
Public delta1(), delta2(), delta3() As Single

Public alphab1(), alphab2(), alphab3() As Single
Public deltab1(), deltab2(), deltab3() As Single

' Error values for each layer
Public errorH1(), errorH2(), errorOut() As Single
' Output for each layer
Public h1(), h2(), o() As Single

' Training set I/O matrix pointer
```

```
Public iTrnMatrix() As Single
Public oTrnMatrix() As Single

' Test set I/O matrix pointer
Public iTestMatrix() As Single
Public oTestMatrix() As Single

Public Momentum As Single        'Momentum parameter
Public LearningRate As Single     'Learning rate
Public MaxLearningRate As Single  'Maximum learning rate(DBD and CG)
Public Tolerance As Single       'Max desired RMS value
Public MaxEpochs As Integer       '# of epochs

'DBD parameters
Public THETA As Single
Public PHI As Single

Public MinMax As Single          'Weight initializitaion range
Public LineThreshold As Single     'Weight < this value is not visible


'Flags for msgbox
Public NetworkExist As Boolean
Public TrainExist As Boolean
Public TestExist As Boolean
Public NetworkSaved As Boolean
Public TrainSaved As Boolean
Public TestSaved As Boolean

Public Continue As Boolean        'Continue button click flag
Public Notfin As Boolean         'Training or testing finished flag
Public a                     'Return value of msgbox

Public FileSelected As Byte       'New training or test set flag

' # of I/O pairs
Public TrnIOs As Integer
Public TestIOs As Integer

Public MaxFactor As Single        'Max weight growth factor (QP)
Public SwitchThreshold As Single    'Switch to normal gradient
descent(QP)

Public IOType As String

Public Speed As Integer
Public Step As Boolean            'Whether run the simulator step-by-step
or not

' Line search parameters
Public IncreaseFactor, DecreaseFactor As Single
Public SuccessRange As Single

'Restores the globals
Public Sub GetDefaults()
  hLayers = 1
  bias = 0
  inputNeurons = 2
  h1Neurons = 2
  h2Neurons = 0
  outputNeurons = 1
  Momentum = 0.9
  LearningRate = 0.6
  Tolerance = 0.05
  MaxEpochs = 500
  MaxLearningRate = 3
  MaxFactor = 1.75
  SwitchThreshold = 0.2
  MinMax = 0.3
  LineThreshold = 0.31
End Sub
```

```
'Reset the accumulated delta values
'for CG with Line Search
Public Sub resetDeltas()

For i = 1 To inputNeurons
   For j = 1 To h1Neurons
      delta1(i, j) = 0#
   Next j
Next i
If bias = 1 Then
   For i = 1 To h1Neurons
      deltab1(i) = 0#
   Next
End If
If hLayers = 2 Then
   For i = 1 To h1Neurons
      For j = 1 To h2Neurons
         delta2(i, j) = 0#
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To h2Neurons
         deltab2(i) = 0#
      Next
   End If
   For i = 1 To h2Neurons
      For j = 1 To outputNeurons
         delta3(i, j) = 0#
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To outputNeurons
         deltab3(i) = 0#
      Next
   End If
Else
```

```
   For i = 1 To h1Neurons
      For j = 1 To outputNeurons
         delta2(i, j) = 0#
      Next j
   Next
   If bias = 1 Then
      For i = 1 To outputNeurons
         deltab1(i) = 0#
      Next
   End If
End If
End Sub
'Save the file Filename as net,trn,or tst
'decided by IOType
Sub SaveFileAs(Filename)
Dim Contents

On Error Resume Next
   ' Open the file.
Open Filename For Output As #1
   ' Display the hourglass mouse pointer.
Screen.MousePointer = 11
Select Case IOType
   Case "net"                    ' Save network file
      NetworkSaved = True
      ' Add # of hidden layers
      Contents = Str(hLayers)
      Contents = Right$(Contents, Len(Contents) - 1)
      Print #1, Contents
         ' Add bias (0 or 1)
      Contents = Str(bias)
      Contents = Right$(Contents, Len(Contents) - 1)
      Print #1, Contents
         ' Add # neurons in each layer
      If hLayers = 2 Then
      Contents = Str(inputNeurons) + Str(h1Neurons) + Str(h2Neurons) +
Str(outputNeurons)
    Else
      Contents = Str(inputNeurons) + Str(h1Neurons) + Str(outputNeurons)
    End If
    Contents = Right$(Contents, Len(Contents) - 1)
    Print #1, Contents

    For i = 1 To inputNeurons   ' First layer to second layer connection weights
      Contents = ""
      For j = 1 To h1Neurons
        Contents = Contents + Format(W1(i, j), "0.0000") + " "
      Next j
      Print #1, Contents
      Next i
      If bias = 1 Then           ' Add bias weights
        Contents = ""
        For j = 1 To h1Neurons
          Contents = Contents + Format(bW1(j), "0.0000") + " "
        Next j
        Print #1, Contents
      End If
      If hLayers = 2 Then        ' # of hidden layers=2
        For i = 1 To h1Neurons  ' Second layer to third layer connection
weights
          Contents = ""
          For j = 1 To h2Neurons
            Contents = Contents + Format(W2(i, j), "0.0000") + " "
          Next j
          Print #1, Contents
        Next i
        If bias = 1 Then        ' add bias weights
          Contents = ""
          For j = 1 To h2Neurons
            Contents = Contents + Format(bW2(j), "0.0000") + " "
          Next j
          Print #1, Contents
        End If
        For i = 1 To h2Neurons  ' Third layer to output layer
connection weights
          Contents = ""
          For j = 1 To outputNeurons
            Contents = Contents + Format(W3(i, j), "0.0000") + " "
          Next j
          Print #1, Contents
        Next i
        If bias = 1 Then        ' add bias weights
          Contents = ""
          For j = 1 To h1Neurons
            Contents = Contents + Format(bW3(j), "0.0000") + " "
          Next j
          Print #1, Contents
        End If
      Else                      ' # of hidden layers=1
        For i = 1 To h1Neurons ' Second layer to output layer
connection weights
          Contents = ""
          For j = 1 To outputNeurons
            Contents = Contents + Format(W2(i, j), "0.0000") + " "
          Next j
          Print #1, Contents
        Next i
        If bias = 1 Then        ' add bias weights
          Contents = ""
          For j = 1 To h1Neurons
            Contents = Contents + Format(bW2(j), "0.0000") + " "
          Next j
          Print #1, Contents
        End If
      End If
      Close #1
      ' Reset the mouse pointer.
      Screen.MousePointer = 0
  Case "trn"                    ' Save training file
      TrainSaved = True
      Contents = Str(TrnIOs) + Str(inputNeurons) + Str(outputNeurons)
      Contents = Right$(Contents, Len(Contents) - 1)
      Print #1, Contents
      For i = 1 To TrnIOs
        Contents = ""
        For j = 1 To inputNeurons
          Contents = Contents + Str(iTrnMatrix(i, j))
        Next j
        For j = 1 To outputNeurons
          Contents = Contents + Str(oTrnMatrix(i, j))
        Next j
        Contents = Right$(Contents, Len(Contents) - 1)
        Print #1, Contents
      Next i
      Close #1
      ' Reset the mouse pointer.
      Screen.MousePointer = 0
  Case "tst"
      TestSaved = True
      Contents = Str(TestIOs) + Str(inputNeurons) + Str(outputNeurons)
      Contents = Right$(Contents, Len(Contents) - 1)
      Print #1, Contents

      For i = 1 To TestIOs
        Contents = ""
        For j = 1 To inputNeurons
          Contents = Contents + Str(iTestMatrix(i, j))
        Next j
        For j = 1 To outputNeurons
          Contents = Contents + Str(oTestMatrix(i, j))
        Next j
        Contents = Right$(Contents, Len(Contents) - 1)
        Print #1, Contents
      Next i
      Close #1
      ' Reset the mouse pointer.
      Screen.MousePointer = 0
  End Select
End Sub
' Displays a Save As dialog box and returns a filename.
' If the user chooses Cancel, returns an empty string.
Function GetFileName(Filename As Variant)
On Error Resume Next
  mainFrm.CD1.Filename = Filename
  mainFrm.CD1.ShowSave
  If Err <> 32755 Then    ' User chose Cancel.
    GetFileName = mainFrm.CD1.Filename
  Else
    GetFileName = ""
  End If
End Function

'Initializes the delta weight, first and second
'derivatives, and alpha and delta arrays
Public Sub initializeDeltaWeights()

For i = 1 To inputNeurons
  For j = 1 To h1Neurons
    dW1(i, j) = 0#
    FirstDer1(i, j) = 0#
    SecondDer1(i, j) = 0#
    delta1(i, j) = 0.1
    alpha1(i, j) = 0.1
  Next j
Next i
If bias = 1 Then
  For i = 1 To h1Neurons
    dbW1(i) = 0#
    FirstDerb1(i) = 0#
```

```
      SecondDerb1(i) = 0#
      deltab1(i) = 0.1
      alphab1(i) = 0.1
   Next
End If
If hLayers = 2 Then
   For i = 1 To h1Neurons
      For j = 1 To h2Neurons
         dW2(i, j) = 0#
         FirstDer2(i, j) = 0#
         SecondDer2(i, j) = 0#
         delta2(i, j) = 0.1
         alpha2(i, j) = 0.1
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To h2Neurons
         dbW2(i) = 0#
         FirstDerb2(i) = 0#
         SecondDerb2(i) = 0#
         deltab2(i) = 0.1
         alphab2(i) = 0.1
      Next
   End If
   For i = 1 To h2Neurons
      For j = 1 To outputNeurons
         dW3(i, j) = 0#
         FirstDer3(i, j) = 0#
         SecondDer3(i, j) = 0#
         delta3(i, j) = 0.1
         alpha3(i, j) = 0.1
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To outputNeurons
         dbW3(i) = 0#
         FirstDerb3(i) = 0#
         SecondDerb3(i) = 0#
         deltab3(i) = 0.1
         alphab3(i) = 0.1
      Next
   End If
Else
   For i = 1 To h1Neurons
      For j = 1 To outputNeurons
         dW2(i, j) = 0#
         FirstDer2(i, j) = 0#
         SecondDer2(i, j) = 0#
         delta2(i, j) = 0.1
         alpha2(i, j) = 0.1
      Next j
   Next
   If bias = 1 Then
      For i = 1 To outputNeurons
         dbW2(i) = 0#
         FirstDerb2(i) = 0#
         SecondDerb2(i) = 0#
         deltab2(i) = 0.1
         alphab2(i) = 0.1
      Next
   End If
End If
End Sub

'Initializes the Values form weight grid
Public Sub initializeGrdValues()
For i = 0 To 9
   For j = 0 To 34
      frmValues.grdW.Col = i
      frmValues.grdW.Row = j
      frmValues.grdW.Text = ""
   Next j
Next i
End Sub
'Initializes the Values form grid1
Public Sub initializeGrds()
Dim i As Byte

frmValues.grdIO.Row = 0
For i = 0 To 5
   frmValues.grdIO.Col = i
   frmValues.grdIO.ColWidth(i) = 750
Next
frmValues.grdIO.Row = 0
frmValues.grdIO.Col = 0
frmValues.grdIO.FixedAlignment(0) = 2
frmValues.grdIO.Text = "Input"

frmValues.grdIO.Col = 1
frmValues.grdIO.FixedAlignment(1) = 2
frmValues.grdIO.Text = "Output"

frmValues.grdIO.Col = 2
frmValues.grdIO.FixedAlignment(2) = 2
frmValues.grdIO.Text = "Computed"

frmValues.grdIO.Col = 3
frmValues.grdIO.FixedAlignment(3) = 2
frmValues.grdIO.Text = "Sqr Error"

frmValues.grdIO.Col = 4
frmValues.grdIO.FixedAlignment(4) = 2
frmValues.grdIO.Text = "RMS"

frmValues.grdIO.Col = 5
frmValues.grdIO.FixedAlignment(5) = 2
frmValues.grdIO.Text = "Epochs"
```

```
frmValues.grdW.Row = 0
frmValues.grdW.Col = 0
frmValues.grdW.ColWidth(0) = 750
frmValues.grdW.FixedAlignment(0) = 2
frmValues.grdW.Text = "Weights"

For i = 1 To 9
   frmValues.grdW.Row = 0
   frmValues.grdW.Col = i
   frmValues.grdW.ColWidth(i) = 650
   frmValues.grdW.FixedAlignment(0) = 2
   frmValues.grdW.Text = Str$(i)
Next
End Sub
'Initializes the pointers to the lines
Public Sub initializeLines()
For i = 1 To h1Neurons
   If inputNeurons >= 1 Then
      Set LinesPointer1(1, i) = mainFrm.Line1(i - 1)
   End If
   If inputNeurons >= 2 Then
      Set LinesPointer1(2, i) = mainFrm.Line2(i - 1)
   End If
   If inputNeurons >= 3 Then
      Set LinesPointer1(3, i) = mainFrm.Line3(i - 1)
   End If
   If inputNeurons >= 4 Then
      Set LinesPointer1(4, i) = mainFrm.Line4(i - 1)
   End If
   If inputNeurons >= 5 Then
      Set LinesPointer1(5, i) = mainFrm.Line5(i - 1)
   End If
   If inputNeurons >= 6 Then
      Set LinesPointer1(6. i) = mainFrm.Line6(i - 1)
   End If
   If inputNeurons >= 7 Then
```

```
      Set LinesPointer1(7, i) = mainFrm.Line7(i - 1)
    End If
    If inputNeurons = 8 Then
      Set LinesPointer1(8, i) = mainFrm.Line8(i - 1)
    End If
  Next i
  If hLayers = 2 Then
   For i = 1 To h2Neurons
    If h1Neurons >= 1 Then
      Set LinesPointer2(1, i) = mainFrm.Line1(16 - i)
    End If
    If h1Neurons >= 2 Then
      Set LinesPointer2(2, i) = mainFrm.Line2(16 - i)
    End If
    If h1Neurons >= 3 Then
      Set LinesPointer2(3, i) = mainFrm.Line3(16 - i)
    End If
    If h1Neurons >= 4 Then
      Set LinesPointer2(4, i) = mainFrm.Line4(16 - i)
    End If
    If h1Neurons >= 5 Then
      Set LinesPointer2(5, i) = mainFrm.Line5(16 - i)
    End If
    If h1Neurons >= 6 Then
      Set LinesPointer2(6, i) = mainFrm.Line6(16 - i)
    End If
    If h1Neurons >= 7 Then
      Set LinesPointer2(7, i) = mainFrm.Line7(16 - i)
    End If
    If h1Neurons = 8 Then
      Set LinesPointer2(8, i) = mainFrm.Line8(16 - i)
    End If
   Next i
   For i = 1 To outputNeurons
    If h2Neurons >= 1 Then
      Set LinesPointer3(1, i) = mainFrm.Line1(24 - i)
    End If
    If h2Neurons >= 2 Then
      Set LinesPointer3(2, i) = mainFrm.Line2(24 - i)
    End If
    If h2Neurons >= 3 Then
      Set LinesPointer3(3, i) = mainFrm.Line3(24 - i)
    End If
    If h2Neurons >= 4 Then
      Set LinesPointer3(4, i) = mainFrm.Line4(24 - i)
    End If
    If h2Neurons >= 5 Then
      Set LinesPointer3(5, i) = mainFrm.Line5(24 - i)
    End If
    If h2Neurons >= 6 Then
      Set LinesPointer3(6, i) = mainFrm.Line6(24 - i)
    End If
    If h2Neurons >= 7 Then
      Set LinesPointer3(7, i) = mainFrm.Line7(24 - i)
    End If
    If h2Neurons = 8 Then
      Set LinesPointer3(8, i) = mainFrm.Line8(24 - i)
    End If
   Next i
  Else
    For i = 1 To outputNeurons
     If h1Neurons >= 1 Then
        Set LinesPointer2(1, i) = mainFrm.Line1(16 - i)
     End If
     If h1Neurons >= 2 Then
        Set LinesPointer2(2, i) = mainFrm.Line2(16 - i)
     End If
     If h1Neurons >= 3 Then
        Set LinesPointer2(3, i) = mainFrm.Line3(16 - i)
     End If
     If h1Neurons >= 4 Then
        Set LinesPointer2(4, i) = mainFrm.Line4(16 - i)
    End If
    If h1Neurons >= 5 Then
        Set LinesPointer2(5, i) = mainFrm.Line5(16 - i)
    End If
    If h1Neurons >= 6 Then
        Set LinesPointer2(6, i) = mainFrm.Line6(16 - i)
    End If
    If h1Neurons >= 7 Then
        Set LinesPointer2(7, i) = mainFrm.Line7(16 - i)
    End If
    If h1Neurons = 8 Then
        Set LinesPointer2(8, i) = mainFrm.Line8(16 - i)
    End If
  Next i
End If
End Sub
'Initially everything on the visual screen
'is disabled(invisible)
Public Sub initializeNet()
For i = 0 To 23    'Lines are not visible
  mainFrm.Line1(i).Visible = False
  mainFrm.Line2(i).Visible = False
  mainFrm.Line3(i).Visible = False
  mainFrm.Line4(i).Visible = False
  mainFrm.Line5(i).Visible = False
  mainFrm.Line6(i).Visible = False
  mainFrm.Line7(i).Visible = False
  mainFrm.Line8(i).Visible = False
Next i
For i = 0 To 7     'The others
  mainFrm.bias1(i).Visible = False
  mainFrm.bline1(i).Visible = False
  mainFrm.bias2(i).Visible = False
  mainFrm.bline2(i).Visible = False
  mainFrm.bias3(i).Visible = False
  mainFrm.bline3(i).Visible = False
```

