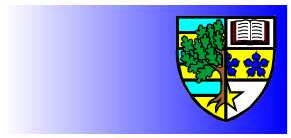

Heriot-Watt University
Edinburgh

NNetwork V2.00 Simulator

Name:	Keith J. Symington
Registration number:	990163440
Company/Address:	Physics Dept., Heriot-Watt University, Edinburgh, EH14 4AS
☎:	+44 (131) 451 3040
Fax.:	+44 (131) 451 3136
e-mail:	kjsymington@iee.org
Supervisor:	Dr. J. F. Snowdon
Date:	22nd October 1999

1 Contents

2 Optoelectronic Neural Network Demonstrator

This program is designed to simulate an optoelectronic neural network demonstrator and test theoretically its limitations. This chapter briefly gives an overview of the demonstrator by dividing it into two segments: the optical system and electronic system. The following chapters then proceed to examine the motivation behind the demonstrator and simulator operation respectively.

2.1 Optical System

The object of using optical interconnection is to supply very high connectivity using a free space optical system in which a set of emitters are connected through a diffractive optic fan-out element to a set of detectors such that each detector integrates over the weighted outputs of several emitters. The system (figure 1) is thus exploiting several of the degrees of freedom made available to computation by free space optics; the raw bandwidth of a massively parallel interconnection, the non-locality that can be achieved in such an interconnection and the capacity for large fan-out and fan-in. Currently the system consists of an 8x8 array of neurons.

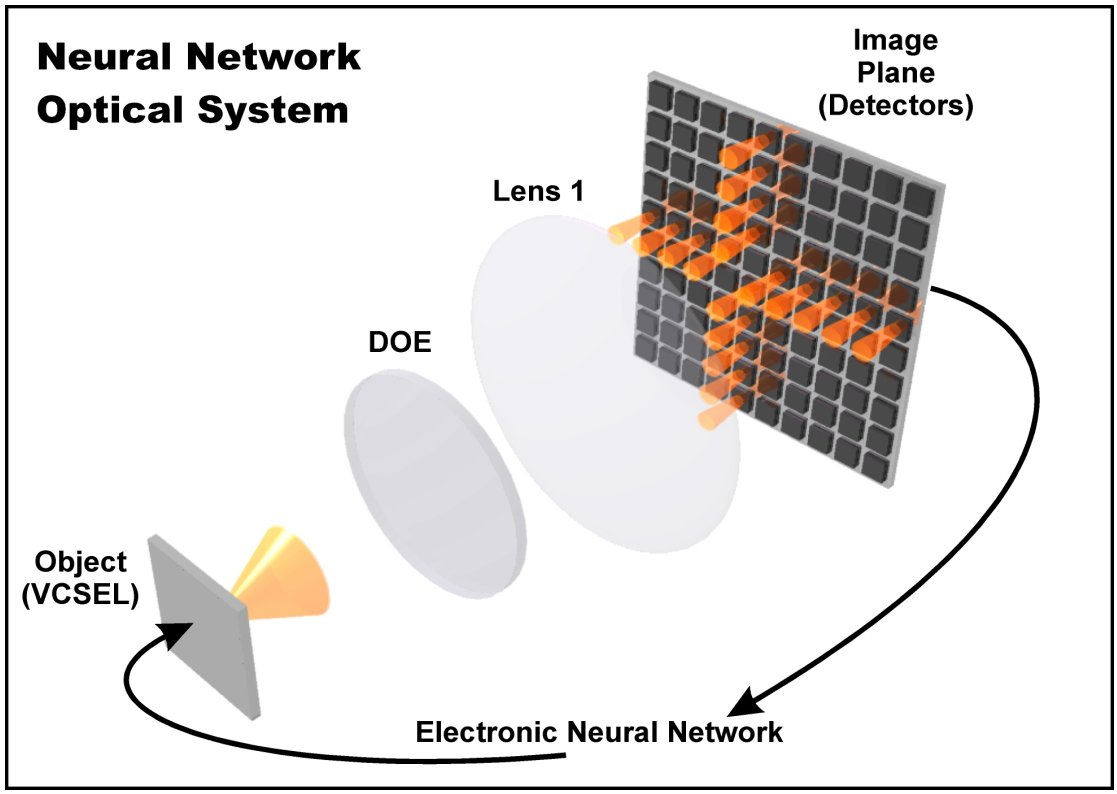

Figure 1

The diagram illustrates the fan-out from a single active VCSEL element.

Such a neural network is however intractable to build to any scalable extent in silicon because of the high degree of connectivity required. Our optical scheme enables the deployment of neural network technology. The system we describe can be reconfigured to solve other NP problems such as route relocation and path determination.

2.2 Electronic System

The electronic system (figure 2) can be considered as consisting of five stages, each performing a specific task.

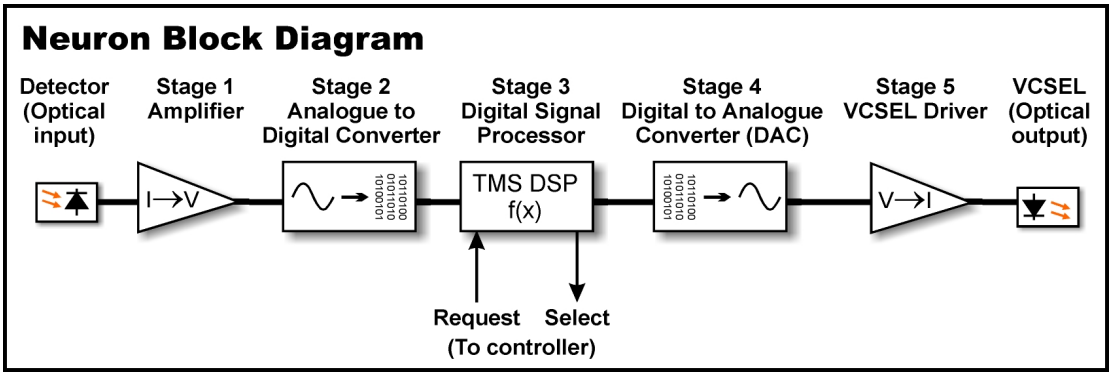

Figure 2

The five stages of the electronic system.

Starting from the optical input end, we have the detection system. This converts a current generated by incident light into a voltage of magnitude specified by the amplification of the transimpedance amplifier. The second stage is an analogue-to-digital converter (ADC) and converts the voltage received from the first stage into digital information (normally 8 bits and multiplexes 16 analogue channels through two octal ADC chips. The third stage consists of a Texas Instruments digital signal processor (DSP) which takes the digital information from the second stage and performs a transfer function on it based on previous and requested values. There are four DSPs in this system each handling 16 neurons (or channels) with each DSP under the control of a master DSP and consequently PC. The fourth stage consists of two octal digital to analogue converters (DACs) which are fed the new activation levels from the third stage and convert this information into voltages. The fifth and final stage takes the analogue values from the fourth stage and converts the voltage into an appropriate drive current for the vertical cavity surface emitting lasers (VCSEL) thus returning the signal into the optical domain. The current system drives the VCSELs in a digital fashion resulting in the combination of stages four and five. A single chip solution for stage four is currently being fabricated for stage five to allow the VCSELs to be driven in an analogue manner.

3 Solving the Assignment Problem Using Neural Networks

Current software systems suffer from an exponential increase in computational complexity when solving the quadratic assignment problem. This chapter considers the problem and proceeds to propose a solution using the inherent parallelism of a neural network to reduce computation times. A specific example is given, in this case a crossbar switch, onto which problem mapping is demonstrated and a solution given.

3.1 The Assignment Problem

As the complexity of modern communications and computational systems increases so does the need to develop new techniques which deal with common assignment problems ([1] and [2]) in situations such as:

- Network and service management.
- Distributed computer systems.
- Work Management systems.
- General scheduling, control or resource allocation problems.

The common assignment problem is essentially optimising task allocation to all available resources thus maximising throughput. In a distributed computer system this results in a many process computation being finished in the shortest possible time whereas, in a network management system, packets are routed to optimise throughput and minimise blocking.

This simulator examines specifically the assignment problem in a crossbar switch for packet routing [3]. These switches are present in many telecommunication systems and computer networks, one good example being ATM (Asynchronous Transfer Mode) networks.

3.2 Neural Network Implementation

The problem of packet routing in crossbar switches is known to be analogous to the travelling salesman problem (TSP). The TSP problem is a renowned NP complete problem [4] which means that although it can be solved by linear programming techniques, such as the Murnkes algorithm [5], it is computationally intensive and complexity grows exponentially as its order increases. Thus, a simple single processor solution will not provide satisfactory scalability.

One alternative is to apply a neural network to the TSP problem [6], [7]. The advantage of a neural net lies in the speed obtained through its inherent parallel operation, especially when dealing with large problems. Such an

implementation will easily outperform any other method at higher orders of network size ([1], [8], [9], [10], [11], [12] and [13]) providing a very good, but not optimal, solution. It has been shown [1] that, at lower orders of network size, the average solution is within 3% of optimal. However, as the network size grows this figure improves slowly and begins to approach the optimal solution.

The problem which remains with any neural network solution is its adaptation to act as a controller for a crossbar switch.

3.3 Crossbar Switches and Notation

A crossbar switch can be simply abstracted as a set of m inputs and n outputs where each input can be switched to any output.

An example of this can be seen in figure 3 where, by simply closing the correct crosspoint switch, any input line may be connected to any output line. This system has the limitation that it is mutually exclusive: any input or output lines that are in use cannot be reused. Thus, two incoming requests for the same output line will result in one becoming blocked regardless of the routing algorithm which is used.

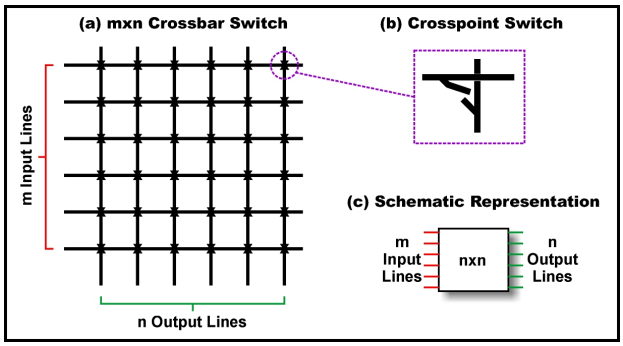

Figure 3

An mxn crossbar switch is shown here at various levels of detail.
(a) Shows an overall connection diagram for a typical crossbar switch.
(b) Details how each of the crosspoint switches work.
(c) Depicts a high level schematic of a crossbar switch.

To clarify the notation used throughout the rest of the document, please examine figure 4. This diagram details how a matrix may be mapped onto the crossbar switch, each crosspoint having a corresponding matrix element. A specific element in any matrix y can therefore be referenced using y_{ij}, where i is the input line and j the output line. Every element in the matrix can take on one of two values: 1 or 'on' when there is a connection (or connection request) or 0 or 'off' otherwise. The values and legality of the matrix is dependent on situation. Please examine the matrices shown in equations 1 and 2 overleaf.

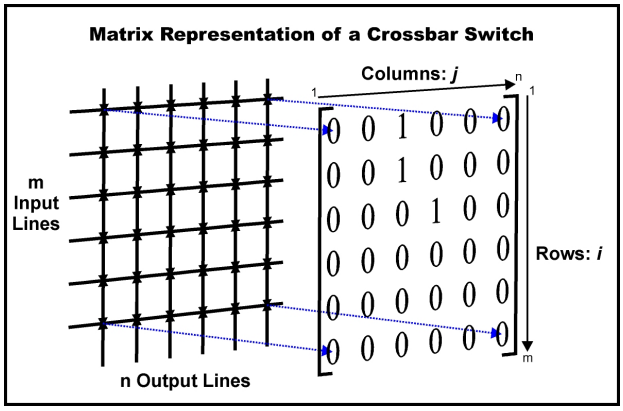

Figure 4

This diagram shows how a matrix can be mapped onto the crossbar switch thus aiding representation.

These matrices represent the crossbar switch in figure 4 but from different points of view. Equation 1 represents a set of desired connections where three input lines have requested connection to two different output lines: one

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Equation 1

This matrix shows a set of requested connections. Input *i*=1 has requested a connection with output *j*=3 and both inputs *i*=2 and *i*=3 have requested a connection to output *j*=4.

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Equation 2

This matrix shows a solution or response to the request in equation 1. It is legal because there are no other connections on the input rows and output columns which have been selected.

request is obviously going to have to wait. Such a matrix is legal regardless of the combination of zeroes and ones. Equation 2 shows a sample response. One request has been discarded in favour of another since only one input line can be connected to one output line at a time. A response is considered to be legal if there are no other closed switches on the same lines, i.e. all other elements in the same row and column as the active element must be zero.

The real optimisation problem comes in when you start to consider a system which has buffered input In such systems there can be multiple packets waiting on a single input line for various output lines, as can be seen in equation 3. Requests for multiple connections can be seen in the left matrix and the only optimal solution which maximises throughput on the right. This request matrix proves useful for testing crossbar control systems.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 3

The left matrix shows a request and the right the only optimal response. This matrix is useful for testing a system.