```
  mainFrm.InputShp(i).Visible = False
  mainFrm.AddGauge1(i).Visible = False
  mainFrm.fOut1(i).Visible = False
  mainFrm.fBox1(i).Visible = False

  mainFrm.AddGauge2(i).Visible = False
  mainFrm.fOut2(i).Visible = False
  mainFrm.fBox2(i).Visible = False

  mainFrm.AddGauge3(i).Visible = False
  mainFrm.fOut3(i).Visible = False
  mainFrm.fBox3(i).Visible = False
Next i
For i = 0 To 2     'Labels
  mainFrm.Label1(i).Visible = False
Next i
End Sub
'Draws the network created
Public Sub drawNet()

'Show input layer neurons
For i = 1 To inputNeurons
  mainFrm.InputShp(i - 1).Visible = True
Next i

For i = 0 To 1
  mainFrm.Label1(i).Visible = True
Next i

If hLayers = 2 Then          'if two hidden layers
  For i = 1 To h1Neurons     'Show hidden one neurons
    mainFrm.AddGauge1(i - 1).Visible = True
    mainFrm.fBox1(i - 1).Visible = True
    mainFrm.fOut1(i - 1).Visible = True
  Next i
  For i = 1 To h2Neurons       'Show hidden two neurons
    mainFrm.AddGauge2(i - 1).Visible = True
    mainFrm.fBox2(i - 1).Visible = True
    mainFrm.fOut2(i - 1).Visible = True
  Next i
  For i = 0 To outputNeurons - 1  'Show output neurons
    mainFrm.AddGauge3(i).Visible = True
    mainFrm.fBox3(i).Visible = True
    mainFrm.fOut3(i).Visible = True
    Next i

  If bias = 1 Then             'If bias show bias units
    mainFrm.Label1(2).Visible = True
    For i = 0 To h1Neurons - 1
      mainFrm.bias1(i).Visible = True
    Next i
    For i = 0 To h2Neurons - 1
      mainFrm.bias2(i).Visible = True
    Next i
    For i = 0 To outputNeurons - 1
      mainFrm.bias3(i).Visible = True
    Next i
  End If
Else                           'if one hidden layer
  For i = 1 To h1Neurons
    mainFrm.AddGauge1(i - 1).Visible = True
    mainFrm.fBox1(i - 1).Visible = True
    mainFrm.fOut1(i - 1).Visible = True
  Next i
  For i = 0 To outputNeurons - 1
    mainFrm.AddGauge2(i).Visible = True
    mainFrm.fBox2(i).Visible = True
    mainFrm.fOut2(i).Visible = True
  Next i

  If bias = 1 Then
```

```
    mainFrm.Label1(2).Visible = True
    For i = 0 To h1Neurons - 1
      mainFrm.bias1(i).Visible = True
    Next i
    For i = 0 To outputNeurons - 1
      mainFrm.bias2(i).Visible = True
    Next i
  End If
  mainFrm.Label1(1).Top = 5400
End If
End Sub

'Set or reinitialize neurons
'in the network
Public Sub initializeNeurons()

For i = 1 To inputNeurons
  mainFrm.InputShp(i - 1).BorderWidth = 1
Next
For i = 1 To h1Neurons
  mainFrm.AddGauge1(i - 1).Value = 0
  mainFrm.AddGauge1(i - 1).Max = GaugeMax
  mainFrm.fOut1(i - 1).BorderWidth = 1
Next

If hLayers = 2 Then
  For i = 1 To h2Neurons
    mainFrm.AddGauge2(i - 1).Value = 0
    mainFrm.AddGauge2(i - 1).Max = GaugeMax
    mainFrm.fOut2(i - 1).BorderWidth = 1
  Next
  For i = 0 To outputNeurons - 1
    mainFrm.AddGauge3(i).Value = 0
    mainFrm.AddGauge3(i).Max = GaugeMax
    mainFrm.fOut3(i).BorderWidth = 1
  Next i
```

```
   If bias = 1 Then
      For i = 0 To h1Neurons - 1
         mainFrm.bias1(i).BorderWidth = 1
      Next i
      For i = 0 To h2Neurons - 1
         mainFrm.bias2(i).BorderWidth = 1
      Next i
      For i = 0 To outputNeurons - 1
         mainFrm.bias3(i).BorderWidth = 1
      Next i
   End If
Else
   For i = 0 To outputNeurons - 1
      mainFrm.AddGauge2(i).Value = 0
      mainFrm.AddGauge2(i).Max = GaugeMax
      mainFrm.fOut2(i).BorderWidth = 1
   Next i
   If bias = 1 Then
      For i = 0 To h1Neurons - 1
         mainFrm.bias1(i).BorderWidth = 1
      Next i
      For i = 0 To outputNeurons - 1
         mainFrm.bias2(i).BorderWidth = 1
      Next i
   End If
End If
End Sub

'Randomize the weights between -MinMax and MinMax
'
Public Sub randomWeights()
Randomize Timer

For i = 1 To inputNeurons
   For j = 1 To h1Neurons
      W1(i, j) = (MinMax * 2 * Rnd) - MinMax
   Next j
Next i
If hLayers = 2 Then
   For i = 1 To h1Neurons
      For j = 1 To h2Neurons
         W2(i, j) = (MinMax * 2 * Rnd) - MinMax
      Next j
   Next i
   For i = 1 To h2Neurons
      For j = 1 To outputNeurons
         W3(i, j) = (MinMax * 2 * Rnd) - MinMax
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To h1Neurons
         bW1(i) = (MinMax * 2 * Rnd) - MinMax
      Next i
      For i = 1 To h2Neurons
         bW2(i) = (MinMax * 2 * Rnd) - MinMax
      Next i
      For i = 1 To outputNeurons
         bW3(i) = (MinMax * 2 * Rnd) - MinMax
      Next i
   End If
Else
   For i = 1 To h1Neurons
      For j = 1 To outputNeurons
         W2(i, j) = (MinMax * 2 * Rnd) - MinMax
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To h1Neurons
         bW1(i) = (MinMax * 2 * Rnd) - MinMax
      Next i
      For i = 1 To outputNeurons
         bW2(i) = (MinMax * 2 * Rnd) - MinMax
```

```
    Next i
  End If
End If
End Sub

'For better memory usage we have dynamic arrays.
'At the beginning they are only pointers, now
'the arrays will be created only the size we need
'There is no static fixed size array
Public Sub redimArrays()

ReDim LinesPointer1(1 To inputNeurons, 1 To h1Neurons)
ReDim W1(1 To inputNeurons, 1 To h1Neurons)
ReDim FirstDer1(1 To inputNeurons, 1 To h1Neurons)
ReDim SecondDer1(1 To inputNeurons, 1 To h1Neurons)
ReDim dW1(1 To inputNeurons, 1 To h1Neurons)
ReDim dW1(1 To inputNeurons, 1 To h1Neurons)

ReDim alpha1(1 To inputNeurons, 1 To h1Neurons)
ReDim delta1(1 To inputNeurons, 1 To h1Neurons)

ReDim h1(1 To h1Neurons)
ReDim o(1 To outputNeurons)

ReDim errorOut(1 To outputNeurons)
ReDim errorH1(1 To h1Neurons)
If bias = 1 Then
  ReDim bW1(1 To h1Neurons)
  ReDim dbW1(1 To h1Neurons)
  ReDim FirstDerb1(1 To h1Neurons)
  ReDim SecondDerb1(1 To h1Neurons)

  ReDim alphab1(1 To h1Neurons)
  ReDim deltab1(1 To h1Neurons)
End If
If hLayers = 2 Then
  ReDim LinesPointer2(1 To h1Neurons, 1 To h2Neurons)
  ReDim LinesPointer3(1 To h2Neurons, 1 To outputNeurons)

  ReDim W2(1 To h1Neurons, 1 To h2Neurons)
  ReDim W3(1 To h2Neurons, 1 To outputNeurons)

  ReDim alpha2(1 To h1Neurons, 1 To h2Neurons)
  ReDim alpha3(1 To h2Neurons, 1 To outputNeurons)
  ReDim delta2(1 To h1Neurons, 1 To h2Neurons)
  ReDim delta3(1 To h2Neurons, 1 To outputNeurons)

  ReDim FirstDer2(1 To h1Neurons, 1 To h2Neurons)
  ReDim FirstDer3(1 To h2Neurons, 1 To outputNeurons)
  ReDim SecondDer2(1 To h1Neurons, 1 To h2Neurons)
  ReDim SecondDer3(1 To h2Neurons, 1 To outputNeurons)
  ReDim dW2(1 To h1Neurons, 1 To h2Neurons)
  ReDim dW3(1 To h2Neurons, 1 To outputNeurons)
  ReDim h2(1 To h2Neurons)
  ReDim errorH2(1 To h2Neurons)
  If bias = 1 Then
    ReDim bW2(1 To h2Neurons)
    ReDim bW3(1 To outputNeurons)

    ReDim alphab2(1 To h2Neurons)
    ReDim deltab2(1 To h2Neurons)
    ReDim alphab3(1 To outputNeurons)
    ReDim deltab3(1 To outputNeurons)

    ReDim FirstDerb2(1 To h2Neurons)
    ReDim FirstDerb3(1 To outputNeurons)
    ReDim SecondDerb2(1 To h2Neurons)
    ReDim SecondDerb3(1 To outputNeurons)
    ReDim dbW2(1 To h2Neurons)
    ReDim dbW3(1 To outputNeurons)
  End If
Else
```

```
      ReDim LinesPointer2(1 To h1Neurons, 1 To outputNeurons)
      ReDim W2(1 To h1Neurons, 1 To outputNeurons)
      ReDim FirstDer2(1 To h1Neurons, 1 To outputNeurons)
      ReDim SecondDer2(1 To h1Neurons, 1 To outputNeurons)
      ReDim dW2(1 To h1Neurons, 1 To outputNeurons)
      ReDim dW2(1 To h1Neurons, 1 To outputNeurons)

      ReDim alpha2(1 To h1Neurons, 1 To outputNeurons)
      ReDim delta2(1 To h1Neurons, 1 To outputNeurons)

      If bias = 1 Then
         ReDim bW2(1 To outputNeurons)
         ReDim FirstDerb2(1 To outputNeurons)
         ReDim SecondDerb2(1 To outputNeurons)
         ReDim dbW2(1 To outputNeurons)

         ReDim alphab2(1 To outputNeurons)
         ReDim deltab2(1 To outputNeurons)
      End If
   End If
End Sub
'Update the lines after weight update
Public Sub showLines()

For i = 1 To inputNeurons
   For j = 1 To h1Neurons
      Set DummyLine = LinesPointer1(i, j)
      LineValue = W1(i, j)
      updateLine
   Next j
Next i
If bias = 1 Then
   For i = 1 To h1Neurons
      Set DummyLine = mainFrm.bline1(i - 1)
      LineValue = bW1(i)
      updateLine
```

```
   Next
End If
If hLayers = 2 Then
   For i = 1 To h1Neurons
      For j = 1 To h2Neurons
         Set DummyLine = LinesPointer2(i, j)
         LineValue = W2(i, j)
         updateLine
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To h2Neurons
         Set DummyLine = mainFrm.bline2(i - 1)
         LineValue = bW2(i)
         updateLine
      Next i
   End If
   For i = 1 To h2Neurons
      For j = 1 To outputNeurons
         Set DummyLine = LinesPointer3(i, j)
         LineValue = W3(i, j)
         updateLine
      Next j
   Next i
   If bias = 1 Then
      For i = 1 To outputNeurons
         Set DummyLine = mainFrm.bline3(i - 1)
         LineValue = bW3(i)
         updateLine
      Next i
   End If
Else
   For i = 1 To h1Neurons
      For j = 1 To outputNeurons
         Set DummyLine = LinesPointer2(i, j)
         LineValue = W2(i, j)
      updateLine
    Next j
  Next i
  If bias = 1 Then
    For i = 1 To outputNeurons
      Set DummyLine = mainFrm.bline2(i - 1)
      LineValue = bW2(i)
      updateLine
    Next i
  End If
End If
End Sub
'Show the current weights
Public Sub showWs()

On Error GoTo Omit
ActiveRow = 1
frmValues.grdW.Row = ActiveRow
frmValues.grdW.Col = 0
frmValues.grdW.FixedAlignment(0) = 2
frmValues.grdW.Text = "Layer 1"
For i = 1 To inputNeurons
  frmValues.grdW.Row = ActiveRow + i
  frmValues.grdW.Col = 0
  frmValues.grdW.FixedAlignment(0) = 2
  frmValues.grdW.Text = Str$(i)
  For j = 1 To h1Neurons
    LineValue = W1(i, j)
    frmValues.grdW.Row = ActiveRow + i
    frmValues.grdW.Col = j
    frmValues.grdW.ColAlignment(j) = 1
    frmValues.grdW.Text = Format(LineValue, "#0.0000")
  Next j
Next i
ActiveRow = ActiveRow + inputNeurons
If bias = 1 Then
```

```
  ActiveRow = ActiveRow + 1
  frmValues.grdW.Row = ActiveRow
  frmValues.grdW.Col = 0
  frmValues.grdW.FixedAlignment(0) = 2
  frmValues.grdW.Text = "Bias 1"
  For i = 1 To h1Neurons
    LineValue = bW1(i)
    frmValues.grdW.Row = ActiveRow
    frmValues.grdW.Col = i
    frmValues.grdW.Text = Format(LineValue, "#0.0000")
  Next
End If
If hLayers = 2 Then
  ActiveRow = ActiveRow + 1
  frmValues.grdW.Row = ActiveRow
  frmValues.grdW.Col = 0
  frmValues.grdW.FixedAlignment(0) = 2
  frmValues.grdW.Text = "Layer 2"
  For i = 1 To h1Neurons
    frmValues.grdW.Row = ActiveRow + i
    frmValues.grdW.Col = 0
    frmValues.grdW.FixedAlignment(0) = 2
    frmValues.grdW.Text = Str$(i)
    For j = 1 To h2Neurons
      LineValue = W2(i, j)
      frmValues.grdW.Row = ActiveRow + i
      frmValues.grdW.Col = j
      frmValues.grdW.Text = Format(LineValue, "#0.0000")
    Next j
  Next i
  ActiveRow = ActiveRow + h1Neurons
  If bias = 1 Then
    ActiveRow = ActiveRow + 1
    frmValues.grdW.Row = ActiveRow
    frmValues.grdW.Col = 0
    frmValues.grdW.FixedAlignment(0) = 2
    frmValues.grdW.Text = "Bias 2"
    For i = 1 To h2Neurons
      LineValue = bW2(i)
      frmValues.grdW.Row = ActiveRow
      frmValues.grdW.Col = i
      frmValues.grdW.Text = Format(LineValue, "#0.0000")
    Next i
  End If
  ActiveRow = ActiveRow + 1
  frmValues.grdW.Row = ActiveRow
  frmValues.grdW.Col = 0
  frmValues.grdW.FixedAlignment(0) = 2
  frmValues.grdW.Text = "Layer 3"
  For i = 1 To h2Neurons
    frmValues.grdW.Row = ActiveRow + i
    frmValues.grdW.Col = 0
    frmValues.grdW.FixedAlignment(0) = 2
    frmValues.grdW.Text = Str$(i)
    For j = 1 To outputNeurons
      LineValue = W3(i, j)
      frmValues.grdW.Row = ActiveRow + i
      frmValues.grdW.Col = j
      frmValues.grdW.Text = Format(LineValue, "#0.0000")
    Next j
  Next i
  ActiveRow = ActiveRow + h2Neurons
  If bias = 1 Then
    ActiveRow = ActiveRow + 1
    frmValues.grdW.Row = ActiveRow
    frmValues.grdW.Col = 0
    frmValues.grdW.FixedAlignment(0) = 2
    frmValues.grdW.Text = "Bias 3"
    For i = 1 To outputNeurons
      LineValue = bW3(i)
      frmValues.grdW.Row = ActiveRow
      frmValues.grdW.Col = i
      frmValues.grdW.Text = Format(LineValue, "#0.0000")
    Next i
  End If
Else
    ActiveRow = ActiveRow + 1
    frmValues.grdW.Row = ActiveRow
    frmValues.grdW.Col = 0
    frmValues.grdW.FixedAlignment(0) = 2
    frmValues.grdW.Text = "Layer 2"