By making the values continuous between 0 and 1, a connection could start off with a partially active value (e.g. 0.5 instead of 1). This means that any packet whose weight was 1 would have an advantage over 0.5 and therefore a priority. In fact, the matrix values need not even stay between 0 and 1 but could be, for example, between 0 and 10. This element could be converted to a value representing the number of packets waiting on each connection.

3.4 The Neural Network

The key to utilising the parallelism of a neural network is matching the network as closely as possible to the problem. For more information please refer to references [14], [16], [17], [18], [19] or [20].

3.4.1 The Neuron or Node

A neural network consists of a large number of processing elements called neurons (see figure 5 overleaf or references [15] and [20]) which are highly

interconnected to each other in a specific fashion. Neurons are the basic building blocks of neural networks and are an approximation of the neuron found in nature. A neuron takes inputs from other neurons' outputs y_{ij} (referenced by *ij*) and multiplies their strengths by a scalar weight w_{ij} known as the synaptic weight.

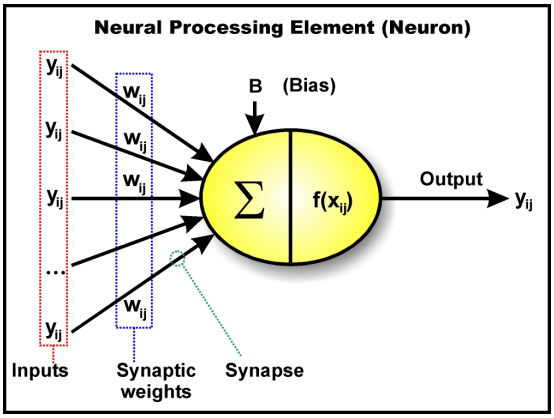

Figure 5

The building block of any neural network: the neuron.

All inputs are summed by the neuron along with a specific bias to find x_{ij}. The neuron's output y_{ij} can then be determined using a monotonic activation function $f(x_{ij})$, such as shown in equation 4. Here β is used to control the gain of the sigmoid function, a higher value resulting in a steeper transition, and o_{min} and o_{max} determine the minimum and maximum output values for y_{ij} respectively.

$$y_{ij} = f(x_{ij}) = o_{\min} + \frac{o_{\max} - o_{\min}}{1 + e^{-\beta x_{ij}}} \qquad \textbf{Equation 4}$$

The exact form of $f(x_{ij})$ is not particularly important and in fact any appropriate non-linear monotonically increasing function could be used. The preferred embodiment is, however, the sigmoid function.

3.4.2 The Updating Rule

Adapting a neural network to any problem requires that an updating rule is defined and thereby the network interconnection structure. The updating rule determines the next value that a neuron will take with respect to time based upon the previous outputs of other neurons, as shown in equation 5:

$$x_{ij}(t) = i_{ij}\left(x_{ij}(t-1) + \lambda_{ij}\left(-A\sum_{k\neq i}^{m} w_{kj}y_{kj} - A\sum_{k\neq j}^{n} w_{ik}y_{ik} + B \right) \right) \qquad \textbf{Equation 5}$$

where:

x_{ij}: is a summation of all inputs to the neuron referenced by *ij* including the bias.

i_{ij}: determines whether a neuron referenced by *ij* is allowed to evolve: it can take a value of either 0 or 1.

λ_{ij}: time constant for neuron referenced by *ij*.

w_{ij}: synaptic weight for neuron input *ij*.

y_{ij}: output from neuron referenced by *ij*.

A: Neuron optimisation value.

B: Neuron specific threshold or bias.

and x_{ij} is related to y_{ij} using equation 4.

To illustrate this rule further, figure 6 shows an interconnection diagram for the modified system. Here the neuron marked with output y_{ij} has inputs from all the other neurons in the same row $-w_{2j}.y_{2j}$ and column $-w_{i2}.y_{i2}$. The important point to note here is that the neural network works in an inhibitory fashion so any active input will inhibit y_{ij}. B describes the external bias supplied to each neuron which is not inhibitory.

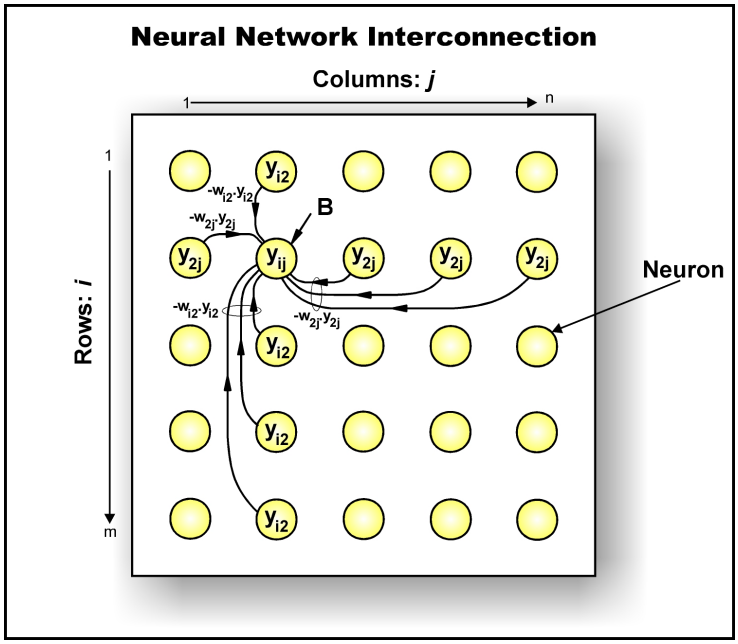

Figure 6

Interconnection diagram for modified Hopfield neural network.

The idea behind this interconnection strategy is that any active neuron will try and turn all the others off, eventually resulting in only one of the requests remaining active in each row and column. However, to demonstrate its ability to find an optimal solution, the example in figure 6 needs to be extended slightly, as in equation 6. The left matrix here represents a request and the right its best case solution with y_{22} switched off. Careful consideration leads us to conclude that the network must converge to the solution shown here since both y_{24} and y_{42} are inhibiting y_{22}, thus resulting in it being switched off before the others and essentially losing. If y_{22} had won in this case then it would have resulted in a poor solution since y_{24} and y_{42} would be off: obviously not maximising potential throughput.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Equation 6

The left matrix is a request and the right its solution.

It has been shown by Hopfield that with symmetric connections and a monotonically increasing activation function $f(x)$, the dynamical system described by the neural network possesses a Lyapunov (energy) function which continually decreases with time. The existence of such a function guarantees that the system converges towards equilibrium which is often referred to as a 'point attractor'.

3.4.3 Local Minima

In any system with a continually reducing energy function, there is always a risk that the system will become trapped in a local minima. In this system, a local minima can be represented as a solution which satisfies the switching constraints but is not a global optimal solution. The best way round this

problem is to introduce noise into the system by varying β. This alteration in the activation curve's gradient is significant enough to provide successful convergence to a global minimum during network simulation. In the system constructed, this variation is provided by the optical system – however the exact magnitude is, as yet, unclear.

3.5 Conclusions

Simulation of the system proves not only that the neural network works but that it is highly scaleable and has underlined two important points:

- Noise plays a very significant role in this model. As the noise level increases, the time taken for network stabilisation decreases. However, when the noise value reaches unity the network becomes unstable and does not provide a valid or steady solution.
- Network size plays an important role in convergence to a solution: the larger it is, the longer it takes to converge.

What makes this system so interesting is its diversity: switching is only one of its many applications. Essentially, this system could be used to solve any quadratic assignment problem where time is of the essence. Its ability to handle larger order problems without serious performance degradation emphasises the contribution such systems could make to the field of computing.

4 The NNetwork Program

This chapter describes how the demonstrator is simulated by NNetwork and the program's usage. Parameter limitations are discussed as well as notes on what their values should be to provide a valid model.

4.1 Start-up

When the program is run, the screen shown in figure 7 is displayed. This box sets up the neural network size in rows *m* and columns *n* as defined in section 3.3. The three marked items allow the values to be changed by different methods. The edit box (1) can be used to directly type in a value, the slider (2) can be dragged to set a value or the speed buttons (3) can be used to set pre-defined minimum and maximum values.

Note that the larger the network is, the longer it will take to simulate. The number of calculations required per iteration increases exponentially as the network size increases.

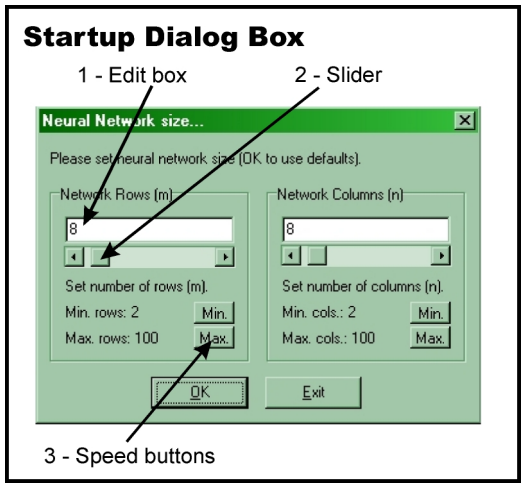

Figure 7

Dialog box shown on program start-up.

4.2 Planes

The simulator divides the demonstrator up into 4 'planes'. Each plane contains information about a neuron at a certain point and is therefore represented by a matrix of the same size as the neural network.

4.2.1 Request Plane

The request plane P_R specifies a set of requested connections and is sent to the DSP from an external controller. If the value of a request is zero than a neuron cannot evolve and will never turn on. The higher the value here, the more priority a neuron has. This is the only plane which can be edited.

4.2.2 Output Plane

The output plane P_O describes the neuron output levels from stage 4 in figure 2. These levels are voltages and are quantised to the resolution of the DAC. These are the values that indicate the current state of the neural network – a set of maxima and minima usually indicating network convergence.

4.2.3 Input Plane

The input plane P_I measures the input from the optical system and is a voltage level from stage 2 in figure 2. This voltage is quantised to the bit depth of the ADC. Noise has been added to this value which is primarily from the optical system.

4.2.4 Memory Plane

Each of the neurons store an internal value from the previous state. This value is held in the memory plane P_M and is a real number.

4.3 Main Program

Figure 8 below shows a screenshot of the main window. For a description of each element please read the appropriate description.

Main Program Interface

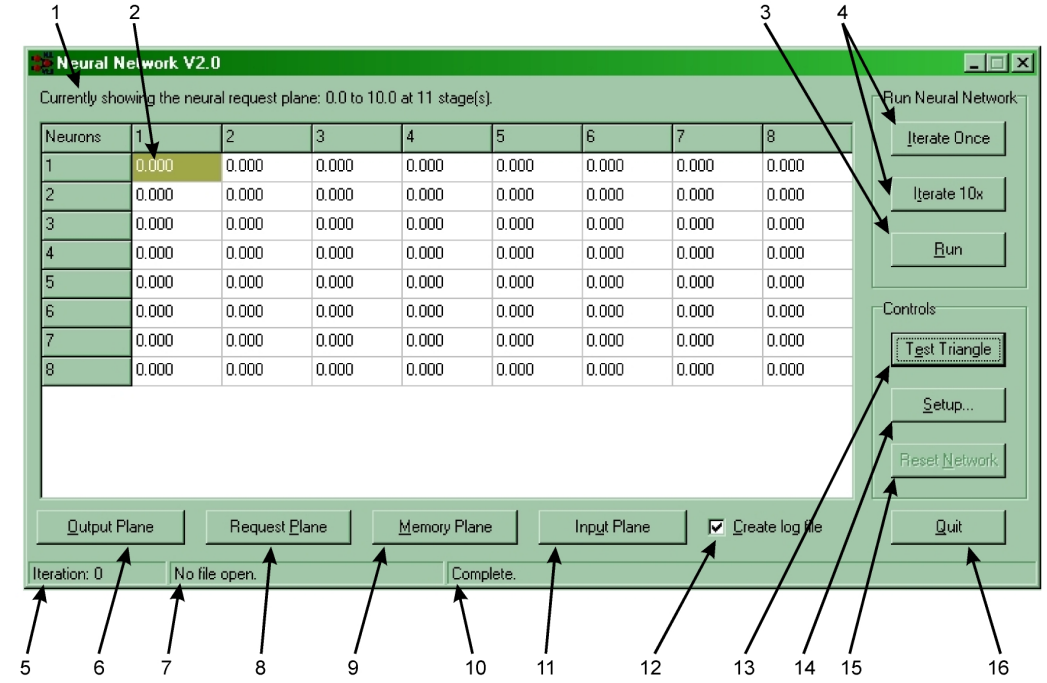

Figure 8

NNetwork main program window.

1. Indicates the plane currently shown and associated parameters.
2. The actual values of the plane. Double clicking while in the request plane allows the values to be changed.
3. Runs the simulation until the program has deemed that it has converged. If the total number of iterations reaches 2000 then the program deems the system to be non-convergent and stops. The system is deemed to have converged when all values in the output plane are within the convergence margin of either minimum or maximum values. This does require that a minimum number of iterations are undertaken, which defaults at 5, and can be set from the Setup dialog.

4. Iterates once or ten times respectively.
5. Shows the number of iterations that have been performed.
6. Displays the output plane in the window.
7. Shows the name of the currently open output file. The files have a name of `nnetxxxx.out` where `xxxx` is a number from 0000 to 9999. Thus 10000 data files can be written without any information being lost when a file is overwritten. See section 4.6 for sample file output.
8. Displays the Request plane in the window. The values can be edited by double-clicking before simulation has begun. If simulation has already begun then the network must be reset thereby keeping solution integrity.
9. Shows the neural memory plane values.
10. The status bar shows what the program is currently doing, be it calculating, redrawing, writing to file etc.
11. Displays the input plane values.
12. When checked, a log file will be generated containing complete information on the simulation when the simulation is started.
13. A quick fill button that fills the request plane triangularly with the value in the top left-hand neuron (1, 1).
14. Setup allows the neural system characteristics to be altered. See section 4.4 for more details.
15. Resets the simulation. Only enabled after a simulation has been started.
16. Does what it says – quits the program.

4.4 Setup Parameters

This section describes what the values in each tab of the setup dialog do and what their default values are.

4.4.1 Neural Network Setup Tab

This tab (figure 9) allows alteration of neural network specific parameters.

- Network *A* Parameter: This parameter specifies a weight by which the summation of input values is multiplied. The default is 1250.
- Network Bias *B*: The bias added to each neuron. The default is 50.
- Sigmoid Function Gradient β: Gradient of the sigmoid activation function *f(x)* used to describe the neuron. The default is 0.02.

Neural Network Setup Tab

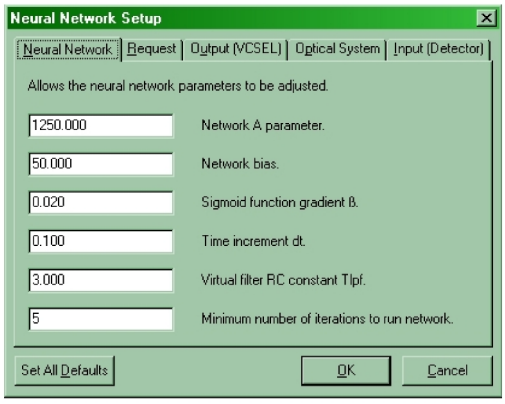

Figure 9

- Time Increment *dt*: Since the discrete system models an analogue one there must be a time interval at which network weights are re-calculated. This is specified by the time increment. The default is 0.1.
- Virtual filter RC constant T_{lpf}: Another virtual constant since the system is discretely modelling an analogue system. This constant is the RC value of a low pass filter stage originally used in the input. It should be possible to combine both *dt* and T_{lpf} into a single value. The default is 3.
- Minimum number of iterations i_{min} to run before network can be considered as having converged. Sometimes the network can be deemed to have converged (section 4.3, point 3) before it actually has. The default is 5.

4.4.2 Request Setup Tab

By double-clicking on the request plane, a slider appears which allows a request value to be set. This tab (figure 10) allows these limits to be set. Note that altering any value in this tab will reset the request plane.

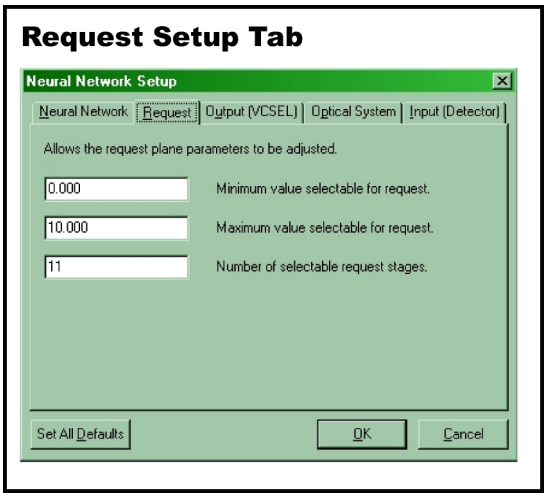

Figure 10

- Minimum value selectable for request R_{min}: Minimum value to which the slider will go. The default is 0.
- Maximum value selectable for request R_{max}: Maximum value to which the slider will go. The default is 10.
- Number of selectable request stages R_{stg}: The number of selectable values on the slider. The higher this value is, the higher the slider resolution. The default is 11.

4.4.3 Output Setup Tab

This tab (figure 11) allows the adjustment of the output system, specifically that of the DAC system (stage 4).

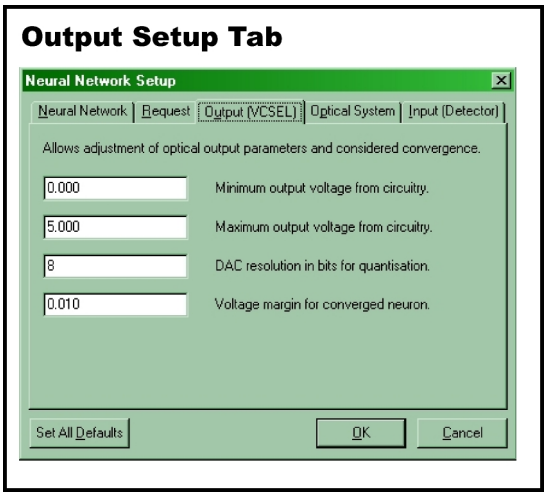

Figure 11

- Minimum output voltage from circuitry V_{omin}: Specifies the voltage used to switch a VCSEL off. The default is 0V.
- Maximum output voltage from circuitry V_{omax}: This is the voltage level used when a VCSEL if fully on. The default is 5V.
- DAC resolution in bits for quantisation Q_{DAC}: This essentially specifies the resolution of the DAC used. The higher the resolution, the better any

approximation to an actual value. For example, an 8 bit DAC would be able to set the VCSEL to one of 256 voltage levels from V_{omin} to V_{omax} whereas a 1 bit DAC (essentially a digital driver) could only set the VCSEL to either V_{omin} or V_{omax}. Quantisation is performed after the target voltage has been calculated. The default resolution is 8 bits.

- Voltage margin for converged neuron V_{conv}: If a neuron is within this voltage margin of either V_{omin} or V_{omax} it is to have converged. Once all neurons have converged the run function will break. Note it is possible to continue calculating using the 'iterate' buttons. The default margin is 0.01V.

4.4.4 Optical System Setup Tab

This tab (figure 12) allows optical system parameters to be adjusted to assess the performance of the system with, perhaps, an inefficient DOE etc.

Optical Setup Tab

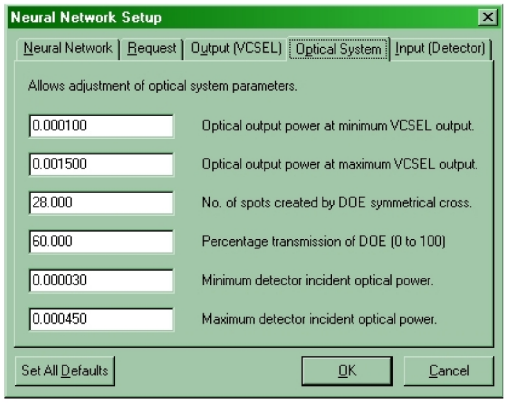

Figure 12

- Optical output power at minimum VCSEL output P_{omin}: This is the optical power in watts generated by the VCSEL when V_{omin} is applied to the DAC. The default is 0.1mW.
- Optical output power at maximum VCSEL output P_{omax}: This is the optical power in watts generated by the VCSEL when V_{omax} is applied to the DAC. The default is 1.5mW.
- No. of spots created by DOE symmetrical cross S_{DOE}: The DOE generates a 'cross' of spots which are incident on other detectors in the same row and columns. For the system to work, m-1 detectors both above and below must be illuminated and n-1 detectors to the left and right. Thus the number of spots required for efficient operation can be calculated using 7:

$$S_{DOE} = 2(m+n-2) \qquad \textbf{Equation 7}$$

  Each spot is considered to have an equal amount of power. The default value is based on the default size (8x8) and works out as 28.

- Percentage transmission of DOE (0 to 100) T_{DOE}: Since the DOE is not perfect, only a certain percentage of the light is transmitted. This can be specified here. For an efficient DOE, transmission is normally around 60%; thus the default is 60.
- Minimum detector incident optical power P_{imin}: The minimum optical power in watts which can fall on a detector when all VCSELs are off. This produces the minimum detector response V_{imin}. For efficient operation this can be calculated as in equation 8:

$$P_{i\min} = \frac{T_{DOE} P_{o\min} (m+n-2)}{100 S_{DOE}} \qquad \textbf{Equation 8}$$

When simulating a real system, actual values can be used. The default is 30μW.

- Maximum detector incident optical power P_{imax}: The maximum optical power in watts which can fall on a detector when all VCSELs are on. This produces the maximum detector response V_{imax}. For efficient operation this can be calculated as in equation 9:

$$P_{i\max} = \frac{T_{DOE} P_{o\max} (m+n-2)}{100 S_{DOE}} \qquad \textbf{Equation 9}$$

When simulating a real system, actual values can be used. The default is 450μW.

4.4.5 Input Setup Tab

This tab (figure 13) is used to specify input parameters of the system as modelled in stage 2.

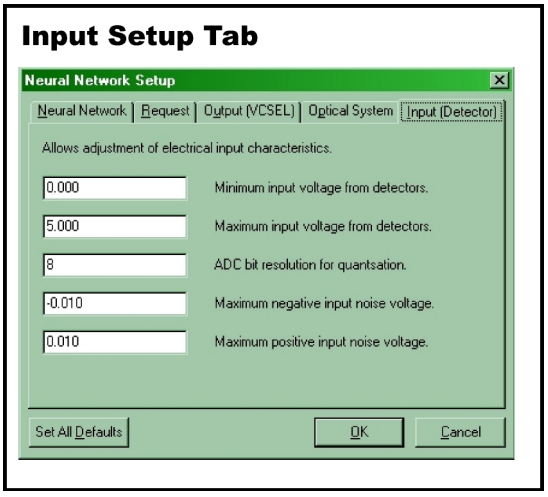

Figure 13

- Minimum input voltage from detectors V_{imin}: Specifies the voltage produced when P_{imin} watts or less is incident on the detectors. The default is 0V.
- Maximum input voltage from detectors V_{imax}: This is the voltage level produced when P_{imax} watts or more is incident on the detectors. The default is 5V.
- ADC bit resolution for quantisation Q_{ADC}: This specifies the resolution of the ADC used. The higher the resolution, the better any approximation to an actual value. For example, an 8 bit ADC would be able to read 256 voltage levels from the detector between V_{imin} and V_{imax} whereas a 1 bit ADC could only read the value as either V_{imin} or V_{imax} – whichever is closest. Quantisation is performed after the detector voltage has been calculated. The default resolution is 8 bits.
- Maximum negative input noise voltage N_{-}: Up to this value can be removed from the detector voltage representing negative noise in the system. The value must contain a negative sign. The default is –0.01V.
- Maximum positive input noise voltage N_{+}: Up to this value can be added to the detector voltage representing positive noise in the system. The default is 0.01V.

4.5 Program Description

This section illustrates one loop iteration and, using the variables defined in section 4.4, examines how one plane may be calculated from the previous. In this example, *ij* is used to reference a location on a plane. There are three stages in the iteration with all planes starting with a value of zero, except for

the request plane which contains the requests. This chapter is written from a programming point of view. For example, $x=x+2$ may be seen. This simply means that the new value of x is the old value plus 2.

4.5.1 Calculating the Output Plane

The first stage in a iteration is to calculate the new values for the output plane (equation 10):

$$P_{O_{ij}} = P_{R_{ij}} \frac{V_{o\max} - V_{o\min}}{1 + e^{-\beta P_{M_{ij}}}} \qquad \textbf{Equation 10}$$

The next stage is to quantise the output values in P_O calculated above to Q_{DAC} bits thus representing the DAC.

4.5.2 Calculating the Input Plane

The next stage requires a temporary plane for calculation which we will call P_{temp}. This plane is used to calculate the VCSEL optical output power (equation 11) by presuming a linear relationship between minimum and maximum powers. Please note that the subscripts are very important as P could either be a plane or a power.

$$P_{temp\,ij} = \left(P_{O_{ij}} - V_{o\min}\right) \times \left(\frac{P_{o\max} - P_{o\min}}{V_{o\max} - V_{o\min}}\right) + P_{o\min} \qquad \textbf{Equation 11}$$

From this we can calculate the amount of power in each spot produced by the VCSELs (equation 12) – presuming the DOE distributes the power evenly.

$$P_{temp_{ij}} = \frac{P_{temp_{ij}} \frac{T_{DOE}}{100}}{S_{DOE}} \qquad \textbf{Equation 12}$$

The next stage is to calculate the total power incident on each detector (equation 13).

$$P_{I_{ij}} = \sum_{k \neq i}^{m} P_{temp_{kj}} + \sum_{k \neq j}^{n} P_{temp_{ik}} \qquad \textbf{Equation 13}$$

This power value is then be translated into a voltage using equation 14:

$$P_{I_{ij}} = \left(P_{I_{ij}} - P_{i\min}\right) \times \left(\frac{V_{i\max} - V_{i\min}}{P_{i\max} - P_{i\min}}\right) + V_{i\min} \qquad \textbf{Equation 14}$$

A random noise from the system between N_- and N_+ is then added since the neural network thrives on noise and system noise must be simulated. Once the noise has been added, the voltage held in P_I is then quantised to Q_{ADC} bits.