    For i = 1 To h1Neurons
      frmValues.grdW.Row = ActiveRow + i
      frmValues.grdW.Col = 0
      frmValues.grdW.FixedAlignment(0) = 2
      frmValues.grdW.Text = Str$(i)
     For j = 1 To outputNeurons
      LineValue = W2(i, j)
      frmValues.grdW.Row = ActiveRow + i
      frmValues.grdW.Col = j
      frmValues.grdW.Text = Format(LineValue, "#0.0000")
    Next j
  Next i
  ActiveRow = ActiveRow + h1Neurons
  If bias = 1 Then
    ActiveRow = ActiveRow + 1
    frmValues.grdW.Row = ActiveRow
    frmValues.grdW.Col = 0
    frmValues.grdW.FixedAlignment(0) = 2
    frmValues.grdW.Text = "Bias 2"
    For i = 1 To outputNeurons
      LineValue = bW2(i)
      frmValues.grdW.Row = ActiveRow
      frmValues.grdW.Col = i
      frmValues.grdW.Text = Format(LineValue, "#0.0000")
    Next i
  End If
End If
Omit:
End Sub
'Change the color and/or size of the line
'decided by the weight value
Public Sub updateLine()

'if less than threshold dont show
If Abs(LineValue) < LineThreshold Then
   DummyLine.Visible = False
'Bright White
ElseIf Abs(LineValue) < 2 Then
   DummyLine.BorderColor = QBColor(15)
   DummyLine.BorderWidth = 1
   DummyLine.Visible = True
'Light Yellow
ElseIf Abs(LineValue) < 3 Then
   DummyLine.BorderColor = QBColor(14)
   DummyLine.BorderWidth = 1
   DummyLine.Visible = True
'Light Blue
ElseIf Abs(LineValue) < 4 Then
   DummyLine.BorderColor = QBColor(9)
   DummyLine.BorderWidth = 1
   DummyLine.Visible = True
'Light Blue Size=2
ElseIf Abs(LineValue) < 6 Then
   DummyLine.BorderColor = QBColor(9)
   DummyLine.BorderWidth = 2
   DummyLine.Visible = True
'Black
ElseIf Abs(LineValue) < 8 Then
   DummyLine.BorderColor = QBColor(0)
   DummyLine.BorderWidth = 2
   DummyLine.Visible = True
'Light Magenta
ElseIf Abs(LineValue) < 11 Then
   DummyLine.BorderColor = QBColor(13)
   DummyLine.BorderWidth = 3
   DummyLine.Visible = True
'Light red
ElseIf Abs(LineValue) < 14 Then
   DummyLine.BorderColor = QBColor(12)
   DummyLine.BorderWidth = 3
   DummyLine.Visible = True
'Light red size=4
ElseIf Abs(LineValue) < 28 Then
   DummyLine.BorderColor = QBColor(12)
   DummyLine.BorderWidth = 4
   DummyLine.Visible = True
'Light red size=5
ElseIf Abs(LineValue) > 28 Then
   DummyLine.BorderColor = QBColor(12)
   DummyLine.BorderWidth = 5
   DummyLine.Visible = True
End If
End Sub

Attribute VB_Name = "mainFrm"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Public inIndex As Integer   'Index of the current I/O pair
Public RMS As Single        'RMS value
Public training As Boolean  'Training or test mode
'Accumulate deltas for batch mode
Private Sub addDeltas()
If hLayers = 2 Then         'if two hidden layers
  For i = 1 To h2Neurons
     For j = 1 To outputNeurons
        delta3(i, j) = delta3(i, j) + dW3(i, j)

     Next j
  Next i
    If bias = 1 Then    'if bias exist
    For j = 1 To outputNeurons
      deltab3(j) = deltab3(j) + dbW3(j)
    Next
    End If
  For i = 1 To h1Neurons
    For j = 1 To h2Neurons
      delta2(i, j) = delta2(i, j) + dW2(i, j)
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To h2Neurons
      deltab2(j) = deltab2(j) + dbW2(j)
    Next
    End If
Else                    'if one hidden layer
  For i = 1 To h1Neurons
    For j = 1 To outputNeurons

      delta2(i, j) = delta2(i, j) + dW2(i, j)
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To outputNeurons
      deltab2(j) = deltab2(j) + dbW2(j)
    Next
    End If
End If

'Whether one or two hidden layers
For i = 1 To inputNeurons
  For j = 1 To h1Neurons
    delta1(i, j) = delta1(i, j) + dW1(i, j)
  Next j
Next i
```

```
  If bias = 1 Then
  For j = 1 To h1Neurons
    deltab1(j) = deltab1(j) + dbW1(j)
  Next
  End If
End Sub
'Compute delta weight values
'
Private Sub computeDeltas()
Dim TakeStep As Single

TakeStep = 0#

If hLayers = 2 Then         'if two hidden layers
 For i = 1 To h2Neurons
    For j = 1 To outputNeurons
      TakeStep = LearningRate * FirstDer3(i, j)
      dW3(i, j) = TakeStep

    Next j
  Next i
    If bias = 1 Then    'if bias exist
    For j = 1 To outputNeurons
      TakeStep = LearningRate * FirstDerb3(j)
      dbW3(j) = TakeStep
    Next
    End If
  For i = 1 To h1Neurons
    For j = 1 To h2Neurons
      TakeStep = LearningRate * FirstDer2(i, j)
      dW2(i, j) = TakeStep
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To h2Neurons
      TakeStep = LearningRate * FirstDerb2(j)
      dbW2(j) = TakeStep
    Next
    End If
Else                    'if one hidden layer
  For i = 1 To h1Neurons
    For j = 1 To outputNeurons

      TakeStep = LearningRate * FirstDer2(i, j)
      dW2(i, j) = TakeStep
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To outputNeurons
      TakeStep = LearningRate * FirstDerb2(j)
      dbW2(j) = TakeStep
    Next
    End If
End If

'Whether one or two hidden layers
For i = 1 To inputNeurons
  For j = 1 To h1Neurons
    TakeStep = LearningRate * FirstDer1(i, j)
    dW1(i, j) = TakeStep
  Next j
Next i
  If bias = 1 Then
  For j = 1 To h1Neurons
    TakeStep = LearningRate * FirstDerb1(j)
    dbW1(j) = TakeStep
  Next
  End If
End Sub
'Delay the simulation by the value decided by "Speed"
Private Sub delay()
  For i = 1 To Speed
```

```
    DoEvents
  Next
End Sub
'Compute the error vectors and
'error vs. weight derivatives
Public Sub errorVectors()
Dim temp As Single
Dim i, j As Byte

'Output layer error vector
For i = 1 To outputNeurons
  errorOut(i) = o(i) * (1 - o(i)) * (oTrnMatrix(inIndex, i) - o(i))
Next
If hLayers = 2 Then        'if two hidden layers
  For i = 1 To h2Neurons
    temp = 0#
    For j = 1 To outputNeurons
      temp = temp + errorOut(j) * W3(i, j)
      'save the previous as second
      SecondDer3(i, j) = FirstDer3(i, j)
      'first derivative
      FirstDer3(i, j) = errorOut(j) * h2(i)
    Next j
    'Second hidden layer error vector
    errorH2(i) = h2(i) * (1 - h2(i)) * temp
  Next i
  For i = 1 To h1Neurons
    temp = 0#
    For j = 1 To h2Neurons
      temp = temp + errorH2(j) * W2(i, j)
      SecondDer2(i, j) = FirstDer2(i, j)
      FirstDer2(i, j) = errorH2(j) * h1(i)
    Next j
    errorH1(i) = h1(i) * (1 - h1(i)) * temp
  Next i
  If bias = 1 Then   'if bias exists
```

```
    For i = 1 To outputNeurons
      'save the previous as second
      SecondDerb3(i) = FirstDerb3(i)
      'first derivative
      FirstDerb3(i) = errorOut(i)
    Next
    For i = 1 To h2Neurons
      SecondDerb2(i) = FirstDerb2(i)
      FirstDerb2(i) = errorH2(i)
    Next
  End If
Else              'if one hidden layer
  For i = 1 To h1Neurons
    temp = 0#
    For j = 1 To outputNeurons
      temp = temp + errorOut(j) * W2(i, j)
      SecondDer2(i, j) = FirstDer2(i, j)
      FirstDer2(i, j) = errorOut(j) * h1(i)
    Next j
    errorH1(i) = h1(i) * (1 - h1(i)) * temp
  Next i
  If bias = 1 Then
    For i = 1 To outputNeurons
      SecondDerb2(i) = FirstDerb2(i)
      FirstDerb2(i) = errorOut(i)
    Next
  End If
End If

'Whether one or two hidden layers
For i = 1 To inputNeurons
  For j = 1 To h1Neurons
    SecondDer1(i, j) = FirstDer1(i, j)
    FirstDer1(i, j) = errorH1(j) * iTrnMatrix(inIndex, i)
  Next j
Next i
```

```
If bias = 1 Then
  For i = 1 To h1Neurons
    SecondDerb1(i) = FirstDerb1(i)
    FirstDerb1(i) = errorH1(i)
  Next
End If
End Sub
Private Sub forwardBP()
Dim UpdateThis As Single

For i = 0 To inputNeurons - 1
  Set CurrentDisc = InputShp(i)                 'Update input layer disc
  If training Then
    CurrentDiscValue = iTrnMatrix(inIndex, i + 1)   'training mode
  Else
    CurrentDiscValue = iTestMatrix(inIndex, i + 1)  'test mode

  End If
  updateDisc
Next i
frmValues.grdIO.Col = 0
For i = 1 To inputNeurons       'Update values form(inputs)
  frmValues.grdIO.Row = i
  If training Then
    UpdateThis = iTrnMatrix(inIndex, i)
  Else
    UpdateThis = iTestMatrix(inIndex, i)
  End If
  frmValues.grdIO.Text = Format(UpdateThis, "0.0###")
Next i
frmValues.grdIO.Col = 1
For i = 1 To outputNeurons      'Update values form(outputs)
  frmValues.grdIO.Row = i
  If training Then
    UpdateThis = oTrnMatrix(inIndex, i)
  Else
```

```
        UpdateThis = oTestMatrix(inIndex, i)
    End If
    frmValues.grdIO.Text = Format(UpdateThis, "0.0###")
Next i

'FORWARD Pass
'Input to first hidden layer
For i = 1 To h1Neurons
    temp = 0
        If training Then
            For j = 1 To inputNeurons
                temp = temp + (iTrnMatrix(inIndex, j) * W1(j, i))
            Next j
        Else
            For j = 1 To inputNeurons
                temp = temp + (iTestMatrix(inIndex, j) * W1(j, i))
            Next j
        End If
            h1(i) = temp
            If bias = 1 Then    'Add if bias
                h1(i) = h1(i) + bW1(i)
            End If
            'Update adders
            CurrentAddValue = h1(i)
            Set CurrentAdder = AddGauge1(i - 1)
            CurrentAddValue = h1(i)
            updateAdder

            h1(i) = sigmoid(h1(i))
            Set CurrentDisc = fOut1(i - 1)
            CurrentDiscValue = h1(i)
            updateDisc
Next i

If hLayers = 2 Then                'if two hidden layers
    'First to second hidden
```

```
For i = 1 To h2Neurons
    temp = 0
    For j = 1 To h1Neurons
        temp = temp + (h1(j) * W2(j, i))
    Next j
    h2(i) = temp
    If bias = 1 Then
        h2(i) = h2(i) + bW2(i)
    End If
    CurrentAddValue = h2(i)
    Set CurrentAdder = AddGauge2(i - 1)
    CurrentAddValue = h1(i)
    updateAdder
    h2(i) = sigmoid(h2(i))
    Set CurrentDisc = fOut2(i - 1)  'Update outputs
    CurrentDiscValue = h2(i)
    updateDisc
Next i
'Second hidden to output layer
For i = 1 To outputNeurons
    temp = 0
    For j = 1 To h2Neurons
        temp = temp + (h2(j) * W3(j, i))
    Next
    o(i) = temp
    If bias = 1 Then
        o(i) = o(i) + bW3(i)
    End If
    CurrentAddValue = o(i)
    Set CurrentAdder = AddGauge3(i - 1)
    CurrentAddValue = o(i)
    updateAdder
    o(i) = sigmoid(o(i))
    Set CurrentDisc = fOut3(i - 1)
    CurrentDiscValue = o(i)
    updateDisc
```

```
   Next i
Else                    'if one hidden layer
   'first hidden to output layer
   For i = 1 To outputNeurons
      temp = 0
      For j = 1 To h1Neurons
         temp = temp + (h1(j) * W2(j, i))
      Next
      o(i) = temp
      If bias = 1 Then
         o(i) = o(i) + bW2(i)
      End If
      CurrentAddValue = o(i)
      Set CurrentAdder = AddGauge2(i - 1)
      CurrentAddValue = o(i)
      updateAdder
      o(i) = sigmoid(o(i))
      Set CurrentDisc = fOut2(i - 1)
      CurrentDiscValue = o(i)
      updateDisc
   Next i
End If
'Update values and outputs
frmValues.grdIO.Col = 2
For i = 1 To outputNeurons
   frmValues.grdIO.Row = i
   UpdateThis = o(i)
   frmValues.grdIO.Text = Format(UpdateThis, "0.0###")
Next i
End Sub
'Returns the delta learning rate for the current weight
'
Function GetDelta(AvgDelta, DeltaNow, AlphaNow)

If AvgDelta * DeltaNow > 0 Then          'if Delta and Avg. Delta have same signs
   GetDelta = LearningRate
```

```
ElseIf AvgDelta * DeltaNow < 0 Then      'if different signs
   GetDelta = -PHI * AlphaNow

Else                                     'if they are both equal to zero
   GetDelta = 0#

End If

End Function

'Returns the output of the current activation(adder)
'
Public Function sigmoid(ByVal Activation As Single) As Single

If Activation > 15 Then
   sigmoid = 1#
ElseIf Activation < -15 Then
   sigmoid = 0#

Else
   sigmoid = 1# / (1# + Exp(-Activation))
End If

End Function

'Returns the output layer square error
'
Private Function thisError() As Single
Dim TempError As Single

   TempError = 0#
   thisError = 0#
   frmValues.grdIO.Col = 3

   'Compute the error for each output layer neuron
```

```
  For i = 1 To outputNeurons
  If training Then
    TempError = (o(i) - oTrnMatrix(inIndex, i)) * (o(i) - oTrnMatrix(inIndex,
i))
  Else
    TempError = (o(i) - oTestMatrix(inIndex, i)) * (o(i) - oTestMatrix(inIndex,
i))
  End If

    frmValues.grdIO.Row = i
    frmValues.grdIO.Text = Format(TempError, "0.0###")
    thisError = thisError + TempError
  Next
  thisError = thisError / outputNeurons
End Function

Private Sub updateAdder()
On Error Resume Next
If CurrentAddValue = Null Then
  CurrentAddValue = 0
End If

If CurrentAddValue < 0 Then           'Adder is negative
  CurrentAdder.ForeColor = NegativeColor
Else
  CurrentAdder.ForeColor = PositiveColor  'Adder is positive
End If
If Abs(CurrentAddValue) > 100 Then
  CurrentAddValue = 100
End If

'Update the current adder
CurrentAdder.Value = CInt(Abs(CurrentAddValue))

End Sub
```

```
Private Sub updateDisc()
Dim i, j As Integer

If CurrentDiscValue = Null Then
  CurrentDiscValue = 1
End If

If CurrentDiscValue < 0 Then           'Disc value is negative
  CurrentDisc.BorderColor = NegativeColor
  CurrentDisc.FillColor = NegativeColor
Else
  CurrentDisc.BorderColor = PositiveColor 'Disc value is positive
  CurrentDisc.FillColor = PositiveColor
End If

'Update the current disc
If Abs(CurrentDiscValue) > 1 Then
  CurrentDisc.BorderWidth = 12
Else
  CurrentDisc.BorderWidth = Abs(CInt(CurrentDiscValue * 10)) + 1
End If
End Sub

Private Sub updateWsBP()
'Delta weight is calculated by the following:
'Dw=Learning rate * First derivative + Momentum * Previous Dw
'Weight update:
'W=W + Dw