4.5.3 Calculating the Memory Plane

The final stage in the loop is to calculate the values in the memory plane P_M. This is done using equation 15 overleaf:

$$P_{M_{ij}} = P_{M_{ij}} + \frac{dt}{T_{lpf}}\left(-A \times P_{I_{ij}} + B\right)$$ **Equation 15**

4.6 Example Simulation

This section gives an example of program output simply by running the program with the default values. The only thing that need be set is the request plane P_R in which the upper left value was set to 1 and "Test Triangle" clicked to fill the remainder of the plane. Examining the output log file allows us to determine the initial values, request matrix and graph neuron evolution using Microsoft Excel as shown in figure 14.

```
Neural Network V2.0 Log File

Network size: 8 rows and 8 columns.
Network parameters: A=1250.000, Bias=50.000, Beta=0.020, dt=0.100, Tlpf=3.000 with a minimum of 5 iterations.
Request values from 0.000 to 10.000 using 11 stages.
Output from 0.000 to 5.000 Volts using 8 bit(s) resolution.
Neural system has a convergence margin of 0.010.
VCSEL output ranges between 0.000100 and 0.001500 Watts of optical power.
DOE creates a symmetrical cross with 28.000 spots with 60.000% transmission.
The detector has a response from 0.000030 to 0.000450 Watts of incident optical power.
Detector input from 0.000 to 5.000 Volts using 8 bit(s) resolution.
Maximum input voltage noise from -0.010 to 0.010 Volts.

R, 1.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 1.000, 1.000, 0.000, ……
1, 2.510, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 2.510, 2.510, 0.000, ……
……
325, 5.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 5.000, 0.000, ……
```

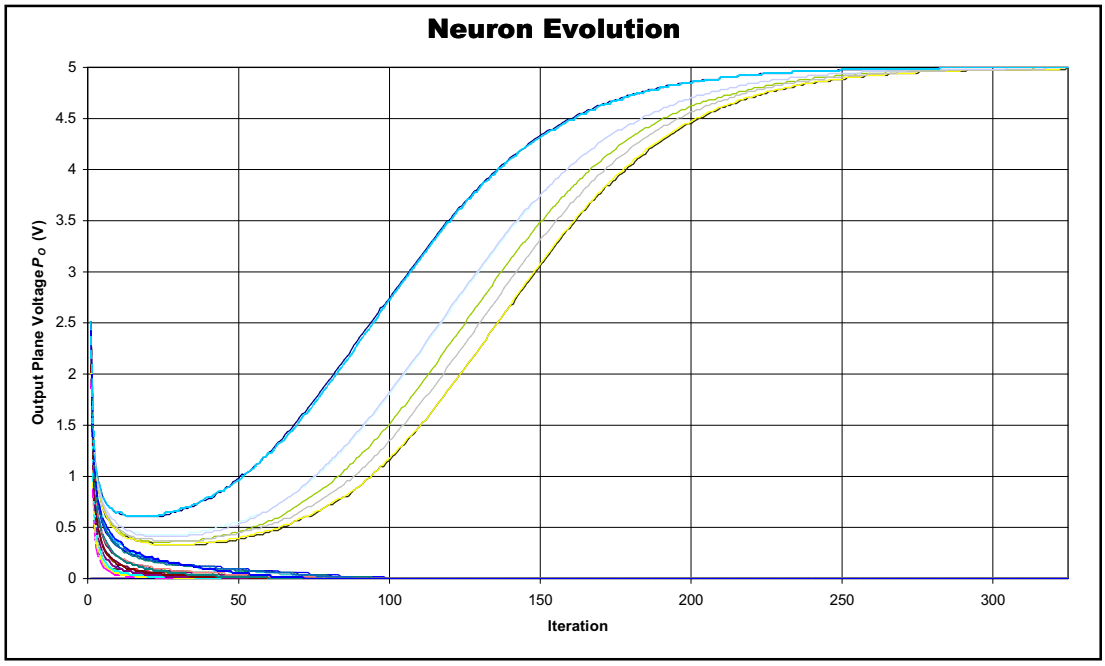

Figure 14

Each line represents a single neuron's evolution. Some neurons turn themselves off (go to 0V) and some on (go to 5V).

4.7 Known Problems

There is one known problem which occurs when excessive iterations are performed or large values of β are use: and exponential overflow. This problem is trapped by the simulator but if the system has a large number of calculations to do then a large number of these messages could be received.

5 Conclusions

The simulator provides a basis for testing the neural network switch controller not only allowing optimisation but enabling the prediction of system limitations. The only deficiency in this simulator is the lack of flexibility in altering DOE configuration. It is hoped that this will be implemented at a future date.

6 Glossary

ADC	Analogue to Digital Converter
DAC	Digital to Analogue Converter
DOE	Diffractive Optic Element
DSP	Digital Signal Processor
PC	Personal Computer
TSP	Travelling Salesman Problem
VCSEL	Vertical Cavity Surface Emitting Laser

7 Bibliography

[1] M. R. W. Manning and M. Gell, *"Evaluation of the Hopfield Neural Network for Service Assignment"*, BT Labs paper, Martlesham Heath, Ipswich, IP5 7RE, publication date unknown.

[2] W. J. Wolfe, J. M. MacMillan, G. Brady, R. Mathews, J. A. Rothman, D. Mathis, M. D. Orosz, C. Anderson and G. Alaghband, *"Inhibitory Grids and the Assignment Problem"*, IEEE Transactions on Neural Networks, volume 4, number 2, March 1993.

[3] T. X. Brown, *"Chapter 3: Controlling Circuit Switching Networks"*, Extract from T. X. Brown's Thesis from CalTech.

[4] M. R. Garey and D. S. Johnson, *"Computers and Intractability"*, New York, W. H. Freeman, 1979.

[5] J. Munkres, *"Algorithms for Assignment and Transportation Problems"*, J. Soc. Ind. Appl. Math., 5, 32-8, 1957.

[6] J. J. Hopfield and D. W. Tank, *"'Neural' Computation of Decisions in Optimisation Problems"*, Biological Cybernetics, volume 52, pages 141-152, 1985.

[7] R. D. Brandt, Y. Wang, A. J. Laub and S. K. Mitra, *"Alternative Networks for Solving the Travelling Salesman Problem"*, IEEE International Conference on Neural Networks, 24th to 28th Feb. 1998, San-Diego.

[8] Peter W. Protzel, Daniel L. Palumbo and Michael K. Arras, *"Performance and Fault-Tolerance of Neural Networks for Optimisation"*, IEEE Transactions on Neural Networks, volume 4, number 4, July 1993.

[9] C. Bousoño-Calzón and M. R. W. Manning, *"The Hopfield Neural Network Applied to the Quadratic Assignment Problem"*, BT Labs paper, Martlesham Heath, Ipswich, IP5 7RE, publication date unknown.

[10] Joydeep Ghosh, Ajat Hukkoo and Anjun Varma, *"Neural Networks for Fast Arbitration and Switching Noise Reduction in Large Crossbars"*, IEEE Transactions on Circuits and Systems, volume 38, number 8, August 1991.

[11] T. X. Brown, *"Neural Networks for Switching"*, IEEE Communications Magazine, November 1989.

[12] A. Marrakchi and T. Troudet, *"A Neural Net Arbitrator for Large Crossbar Packet Switches"*, Circuits and Systems Letters, IEEE Transactions on Circuits and Systems, volume 36, number 7, July 1989.

[13] S. B. Aiyer, M. Niranjan and F. Fallside, *"A Theoretical Investigation into the Performance of the Hopfield Model"*, IEEE Transactions on Neural Networks, volume 1, number 2, June 1990.

[14] J. J. Hopfield, *"Neural Networks and Physical Systems with Emergent Collective Computational Abilities"*, Proc. Natl. Acad. Sci. USA, volume 79, pages 2554-2558, April 1982.

[15] J. J. Hopfield, *"Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons"*, Proc. Natl. Acad. Sci. USA, volume 81, pages 3088-3092, May 1984.

[16] Robert L. Harvey, *"Neural Network Principles"*, Prentice Hall International Editions, 1994.

[17] James A. Anderson, *"An Introduction to Neural Nets"*, IT Press, 1995.

[18] Simon Haykin, *"Neural Networks"*, Macmillan Publishing Company, 1994.

[19] Clifford Lau, *"Neural Networks: Theoretical Foundations and Analysis"*, IEEE Press, 1991.

[20] J. Hertz, A. Krough and R. G. Palmer, *"Introduction to the Theory of Neural Computation"*, Addison-Wesley, 1991.

8 Appendix

This section contains source code for NNetwork V2.00.

8.1 Data.h

```
//---------------------------------------------------------------------------
#ifndef DataH
#define DataH
//---------------------------------------------------------------------------
#include <vcl\System.hpp>
#include <vcl\Windows.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
#include <ComCtrls.hpp>

//---------------------------------------------------------------------------
// Limit defines for data network data.
#define MIN_ROWS                2  // Minimum number of rows.
#define MAX_ROWS              100  // Maximum number of rows.
#define MIN_COLS                2  // Minimum number of columns.
#define MAX_COLS              100  // Maximum number of columns.
#define DEF_NO_ROWS             8  // Default number of neuron rows.
#define DEF_NO_COLS             8  // Default number of neuron columns.

// Default defines for Neural network.
#define DEF_A              1250.0  // Value of A parameters.
#define DEF_BIAS             50.0  // Bias parameter.
#define DEF_BETA             0.02  // Sigmoid function gradient.
#define DEF_DT                0.1  // Time increment delta T.
#define DEF_TLPF              3.0  // RC constant of virtual filter.
#define DEF_MINITER             5  // Instability region to overcome.  Only in the "Run" section.

#define DEF_MIN_REQUEST       0.0  // Minimum value for a request.
#define DEF_MAX_REQUEST      10.0  // Maximum value for a request.
#define DEF_REQUEST_STAGES     11  // Number of different values selectable for a request.

#define DEF_MIN_OUTPUT        0.0  // Minumum output voltage.
#define DEF_MAX_OUTPUT        5.0  // Maximum output voltage.
#define DEF_OUTPUT_BIT_RES      8  // Bit resolution available at output.
#define DEF_CONV_MARGIN      0.01  // If a neuron is within this of minimum or maximum it has
stabilised.

#define DEF_MIN_OPTICAL    0.0001  // Optical output power at min output voltage.
#define DEF_MAX_OPTICAL    0.0015  // Optical output power at max output voltage.
#define DEF_DOE_SPOTS        28.0  // No of spots created by DOE (presume symmetrical cross).
#define DEF_DOE_EFFICIENCY   60.0  // Percentage of optical power transmitted.
#define DEF_MIN_DET       0.00003  // Produces MIN_INPUT_VOLTAGE.
#define DEF_MAX_DET       0.00045  // Produces MAX_INPUT_VOLTAGE.

#define DEF_MIN_INPUT         0.0  // Minimum input voltage.
#define DEF_MAX_INPUT         5.0  // Maximum input voltage.
#define DEF_INPUT_BIT_RES       8  // Bit resolution available at input.
#define DEF_VMINUS_NOISE    -0.01  // Max. negative voltage noise on voltage from detector.
#define DEF_VPLUS_NOISE      0.01  // Max. positive voltage noise on voltage from detector.

//---------------------------------------------------------------------------
// Text string defines.
#define MIN_ROWS_TEXT "Min. rows: %d"
#define MAX_ROWS_TEXT "Max. rows: %d"
#define MIN_COLS_TEXT "Min. cols.: %d"
#define MAX_COLS_TEXT "Max. cols.: %d"
#define MEM_ALLOC_ERR "Could not allocate memory.  Terminating."

//---------------------------------------------------------------------------
class TnnData : public TForm
{
__published:
        TButton *OKBtn;
        TButton *CancelBtn;
  TGroupBox *GroupBoxRows;
```

```
  TLabel *Rows;
  TGroupBox *GroupBoxCols;
  TEdit *EditCols;
  TLabel *Cols;
  TLabel *Purpose;
  TLabel *MinRows;
  TLabel *MaxRows;
  TLabel *MinCols;
  TLabel *MaxCols;
  TEdit *EditRows;
  TSpeedButton *SpeedMinRows;
  TSpeedButton *SpeedMaxRows;
  TSpeedButton *SpeedMinCols;
  TSpeedButton *SpeedMaxCols;
  TScrollBar *ScrollCols;
  TScrollBar *ScrollRows;
  void __fastcall OKBtnClick(TObject *Sender);
  void __fastcall CancelBtnClick(TObject *Sender);
  void __fastcall SpeedMinRowsClick(TObject *Sender);
  void __fastcall SpeedMaxRowsClick(TObject *Sender);
  void __fastcall SpeedMinColsClick(TObject *Sender);
  void __fastcall SpeedMaxColsClick(TObject *Sender);
  void __fastcall ScrollRowsScroll(TObject *Sender, TScrollCode ScrollCode,
          int &ScrollPos);
  void __fastcall EditRowsKeyPress(TObject *Sender, char &Key);
  void __fastcall EditRowsExit(TObject *Sender);
  void __fastcall ScrollColsScroll(TObject *Sender, TScrollCode ScrollCode,
          int &ScrollPos);
  void __fastcall EditColsKeyPress(TObject *Sender, char &Key);
  void __fastcall EditColsExit(TObject *Sender);
private:
public:
  // Constructor.
  virtual __fastcall TnnData(TComponent* AOwner);
  // Used for startup.
  void SetupArrays();
  // Temporary holding variables.
  AnsiString holding;
  double **tempMatrix;  // For optical system.
  // Neural Network maxtrix sizes.
  int noRows, noCols;
  // Matrices are initialised by SetupArrays()
  double **nOutput;
  double **nInput;
  double **nRequest;
  double **nMemory;
  // Neural network parameters.
  double nn_A, nn_Bias, nn_beta, nn_dt, nn_Tlpf;
  int nn_MinIter;
  // Request tab.
  double rq_Min, rq_Max;
  int rq_Stages;
  // Output tab.
  double op_MinV, op_MaxV, op_Conv;
  int op_Res;
  // Optical tab.
  double os_MinOO, os_MaxOO, os_Spots, os_Trans, os_MinDet, os_MaxDet;
  // Input tab.
  double ip_MinV, ip_MaxV, ip_MinusNoise, ip_PlusNoise;
  int ip_Res;
};
//---------------------------------------------------------------------------

extern PACKAGE TnnData *nnData;
//---------------------------------------------------------------------------
#endif
```