Dim TakeStep As Single

NetworkSaved = False
TakeStep = 0#

If hLayers = 2 Then
  For i = 1 To h2Neurons
```

```
    For j = 1 To outputNeurons

        'Dw=Learning rate * First derivative + Momentum * Previous Dw
        'W=W + Dw
        TakeStep = LearningRate * FirstDer3(i, j) + Momentum * dW3(i, j)
        W3(i, j) = W3(i, j) + TakeStep
        dW3(i, j) = TakeStep

        'Update the line(weight)
        Set DummyLine = LinesPointer3(i, j)
        LineValue = W3(i, j)
        updateLine
    Next j
Next i
    If bias = 1 Then    'If bias exist
    For j = 1 To outputNeurons
        TakeStep = LearningRate * FirstDerb3(j) + Momentum * dbW3(j)
        bW3(j) = bW3(j) + TakeStep

        dbW3(j) = TakeStep

        Set DummyLine = bline3(j - 1)
        LineValue = bW3(j)
        updateLine
    Next
    End If

For i = 1 To h1Neurons
    For j = 1 To h2Neurons
        TakeStep = LearningRate * FirstDer2(i, j) + Momentum * dW2(i, j)
        W2(i, j) = W2(i, j) + TakeStep
        dW2(i, j) = TakeStep
        Set DummyLine = LinesPointer2(i, j)
        LineValue = W2(i, j)

        updateLine
```

```
        Next j
    Next i
        If bias = 1 Then
        For j = 1 To h2Neurons
            TakeStep = LearningRate * FirstDerb2(j) + Momentum *
dbW2(j)
            bW2(j) = bW2(j) + TakeStep
            dbW2(j) = TakeStep
            Set DummyLine = bline2(j - 1)
            LineValue = bW2(j)
            updateLine
        Next
        End If

Else

    For i = 1 To h1Neurons
        For j = 1 To outputNeurons

            TakeStep = LearningRate * FirstDer2(i, j) + Momentum *
dW2(i, j)
            W2(i, j) = W2(i, j) + TakeStep

            dW2(i, j) = TakeStep
            Set DummyLine = LinesPointer2(i, j)
            LineValue = W2(i, j)

            updateLine
        Next j
    Next i
        If bias = 1 Then
        For j = 1 To outputNeurons
            TakeStep = LearningRate * FirstDerb2(j) + Momentum *
dbW2(j)
            bW2(j) = bW2(j) + TakeStep
            dbW2(j) = TakeStep
```

```
      Set DummyLine = bline2(j - 1)
      LineValue = bW2(j)
      updateLine
    Next
    End If
End If
For i = 1 To inputNeurons
  For j = 1 To h1Neurons
    TakeStep = LearningRate * FirstDer1(i, j) + Momentum * dW1(i, j)
    W1(i, j) = W1(i, j) + TakeStep
    dW1(i, j) = TakeStep

    Set DummyLine = LinesPointer1(i, j)
    LineValue = W1(i, j)
    updateLine
  Next j
Next i
  If bias = 1 Then
  For j = 1 To h1Neurons
    TakeStep = LearningRate * FirstDerb1(j) + Momentum * dbW1(j)
    bW1(j) = bW1(j) + TakeStep
    dbW1(j) = TakeStep
    Set DummyLine = bline1(j - 1)
    LineValue = bW1(j)
    updateLine
  Next
  End If
End Sub

Private Sub updateWsCG()

'Weight update:
'W=W + Dw
'Dw is the accumulated delta weight for an epoch
```

```
NetworkSaved = False

If hLayers = 2 Then
  For i = 1 To h2Neurons
    For j = 1 To outputNeurons
      W3(i, j) = W3(i, j) + delta3(i, j)

      Set DummyLine = LinesPointer3(i, j)
      LineValue = W3(i, j)
      updateLine
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To outputNeurons
      bW3(j) = bW3(j) + deltab3(j)

      Set DummyLine = bline3(j - 1)
      LineValue = bW3(j)
      updateLine
    Next
    End If

  For i = 1 To h1Neurons
    For j = 1 To h2Neurons
      W2(i, j) = W2(i, j) + delta2(i, j)

      Set DummyLine = LinesPointer2(i, j)
      LineValue = W2(i, j)
      updateLine
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To h2Neurons
      bW2(j) = bW2(j) + deltab2(j)

      Set DummyLine = bline2(j - 1)
      LineValue = bW2(j)
      updateLine
    Next
    End If

Else

  For i = 1 To h1Neurons
    For j = 1 To outputNeurons

      W2(i, j) = W2(i, j) + delta2(i, j)

      Set DummyLine = LinesPointer2(i, j)
      LineValue = W2(i, j)
      updateLine
    Next j
  Next i
    If bias = 1 Then
    For j = 1 To outputNeurons
      bW2(j) = bW2(j) + deltab2(j)

      Set DummyLine = bline2(j - 1)
      LineValue = bW2(j)
      updateLine
    Next
    End If
End If
For i = 1 To inputNeurons
  For j = 1 To h1Neurons
    W1(i, j) = W1(i, j) + delta1(i, j)

    Set DummyLine = LinesPointer1(i, j)
    LineValue = W1(i, j)
    updateLine
  Next j
Next i
```

```
  If bias = 1 Then
  For j = 1 To h1Neurons
    bW1(j) = bW1(j) + deltab1(j)

    Set DummyLine = bline1(j - 1)
    LineValue = bW1(j)
    updateLine
  Next
  End If
End Sub

Private Sub updateWsDBD()
Dim AvgDelta, ProducedDelta, DeltaNow, AlphaNow

NetworkSaved = False

If hLayers = 2 Then
  For i = 1 To h2Neurons
    For j = 1 To outputNeurons

      'Calculate the delta for current weight
      AvgDelta = (1 - THETA) * FirstDer3(i, j) + THETA * delta3(i,
j)

      DeltaNow = delta3(i, j)
      AlphaNow = alpha3(i, j)

      'Calculate the delta learning rate for this weight
      ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

      delta3(i, j) = FirstDer3(i, j)

      'Learning rate = Learning rate + Delta learning rate
      alpha3(i, j) = alpha3(i, j) + ProducedDelta

      'Set learning rate to the maximum
```

```
         If alpha3(i, j) > MaxLearningRate Then
            alpha3(i, j) = MaxLearningRate
         End If

         'Calculate the delta weight
         dW3(i, j) = alpha3(i, j) * FirstDer3(i, j)

         'Update the weight
         W3(i, j) = W3(i, j) + dW3(i, j)

         'Update the weight line
         Set DummyLine = LinesPointer3(i, j)
         LineValue = W3(i, j)
         updateLine
      Next j
   Next i
   If bias = 1 Then      'If bias exists
    For j = 1 To outputNeurons
      AvgDelta = (1 - THETA) * FirstDerb3(j) + THETA * deltab3(j)

      DeltaNow = deltab3(j)
      AlphaNow = alphab3(j)

      ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

      deltab3(j) = FirstDerb3(j)
      alphab3(j) = alphab3(j) + ProducedDelta

      If alphab3(j) > MaxLearningRate Then
         alphab3(j) = MaxLearningRate
      End If

      dbW3(j) = alphab3(j) * FirstDerb3(j)
      bW3(j) = bW3(j) + dbW3(j)

      Set DummyLine = bline3(j - 1)
```

```
      LineValue = bW3(j)
      updateLine
    Next
   End If

   For i = 1 To h1Neurons
      For j = 1 To h2Neurons
         AvgDelta = (1 - THETA) * FirstDer2(i, j) + THETA * delta2(i,
j)

         DeltaNow = delta2(i, j)
         AlphaNow = alpha2(i, j)

         ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

         delta2(i, j) = FirstDer2(i, j)
         alpha2(i, j) = alpha2(i, j) + ProducedDelta

         If alpha2(i, j) > MaxLearningRate Then
            alpha2(i, j) = MaxLearningRate
         End If
         dW2(i, j) = alpha2(i, j) * FirstDer2(i, j)
         W2(i, j) = W2(i, j) + dW2(i, j)

         Set DummyLine = LinesPointer2(i, j)
         LineValue = W2(i, j)

         updateLine
      Next j
   Next i
   If bias = 1 Then
      For j = 1 To h2Neurons
         AvgDelta = (1 - THETA) * FirstDerb2(j) + THETA * deltab2(j)

         DeltaNow = deltab2(j)
         AlphaNow = alphab2(j)
         ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

         deltab2(j) = FirstDerb2(j)
         alphab2(j) = alphab2(j) + ProducedDelta

         If alphab2(j) > MaxLearningRate Then
            alphab2(j) = MaxLearningRate
         End If

         dbW2(j) = alphab2(j) * FirstDerb2(j)
         bW2(j) = bW2(j) + dbW2(j)
         Set DummyLine = bline2(j - 1)
         LineValue = bW2(j)
         updateLine
      Next
   End If

Else                    'if one hidden layer

   For i = 1 To h1Neurons
      For j = 1 To outputNeurons
         AvgDelta = (1 - THETA) * FirstDer2(i, j) + THETA * delta2(i, j)

         DeltaNow = delta2(i, j)
         AlphaNow = alpha2(i, j)

         ProducedDelta = GetDelta(AvgDelta. DeltaNow, AlphaNow)

         delta2(i, j) = FirstDer2(i. j)
         alpha2(i, j) = alpha2(i. j) + ProducedDelta

         If alpha2(i. j) > MaxLearningRate Then
            alpha2(i, j) = MaxLearningRate
         End If

         dW2(i, j) = alpha2(i, j) * FirstDer2(i, j)
         W2(i, j) = W2(i, j) + dW2(i, j)

         Set DummyLine = LinesPointer2(i, j)
         LineValue = W2(i, j)

         updateLine
      Next j
   Next i
   If bias = 1 Then
      For j = 1 To outputNeurons
         AvgDelta = (1 - THETA) * FirstDerb2(j) + THETA * deltab2(j)

         DeltaNow = deltab2(j)
         AlphaNow = alphab2(j)

         ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

         deltab2(j) = FirstDerb2(j)
         alphab2(j) = alphab2(j) + ProducedDelta

         If alphab2(j) > MaxLearningRate Then
            alphab2(j) = MaxLearningRate
         End If

         dbW2(j) = alphab2(j) * FirstDerb2(j)
         bW2(j) = bW2(j) + dbW2(j)

         Set DummyLine = bline2(j - 1)
         LineValue = bW2(j)
         updateLine
      Next
   End If
End If

'Whether one or two hidden layers
```

```
For i = 1 To inputNeurons
   For j = 1 To h1Neurons
       AvgDelta = (1 - THETA) * FirstDer1(i, j) + THETA * delta1(i, j)

       DeltaNow = delta1(i, j)
       AlphaNow = alpha1(i, j)

       ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

       delta1(i, j) = FirstDer1(i, j)
       alpha1(i, j) = alpha1(i, j) + ProducedDelta

       If alpha1(i, j) > MaxLearningRate Then
            alpha1(i, j) = MaxLearningRate
       End If

       dW1(i, j) = alpha1(i, j) * FirstDer1(i, j)
       W1(i, j) = W1(i, j) + dW1(i, j)

      Set DummyLine = LinesPointer1(i, j)
      LineValue = W1(i, j)
      updateLine
   Next j
Next i
If bias = 1 Then
   For j = 1 To h1Neurons
      AvgDelta = (1 - THETA) * FirstDerb1(j) + THETA * deltab1(j)

      DeltaNow = deltab1(j)
      AlphaNow = alphab1(j)

      ProducedDelta = GetDelta(AvgDelta, DeltaNow, AlphaNow)

      deltab1(j) = FirstDerb1(j)
      alphab1(j) = alphab1(j) + ProducedDelta

      If alphab1(j) > MaxLearningRate Then
            alphab1(j) = MaxLearningRate
      End If

      dbW1(j) = alphab1(j) * FirstDerb1(j)
      bW1(j) = bW1(j) + dbW1(j)

      Set DummyLine = bline1(j - 1)
      LineValue = bW1(j)
      updateLine
   Next
End If
End Sub

Private Sub updateWsQP()
Dim TakeStep As Single
Dim ShrinkFactor As Single

NetworkSaved = False
TakeStep = 0#

'Calculate the factor for taking maximum step size
ShrinkFactor = MaxFactor / (1# + MaxFactor)

If hLayers = 2 Then
  For i = 1 To h2Neurons
      For j = 1 To outputNeurons
         TakeStep = 0#
         If dW3(i, j) > SwitchThreshold Then  'Do QP

            If FirstDer3(i, j) > 0# Then    'Take step with learning rate
               TakeStep = LearningRate * FirstDer3(i, j)
            End If

            If FirstDer3(i, j) > (ShrinkFactor * SecondDer3(i, j)) Then
               'Don't take more than this
```

```
          TakeStep = TakeStep + MaxFactor * dW3(i, j)
        Else
          'Do QP
          'This step is less than maximum one
          TakeStep = TakeStep + (FirstDer3(i, j) / (SecondDer3(i, j) -
FirstDer3(i, j))) * dW3(i, j)
        End If

      ElseIf dW3(i, j) < -SwitchThreshold Then 'Do QP
      '(Same as above)
        If FirstDer3(i, j) < 0# Then
          TakeStep = LearningRate * FirstDer3(i, j)
        End If
        If FirstDer3(i, j) < (ShrinkFactor * SecondDer3(i, j)) Then
          TakeStep = TakeStep + MaxFactor * dW3(i, j)
        Else
          TakeStep = TakeStep + (FirstDer3(i, j) / (SecondDer3(i, j) -
FirstDer3(i, j))) * dW3(i, j)
        End If
      Else   'Do gradient descent, complete with momentum
        TakeStep = LearningRate * FirstDer3(i, j) + Momentum * dW3(i, j)
      End If

      'Update the current weight
      W3(i, j) = W3(i, j) + TakeStep
      dW3(i, j) = TakeStep

      'Update the current line(weight)
      Set DummyLine = LinesPointer3(i, j)
      LineValue = W3(i, j)
      updateLine
    Next j
  Next i

  If bias = 1 Then             'if bias exist(same)
    For j = 1 To outputNeurons
```

```
      TakeStep = 0#
      If dbW3(j) > SwitchThreshold Then
        If FirstDerb3(j) > 0# Then
          TakeStep = LearningRate * FirstDerb3(j)
        End If
        If FirstDerb3(j) > (ShrinkFactor * SecondDerb3(j)) Then
          TakeStep = TakeStep + MaxFactor * dbW3(j)
        Else
          TakeStep = TakeStep + (FirstDerb3(j) / (SecondDerb3(j) -
FirstDerb3(j))) * dbW3(j)
        End If

      ElseIf dbW3(j) < -SwitchThreshold Then
        If FirstDerb3(j) < 0# Then
          TakeStep = LearningRate * FirstDerb3(j)
        End If
        If FirstDerb3(j) < (ShrinkFactor * SecondDerb3(j)) Then
          TakeStep = TakeStep + MaxFactor * dbW3(j)
        Else
          TakeStep = TakeStep + (FirstDerb3(j) / (SecondDerb3(j) -
FirstDerb3(j))) * dbW3(j)
        End If
      Else
        TakeStep = LearningRate * FirstDerb3(j) + Momentum *
dbW3(j)
      End If
      bW3(j) = bW3(j) + TakeStep

      dbW3(j) = TakeStep

      Set DummyLine = bline3(j - 1)
      LineValue = bW3(j)
      updateLine
    Next
  End If    ' end bias
  For i = 1 To h1Neurons
     For j = 1 To h2Neurons
        TakeStep = 0#

        If dW2(i, j) > SwitchThreshold Then
           If FirstDer2(i, j) > 0# Then
              TakeStep = LearningRate * FirstDer2(i, j)
           End If
           If FirstDer2(i, j) > (ShrinkFactor * SecondDer2(i, j)) Then
              TakeStep = TakeStep + MaxFactor * dW2(i, j)
           Else
              TakeStep = TakeStep + (FirstDer2(i, j) / (SecondDer2(i, j) -
FirstDer2(i, j))) * dW2(i, j)
           End If

        ElseIf dW2(i, j) < -SwitchThreshold Then
           If FirstDer2(i, j) < 0# Then
              TakeStep = LearningRate * FirstDer2(i, j)
           End If
           If FirstDer2(i, j) < (ShrinkFactor * SecondDer2(i, j)) Then
              TakeStep = TakeStep + MaxFactor * dW2(i, j)
           Else
              TakeStep = TakeStep + (FirstDer2(i, j) / (SecondDer2(i, j) -
FirstDer2(i, j))) * dW2(i, j)
           End If
        Else
           TakeStep = LearningRate * FirstDer2(i, j) + Momentum * dW2(i, j)
        End If
        W2(i, j) = W2(i, j) + TakeStep
        dW2(i, j) = TakeStep
        Set DummyLine = LinesPointer2(i, j)
        LineValue = W2(i, j)

        updateLine
     Next j
  Next i
```

```
  If bias = 1 Then
     For j = 1 To h2Neurons
        TakeStep = 0#
        If dbW2(j) > SwitchThreshold Then
           If FirstDerb2(j) > 0# Then
              TakeStep = LearningRate * FirstDerb2(j)
           End If
           If FirstDerb2(j) > (ShrinkFactor * SecondDerb2(j)) Then
              TakeStep = TakeStep + MaxFactor * dbW2(j)
           Else
              TakeStep = TakeStep + (FirstDerb2(j) / (SecondDerb2(j) -
FirstDerb2(j))) * dbW2(j)
           End If

        ElseIf dbW2(j) < -SwitchThreshold Then
           If FirstDerb2(j) < 0# Then
              TakeStep = LearningRate * FirstDerb2(j)
           End If
           If FirstDerb2(j) < (ShrinkFactor * SecondDerb2(j)) Then
              TakeStep = TakeStep + MaxFactor * dbW2(j)
           Else
              TakeStep = TakeStep + (FirstDerb2(j) / (SecondDerb2(j) -
FirstDerb2(j))) * dbW2(j)
           End If
        Else
           TakeStep = LearningRate * FirstDerb2(j) + Momentum *
dbW2(j)
        End If

        bW2(j) = bW2(j) + TakeStep
        dbW2(j) = TakeStep
        Set DummyLine = bline2(j - 1)
        LineValue = bW2(j)
        updateLine
     Next j
  End If 'end bias
```

```
Else   ' # of hidden layers is 1
   For i = 1 To h1Neurons
      For j = 1 To outputNeurons
         TakeStep = 0#

         If dW2(i, j) > SwitchThreshold Then
            If FirstDer2(i, j) > 0# Then
               TakeStep = LearningRate * FirstDer2(i, j)
            End If
            If FirstDer2(i, j) > (ShrinkFactor * SecondDer2(i, j)) Then
               TakeStep = TakeStep + MaxFactor * dW2(i, j)
            Else
               TakeStep = TakeStep + (FirstDer2(i, j) / (SecondDer2(i, j) - FirstDer2(i, j))) * dW2(i, j)
            End If

         ElseIf dW2(i, j) < -SwitchThreshold Then
            If FirstDer2(i, j) < 0# Then
               TakeStep = LearningRate * FirstDer2(i, j)
            End If
            If FirstDer2(i, j) < (ShrinkFactor * SecondDer2(i, j)) Then
               TakeStep = TakeStep + MaxFactor * dW2(i, j)
            Else
               TakeStep = TakeStep + (FirstDer2(i, j) / (SecondDer2(i, j) - FirstDer2(i, j))) * dW2(i, j)
            End If
         Else
            TakeStep = LearningRate * FirstDer2(i, j) + Momentum * dW2(i, j)
         End If
         W2(i, j) = W2(i, j) + TakeStep

         dW2(i, j) = TakeStep
         Set DummyLine = LinesPointer2(i, j)
         LineValue = W2(i, j)

         updateLine
      Next j
   Next i
   If bias = 1 Then
      For j = 1 To outputNeurons
         TakeStep = 0#
         If dbW2(j) > SwitchThreshold Then
            If FirstDerb2(j) > 0# Then
               TakeStep = LearningRate * FirstDerb2(j)
            End If
            If FirstDerb2(j) > (ShrinkFactor * SecondDerb2(j)) Then
               TakeStep = TakeStep + MaxFactor * dbW2(j)
            Else
               TakeStep = TakeStep + (FirstDerb2(j) / (SecondDerb2(j) - FirstDerb2(j))) * dbW2(j)
            End If

         ElseIf dbW2(j) < -SwitchThreshold Then
            If FirstDerb2(j) < 0# Then
               TakeStep = LearningRate * FirstDerb2(j)
            End If
            If FirstDerb2(j) < (ShrinkFactor * SecondDerb2(j)) Then
               TakeStep = TakeStep + MaxFactor * dbW2(j)
            Else
               TakeStep = TakeStep + (FirstDerb2(j) / (SecondDerb2(j) - FirstDerb2(j))) * dbW2(j)
            End If
         Else
            TakeStep = LearningRate * FirstDerb2(j) + Momentum * dbW2(j)
         End If
         bW2(j) = bW2(j) + TakeStep
         dbW2(j) = TakeStep

         Set DummyLine = bline2(j - 1)
         LineValue = bW2(j)
         updateLine
```

```
        Next j
    End If ' end bias
End If
' Whether 1 or 2 hidden layers process this part
For i = 1 To inputNeurons
    For j = 1 To h1Neurons
        TakeStep = 0#

        If dW1(i, j) > SwitchThreshold Then
            If FirstDer1(i, j) > 0# Then
                TakeStep = LearningRate * FirstDer1(i, j)
            End If
            If FirstDer1(i, j) > (ShrinkFactor * SecondDer1(i, j)) Then
                TakeStep = TakeStep + MaxFactor * dW1(i, j)
            Else
                TakeStep = TakeStep + (FirstDer1(i, j) / (SecondDer1(i, j) -
FirstDer1(i, j))) * dW1(i, j)
            End If

        ElseIf dW1(i, j) < -SwitchThreshold Then
            If FirstDer1(i, j) < 0# Then
                TakeStep = LearningRate * FirstDer1(i, j)
            End If
            If FirstDer1(i, j) < (ShrinkFactor * SecondDer1(i, j)) Then
                TakeStep = TakeStep + MaxFactor * dW1(i, j)
            Else
                TakeStep = TakeStep + (FirstDer1(i, j) / (SecondDer1(i, j) -
FirstDer1(i, j))) * dW1(i, j)
            End If
        Else
            TakeStep = LearningRate * FirstDer1(i, j) + Momentum * dW1(i, j)
        End If
        W1(i, j) = W1(i, j) + TakeStep
        dW1(i, j) = TakeStep

        Set DummyLine = LinesPointer1(i, j)
        LineValue = W1(i, j)
        updateLine
    Next j
Next i
If bias = 1 Then
For j = 1 To h1Neurons
        TakeStep = 0#
        If dbW1(j) > SwitchThreshold Then
                If FirstDerb1(j) > 0# Then
                    TakeStep = LearningRate * FirstDerb1(j)
                End If
                If FirstDerb1(j) > (ShrinkFactor * SecondDerb1(j)) Then
                    TakeStep = TakeStep + MaxFactor * dbW1(j)
                Else
                    TakeStep = TakeStep + (FirstDerb1(j) / (SecondDerb1(j) -
FirstDerb1(j))) * dbW1(j)
                End If

        ElseIf dbW1(j) < -SwitchThreshold Then
                If FirstDerb1(j) < 0# Then
                    TakeStep = LearningRate * FirstDerb1(j)
                End If
                If FirstDerb1(j) < (ShrinkFactor * SecondDerb1(j)) Then
                    TakeStep = TakeStep + MaxFactor * dbW1(j)
                Else
                    TakeStep = TakeStep + (FirstDerb1(j) / (SecondDerb1(j) -
FirstDerb1(j))) * dbW1(j)
                End If
        Else
                TakeStep = LearningRate * FirstDerb1(j) + Momentum *
dbW1(j)
        End If
        bW1(j) = bW1(j) + TakeStep
        dbW1(j) = TakeStep
        Set DummyLine = bline1(j - 1)
        LineValue = bW1(j)
        updateLine
    Next j
End If '  end bias
End Sub


Private Sub Downstatus_PanelClick(ByVal Panel As Panel)

    Panel.Bevel = sbrInset
    If Downstatus.Panels(4).Bevel = sbrInset Then

        Continue = True
        Downstatus.Panels(4).Bevel = sbrRaised
    ElseIf Downstatus.Panels(5).Bevel = sbrInset Then
        Notfin = False
        Downstatus.Panels(5).Bevel = sbrRaised
    End If



End Sub

Private Sub Form_Load()

'When got focus redraws the form
mainFrm.AutoRedraw = True
clicked = False
mainFrm.mnuFileNewSet.Enabled = False
mainFrm.mnuFileNewTest.Enabled = False
mainFrm.mnuFileOpenSet.Enabled = False
mainFrm.mnuFileOpenTest.Enabled = False
NegativeColor = &HFF&
PositiveColor = &H8000&
GaugeMax = 15
THETA = 0.1
PHI = 0.1
```

```
Step = False

mainFrm.Downstatus.Panels(4).Visible = False
mainFrm.Downstatus.Panels(5).Visible = True
mainFrm.Downstatus.Panels(5).Bevel = sbrRaised

Speed = 100
IncreaseFactor = 1.5
DecreaseFactor = 0.75
SuccessRange = 0.001

' Sets the initial color value for the dialog box.
frmOptions.CD1.Flags = cdlCCRGBInit
initializeNet
GetDefaults
mainFrm.Enabled = True

NetworkExist = False
TrainExist = False
TestExist = False
End Sub
Private Sub Form_Unload(Cancel As Integer)
mnuFileExit_Click
If a = vbCancel Then Cancel = -1
End Sub

Private Sub mnuEdit_Click()
Downstatus.Panels(6).Text = "Edit a network, training, or test file."
End Sub
Private Sub mnuEditNet_Click()
Downstatus.Panels(6).Text = "Edit a network file "
IOType = "net"
frmEdit.Caption = "Edit Network File"
frmEdit.Show
End Sub
Private Sub mnuEditSet_Click()
```

```
Downstatus.Panels(6).Text = "Edit a training file "
IOType = "trn"
frmEdit.Caption = "Edit Training File"
frmEdit.Show
End Sub
Private Sub mnuEditTest_Click()
Downstatus.Panels(6).Text = "Edit a test file "
IOType = "tst"
frmEdit.Caption = "Edit Test File"
frmEdit.Show
End Sub
Private Sub mnuFile_Click()
Downstatus.Panels(6).Text = "Create,Open,Close,or Save the files. Quit the
simulator."
End Sub

Private Sub mnuFileAsNet_Click()
Dim Filename As String

Downstatus.Panels(6).Text = "Write the current network file to ... "
If NetworkExist Then
   IOType = "net"
   CD1.DialogTitle = "Save Network as"
   CD1.Filter = "Network Files (*.net)|*.net|All Files (*.*)|*.*"
  ' Set the default File Type to network Files (*.net)
   CD1.FilterIndex = 1

   Filename = Downstatus.Panels(1).Text
   Filename = GetFileName(Filename)

  ' Call the save procedure. If Filename = Empty, then
  ' the user chose Cancel in the Save As dialog box; otherwise,
  ' save the file.
   If Filename <> "" Then
      SaveFileAs Filename
      Downstatus.Panels(1).Text = Filename
```

```
   End If
Else
    a = MsgBox("There is no network to save!", vbOKOnly +
vbInformation, "Saving Report")
End If
End Sub
Private Sub mnuFileAsSet_Click()
Dim Filename As String

Downstatus.Panels(6).Text = "Write the current training file to ... "
If TrainExist Then
   IOType = "trn"
   CD1.DialogTitle = "Save Training File as"
   CD1.Filter = "Training Files (*.trn)|*.trn|All Files (*.*)|*.*"
  ' Set the default File Type to training Files (*.trn)
   CD1.FilterIndex = 1

   Filename = Downstatus.Panels(2).Text
   Filename = GetFileName(Filename)

   ' Call the save procedure. If Filename = Empty, then
   ' the user chose Cancel in the Save As dialog box; otherwise,
   ' save the file.
   If Filename <> "" Then
      SaveFileAs Filename
      Downstatus.Panels(2).Text = Filename
   End If
Else
    a = MsgBox("There is no training file to save!", vbOKOnly +
vbInformation, "Saving Report")
End If
End Sub
Private Sub mnuFileAsTest_Click()
Dim Filename As String

Downstatus.Panels(6).Text = "Write the current test file to ... "
```

```
If TestExist Then
   IOType = "tst"
   CD1.DialogTitle = "Save Test File as"
   CD1.Filter = "Test Files (*.tst)|*.tst|All Files (*.*)|*.*"
  ' Set the default File Type to test Files (*.trn)
   CD1.FilterIndex = 1

   Filename = Downstatus.Panels(3).Text
   Filename = GetFileName(Filename)

   ' Call the save procedure. If Filename = Empty, then
   ' the user chose Cancel in the Save As dialog box; otherwise,
   ' save the file.
   If Filename <> "" Then
      SaveFileAs Filename
      Downstatus.Panels(3).Text = Filename
   End If
Else
     a = MsgBox("There is no test file to save!", vbOKOnly + vbInformation,
"Saving Report")
End If
End Sub
Private Sub mnuFileClose_Click()
Downstatus.Panels(6).Text = "Close the netowrk, training set, or test set "
End Sub


Private Sub mnuFileCloseAll_Click()
Downstatus.Panels(6).Text = "Reinitialize the simulator"
If NetworkExist And Not NetworkSaved Then
   a = MsgBox("Save the changes in the network to file? ", vbYesNoCancel +
vbQuestion)
   If a = vbYes Then
      mnuFileSaveNet_Click
   ElseIf a = vbCancel Then
      GoTo Omit
   End If
End If
   NetworkExist = False
   mnuFileNewSet.Enabled = False
   mnuFileNewTest.Enabled = False
   mnuFileOpenSet.Enabled = False
   mnuFileOpenTest.Enabled = False
   GetDefaults
   initializeNet
   initializeGrds
   initializeGrdValues

If TrainExist And Not TrainSaved Then
   a = MsgBox("Save the changes in the training set to file? ",
vbYesNoCancel + vbQuestion)
   If a = vbYes Then
      mnuFileSaveSet_Click
   ElseIf a = vbCancel Then
      GoTo Omit
   End If
End If
TrainExist = False
If TestExist And Not TestSaved Then
   a = MsgBox("Save the changes in the test set to file? ",
vbYesNoCancel + vbQuestion)
   If a = vbYes Then
      mnuFileSaveTest_Click
   ElseIf a = vbCancel Then
      GoTo Omit
   End If
End If
TestExist = False
Omit:
End Sub
Private Sub mnuFileCloseNet_Click()
Downstatus.Panels(6).Text = "Reinitialize the simulator"
```

```
If NetworkExist And Not NetworkSaved Then
   a = MsgBox("Save the changes in the network to file? ", vbYesNoCancel +
vbQuestion)
   If a = vbYes Then
      mnuFileSaveNet_Click
   ElseIf a = vbCancel Then
      GoTo Omit
   End If
End If
   mnuFileNewSet.Enabled = False
   mnuFileNewTest.Enabled = False
   mnuFileOpenSet.Enabled = False
   mnuFileOpenTest.Enabled = False
   NetworkExist = False
   TrainExist = False
   TestExist = False
   GetDefaults
   initializeNet
   initializeGrds
   initializeGrdValues
Omit:
mainFrm.SetFocus
End Sub
Private Sub mnuFileCloseSet_Click()
Downstatus.Panels(6).Text = "Remove the training set"
If TrainExist And Not TrainSaved Then
   a = MsgBox("Save the changes in the training set to file? ", vbYesNoCancel +
vbQuestion)
   If a = vbYes Then
      mnuFileSaveSet_Click
   ElseIf a = vbCancel Then
      GoTo Omit
   End If
End If
   TrainExist = False
Omit:
End Sub
Private Sub mnuFileCloseTest_Click()
Downstatus.Panels(6).Text = "Remove the test set"
If TestExist And Not TestSaved Then
   a = MsgBox("Save the changes in the test set to file? ",
vbYesNoCancel + vbQuestion)
   If a = vbYes Then
      mnuFileSaveTest_Click
   ElseIf a = vbCancel Then
      GoTo Omit
   End If
End If
   TestExist = False
Omit:
End Sub

Private Sub mnuFileExit_Click()
Downstatus.Panels(6).Text = "Quit the simulator"
mnuFileCloseAll_Click
If a <> vbCancel Then
   Cls
   End
End If
End Sub

Private Sub mnuFileNew_Click()
Downstatus.Panels(6).Text = "Create new network, training, or test set"
End Sub


Private Sub mnuFileNewNet_Click()
Downstatus.Panels(6).Text = "Create a network"
If NetworkExist And Not NetworkSaved Then
   a = MsgBox("Save the changes in the network to file? ",
vbYesNoCancel + vbQuestion)
   If a = vbYes Then
```

```
      mnuFileSaveNet_Click
    ElseIf a = vbCancel Then
      GoTo Omit
    End If
  End If
  Downstatus.Panels(6).Text = "Create new network file"
  newFrm.Show
  Omit:
  End Sub
  Private Sub mnuFileNewSet_Click()
  Downstatus.Panels(6).Text = "Create a training set"
  If TrainExist And Not TrainSaved Then
    a = MsgBox("Save the training set changes ", vbYesNoCancel + vbQuestion)
    If a = vbYes Then
      mnuFileSaveSet_Click
    ElseIf a = vbCancel Then
      GoTo Omit
    End If
  End If
  Downstatus.Panels(6).Text = "Create new training file"
  FileSelected = 1
  frmInOut.Show
  Omit:
  End Sub
  Private Sub mnuFileNewTest_Click()
  Downstatus.Panels(6).Text = "Create a test set"
  If TestExist And Not TestSaved Then
    a = MsgBox("Save the test set changes ", vbYesNoCancel + vbQuestion)
    If a = vbYes Then
      mnuFileSaveTest_Click
    ElseIf a = vbCancel Then
      GoTo Omit
    End If
  End If
  Downstatus.Panels(6).Text = "Create new test file"
  FileSelected = 2
```

```
frmInOut.Show
Omit:
End Sub
Private Sub mnuFileOpen_Click()
Downstatus.Panels(6).Text = "Load a network, training set, or test set
from a file"
End Sub


Private Sub mnuFileOpenNetwork_Click()
Downstatus.Panels(6).Text = "Load a network"
Dim OneLine As String

If NetworkExist And Not NetworkSaved Then
  a = MsgBox("Save the network changes ", vbYesNoCancel +
vbQuestion)
  If a = vbYes Then
    mnuFileSaveNet_Click
  ElseIf a = vbCancel Then
    GoTo OmitFileOpen
  End If
End If
' Set an error trap to detect the pressing of the cancel button
On Error GoTo OmitFileOpen
  ' Fill the items of the File Type list box
  ' of the Open dialog box
CD1.Filter = "Network Files (*.net)|*.net|All Files (*.*)|*.*"
  ' Set the default File Type to Network Files (*.net)
CD1.FilterIndex = 1
CD1.DialogTitle = "Open Network File"
CD1.ShowOpen  ' display Open common dialog box.