8.2 Data.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <iostream.h>
#pragma hdrstop

#include "Data.h"
#include "NNet.h"
#include "Setup.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TnnData *nnData;
//---------------------------------------------------------------------------
// Set all text strings based on defines.
__fastcall TnnData::TnnData(TComponent* AOwner)
        : TForm(AOwner)
{
```

```
  // Display minimum and maximums.
  holding=""; holding.sprintf(MIN_ROWS_TEXT, MIN_ROWS);
  MinRows->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf(MAX_ROWS_TEXT, MAX_ROWS);
  MaxRows->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf(MIN_COLS_TEXT, MIN_COLS);
  MinCols->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf(MAX_COLS_TEXT, MAX_COLS);
  MaxCols->SetTextBuf(holding.c_str());

  // Initialise text buffers.
  holding=""; holding.sprintf("%d", DEF_NO_ROWS);
  EditRows->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%d", DEF_NO_COLS);
  EditCols->SetTextBuf(holding.c_str());

  // Initialise scroll bars.
  ScrollRows->Min=MIN_ROWS;
  ScrollRows->Max=MAX_ROWS;
  ScrollRows->Position=DEF_NO_ROWS;
  ScrollCols->Min=MIN_COLS;
  ScrollCols->Max=MAX_COLS;
  ScrollCols->Position=DEF_NO_COLS;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Copy data for initialising rows and columns.
// Tested 2/8/1999
void __fastcall TnnData::OKBtnClick(TObject *Sender)
{
  // Set defaults.
  noRows=DEF_NO_ROWS;
  noCols=DEF_NO_COLS;

  // Get rows and check within limits.
  noRows=EditRows->Text.ToInt();
  if (noRows<MIN_ROWS) noRows=MIN_ROWS;
  if (noRows>MAX_ROWS) noRows=MAX_ROWS;

  // Get columns and check within limits.
  noCols=EditCols->Text.ToInt();
  if (noCols<MIN_COLS) noCols=MIN_COLS;
  if (noCols>MAX_COLS) noCols=MAX_COLS;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// To exit the program press cancel!
void __fastcall TnnData::CancelBtnClick(TObject *Sender)
{
  // The user has requested program termination.
  PostQuitMessage(0);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Allocates memory to the arrays a requested.
// Deallocation of memory is handled automaticaly by system.
// Tested 3/8/1999 KJS
void TnnData::SetupArrays()
{
  // Test for exceptions.
  try
  { // First set up the rows.
    tempMatrix=new double *[noRows];
    for (int j=0; j<noRows; j++)
      tempMatrix[j]=new double[noCols];  // Then set up the columns

    // First set up the rows.
    nOutput=new double *[noRows];
    for (int j=0; j<noRows; j++)
      nOutput[j]=new double[noCols];  // Then set up the columns

    // First set up the rows.
    nInput=new double *[noRows];
    for (int j=0; j<noRows; j++)
      nInput[j]=new double[noCols];  // Set up the columns

    // First set up the rows.
    nRequest=new double *[noRows];
    for (int j=0; j<noRows; j++)
      nRequest[j]=new double[noCols];  // Set up the columns

    // First set up the rows.
    nMemory=new double *[noRows];
    for (int j=0; j<noRows; j++)
      nMemory[j]=new double[noCols];  // Set up the columns
```

```
  }

  // Enter this block if a bad allocation is detected.
  catch (std::bad_alloc)
  { // Memory allocation error.
    Application->MessageBox(MEM_ALLOC_ERR, PROG_NAME, MB_OK);
    PostQuitMessage(0);
  }

  // Arbitary initialisation of all matrices.
  for (int i=0; i<noRows; i++)
    for (int j=0; j<noCols; j++)
      nOutput[i][j]=nInput[i][j]=nRequest[i][j]=nMemory[i][j]=0;

  // Initialise the proper setup default values.
  SetupBox->InitialiseDefaults();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Update the rows text box with the current position.
// Tested 4/8/1999 KJS
void __fastcall TnnData::ScrollRowsScroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{ // Take the current scroll bar position and change text buffer.
  holding=""; holding.sprintf("%d", ScrollPos);
  EditRows->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Speed button for minumum number of rows.
// Tested 4/8/1999 KJS
void __fastcall TnnData::SpeedMinRowsClick(TObject *Sender)
{ // Set both text box and scroll bar to the minimum position.
  ScrollRows->Position=MIN_ROWS;
  holding=""; holding.sprintf("%d", MIN_ROWS);
  EditRows->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Speed button for maximum number of rows.
// Tested 4/8/1999 KJS
void __fastcall TnnData::SpeedMaxRowsClick(TObject *Sender)
{ // Set both text box and scroll bar to the minimum position.
  ScrollRows->Position=MAX_ROWS;
  holding=""; holding.sprintf("%d", MAX_ROWS);
  EditRows->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Waits for return key and then changes the position of the scroll bar.
// Tested 4/8/1999 KJS
void __fastcall TnnData::EditRowsKeyPress(TObject *Sender, char &Key)
{
  // Check to see if it's the return key.
  if (Key==VK_RETURN)
  { // Eliminate keypress return to windows.
    Key=0;

    // Get rows and check within limits.
    noRows=EditRows->Text.ToInt();
    if (noRows<MIN_ROWS) noRows=MIN_ROWS;
    if (noRows>MAX_ROWS) noRows=MAX_ROWS;

    // Set scroll bar.
    ScrollRows->Position=noRows;
    holding=""; holding.sprintf("%d", noRows);
    EditRows->SetTextBuf(holding.c_str());
  }
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Change the scroll bar position on de-focus.
// Tested 4/8/1999 KJS
void __fastcall TnnData::EditRowsExit(TObject *Sender)
{
  // Get rows and check within limits.
  noRows=EditRows->Text.ToInt();
  if (noRows<MIN_ROWS) noRows=MIN_ROWS;
  if (noRows>MAX_ROWS) noRows=MAX_ROWS;

  // Set scroll bar.
  ScrollRows->Position=noRows;
  holding=""; holding.sprintf("%d", noRows);
```

```
  EditRows->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Update the cols. text box with the current position.
// Tested 4/8/1999 KJS
void __fastcall TnnData::ScrollColsScroll(TObject *Sender,
      TScrollCode ScrollCode, int &ScrollPos)
{ // Take the current scroll bar position and change text buffer.
  holding=""; holding.sprintf("%d", ScrollPos);
  EditCols->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Speed button for minumum number of columns.
// Tested 4/8/1999 KJS
void __fastcall TnnData::SpeedMinColsClick(TObject *Sender)
{
  ScrollCols->Position=MIN_COLS;
  holding=""; holding.sprintf("%d", MIN_COLS);
  EditCols->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Speed button for maximum number of columns.
// Tested 4/8/1999 KJS
void __fastcall TnnData::SpeedMaxColsClick(TObject *Sender)
{
  ScrollCols->Position=MAX_COLS;
  holding=""; holding.sprintf("%d", MAX_COLS);
  EditCols->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Waits for return key and then changes the position of the scroll bar.
// Tested 4/8/1999 KJS
void __fastcall TnnData::EditColsKeyPress(TObject *Sender, char &Key)
{
  // Check to see if it's the return key.
  if (Key==VK_RETURN)
  { // Eliminate keypress return to windows.
    Key=0;
    // Get cols and check within limits.
    noCols=EditCols->Text.ToInt();
    if (noCols<MIN_COLS) noCols=MIN_COLS;
    if (noCols>MAX_COLS) noCols=MAX_COLS;
    // Set scroll bar.
    ScrollCols->Position=noCols;
    holding=""; holding.sprintf("%d", noCols);
    EditCols->SetTextBuf(holding.c_str());
  }
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Change the scroll bar position on de-focus.
// Tested 4/8/1999 KJS
void __fastcall TnnData::EditColsExit(TObject *Sender)
{
  // Get cols and check within limits.
  noCols=EditCols->Text.ToInt();
  if (noCols<MIN_COLS) noCols=MIN_COLS;
  if (noCols>MAX_COLS) noCols=MAX_COLS;

  // Set scroll bar.
  ScrollCols->Position=noCols;
  holding=""; holding.sprintf("%d", noCols);
  EditCols->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------
```

8.3 NNet.h

```
//---------------------------------------------------------------------------
// Neural network program.
//---------------------------------------------------------------------------
#ifndef NNetH
#define NNetH

//---------------------------------------------------------------------------
// Includes.
```

```
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Grids.hpp>
#include "rmrandom.h"
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>

//---------------------------------------------------------------------------
// Defines for Neural Setup.

// Program textual defines.
#define ITERATION_TEXT     "Iteration: %d"
#define INPUT_PLANE_TEXT   "Currently showing the neural input plane: %0.1f to %0.1f Volts at %d
bit(s)."
#define OUTPUT_PLANE_TEXT  "Currently showing the neural output plane: %0.1f to %0.1f Volts at %d
bit(s)."
#define MEMORY_PLANE_TEXT  "Currently showing the neural memory plane at double resolution (64
bits)."
#define REQUEST_PLANE_TEXT "Currently showing the neural request plane: %0.1f to %0.1f at %d
stage(s)."
#define ERR_NOT_OP         "Can only edit values in the request plane."
#define ERR_ITERATION      "Cannot edit values during simulation run-time."
#define REQUEST_CHANGE     "Request limits have changed.\nResetting all values in request plane."
#define INIT_SYS           "Initialising system..."
#define NEU                "Neurons"
#define COMP               "Complete."
#define REDRAW             "Redrawing window..."
#define LOWER_TRI          "Creating lower triangular matrix..."
#define MAX_IT_TEXT        "Maximum number of iterations reached."
#define MAX_TOEXCEED_TEXT  "Maximum number of iterations would be exceeded."
#define PROG_NAME          "Neural Network V2.0"
#define CALC               "Calculating..."
#define RUN_CONV           "Network has been run and has converged.\nPlease use 'Reset Network' to
continue."
#define MAT_FILL           "Lower triangular matrix created using request value in (1,1)"
#define ERR_CLOSE          "File I/O error: Cannot close file."
#define ERR_CREATE         "File I/O error: Cannot create file."
#define ERR_WRI            "File I/O error: Cannot write information."
#define NO_FREE_NAMES      "Cannot find a free file name.\nPlease delete some of the NNetXXXX.out
files in this directory."
#define DEFAULT_NAME       "NNet%04d.out"
#define NO_FILE            "No file open."

// Enum defines for current plane.  Do not alter.
#define INPUT_PLANE          0
#define OUTPUT_PLANE         1
#define MEMORY_PLANE         2
#define REQUEST_PLANE        3
#define IT_PANEL             0
#define FILE_PANEL           1
#define GEN_PANEL            2
#define MAX_NO_FILES     10000
#define MAX_ITERATIONS    2000  // Prevents an accidental exp overflow.