If CD1.Filename <> "" Then
  mnuFileNewSet.Enabled = True
  mnuFileNewTest.Enabled = True
  mnuFileOpenSet.Enabled = True
```

```
mnuFileOpenTest.Enabled = True
' Display the hourglass mouse pointer.
Screen.MousePointer = 11
NetworkExist = True
Open CD1.Filename For Input As #1
Line Input #1, OneLine     ' Read # of hidden layers
hLayers = Val(OneLine)
Line Input #1, OneLine     ' Read bias
bias = Val(OneLine)
Line Input #1, OneLine     ' Read # of neurons in each layer
If hLayers = 2 Then
   temp = Left(OneLine, InStr(OneLine, " "))
   inputNeurons = Val(temp)
   OneLine = Right(OneLine, Len(OneLine) - Len(temp))
   temp = Left(OneLine, InStr(OneLine, " "))
   h1Neurons = Val(temp)
   OneLine = Right(OneLine, Len(OneLine) - Len(temp))
   temp = Left(OneLine, InStr(OneLine, " "))
   h2Neurons = Val(temp)
   OneLine = Right(OneLine, Len(OneLine) - Len(temp))
   outputNeurons = Val(OneLine)
Else
   temp = Left(OneLine, InStr(OneLine, " "))
   inputNeurons = Val(temp)
   OneLine = Right(OneLine, Len(OneLine) - Len(temp))
   temp = Left(OneLine, InStr(OneLine, " "))
   h1Neurons = Val(temp)
   OneLine = Right(OneLine, Len(OneLine) - Len(temp))
   outputNeurons = Val(OneLine)
End If
redimArrays
For i = 1 To inputNeurons
   Line Input #1, OneLine
   For j = 1 To h1Neurons
      If j = h1Neurons Then
         W1(i, j) = Val(OneLine)
      Else
         temp = Left(OneLine, InStr(OneLine, " "))
         W1(i, j) = Val(temp)
         OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      End If
   Next j
Next i
If bias = 1 Then
   Line Input #1, OneLine
   For j = 1 To h1Neurons
      If j = h1Neurons Then
         bW1(j) = Val(OneLine)
      Else
         temp = Left(OneLine, InStr(OneLine, " "))
         bW1(j) = Val(temp)
         OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      End If
   Next j
End If
If hLayers = 2 Then
   For i = 1 To h1Neurons
      Line Input #1, OneLine
      For j = 1 To h2Neurons
         If j = h2Neurons Then
            W2(i, j) = Val(OneLine)
         Else
            temp = Left(OneLine, InStr(OneLine, " "))
            W2(i, j) = Val(temp)
            OneLine = Right(OneLine, Len(OneLine) - Len(temp))
         End If
      Next j
   Next i
   If bias = 1 Then
      Line Input #1, OneLine
      For j = 1 To h2Neurons
         If j = h2Neurons Then
```

```
        bW2(j) = Val(OneLine)
      Else
        temp = Left(OneLine, InStr(OneLine, " "))
        bW2(j) = Val(temp)
        OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      End If
    Next j
  End If
  For i = 1 To h2Neurons
    Line Input #1, OneLine
    For j = 1 To outputNeurons
      If j = outputNeurons Then
        W3(i, j) = Val(OneLine)
      Else
        temp = Left(OneLine, InStr(OneLine, " "))
        W3(i, j) = Val(temp)
        OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      End If
    Next j
  Next i
  If bias = 1 Then
    Line Input #1, OneLine
    For j = 1 To outputNeurons
      If j = outputNeurons Then
        bW3(j) = Val(OneLine)
      Else
        temp = Left(OneLine, InStr(OneLine, " "))
        bW3(j) = Val(temp)
        OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      End If
    Next j
  End If
Else
  For i = 1 To h1Neurons
    Line Input #1, OneLine
    For j = 1 To outputNeurons
```

```
        If j = outputNeurons Then
          W2(i, j) = Val(OneLine)
        Else
          temp = Left(OneLine, InStr(OneLine, " "))
          W2(i, j) = Val(temp)
          OneLine = Right(OneLine, Len(OneLine) - Len(temp))
        End If
      Next j
    Next i
    If bias = 1 Then
      Line Input #1, OneLine
      For j = 1 To outputNeurons
        If j = outputNeurons Then
          bW2(j) = Val(OneLine)
        Else
          temp = Left(OneLine, InStr(OneLine, " "))
          bW2(j) = Val(temp)
          OneLine = Right(OneLine, Len(OneLine) - Len(temp))
        End If
      Next j
    End If
  End If
  Close #1
End If
  NetworkSaved = True
  initializeLines
  initializeNeurons
  initializeDeltaWeights

  showLines
  initializeGrds
  showWs

  drawNet
  frmError.Graph1.DataReset = 1
  ' Reset the mouse pointer.
```

```
   Screen.MousePointer = 0
OmitFileOpen:
End Sub
Private Sub mnuFileOpenSet_Click()
Downstatus.Panels(6).Text = "Load a training set"
Dim OneLine As String

If TrainExist And Not TrainSaved Then
   a = MsgBox("Save the training set changes ", vbYesNoCancel + vbQuestion)
   If a = vbYes Then
      mnuFileSaveSet_Click
   ElseIf a = vbCancel Then
      GoTo OmitFileOpen
   End If
End If
' Set an error trap to detect the pressing of the cancel button
On Error GoTo OmitFileOpen
  ' Fill the items of the File Type list box
  ' of the Open dialog box
CD1.Filter = "Training Files (*.trn)|*.trn|All Files (*.*)|*.*"
  ' Set the default File Type to Training Files (*.trn)
CD1.FilterIndex = 1
CD1.DialogTitle = "Open Training File"
CD1.ShowOpen  ' display Open common dialog box.

If CD1.Filename <> "" Then
   TrainExist = True
   TrainSaved = True
   Open CD1.Filename For Input As #1
      Line Input #1, OneLine
   temp = Left(OneLine, InStr(OneLine, " "))
   TrnIOs = Val(temp)

   ReDim iTrnMatrix(1 To TrnIOs, 1 To inputNeurons)
   ReDim oTrnMatrix(1 To TrnIOs, 1 To outputNeurons)
   For i = 1 To TrnIOs
      Line Input #1, OneLine
      For j = 1 To inputNeurons
         temp = Left(OneLine, InStr(OneLine, " "))
         iTrnMatrix(i, j) = Val(temp)
         OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      Next j
      For j = 1 To outputNeurons
         If j = outputNeurons Then
            oTrnMatrix(i, j) = Val(OneLine)
         Else
            temp = Left(OneLine, InStr(OneLine, " "))
            oTrnMatrix(i, j) = Val(temp)
            OneLine = Right(OneLine, Len(OneLine) - Len(temp))
         End If
      Next j
   Next i
   Close #1
End If
OmitFileOpen:
End Sub
Private Sub mnuFileOpenTest_Click()
Downstatus.Panels(6).Text = "Load a test set"

If TestExist And Not TestSaved Then
   a = MsgBox("Save the test set changes ", vbYesNoCancel +
vbQuestion)
   If a = vbYes Then
      mnuFileSaveTest_Click
   ElseIf a = vbCancel Then
      GoTo OmitFileOpen
   End If
End If
' Set an error trap to detect the pressing of the cancel button
On Error GoTo OmitFileOpen
  ' Fill the items of the File Type list box
  ' of the Open dialog box
```

```
CD1.Filter = "Test Files (*.tst)|*.tst|All Files (*.*)|*.*"
 ' Set the default File Type to Test Files (*.tst)
CD1.FilterIndex = 1
CD1.DialogTitle = "Open Test File"
CD1.ShowOpen  ' display Open common dialog box.
 'ReadFile
If CD1.Filename <> "" Then
  TestExist = True
  TestSaved = True
  Open CD1.Filename For Input As #1
    Line Input #1, OneLine
  temp = Left(OneLine, InStr(OneLine, " "))
  TestIOs = Val(temp)

  ReDim iTestMatrix(1 To TestIOs, 1 To inputNeurons)
  ReDim oTestMatrix(1 To TestIOs, 1 To outputNeurons)
  For i = 1 To TestIOs
    Line Input #1, OneLine
    For j = 1 To inputNeurons
      temp = Left(OneLine, InStr(OneLine, " "))
      iTestMatrix(i, j) = Val(temp)
      OneLine = Right(OneLine, Len(OneLine) - Len(temp))
    Next j
    For j = 1 To outputNeurons
      If j = outputNeurons Then
        oTestMatrix(i, j) = Val(OneLine)
      Else
        temp = Left(OneLine, InStr(OneLine, " "))
        oTestMatrix(i, j) = Val(temp)
        OneLine = Right(OneLine, Len(OneLine) - Len(temp))
      End If
    Next j
  Next i
  Close #1
End If
OmitFileOpen:
```

```
End Sub
Private Sub mnuFileSave_Click()
Downstatus.Panels(6).Text = "Save the network,training set,or test set
to a file"
End Sub


Private Sub mnuFileSaveAs_Click()
Downstatus.Panels(6).Text = "Write current files to ..."
End Sub


Private Sub mnuFileSaveNet_Click()
Dim Filename As String

Downstatus.Panels(6).Text = "Save the current network to a file"
If NetworkExist Then
  IOType = "net"
  CD1.DialogTitle = "Save Network"
  CD1.Filter = "Network Files (*.net)|*.net|All Files (*.*)|*.*"
 ' Set the default File Type to network Files (*.net)
  CD1.FilterIndex = 1
  If Left(Downstatus.Panels(1).Text, 7) = "network" Then
    ' The file hasn't been saved yet.
    ' Get the filename, and then call the save procedure, GetFileName.
    Filename = Downstatus.Panels(1).Text
    Filename = GetFileName(Filename)
  Else
    ' The form's Caption contains the name of the open file.
    Filename = Downstatus.Panels(1).Text
  End If
 ' Call the save procedure. If Filename = Empty, then
 ' the user chose Cancel in the Save As dialog box; otherwise,
 ' save the file.
  If Filename <> "" Then
    SaveFileAs Filename
```

```
        Downstatus.Panels(1).Text = Filename
    End If
Else
    a = MsgBox("There is no network to save!", vbOKOnly + vbInformation,
"Saving Report")
End If
End Sub
Private Sub mnuFileSaveSet_Click()
Dim Filename As String

Downstatus.Panels(6).Text = "Save the current training set to a file"
If TrainExist Then
    IOType = "trn"
    CD1.DialogTitle = "Save Training Set"
    CD1.Filter = "Training Files (*.trn)|*.trn|All Files (*.*)|*.*"
   ' Set the default File Type to Training Files (*.trn)
    CD1.FilterIndex = 1
    If Left(Downstatus.Panels(2).Text, 5) = "train" Then
        ' The file hasn't been saved yet.
        ' Get the filename, and then call the save procedure, GetFileName.
        Filename = Downstatus.Panels(2).Text
        Filename = GetFileName(Filename)
    Else
        ' The form's Caption contains the name of the open file.
        Filename = Downstatus.Panels(2).Text
    End If
    ' Call the save procedure. If Filename = Empty, then
    ' the user chose Cancel in the Save As dialog box; otherwise,
    ' save the file.
    If Filename <> "" Then
        SaveFileAs Filename
        Downstatus.Panels(2).Text = Filename
    End If
Else
    a = MsgBox("There is no training set to save!", vbOKOnly + vbInformation,
"Saving Report")
End If
End Sub
Private Sub mnuFileSaveTest_Click()
Dim Filename As String

Downstatus.Panels(6).Text = "Save the current test set to a file"
If TestExist Then
    IOType = "tst"
    CD1.DialogTitle = "Save Test Data Set"
    CD1.Filter = "Test Files (*.tst)|*.tst|All Files (*.*)|*.*"
   ' Set the default File Type to test Files (*.tst)
    CD1.FilterIndex = 1
    If Left(Downstatus.Panels(3).Text, 4) = "test" Then
        ' The file hasn't been saved yet.
        ' Get the filename, and then call the save procedure, GetFileName.
        Filename = Downstatus.Panels(3).Text
        Filename = GetFileName(Filename)
    Else
        ' The form's Caption contains the name of the open file.
        Filename = Downstatus.Panels(3).Text
    End If
    ' Call the save procedure. If Filename = Empty, then
    ' the user chose Cancel in the Save As dialog box; otherwise,
    ' save the file.
    If Filename <> "" Then
        SaveFileAs Filename
        Downstatus.Panels(3).Text = Filename
    End If
Else
    a = MsgBox("There is no test set to save!", vbOKOnly +
vbInformation, "Saving Report")
End If
End Sub
Private Sub mnuHelp_Click()
Downstatus.Panels(6).Text = "See the instructions or info about
VisualNN"
```

```
End Sub

Private Sub mnuHelpAbout_Click()
Downstatus.Panels(6).Text = "About VisualNN 1.0..."
aboutFrm.Show
mainFrm.Enabled = False
End Sub

Private Sub mnuHelpIns_Click()
Downstatus.Panels(6).Text = "VisualNN instructions"
frmInstructions.Show
End Sub
Private Sub mnuHelpWhat_Click()
Downstatus.Panels(6).Text = "Information about neurons and weights"
frmWhatIs.Show
End Sub
Private Sub mnuRun_Click()
Downstatus.Panels(6).Text = "Test the network "
End Sub
Private Sub mnuRunEnd_Click()
Downstatus.Panels(6).Text = "Testing the network ended "
Notfin = False
End Sub
Private Sub mnuRunStart_Click()
Dim Result As String

Downstatus.Panels(6).Text = "Testing the network started "
If TestExist And NetworkExist Then
  training = False
  inIndex = 0
  frmValues.grdIO.Col = 4
  frmValues.grdIO.Row = 1
  frmValues.grdIO.Text = " "
  frmValues.grdIO.Col = 5
  frmValues.grdIO.Row = 1
  frmValues.grdIO.Text = " "
```

```
frmValues.grdIO.Col = 5
frmValues.grdIO.Row = 0
frmValues.grdIO.FixedAlignment(5) = 2
frmValues.grdIO.Text = " "
RMS = 0#
Notfin = True
Continue = False
Result = "Input(s)     Output(s)     Result(s)" + Chr(13) + Chr(10)
For k = 1 To TestIOs
 inIndex = k
  For i = 1 To inputNeurons
    Result = Result + Format(iTestMatrix(k, i), "0.000") + " "
  Next
  Result = Result + "     "
  For i = 1 To outputNeurons
    Result = Result + Format(oTestMatrix(k, i), "0.000") + " "
  Next
  Result = Result + "     "
  forwardBP
  For i = 1 To outputNeurons
    Result = Result + Format(o(i), "0.0000") + " "
    If i = outputNeurons Then Result = Result + Chr(13) + Chr(10)
  Next
 RMS = RMS + thisError
If Step Then
  Downstatus.Panels(6).Text = "CONTINUE"
  Beep
  Do Until Continue
    If Notfin = False Then Exit Do
    DoEvents
  Loop
  Continue = False
End If
If Notfin = False Then Exit Sub
Next
```