//---------------------------------------------------------------------------
// Object definition.
class TMain : public TForm
{
__published:     // IDE-managed Components
  TStringGrid *StateGrid;
  TLabel *TableTitle;
  TStatusBar *StatusBar;
  TCheckBox *LogCheckBox;
  TGroupBox *NetGroupBox;
  TButton *Iterate;
  TButton *Iterate10;
  TButton *Run;
  TGroupBox *ControlGroupBox;
  TButton *LowerTriang;
  TButton *ResetAll;
  TButton *SetupButton;
  TButton *QuitButton;
  TButton *OutputPlane;
  TButton *RequestPlane;
  TButton *MemoryPlane;
  TButton *InputPlane;
  void __fastcall QuitButtonClick(TObject *Sender);
  void __fastcall OutputPlaneClick(TObject *Sender);
  void __fastcall ResetAllClick(TObject *Sender);
  void __fastcall IterateClick(TObject *Sender);
  void __fastcall Iterate10Click(TObject *Sender);
  void __fastcall StateGridDblClick(TObject *Sender);
  void __fastcall RunClick(TObject *Sender);
  void __fastcall LowerTriangClick(TObject *Sender);
  void __fastcall SetupButtonClick(TObject *Sender);
  void __fastcall FormActivate(TObject *Sender);
```

```
  void __fastcall RequestPlaneClick(TObject *Sender);
  void __fastcall MemoryPlaneClick(TObject *Sender);
  void __fastcall InputPlaneClick(TObject *Sender);
private: // User declarations
  // Object variables.
  int showTable;
  int iteration;
  bool converged;
  CRand rand;
  AnsiString holding;
  HANDLE file;
public:          // User declarations
  __fastcall TMain(TComponent* Owner);
  void __fastcall UpdateAll();
  // Use inline for speed.
  double __inline Quantise(double min, double max, double bits, double value);
  void __inline OpticalSystem();
  void __inline NeuralMemoryToOutput();
  void __inline NeuralInputToMemory();
  // Save to disk code.
  void __fastcall OpenDiskFile();
  void __fastcall CloseDiskFile();
  __fastcall ~TMain();
  // Disable all the buttons which should be unavailable while iterating.
  void __fastcall DisableButtons();
  void __fastcall EnableButtons();
};
//---------------------------------------------------------------------------
extern PACKAGE TMain *Main;
//---------------------------------------------------------------------------
#endif
```

8.4 NNet.cpp

```
//---------------------------------------------------------------------------
// Neural network program.
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------
// Includes.
#include "nnet.h"
#include "value.h"
#include "setup.h"
#include "data.h"
#include <values.h>
#include <math.h>
#include <stdio.h>
#include <iostream.h>

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMain *Main;
//---------------------------------------------------------------------------
// Program constructor.  Initialises the system.
// Tested 5/7/1999 KJS
__fastcall TMain::TMain(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Set up network size.
// Tested 10/8/1999 KJS
void __fastcall TMain::FormActivate(TObject *Sender)
{
  // Get network size for initialisation.
  if (nnData->ShowModal()!=IDOK) Close(); // Anything else means close.

  // Allocate memory
  nnData->SetupArrays();

  // Clear other variables.
  iteration=0;
  converged=false;
  file=INVALID_HANDLE_VALUE;

  // Set up program variables.
  SetupBox->InitialiseDefaults();

  // Initialise the status grid to the correct size.
  StateGrid->RowCount=nnData->noRows+1;
```

```
  StateGrid->ColCount=nnData->noCols+1;

  // Initialise neural elements.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
    { // All elements to zero.
      nnData->nOutput[m][n]=nnData->op_MinV;
      nnData->nInput[m][n]=nnData->ip_MinV;
      nnData->nMemory[m][n]=0;
      nnData->nRequest[m][n]=nnData->rq_Min;
    }

  // Print information in the fixed boxes.
  for (int pos=1; nnData->noCols>=pos; pos++)
  { // Print column markers
    holding="";  // Clear the string.
    holding.sprintf("%i", pos);
    StateGrid->Cells[pos][0]=holding.c_str();
  }
  for (int pos=1; nnData->noRows>=pos; pos++)
  { // Print row markers
    holding="";  // Clear the string.
    holding.sprintf("%i", pos);
    StateGrid->Cells[0][pos]=holding.c_str();
  }
  // Write that they are neurons.
  StateGrid->Cells[0][0]=NEU;

  // Update table: always show the request plane first.
  showTable=REQUEST_PLANE;
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Exit the program.
// Tested 5/7/1999 KJS
void __fastcall TMain::QuitButtonClick(TObject *Sender)
{
  // Exit the program.
  Close();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Updates the tables with appropriate information.
// Tested 10/8/1999 KJS
void __fastcall TMain::UpdateAll()
{
  // Tell the user that the window is being redrawn.
  StatusBar->Panels->Items[GEN_PANEL]->Text=REDRAW;
  StatusBar->Refresh();

  // Print the information in selected table.
  if (showTable==INPUT_PLANE)
  { // Change the status text.
    holding="";  // Clear the string.
    holding.sprintf(INPUT_PLANE_TEXT, nnData->ip_MinV, nnData->ip_MaxV, nnData->ip_Res);
    TableTitle->SetTextBuf(holding.c_str());
    for (int m=0; nnData->noRows>m; m++)
      for (int n=0; nnData->noCols>n; n++)
      { // Print column markers.
        holding="";  // Clear the string.
        holding.sprintf("%0.3f", nnData->nInput[m][n]);
        StateGrid->Cells[(n+1)][(m+1)]=holding.c_str();
      }
  }

  // Print the information in selected table.
  if (showTable==MEMORY_PLANE)
  { // Change the status text.
    TableTitle->SetTextBuf(MEMORY_PLANE_TEXT);
    for (int m=0; nnData->noRows>m; m++)
      for (int n=0; nnData->noCols>n; n++)
      { // Print column markers.
        holding="";  // Clear the string.
        holding.sprintf("%0.3f", nnData->nMemory[m][n]);
        StateGrid->Cells[(n+1)][(m+1)]=holding.c_str();
      }
  }

  // Print the information in selected table.
  if (showTable==REQUEST_PLANE)
  { // Change the status text.
    holding="";  // Clear the string.
    holding.sprintf(REQUEST_PLANE_TEXT, nnData->rq_Min, nnData->rq_Max, nnData->rq_Stages);
    TableTitle->SetTextBuf(holding.c_str());
    for (int m=0; nnData->noRows>m; m++)
```

```
      for (int n=0; nnData->noCols>n; n++)
      { // Print column markers.
        holding="";  // Clear the string.
        holding.sprintf("%0.3f", nnData->nRequest[m][n]);
        StateGrid->Cells[(n+1)][(m+1)]=holding.c_str();
      }
  }

  // Print the information in selected table.
  if (showTable==OUTPUT_PLANE)
  { // Change the status text.
    holding="";  // Clear the string.
    holding.sprintf(OUTPUT_PLANE_TEXT, nnData->op_MinV, nnData->op_MaxV, nnData->op_Res);
    TableTitle->SetTextBuf(holding.c_str());
    for (int m=0; nnData->noRows>m; m++)
      for (int n=0; nnData->noCols>n; n++)
      { // Print column markers.
        holding="";  // Clear the string.
        holding.sprintf("%0.3f", nnData->nOutput[m][n]);
        StateGrid->Cells[(n+1)][(m+1)]=holding.c_str();
      }
  }

  // Print current iteration
  holding="";  // Clear the string.
  holding.sprintf(ITERATION_TEXT, iteration);
  StatusBar->Panels->Items[IT_PANEL]->Text=holding.c_str();
  StatusBar->Panels->Items[GEN_PANEL]->Text=COMP;
  StatusBar->Refresh();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Switches to the output plane view.
// Tested 10/8/1999 KJS
void __fastcall TMain::OutputPlaneClick(TObject *Sender)
{
  // Change the table selector.
  showTable=OUTPUT_PLANE;
  // Update the window.
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Changes to the request plane window.
// Tested 10/8/1999 KJS
void __fastcall TMain::RequestPlaneClick(TObject *Sender)
{
  // Change to the request plane and updates the window.
  showTable=REQUEST_PLANE;
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Changes to the memory plane window.
// Tested 10/8/1999 KJS
void __fastcall TMain::MemoryPlaneClick(TObject *Sender)
{
  // Change to the memory plane and updates the window.
  showTable=MEMORY_PLANE;
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Changes to the input plane window.
// Tested 10/8/1999 KJS
void __fastcall TMain::InputPlaneClick(TObject *Sender)
{
  // Change to the memory plane and updates the window.
  showTable=INPUT_PLANE;
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Disables all the buttons which should be unavailable while iterating.
// Tested 14/9/99 KJS.
void __fastcall TMain::DisableButtons()
{
  LowerTriang->Enabled=false;
  SetupBox->OKBtn->Enabled=false;
  SetupBox->NoAlter->Visible=true;
  SetupBox->Default->Visible=false;
  LogCheckBox->Enabled=false;
  ResetAll->Enabled=true;
```

```
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Enables all the buttons which should be unavailable while iterating.
// Tested 14/9/99 KJS.
void __fastcall TMain::EnableButtons()
{
  LowerTriang->Enabled=true;
  SetupBox->OKBtn->Enabled=true;
  SetupBox->NoAlter->Visible=false;
  SetupBox->Default->Visible=true;
  LogCheckBox->Enabled=true;
  ResetAll->Enabled=false;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Resets all values ready to start again.
// Tested 15/8/1999 KJS Altered
void __fastcall TMain::ResetAllClick(TObject *Sender)
{
  // Reset iterations.
  iteration=0;
  converged=false;

  // Clear neural elements.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
    { // All but the request plane need be cleared.
      nnData->nOutput[m][n]=nnData->op_MinV;
      nnData->nInput[m][n]=nnData->ip_MinV;
      nnData->nMemory[m][n]=0;
    }

  // Close an open disk file.
  CloseDiskFile();

  // Redraw all views.
  UpdateAll();
  // Enable buttons.
  EnableButtons();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Quantises a system to the nearest value.  Rounds down.
// Tested 29/6/1999 KJS
double __inline TMain::Quantise(double min, double max, double bits, double value)
{
  // Variable to return.
  double toreturn;
  double aboveDelta;
  double belowDelta;
  double increment;

  // See if the value is below or equal to the minimum.
  if (min>=value) return min; // Clip the value.
  // See if the value is above or equal to the maximum.
  if (value>=max) return max; // Clip the value.

  // Calculate the increment.  Note we take zero into account.
  increment=(max-min)/(pow(2, bits)-1);

  // OK, find out the value just above by continually incrementing from min.
  toreturn=min;
  while (value>=toreturn)
    toreturn+=increment;

  // Calculate the difference between the two values.
  aboveDelta=fabs((toreturn-value));
  belowDelta=fabs(((toreturn-increment)-value));

  // Return the appropriate value by rounding down.
  if (belowDelta>aboveDelta) return toreturn;
  else return (toreturn-increment);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Takes the values in nnData->nOutput and calculates the new values in neuralInput.
// Tested 30/6/1999 KJS
void __inline TMain::OpticalSystem()
{
  // Local variables.
  double gradient, constant;

  // Calculate graph gradient.
```

```
  gradient=(nnData->os_MaxOO-nnData->os_MinOO)/(nnData->op_MaxV-nnData->op_MinV);
  constant=nnData->os_MinOO-(gradient*nnData->op_MinV);
  // Convert neural output to optical powers.  Store in the tempMatrix.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->tempMatrix[m][n]=(gradient*nnData->nOutput[m][n])+constant;

  // Calculate DOE fanout and store in tempMatrix.
  // Presuming 1:1 mapping and equal power in each spot.
  // Calculate the power in each spot from a specific VCSEL.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->tempMatrix[m][n]=(nnData->tempMatrix[m][n]*(nnData->os_Trans/100.0))/nnData-
>os_Spots;

  // Clear the neuralInput matrix before totalling.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->nInput[m][n]=0.0;
  // Now total the incident power on each detector and store in the neuralInput matrix.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
    { // Sum all elements in column first.
      for (int y=0; nnData->noRows>y; y++)
        if (y!=m) nnData->nInput[m][n]+=nnData->tempMatrix[y][n];
      // Sum all elements in row next.
      for (int x=0; nnData->noCols>x; x++)
        if (x!=n) nnData->nInput[m][n]+=nnData->tempMatrix[m][x];
    }

  // We know the upper and lower power responses for the detector, so calculate voltage.
  // Calculate graph gradient.
  gradient=(nnData->ip_MaxV-nnData->ip_MinV)/(nnData->os_MaxDet-nnData->os_MinDet);
  constant=nnData->ip_MinV-(gradient*nnData->os_MinDet);
  // Convert neural output to optical powers.  Store in the tempMatrix.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->nInput[m][n]=(gradient*nnData->nInput[m][n])+constant;

  // Now add a set voltage noise.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->nInput[m][n]=nnData->nInput[m][n]+rand.GetRandom(nnData->ip_MinusNoise, nnData-
>ip_PlusNoise);

  // Quantise input voltage.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->nInput[m][n]=Quantise(nnData->ip_MinV, nnData->ip_MaxV, nnData->ip_Res, nnData-
>nInput[m][n]);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Takes the neural memory and calculates the neural output.
// Tested 6/7/1999 KJS
void __inline TMain::NeuralMemoryToOutput()
{
  // Local variables.
  int stableNeurons=0;
  DWORD written;  // Flags number of bytes written to disk.
  holding=""; // For file I/O.

  // Put the memory through the neuron and multiply by request matrix.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
    { // Sigmoid neural response.
      nnData->nOutput[m][n]=(nnData->nRequest[m][n]*(nnData->op_MaxV-nnData->op_MinV))
        /(1.0+exp((-1.0*nnData->nn_beta*nnData->nMemory[m][n])));
    }

  // Print the iteration number in preparation for writing to file.
  if (file!=INVALID_HANDLE_VALUE)
    holding.sprintf("%d", iteration);

  // Quantise values in neural output plane and flag convergence.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
    { // Flag number of converged neurons before quantisation.
      if (((nnData->op_MinV+nnData->op_Conv)>nnData->nOutput[m][n]) ||
          (nnData->nOutput[m][n]>(nnData->op_MaxV-nnData->op_Conv))) stableNeurons++;
      // Quantise.
      nnData->nOutput[m][n]=Quantise(nnData->op_MinV, nnData->op_MaxV, nnData->op_Res, nnData-
>nOutput[m][n]);
      // Write the current value to file.
      if (file!=INVALID_HANDLE_VALUE)
        holding.sprintf(", %0.3f", nnData->nOutput[m][n]);
    }
```

```
  if (file!=INVALID_HANDLE_VALUE)
  { // Write a new line character.
    holding.sprintf("\n");
    // Commit the information.
    WriteFile(file, holding.c_str(), holding.Length(), &written, NULL);
    // Check that it was written correctly.
    if (written!=(DWORD)holding.Length()) Application->MessageBox(ERR_WRI, PROG_NAME,
MB_ICONERROR);
  }

  // Flag system convergence if over minimum iterations.
  if ((stableNeurons==(nnData->noCols*nnData->noRows)) && (iteration>nnData->nn_MinIter))
converged=true;

  // Return.
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Takes the neural input and calculates the neural memory.
// Tested 5/7/1999 KJS
void __inline TMain::NeuralInputToMemory()
{
  // Convert neural input to memory values.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      nnData->nMemory[m][n]=nnData->nMemory[m][n]+(nnData->nn_dt/nnData->nn_Tlpf)
                        *(-1.0*nnData->nn_A*nnData->nInput[m][n]+nnData->nn_Bias);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Handles a single iteration.
// Tested 10/8/1999 KJS
void __fastcall TMain::IterateClick(TObject *Sender)
{
  // Point out error if iteration limit will be exceeded.
  if ((iteration+1)>MAX_ITERATIONS)
  { // Report error and return.
    Application->MessageBox(MAX_TOEXCEED_TEXT, PROG_NAME, NULL);
    return;
  }

  // Indicate calculating.
  StatusBar->Panels->Items[GEN_PANEL]->Text=CALC;
  StatusBar->Refresh();
  // Disable buttons.
  DisableButtons();
  // Open log file if requested.
  if ((iteration==0) && (LogCheckBox->Checked==true)) OpenDiskFile();
  // Increment the number of iterations.
  iteration++;
  // Process neural memory to optical output.
  NeuralMemoryToOutput();
  // Process the optical system.
  OpticalSystem();
  // Process neural input to neural memory.
  NeuralInputToMemory();
  // Update all outputs.
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Calls 10 iterations.
// Tested 10/8/1999 KJS
void __fastcall TMain::Iterate10Click(TObject *Sender)
{
  // Point out error if iteration limit will be exceeded.
  if ((iteration+10)>MAX_ITERATIONS)
  { // Report error and return.
    Application->MessageBox(MAX_TOEXCEED_TEXT, PROG_NAME, NULL);
    return;
  }

  // Indicate the program is working.
  StatusBar->Panels->Items[GEN_PANEL]->Text=CALC;
  StatusBar->Refresh();

  // Disable buttons.
  DisableButtons();

  // Open log file if requested.
  if ((iteration==0) && (LogCheckBox->Checked==true)) OpenDiskFile();

  // Repeat.
  for (int x=0; 10>x; x++)
```

```
  { // Increment the number of iterations.
    iteration++;
    // Process neural memory to optical output.
    NeuralMemoryToOutput();
    // Process the optical system.
    OpticalSystem();
    // Process neural input to neural memory.
    NeuralInputToMemory();
    // Indicate current iteration.
    holding="";  // Clear the string.
    holding.sprintf(ITERATION_TEXT, iteration);
    StatusBar->Panels->Items[IT_PANEL]->Text=holding.c_str();
    StatusBar->Refresh();
  }

  // Update all outputs.
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Keeps running until the converged variable is set by NeuralMemoryToOutput().
// Tested 10/8/1999 KJS
void __fastcall TMain::RunClick(TObject *Sender)
{
  // Check network has not already been run.
  if (converged)
  { // State it has converged and return.
    Application->MessageBox(RUN_CONV, PROG_NAME, MB_ICONINFORMATION);
    return;
  }

  // Indicate the program is working.
  StatusBar->Panels->Items[GEN_PANEL]->Text=CALC;
  StatusBar->Refresh();

  // Disable buttons.
  DisableButtons();

  // Open log file if requested.
  if ((iteration==0) && (LogCheckBox->Checked==true)) OpenDiskFile();

  // Repeat while still not converged.
  while ((!converged) && (MAX_ITERATIONS>iteration))
  { // Increment the number of iterations.
    iteration++;
    // Process neural memory to optical output.
    NeuralMemoryToOutput();
    // Process the optical system.
    OpticalSystem();
    // Process neural input to neural memory.
    NeuralInputToMemory();
    // Indicate current iteration.
    holding="";  // Clear the string.
    holding.sprintf(ITERATION_TEXT, iteration);
    StatusBar->Panels->Items[IT_PANEL]->Text=holding.c_str();
    StatusBar->Refresh();
  }

  // Update all outputs.
  UpdateAll();

  // Point out error if iteration limit exceeded.
  if (iteration>=MAX_ITERATIONS) Application->MessageBox(MAX_IT_TEXT, PROG_NAME, NULL);

  // Tell the user what's happening.
  StatusBar->Panels->Items[GEN_PANEL]->Text=COMP;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Allow the user to edit a request value if the request plane is selected.
// Tested 6/7/1999 KJS
void __fastcall TMain::StateGridDblClick(TObject *Sender)
{
  // Local variables.
  TGridRect selected;

  // Cannot edit anything but the request plane.
  if (showTable==REQUEST_PLANE)
  { // Ensure that this is the first iteration.
    if (iteration!=0)
    { // Cannot alter requests during run time.
      Application->MessageBox(ERR_ITERATION, PROG_NAME, MB_ICONEXCLAMATION);
      return;
    }

    // First get the selected position.
```

```
    selected=StateGrid->Selection;
    // Set up the valuebox.
    ValueBox->TrackBar->Min=0;
    ValueBox->TrackBar->Max=nnData->rq_Stages-1;
    ValueBox->increment=(nnData->rq_Max-nnData->rq_Min)/(double)(nnData->rq_Stages-1);
    ValueBox->TrackBar->Frequency=1;

    // Show the box.  If OK then change the value.
    if (ValueBox->ShowModal()==mrOk)
      nnData->nRequest[(selected.Top-1)][(selected.Left-1)]=ValueBox->value;
    // Update the system.
    UpdateAll();
  }
  else
    // Can only edit output plane.
    Application->MessageBox(ERR_NOT_OP, PROG_NAME, MB_ICONEXCLAMATION);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Creates a lower triangular request matrix - best for testing.
// Tested 10/8/1999 KJS
void __fastcall TMain::LowerTriangClick(TObject *Sender)
{
  // Local variable.
  int number=0;

  // Status bar updates.
  StatusBar->Panels->Items[GEN_PANEL]->Text=LOWER_TRI;
  StatusBar->Refresh();

  // Fill with triangular matrix.
  for (int n=0; nnData->noCols>n; n++)
  {
    for (int m=0; nnData->noRows>m; m++)
      if ((m+1)>number) nnData->nRequest[m][n]=nnData->nRequest[0][0];
    number++;
  }

  // Update all outputs.
  UpdateAll();

  // Point out error if iteration limit exceeded.
  Application->MessageBox(MAT_FILL, PROG_NAME, MB_ICONINFORMATION);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Handles parameter setup in system.
// Tested 14/9/1999 KJS
void __fastcall TMain::SetupButtonClick(TObject *Sender)
{
  // Sets the string values from data object.
  SetupBox->SetStringValues();
  // Set no change flag.
  SetupBox->rqChange=false;

  // Execute setup box.
  if (SetupBox->ShowModal()==mrOk)
  { // Read back the values.
    SetupBox->SetDoubleValues();
    if (SetupBox->rqChange==true)
    { // Reset the request plane.
      Application->MessageBox(REQUEST_CHANGE, PROG_NAME, MB_ICONINFORMATION);
      // Initialise neural elements.
      for (int m=0; nnData->noRows>m; m++)
        for (int n=0; nnData->noCols>n; n++)
          nnData->nRequest[m][n]=nnData->rq_Min;
      // Reset valuebox slider.
      ValueBox->TrackBar->Position=0;
    }
  }

  // Update all values.
  UpdateAll();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Opens the disk file with default name.
// Tested 15/9/1999 KJS
void __fastcall TMain::OpenDiskFile()
{
  // Local variable for writing.
  DWORD written;
  int current=0;

  // Close any open files first.
```

```
  CloseDiskFile();

  // Search for a free file name and write.
  do
  { // Select name.
    holding="";
    holding.sprintf(DEFAULT_NAME, current++);
    // Open and close file to see if it exists.
    file=CreateFile(holding.c_str(), GENERIC_READ, NULL, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (file!=INVALID_HANDLE_VALUE) CloseHandle(file); // If it does then close it again.
  }
  while ((file!=INVALID_HANDLE_VALUE) && (MAX_NO_FILES>current));

  // If current has reached MAX_NO_FILES then there are a lot of files in this directory!
  if (current==MAX_NO_FILES)
    { // Indicate error and return.
    Application->MessageBox(NO_FREE_NAMES, PROG_NAME, MB_ICONERROR);
    return;
  }

  // Create a new file.
  file=CreateFile(holding.c_str(), GENERIC_WRITE, NULL, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
  if (file==INVALID_HANDLE_VALUE) // Check for errors.
  { // Indicate error and return.
    Application->MessageBox(ERR_CREATE, PROG_NAME, MB_ICONERROR);
    return;
  }
  // Display file name in file box.
  StatusBar->Panels->Items[FILE_PANEL]->Text=holding.c_str();
  StatusBar->Refresh();

  // Write program information.
  holding="";
  holding.sprintf("%s Log File\n\n", PROG_NAME);
  holding.sprintf("Network size: %d rows and %d columns.\n", nnData->noRows, nnData->noCols);
  holding.sprintf("Network parameters: A=%0.3f, Bias=%0.3f, Beta=%0.3f, dt=%0.3f, Tlpf=%0.3f with
a minimum of %d iterations.\n",
                  nnData->nn_A, nnData->nn_Bias, nnData->nn_beta,
                  nnData->nn_dt, nnData->nn_Tlpf, nnData->nn_MinIter);
  holding.sprintf("Request values from %0.3f to %0.3f using %d stages.\n",
                  nnData->rq_Min, nnData->rq_Max, nnData->rq_Stages);
  holding.sprintf("Output from %0.3f to %0.3f Volts using %d bit(s) resolution.\n",
                  nnData->op_MinV, nnData->op_MaxV, nnData->op_Res);
  holding.sprintf("Neural system has a convergence margin of %0.3f.\n",
                  nnData->op_Conv);
  holding.sprintf("VCSEL output ranges between %0.6f and %0.6f Watts of optical power.\n",
                  nnData->os_MinOO, nnData->os_MaxOO);
  holding.sprintf("DOE creates a symmetrical cross with %0.3f spots with %0.3f% transmission.\n",
                  nnData->os_Spots, nnData->os_Trans);
  holding.sprintf("The detector has a response from %0.6f to %0.6f Watts of incident optical
power.\n",
                  nnData->os_MinDet, nnData->os_MaxDet);
  holding.sprintf("Detector input from %0.3f to %0.3f Volts using %d bit(s) resolution.\n",
                  nnData->ip_MinV, nnData->ip_MaxV, nnData->ip_Res);
  holding.sprintf("Maximum input voltage noise from %0.3f to %0.3f Volts.\n\n",
                  nnData->ip_MinusNoise, nnData->ip_PlusNoise);

  // Write the request matrix.
  holding.sprintf("R");
  // Write the request matrix value.
  for (int m=0; nnData->noRows>m; m++)
    for (int n=0; nnData->noCols>n; n++)
      holding.sprintf(", %0.3f", nnData->nRequest[m][n]);
  // Write a new line character.
  holding.sprintf("\n");