```
   RMS = Sqr(RMS / TestIOs)
   frmValues.grdIO.Col = 4
   frmValues.grdIO.Row = 1
   frmValues.grdIO.Text = Format(RMS, "0.0000")
   Result = Result + "RMS Value : " + Format(RMS, "0.00000")
   frmResult.Show
   frmResult.TextResult.Text = Result
Else
    a = MsgBox("You must have a network and test set to run.", vbOKOnly +
vbInformation, "Training Report")
End If
End Sub
Private Sub mnuTrain_Click()
Downstatus.Panels(6).Text = "Train the network"
End Sub


Private Sub mnuTrainAlg_Click()
Downstatus.Panels(6).Text = "Training algorithms "
End Sub


Private Sub mnuTrainAlgBack_Click()
Downstatus.Panels(6).Text = "Backpropagation with momentum"
Algorithm = "BP"
temp = paramFrm.Caption
paramFrm.Caption = temp + "[Backprop]"
paramFrm.Show
End Sub
Private Sub mnuTrainAlgCon_Click()
Downstatus.Panels(6).Text = "Conjugate gradient with line search"
Algorithm = "CG"
temp = paramFrm.Caption
paramFrm.Caption = temp + "[Conjugate Gradient]"
paramFrm.Show
End Sub
```

```
Private Sub mnuTrainAlgDelta_Click()
Downstatus.Panels(6).Text = "Delta-Bar-Delta"
Algorithm = "DBD"
temp = paramFrm.Caption
paramFrm.Caption = temp + "[Delta-Bar-Delta]"
paramFrm.Show
End Sub
Private Sub mnuTrainAlgQuick_Click()
Downstatus.Panels(6).Text = "Quickprop"
Algorithm = "QP"
temp = paramFrm.Caption
paramFrm.Caption = temp + "[Quickprop]"
paramFrm.Show
End Sub
Private Sub mnuTrainEnd_Click()
Downstatus.Panels(6).Text = "Training the network ended"
Notfin = False
End Sub
Private Sub mnuTrainStart_Click()
Dim temp As Single
Dim epochs As Integer
Dim i, j, k As Integer
Dim previousRMS As Single

Downstatus.Panels(6).Text = "Training started"
Continue = False
If NetworkExist And TrainExist Then
   training = True
   epochs = 1
   Select Case Algorithm
      Case "BP"
         Notfin = True
         RMS = 1#
         While Notfin And epochs <= MaxEpochs And RMS >
Tolerance
            RMS = 0#
```

```
inIndex = 0
For k = 1 To TrnIOs 'Present each I/O pair
   inIndex = inIndex + 1
   delay
   forwardBP      'Present each I/O pair
   temp = thisError
   RMS = RMS + temp
   errorVectors
   updateWsBP

   showWs
   DoEvents
   If Step Then
      Downstatus.Panels(6).Text = "CONTINUE"
      Beep
      Do Until Continue
         If Notfin = False Then Exit Do
         DoEvents
      Loop
      Continue = False
   End If
Next k
RMS = Sqr(RMS / TrnIOs)
frmValues.grdIO.Col = 4
frmValues.grdIO.Row = 1
frmValues.grdIO.Text = Format(RMS, "0.0000")
frmError.Graph1.ThisPoint = epochs
frmError.Graph1.GraphData = RMS
frmValues.grdIO.Col = 5
frmValues.grdIO.Text = Str$(epochs)
If epochs = 1 Then
   frmError.Graph1.YAxisMax = RMS
End If
If epochs / 10 = Int(epochs / 10) Then
   'Update the error graph
   frmError.Graph1.DrawMode = 3
```

```
            End If
            epochs = epochs + 1
        Wend
        frmError.Graph1.DrawMode = 3
        Beep

        Case "QP"
          Notfin = True
          RMS = 1#
          initializeDeltaWeights

          While Notfin And epochs <= MaxEpochs And RMS >
Tolerance
            RMS = 0#
            inIndex = 0

            For k = 1 To TrnIOs
            inIndex = inIndex + 1
            delay
            forwardBP
            temp = thisError
            RMS = RMS + temp
            errorVectors
            updateWsQP
            showWs
            DoEvents
            If Step Then
                 Downstatus.Panels(6).Text = "CONTINUE"
                 Beep
                 Do Until Continue
                    If Notfin = False Then Exit Do
                    DoEvents
                 Loop
                 Continue = False
            End If
            Next k
```

```
    RMS = Sqr(RMS / TrnIOs)
    frmValues.grdIO.Col = 4
    frmValues.grdIO.Row = 1
    frmValues.grdIO.Text = Format(RMS, "0.0000")

    frmError.Graph1.ThisPoint = epochs
    frmError.Graph1.GraphData = RMS
    frmValues.grdIO.Col = 5
    frmValues.grdIO.Text = Str$(epochs)
    If epochs = 1 Then
      frmError.Graph1.YAxisMax = RMS + 0.1
    End If
    If epochs / 10 = Int(epochs / 10) Then
      frmError.Graph1.DrawMode = 3
    End If
    epochs = epochs + 1
  Wend
  frmError.Graph1.DrawMode = 3
  Beep

Case "DBD"  ' Delta-Bar-Delta Algorithm
  Notfin = True
  RMS = 1#

  While Notfin And epochs <= MaxEpochs And RMS > Tolerance
    RMS = 0#
    inIndex = 0
    For k = 1 To TrnIOs
    inIndex = inIndex + 1
    delay

    forwardBP
    temp = thisError
    RMS = RMS + temp
    errorVectors
    updateWsDBD
    showWs
    DoEvents
    If Step Then
        Downstatus.Panels(6).Text = "CONTINUE"
        Beep
        Do Until Continue
          If Notfin = False Then Exit Do
          DoEvents
        Loop
        Continue = False
    End If
    Next k
    RMS = Sqr(RMS / TrnIOs)
    frmValues.grdIO.Col = 4
    frmValues.grdIO.Row = 1
    frmValues.grdIO.Text = Format(RMS, "0.0000")

    frmError.Graph1.ThisPoint = epochs
    frmError.Graph1.GraphData = RMS
    frmValues.grdIO.Col = 5
    frmValues.grdIO.Text = Str$(epochs)
    If epochs = 1 Then
      frmError.Graph1.YAxisMax = RMS
    End If
    If epochs / 10 = Int(epochs / 10) Then
      frmError.Graph1.DrawMode = 3
    End If
    epochs = epochs + 1
  Wend
  frmError.Graph1.DrawMode = 3
  Beep

Case "CG"            'CG with line search
  Notfin = True
  initializeDeltaWeights
```

```
RMS = 0#
inIndex = 0
resetDeltas
For k = 1 To TrnIOs
  inIndex = inIndex + 1
  delay
  forwardBP
  temp = thisError
  RMS = RMS + temp
  errorVectors
  computeDeltas
  addDeltas
  If Step Then
      Downstatus.Panels(6).Text = "CONTINUE"
      Beep
      Do Until Continue
        If Notfin = False Then Exit Do
        DoEvents
      Loop
      Continue = False
  End If
Next k
RMS = Sqr(RMS / TrnIOs)
frmValues.grdIO.Col = 4
frmValues.grdIO.Row = 1
frmValues.grdIO.Text = Format(RMS, "0.0000")

frmError.Graph1.ThisPoint = epochs
frmError.Graph1.GraphData = RMS
frmValues.grdIO.Col = 5
frmValues.grdIO.Text = Str$(epochs)
If epochs = 1 Then
  frmError.Graph1.YAxisMax = RMS + 0.1
End If
epochs = epochs + 1
If RMS < Tolerance Or epochs > MaxEpochs Then
```

```
        GoTo OmitNext
      End If

      previousRMS = RMS   'Store RMS value
      While Notfin And epochs <= MaxEpochs And RMS >
Tolerance
        RMS = 0#
        inIndex = 0
        resetDeltas
        For k = 1 To TrnIOs
          inIndex = inIndex + 1
          delay
          forwardBP
          temp = thisError
          RMS = RMS + temp
          errorVectors
          computeDeltas
          addDeltas
          If Step Then
            Downstatus.Panels(6).Text = "CONTINUE"
            Beep
            Do Until Continue
              If Notfin = False Then Exit Do
              DoEvents
            Loop
            Continue = False
          End If
        Next k
        RMS = Sqr(RMS / TrnIOs)
        frmValues.grdIO.Col = 4
        frmValues.grdIO.Row = 1
        frmValues.grdIO.Text = Format(RMS, "0.0000")

        frmError.Graph1.ThisPoint = epochs
        frmError.Graph1.GraphData = RMS
        frmValues.grdIO.Col = 5
```

```
          frmValues.grdIO.Text = Str$(epochs)
          If epochs / 10 = Int(epochs / 10) Then
            frmError.Graph1.DrawMode = 3
          End If
          epochs = epochs + 1
          If RMS < Tolerance Or epochs > MaxEpochs Then
            GoTo OmitNext
          End If
          'Line search
          If RMS - previousRMS < SuccessRange Then
            'Success
            updateWsCG
            showWs
            'Increase the learning rate
            LearningRate = IncreaseFactor * LearningRate
            If LearningRate > MaxLearningRate Then
              LearningRate = MaxLearningRate
            End If
          Else
            'Fail
            'Decrease the learning rate
            LearningRate = DecreaseFactor * LearningRate
          End If
          previousRMS = RMS
OmitNext:
        Wend
        frmError.Graph1.DrawMode = 3
        Beep
End Select
Else
   a = MsgBox("You must have a network and training set to start training.",
vbOKOnly + vbInformation, "Training Report")
End If
End Sub
Private Sub mnuView_Click()
Downstatus.Panels(6).Text = "View the settings or different screens"
```

```
End Sub



Private Sub mnuViewOptions_Click()
Downstatus.Panels(6).Text = "View or edit the settings"
frmOptions.Show
End Sub

Private Sub mnuViewSplit_Click()
Downstatus.Panels(6).Text = "View error graph and network values"
  mainFrm.Width = 7185
  frmValues.Show
  frmError.Show
  initializeGrdValues
  initializeGrds
  showWs
End Sub

Private Sub mnuViewVisual_Click()
Downstatus.Panels(6).Text = "View the visual screen"
mainFrm.Width = 10500
End Sub

Attribute VB_Name = "aboutFrm"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Form_Unload(Cancel As Integer)
  mainFrm.Enabled = True
End Sub
Private Sub SScmdOK_Click()
  aboutFrm.Hide
  mainFrm.Enabled = True
  mainFrm.SetFocus
End Sub
```

```
Attribute VB_Name = "frmEdit"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Public FName As Variant        'Current file name for I/O operations
Public FileChanged As Boolean  'Flag for text change

Private Sub cmdExit_Click()
If FileChanged Then            'If text is changed inform the user
   a = MsgBox("Save the changes to the file " + FName, vbYesNoCancel +
vbQuestion)
   If a = vbYes Then
      cmdSave_Click
   ElseIf a = vbCancel Then    'If cancel is selected do not exit.
      GoTo OmitExit
   End If
End If
Unload frmEdit
OmitExit:
End Sub

Private Sub cmdOpen_Click()

'If cancel clicked exit the sub
On Error GoTo OmitFileOpen

Select Case IOType
   Case "net"          'File is network type
      CD1.DialogTitle = "Open Network File"
      CD1.Filter = "Network Files (*.net)|*.net|All Files (*.*)|*.*"
      ' Set the default File Type to network Files (*.net)
      CD1.FilterIndex = 1

      'Get the file name
      CD1.ShowOpen
      FName = CD1.Filename
      frmEdit.Caption = frmEdit.Caption + " [" + FName + "]"

      'Load the network file
      Rtext1.LoadFile FName, rtfText

   Case "trn"          'File is training type
      CD1.DialogTitle = "Open Training File"
      CD1.Filter = "Training Files (*.trn)|*.trn|All Files (*.*)|*.*"
      ' Set the default File Type to training Files (*.trn)
      CD1.FilterIndex = 1

      'Get the file name
      CD1.ShowOpen
      FName = CD1.Filename
      frmEdit.Caption = frmEdit.Caption + " [" + FName + "]"

      'Load the training file
      Rtext1.LoadFile FName, rtfText
   Case "tst"          'File is test type
      CD1.DialogTitle = "Open Test File"
      CD1.Filter = "Test Files (*.tst)|*.tst|All Files (*.*)|*.*"

      ' Set the default File Type to test Files (*.tst)
      CD1.FilterIndex = 1
          'Get the file name
      CD1.ShowOpen
      FName = CD1.Filename
      frmEdit.Caption = frmEdit.Caption + " [" + FName + "]"
          'Load the test file
      Rtext1.LoadFile FName, rtfText
End Select
FileChanged = False     'Initially text is not changed
OmitFileOpen:

End Sub

Private Sub cmdSave_Click()
```

```
'Save the text as text file
Rtext1.SaveFile FName, rtfText

FileChanged = False
End Sub

Private Sub Rtext1_Change()
'If text changes set flag to true
FileChanged = True
End Sub

Attribute VB_Name = "frmInOut"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
'Temporary arrays for I/O pairs
Private iMatrix() As Single
Private oMatrix() As Single
Private oTemp() As Single
Private iTemp() As Single
'Initially 20 and increased by 20
Private MaxIndex As Integer
'I/O matrix index
Private Index As Integer

Private Sub increaseMatrixSize()
'Increase the I/O matrices by 20
MaxIndex = MaxIndex + 20
ReDim iMatrix(1 To MaxIndex, 1 To inputNeurons) As Single
ReDim oMatrix(1 To MaxIndex, 1 To outputNeurons) As Single
End Sub

'Print the I/O values to text box
'
Private Sub write2Text()
Dim i As Integer
Dim j As Integer
```

```
rtextInOut.SetFocus
SendKeys "{END}" & "{ENTER}", True

For j = 1 To inputNeurons
   SendKeys Format(Val(textInput(j - 1).Text), "0.0") & " ", True
Next j
   SendKeys "     ", True

For j = 1 To outputNeurons
   SendKeys Format(Val(textOutput(j - 1).Text), "0.0") & " ", True
Next j

textInput(0).SetFocus
End Sub

'Add the I/O pair to the matrices
'
Private Sub cmdAdd_Click()
Dim i As Integer

For i = 0 To inputNeurons - 1
   If textInput(i).Text = "" Or Val(textInput(i).Text) = -1 Then
      a = MsgBox("Enter a valid number", vbOKOnly + vbCritical,
"Invalid Entry")
      textInput(i).SetFocus
       Exit Sub
   End If
   If Abs(Val(textInput(i).Text)) >= 10 Then
      a = MsgBox("Input value too large", vbOKOnly + vbCritical,
"Invalid Entry")
      textInput(i).SetFocus

        Exit Sub
   End If
Next
```

```
For i = 0 To outputNeurons - 1
  If textOutput(i).Text = "" Or Val(textOutput(i).Text) = -1 Then
    a = MsgBox(" Enter a valid number ", vbOKOnly + vbCritical, "Invalid
Entry")
    textOutput(i).SetFocus
    Exit Sub
  End If

  If Val(textOutput(i).Text) < 0 Or Val(textOutput(i).Text) > 1 Then
    a = MsgBox("Output value must be between 0 and 1 ", vbOKOnly +
vbCritical, "Invalid Entry")
    textOutput(i).SetFocus
    Exit Sub
  End If
Next

  Index = Index + 1
  If Index > MaxIndex Then
    increaseMatrixSize
  End If

  For i = 1 To inputNeurons
    iMatrix(Index, i) = Val(textInput(i - 1).Text)
  Next
  For i = 1 To outputNeurons
    oMatrix(Index, i) = Val(textOutput(i - 1).Text)
  Next
  write2Text

For i = 0 To inputNeurons - 1
  textInput(i).Text = ""
Next

For i = 0 To outputNeurons - 1
  textOutput(i).Text = ""
```

```
Next

End Sub

Private Sub cmdCancel_Click()
  Unload frmInOut
  mainFrm.Enabled = True
  mainFrm.SetFocus
End Sub

'Delete the previously entered I/O pair
'
Private Sub cmdDelete_Click()
If Index > 0 Then
  Index = Index - 1

  rtextInOut.SetFocus

  For i = 1 To 4 * (inputNeurons + outputNeurons) + 6
    SendKeys "{BS}", True
  Next
  textInput(0).SetFocus

Else
   a = MsgBox(" No data to delete", vbOKOnly + vbInformation,
"Delete Report")
    textInput(0).SetFocus
End If

End Sub

Private Sub cmdOK_Click()
If Index <= 0 Then
  Unload frmInOut
  mainFrm.Enabled = True
  mainFrm.SetFocus
```

```
        Exit Sub
    End If

    If FileSelected = 1 Then        'If training set
        TrainSaved = False          'set is not saved

        'Initialize the training set matrices
        ReDim iTrnMatrix(1 To Index, 1 To inputNeurons)
        ReDim oTrnMatrix(1 To Index, 1 To outputNeurons)

        'Get the pairs from temporary matrices to training matrices
        For i = 1 To Index
            For j = 1 To inputNeurons
                iTrnMatrix(i, j) = iMatrix(i, j)
            Next j
            For j = 1 To outputNeurons
                oTrnMatrix(i, j) = oMatrix(i, j)
            Next j
        Next i

        TrainExist = True
        TrnIOs = Index      'Set the number of training pairs


    ElseIf FileSelected = 2 Then    'If test set
        TestSaved = False
        ReDim iTestMatrix(1 To Index, 1 To inputNeurons) As Single
        ReDim oTestMatrix(1 To Index, 1 To outputNeurons) As Single

        For i = 1 To Index
            For j = 1 To inputNeurons
                iTestMatrix(i, j) = iMatrix(i, j)
            Next j
            For j = 1 To outputNeurons
                oTestMatrix(i, j) = oMatrix(i, j)
            Next j
```

```
        Next i
        TestExist = True
        TestIOs = Index
    End If

    Unload frmInOut
    mainFrm.Enabled = True
    mainFrm.SetFocus

    End Sub

    Private Sub Form_Load()

    Dim i As Integer

    mainFrm.Enabled = False
    MaxIndex = 20

    ReDim iMatrix(1 To MaxIndex, 1 To inputNeurons) As Single
    ReDim oMatrix(1 To MaxIndex, 1 To outputNeurons) As Single

    Index = 0

    For i = 0 To 7
        textInput(i).Enabled = False

        textOutput(i).Enabled = False
    Next


    If FileSelected = 1 Then
        frmInOut.Caption = " New training file"
    Else
        frmInOut.Caption = " New test file"
    End If
```

```
'Enable the necessary boxes only
For i = 0 To inputNeurons - 1
   textInput(i).Enabled = True
Next

For i = 0 To outputNeurons - 1
   textOutput(i).Enabled = True
Next

End Sub

Private Sub Form_Unload(Cancel As Integer)
   mainFrm.Enabled = True
   mainFrm.SetFocus
End Sub

Attribute VB_Name = "frmInstructions"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Form_Load()
           On Error Resume Next
           R1.LoadFile "instructions.rtf"
 End Sub

Attribute VB_Name = "newFrm"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Form_Load()
mainFrm.Enabled = False
'Put the current values to NewNet form
TextHidden.Text = hLayers
TextBias.Text = bias
TextInputNeurons.Text = inputNeurons
TextOutputNeurons.Text = outputNeurons
hidden1Text.Text = h1Neurons
```

```
'if one hidden layer disable second hidden
If hLayers = 1 Then
   newFrm.hidden2Text.Enabled = False
   hidden2Text.Text = 0
Else
   newFrm.hidden2Text.Enabled = True
   hidden2Text.Text = h2Neurons
End If

TextTresh.Text = LineThreshold
TextMinMax.Text = MinMax
End Sub

Private Sub Form_Unload(Cancel As Integer)
   mainFrm.Enabled = True
   mainFrm.SetFocus
End Sub

Private Sub hidden1Text_LostFocus()
temp = Val(hidden1Text.Text)
   If temp < 1 Or temp > 8 Then
         a = MsgBox("Minimum 1, maximum 8 hidden layer neurons",
vbOKOnly + vbCritical, "Invalid Entry")
    hidden1Text.Text = 2
    hidden1Text.SetFocus
   End If
End Sub

Private Sub hidden2Text_LostFocus()
temp = Val(hidden2Text.Text)
   If temp < 1 Or temp > 8 Then
         a = MsgBox("Minimum 1, maximum 8 hidden layer neurons",
vbOKOnly + vbCritical, "Invalid Entry")
    hidden2Text.Text = 1
    hidden2Text.SetFocus
   End If
End Sub

Private Sub newCancel_Click()
   Unload newFrm
   mainFrm.Enabled = True
   mainFrm.SetFocus
End Sub

Private Sub newCreate_Click()
'Get the values from the form
h1Neurons = Val(hidden1Text.Text)
h2Neurons = Val(hidden2Text.Text)
bias = Val(TextBias.Text)
hLayers = Val(TextHidden.Text)
inputNeurons = Val(TextInputNeurons.Text)
MinMax = Val(TextMinMax.Text)
outputNeurons = Val(TextOutputNeurons.Text)
LineThreshold = Val(TextTresh.Text)

'Now a network exist. Enable the Training and Test
'Set creation and opening options
   mainFrm.mnuFileNewSet.Enabled = True
   mainFrm.mnuFileNewTest.Enabled = True
   mainFrm.mnuFileOpenSet.Enabled = True
   mainFrm.mnuFileOpenTest.Enabled = True

'Initialize everything
   initializeGrdValues
   initializeNet
   redimArrays     'Reinitialize the arrays
   initializeLines
   initializeNeurons
   randomWeights
   initializeDeltaWeights
   showLines
   initializeGrds
```

```
   showWs
   drawNet

   'Reset the error graph
   frmError.Graph1.DataReset = 1

   NetworkExist = True
   NetworkSaved = False    'Network is not saved
   Unload newFrm
   mainFrm.SetFocus
End Sub

Private Sub TextBias_LostFocus()
If TextBias.Text = "" Then
  a = MsgBox(" 1.With bias 0.Without bias", vbOKOnly + vbCritical,
"Invalid Entry")
    TextBias.Text = 0
    TextBias.SetFocus
Else
temp = Val(TextBias.Text)
   If temp <> 0 And temp <> 1 Then
         a = MsgBox(" 1.With bias 0.Without bias", vbOKOnly +
vbCritical, "Invalid Entry")
    TextBias.Text = 0
    TextBias.SetFocus
   End If
End If
End Sub

Private Sub TextHidden_LostFocus()
If TextHidden.Text = "" Then
  a = MsgBox(" 1 or 2 hidden layer", vbOKOnly + vbCritical, "Invalid
entry")
    TextHidden.Text = 1
    TextHidden.SetFocus
Else
```

```
temp = Val(TextHidden.Text)
  If temp = 2 Then
    hidden2Text.Enabled = True
    hidden2Text.Text = 1
  ElseIf temp = 1 Then
    hidden2Text.Enabled = False
  Else
   a = MsgBox(" 1 or 2 hidden layer", vbOKOnly + vbCritical, "Invalid entry")
   TextHidden.Text = 1
   TextHidden.SetFocus
  End If
End If
End Sub

Private Sub TextInputNeurons_LostFocus()
temp = Val(TextInputNeurons.Text)
  If temp < 1 Or temp > 8 Then
       a = MsgBox("Minimum 1, maximum 8 input neurons", vbOKOnly +
vbCritical, "Invalid Entry")
   TextInputNeurons.Text = 2
   TextInputNeurons.SetFocus
  End If
End Sub

Private Sub TextMinMax_LostFocus()
Style = vbOKOnly + vbCritical ' Define buttons.
Title = "Invalid Entry"  ' Define title.
temp = Val(TextMinMax.Text)
  If temp < 0 Then
       a = MsgBox(" Less than 0 is not a valid range", Style, Title)
   TextMinMax.Text = 0.2
   TextMinMax.SetFocus
  End If
End Sub
Private Sub TextOutputNeurons_LostFocus()
temp = Val(TextOutputNeurons.Text)
```

```
  If temp < 1 Or temp > 8 Then
       a = MsgBox("Minimum 1, maximum 8 output neurons",
vbOKOnly + vbCritical, "Invalid Entry")
   TextOutputNeurons.Text = 1
   TextOutputNeurons.SetFocus
  End If
End Sub

Private Sub TextTresh_LostFocus()
Style = vbOKOnly + vbCritical ' Define buttons.
Title = "Invalid Entry"  ' Define title.
temp = Val(TextTresh.Text)
  If temp < 0 Then
       a = MsgBox(" Less than 0 is not a valid threshold value", Style,
Title)
   TextTresh.Text = 0.21
   TextTresh.SetFocus
  End If
End Sub

Attribute VB_Name = "frmOptions"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Dim MaxChanged As Boolean
Private Sub cmdCancel_Click()
  Unload frmOptions
  mainFrm.SetFocus
End Sub

Private Sub cmdOK_Click()
'Get the values from the form
Speed = scrolSpeed.Value
Step = OptionStep.Value
PositiveColor = Combo1.Text
NegativeColor = Combo2.Text
GaugeMax = Val(TextAdder.Text)
```

```
If MaxChanged Then initializeNeurons
IncreaseFactor = Val(TextIF.Text)
DecreaseFactor = Val(TextDF.Text)
SuccessRange = Val(TextSR.Text)

THETA = Val(TextTHETA.Text)
PHI = Val(TextPHI.Text)
If Step Then    'Show continue button
      mainFrm.Downstatus.Panels(4).Visible = True
      mainFrm.Downstatus.Panels(5).Visible = False
      mainFrm.Downstatus.Panels(4).Bevel = sbrRaised
Else          'Show stop button
      mainFrm.Downstatus.Panels(4).Visible = False
      mainFrm.Downstatus.Panels(5).Visible = True
      mainFrm.Downstatus.Panels(5).Bevel = sbrRaised
End If
   Unload frmOptions
   mainFrm.SetFocus
End Sub
Private Sub Combo1_DropDown()
CD1.Color = Combo1.Text
CD1.ShowColor
Combo1.Text = CD1.Color
End Sub

Private Sub Combo1_GotFocus()
OptionStatus.SimpleText = "Color for positive values (input, output, and adder
function)"
End Sub

Private Sub Combo2_DropDown()
CD1.Color = Combo2.Text
CD1.ShowColor
Combo2.Text = CD1.Color
End Sub
```

```
Private Sub Combo2_GotFocus()
OptionStatus.SimpleText = "Color for negative values (input, output,
and adder function)"
End Sub

'Load the values to the form
Private Sub Form_Load()
   scrolSpeed.Value = Speed
   OptionStep.Value = Step
   OptionFull.Value = Not Step

   Combo1.Text = PositiveColor
   Combo2.Text = NegativeColor
   TextAdder.Text = GaugeMax
   MaxChanged = False

   TextIF.Text = IncreaseFactor
   TextDF.Text = DecreaseFactor
   TextSR.Text = SuccessRange

   TextTHETA.Text = THETA
   TextPHI.Text = PHI
End Sub

Private Sub OptionFull_Click()
OptionStatus.SimpleText = "Simulation will continue until it
ends(training and testing)"
   OptionStep.Value = False
   OptionFull.Value = True
End Sub


Private Sub OptionStep_Click()
OptionStatus.SimpleText = "Simulation will stop after each
presentation of input set(training and testing)"
OptionFull.Value = False
```

```
OptionStep.Value = True
End Sub
Private Sub scrolSpeed_GotFocus()
OptionStatus.SimpleText = "Simulation speed"
End Sub

Private Sub TextAdder_Change()
If Val(TextAdder.Text) > 0 Then
   MaxChanged = True
Else
   TextAdder.Text = GaugeMax
End If
End Sub

Private Sub TextAdder_GotFocus()
OptionStatus.SimpleText = "Maximum value of the adders(visual part only)"
End Sub

Private Sub TextDF_Change()
OptionStatus.SimpleText = "Simulation will continue until it ends(training and
testing)"
End Sub

Private Sub TextDF_GotFocus()
OptionStatus.SimpleText = "Learning rate decrease factor in the case of search
fail"
End Sub

Private Sub TextIF_GotFocus()
OptionStatus.SimpleText = "Learning rate increase factor in the case of
successful search"
End Sub
Private Sub TextPHI_GotFocus()
OptionStatus.SimpleText = "Learning rate decrement factor"
End Sub
```

```
Private Sub TextSR_Change()
OptionStatus.SimpleText = "Maximum RMS difference for successful
search"
End Sub

Private Sub TextSR_GotFocus()
OptionStatus.SimpleText = "Maximum RMS difference for successful
search"
End Sub

Private Sub TextTHETA_GotFocus()
OptionStatus.SimpleText = "Weight factor for previous gradient"
End Sub

Attribute VB_Name = "paramFrm"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Form_Load()
'Enable everything at the beginning
textLRate.Visible = True
   Label2.Visible = True
Label8.Visible = False
textMomentum.Visible = True
   Label3.Visible = True
textMaxGrowth.Visible = True
   Label5.Visible = True
textSwitch.Visible = True
   Label6.Visible = True
TextMax.Visible = True
   Label7.Visible = True

'Put the values to the form
textLRate.Text = LearningRate
textMomentum.Text = Momentum
textEpochs.Text = MaxEpochs
textTolerance.Text = Tolerance
```

```
textMaxGrowth.Text = MaxFactor
textSwitch.Text = SwitchThreshold
TextMax.Text = MaxLearningRate

'Disable the unnecessary parameters for the algorithm
If Algorithm = "BP" Then
   textMaxGrowth.Visible = False
   Label5.Visible = False
   textSwitch.Visible = False
   Label6.Visible = False
   TextMax.Visible = False
   Label7.Visible = False
End If
If Algorithm = "QP" Then
   TextMax.Visible = False
   Label7.Visible = False
End If
If Algorithm = "DBD" Then
   Label8.Visible = True
   textMomentum.Visible = False
   Label3.Visible = False
   textMaxGrowth.Visible = False
   Label5.Visible = False
   textSwitch.Visible = False
   Label6.Visible = False
End If
If Algorithm = "CG" Then
   Label8.Visible = True
   textMomentum.Visible = False
   Label3.Visible = False
   textMaxGrowth.Visible = False
   Label5.Visible = False
   textSwitch.Visible = False
   Label6.Visible = False
End If
```

```
mainFrm.Downstatus.Panels(6).Text = "Enter the " + Algorithm + "
parameters for training"
End Sub

Private Sub paramTab_Click(PreviousTab As Integer)
'paramFrm.paramPan.Caption = "Enter " +
paramFrm.paramTab.Caption + " parameters"
End Sub

Private Sub paramTab_DblClick()
 paramFrm.paramPan.Refresh
End Sub

Private Sub SSCommand1_Click()
'OK button clicked
'Get the values from the form
LearningRate = Val(textLRate.Text)
Momentum = Val(textMomentum.Text)
MaxEpochs = Val(textEpochs.Text)
Tolerance = Val(textTolerance.Text)
MaxFactor = Val(textMaxGrowth.Text)
SwitchThreshold = Val(textSwitch.Text)
MaxLearningRate = Val(TextMax.Text)

'Set error graph
frmError.Graph1.NumPoints = MaxEpochs
frmError.Graph1.LabelEvery = Int(MaxEpochs / 10)
 mainFrm.mnuTrainStart.Enabled = True
 paramFrm.Caption = temp
 Unload paramFrm
End Sub

Private Sub SSCommand2_Click()
'Cancel button clicked
   paramFrm.Caption = temp
   Unload paramFrm
```

```
End Sub

Private Sub textEpochs_GotFocus()
paramPan.Caption = "Maximum number of presentations of training set"
End Sub

Private Sub textLRate_GotFocus()
   paramPan.Caption = "Learning rate parameter for training the network"
End Sub

Private Sub TextMax_GotFocus()
paramPan.Caption = "Max. allowable learning rate value"
End Sub

Private Sub textMaxGrowth_GotFocus()
paramPan.Caption = "A weight cannot grow more than this value * previous"
End Sub

Private Sub textMomentum_GotFocus()
paramPan.Caption = "Momentum parameter for training the network"
End Sub

Private Sub textSwitch_GotFocus()
paramPan.Caption = "Weight update is done by backprop between - and + this
value"
End Sub

Private Sub textTolerance_GotFocus()
paramPan.Caption = "Max. allowable RMS error value"
End Sub


Attribute VB_Name = "frmResult"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Form_Unload(Cancel As Integer)
mainFrm.SetFocus
End Sub

Attribute VB_Name = "frmWhatIs"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Private Sub Form_Unload(Cancel As Integer)
mainFrm.SetFocus
End Sub
```

VITA

Osman Öz

Candidate for the Degree of

Master of Science

Thesis: VISUALIZATION OF LEARNING IN NEURAL NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Simav, Kütahya, Turkey, On March 1, 1971, the son of Ísmail and Esma Öz.

Education: Received Bachelor of Education degree in Computer Technology from Marmara University, Istanbul, Turkey in February 1992. Completed the requirements for the Master of Science with a major in Computer Science at Oklahoma State University in July 1997.

Experience: Employed as a computer programming teacher by Ozsahin Private Computer Education Center in Istanbul from July 1991 to December 1992; employed by the Turkish Ministry of National Education as a teacher in Antalya from December 1992 to August 1993; employed by Simav Technical Education Faculty, Dumlupinar University, Turkey as a research assistant from August 1993 to present.