  // Write the string.
  WriteFile(file, holding.c_str(), holding.Length(), &written, NULL);
  // Check for write error.
  if (written!=(DWORD)holding.Length()) Application->MessageBox(ERR_WRI, PROG_NAME, MB_ICONERROR);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Closes the file pointed to by 'file' and reports any errors.
// Tested 15/9/1999 KJS
void __fastcall TMain::CloseDiskFile()
{ // Check that the file exists.
  if (file!=INVALID_HANDLE_VALUE)
  { // Test for success.
    if (CloseHandle(file)==0) // Cannot close file.
      Application->MessageBox(ERR_CLOSE, PROG_NAME, MB_ICONERROR);
    // Update status bar.
    StatusBar->Panels->Items[FILE_PANEL]->Text=NO_FILE;
    StatusBar->Refresh();
  }
```

```
  // Reset file handle.
  file=INVALID_HANDLE_VALUE;
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Ensure open file is closed.
// Tested 6/7/1999 KJS
__fastcall TMain::~TMain()
{
   // Close the disk file.
   CloseDiskFile();
}
//---------------------------------------------------------------------------
```

8.5 NNetwork.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("NNetwork.res");
USEFORM("NNet.cpp", Main);
USEFORM("Value.cpp", ValueBox);
USEFORM("Setup.cpp", SetupBox);
USEFORM("Data.cpp", nnData);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  try
  {
    Application->Initialize();
    Application->Title = "Neural Network";
    Application->HelpFile = "";
     Application->CreateForm(__classid(TMain), &Main);
     Application->CreateForm(__classid(TSetupBox), &SetupBox);
     Application->CreateForm(__classid(TValueBox), &ValueBox);
     Application->CreateForm(__classid(TnnData), &nnData);
     Application->Run();
  }
  catch (Exception &exception)
  {
    Application->ShowException(&exception);
  }
  return 0;
}
//---------------------------------------------------------------------------
```

8.6 Rmrandom.h

```
//---------------------------------------------------------------------------
// Module: rmrandom.h
// Purpose: Interface and implementation of a random number generating class.
//---------------------------------------------------------------------------
#ifndef __RMRANDOM_H__
#define __RMRANDOM_H__
#include <cstdlib>  // For srand(), rand()
#include <ctime>    // For time()

//---------------------------------------------------------------------------
// CRand class
class CRand
{
public:
  // Seed the random number generator
  // during construction.
  // First constructor (default) uses time for random.
  CRand() { srand((unsigned)time(0)); }
  // Second constructor uses parameter as seed.
  CRand(unsigned seed) { srand(seed); }

  // Generate a random number within defined limits.
  double GetRandom(double min, double max)
  {
    // Generate number and return.
    return ((((double)rand()/(double)RAND_MAX)*(max-min))+min);
  }
};

#endif
```

8.7 Setup.h

```
//---------------------------------------------------------------------------
#ifndef SetupH
#define SetupH
//---------------------------------------------------------------------------
#include <vcl\ExtCtrls.hpp>
#include <vcl\ComCtrls.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\Classes.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Windows.hpp>
#include <vcl\System.hpp>
//---------------------------------------------------------------------------
class TSetupBox : public TForm
{
__published:
        TPanel *Panel1;
        TPanel *Panel2;
  TPageControl *Page;
  TTabSheet *NetworkParams;
  TTabSheet *Request;
  TTabSheet *Output;
        TButton *OKBtn;
        TButton *CancelBtn;
  TEdit *Edit_nn_A;
  TEdit *Edit_nn_Bias;
  TEdit *Edit_nn_Beta;
  TEdit *Edit_nn_dt;
  TEdit *Edit_nn_Tlpf;
  TLabel *nn3;
  TLabel *nn4;
  TLabel *nn5;
  TLabel *nn6;
  TTabSheet *Optical;
  TTabSheet *Input;
  TButton *Default;
  TLabel *nn2;
  TLabel *nn1;
  TLabel *rq1;
  TEdit *Edit_rq_Min;
  TEdit *Edit_rq_Stages;
  TEdit *Edit_rq_Max;
  TEdit *Edit_nn_MinIter;
  TLabel *Label1;
  TLabel *rq_2;
  TLabel *rq_3;
  TLabel *rq_4;
  TEdit *Edit_op_MinV;
  TEdit *Edit_op_MaxV;
  TEdit *Edit_op_Res;
  TEdit *Edit_op_Conv;
  TEdit *Edit_os_MinOO;
  TEdit *Edit_os_MaxOO;
  TEdit *Edit_os_Spots;
  TEdit *Edit_os_Trans;
  TEdit *Edit_os_MinDet;
  TEdit *Edit_os_MaxDet;
  TEdit *Edit_ip_MinV;
  TEdit *Edit_ip_MaxV;
  TEdit *Edit_ip_Res;
  TEdit *Edit_ip_MinusNoise;
  TEdit *Edit_ip_PlusNoise;
  TLabel *op_1;
  TLabel *op_2;
  TLabel *op_3;
  TLabel *op_4;
  TLabel *op_5;
  TLabel *os_1;
  TLabel *os_2;
  TLabel *os_3;
  TLabel *os_4;
  TLabel *os_5;
  TLabel *os_6;
  TLabel *os_7;
  TLabel *ip_1;
  TLabel *ip_2;
  TLabel *ip_3;
  TLabel *ip_4;
  TLabel *ip_5;
  TLabel *ip_6;
```

```
  TLabel *NoAlter;
  void __fastcall DefaultClick(TObject *Sender);
  void __fastcall RequestChange(TObject *Sender);
private:
public:
  // Constructor.
        virtual __fastcall TSetupBox(TComponent* AOwner);
  // Initialise.
  void __fastcall InitialiseDefaults();
  // Copy values.
  void __fastcall SetDoubleValues();
  // Set text strings.
  void __fastcall SetStringValues();
  // Temp variables.
  AnsiString holding;
  bool rqChange;
};
//---------------------------------------------------------------------------
extern PACKAGE TSetupBox *SetupBox;
//---------------------------------------------------------------------------
#endif
```

8.8 Setup.cpp

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Setup.h"
#include "Data.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TSetupBox *SetupBox;
//---------------------------------------------------------------------
__fastcall TSetupBox::TSetupBox(TComponent* AOwner)
        : TForm(AOwner)
{
}
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Resets all defaults.
// Tested 9/8/1999 KJS
void __fastcall TSetupBox::InitialiseDefaults()
{
  // Set defaults for all strings.
  // First for Neural Network Tab
  nnData->nn_A=DEF_A;
  nnData->nn_Bias=DEF_BIAS;
  nnData->nn_beta=DEF_BETA;
  nnData->nn_dt=DEF_DT;
  nnData->nn_Tlpf=DEF_TLPF;
  nnData->nn_MinIter=DEF_MINITER;

  // Request Tab.
  nnData->rq_Min=DEF_MIN_REQUEST;
  nnData->rq_Max=DEF_MAX_REQUEST;
  nnData->rq_Stages=DEF_REQUEST_STAGES;

  // Output Tab.
  nnData->op_MinV=DEF_MIN_OUTPUT;
  nnData->op_MaxV=DEF_MAX_OUTPUT;
  nnData->op_Res=DEF_OUTPUT_BIT_RES;
  nnData->op_Conv=DEF_CONV_MARGIN;

  // Optical Tab.
  nnData->os_MinOO=DEF_MIN_OPTICAL;
  nnData->os_MaxOO=DEF_MAX_OPTICAL;
  nnData->os_Spots=DEF_DOE_SPOTS;
  nnData->os_Trans=DEF_DOE_EFFICIENCY;
  nnData->os_MinDet=DEF_MIN_DET;
  nnData->os_MaxDet=DEF_MAX_DET;

  // Input Tab.
  nnData->ip_MinV=DEF_MIN_INPUT;
  nnData->ip_MaxV=DEF_MAX_INPUT;
  nnData->ip_Res=DEF_INPUT_BIT_RES;
  nnData->ip_MinusNoise=DEF_VMINUS_NOISE;
  nnData->ip_PlusNoise=DEF_VPLUS_NOISE;
}
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Set double values to currently held text strings.
// Tested 9/8/1999 KJS
void __fastcall TSetupBox::SetDoubleValues()
```

```
{
  // Turn the text strings into double values.
  nnData->nn_A=Edit_nn_A->Text.ToDouble();
  nnData->nn_Bias=Edit_nn_Bias->Text.ToDouble();
  nnData->nn_beta=Edit_nn_Beta->Text.ToDouble();
  nnData->nn_dt=Edit_nn_dt->Text.ToDouble();
  nnData->nn_Tlpf=Edit_nn_Tlpf->Text.ToDouble();
  nnData->nn_MinIter=Edit_nn_MinIter->Text.ToInt();

  // Request tab.
  nnData->rq_Min=Edit_rq_Min->Text.ToDouble();
  nnData->rq_Max=Edit_rq_Max->Text.ToDouble();
  nnData->rq_Stages=Edit_rq_Stages->Text.ToInt();

  // Output Tab.
  nnData->op_MinV=Edit_op_MinV->Text.ToDouble();
  nnData->op_MaxV=Edit_op_MaxV->Text.ToDouble();
  nnData->op_Res=Edit_op_Res->Text.ToInt();
  nnData->op_Conv=Edit_op_Conv->Text.ToDouble();

  // Optical Tab.
  nnData->os_MinOO=Edit_os_MinOO->Text.ToDouble();
  nnData->os_MaxOO=Edit_os_MaxOO->Text.ToDouble();
  nnData->os_Spots=Edit_os_Spots->Text.ToDouble();
  nnData->os_Trans=Edit_os_Trans->Text.ToDouble();
  nnData->os_MinDet=Edit_os_MinDet->Text.ToDouble();
  nnData->os_MaxDet=Edit_os_MaxDet->Text.ToDouble();

  // Input Tab.
  nnData->ip_MinV=Edit_ip_MinV->Text.ToDouble();
  nnData->ip_MaxV=Edit_ip_MaxV->Text.ToDouble();
  nnData->ip_Res=Edit_ip_Res->Text.ToInt();
  nnData->ip_MinusNoise=Edit_ip_MinusNoise->Text.ToDouble();
  nnData->ip_PlusNoise=Edit_ip_PlusNoise->Text.ToDouble();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Set the text strings to currently held double values.
// Tested 9/8/1999 KJS
void __fastcall TSetupBox::SetStringValues()
{
  // First Neural Network Tab.
  holding=""; holding.sprintf("%0.3f", nnData->nn_A);
  Edit_nn_A->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->nn_Bias);
  Edit_nn_Bias->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->nn_beta);
  Edit_nn_Beta->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->nn_dt);
  Edit_nn_dt->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->nn_Tlpf);
  Edit_nn_Tlpf->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%d", nnData->nn_MinIter);
  Edit_nn_MinIter->SetTextBuf(holding.c_str());

  // Second request tab.
  holding=""; holding.sprintf("%0.3f", nnData->rq_Min);
  Edit_rq_Min->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->rq_Max);
  Edit_rq_Max->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%d", nnData->rq_Stages);
  Edit_rq_Stages->SetTextBuf(holding.c_str());

  // Thirdly output tab.
  holding=""; holding.sprintf("%0.3f", nnData->op_MinV);
  Edit_op_MinV->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->op_MaxV);
  Edit_op_MaxV->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%d", nnData->op_Res);
  Edit_op_Res->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->op_Conv);
  Edit_op_Conv->SetTextBuf(holding.c_str());

  // Fourth optical tab.
  holding=""; holding.sprintf("%0.6f", nnData->os_MinOO);
  Edit_os_MinOO->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.6f", nnData->os_MaxOO);
  Edit_os_MaxOO->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->os_Spots);
  Edit_os_Spots->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->os_Trans);
  Edit_os_Trans->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.6f", nnData->os_MinDet);
  Edit_os_MinDet->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.6f", nnData->os_MaxDet);
  Edit_os_MaxDet->SetTextBuf(holding.c_str());
```

```
  // Fifth input tab.
  holding=""; holding.sprintf("%0.3f", nnData->ip_MinV);
  Edit_ip_MinV->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->ip_MaxV);
  Edit_ip_MaxV->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%d", nnData->ip_Res);
  Edit_ip_Res->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->ip_MinusNoise);
  Edit_ip_MinusNoise->SetTextBuf(holding.c_str());
  holding=""; holding.sprintf("%0.3f", nnData->ip_PlusNoise);
  Edit_ip_PlusNoise->SetTextBuf(holding.c_str());

}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Initialises all the default values and redraws the box.
// Tested 9/8/1999 KJS
void __fastcall TSetupBox::DefaultClick(TObject *Sender)
{
  InitialiseDefaults();
  SetStringValues();
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// The variable rqChange should be set to false by the user before the box
// is executed.  It will flag any change in request variables.
// Tested 10/9/1999 KJS
void __fastcall TSetupBox::RequestChange(TObject *Sender)
{
  // Set rqChange.
  rqChange=true;
}
//---------------------------------------------------------------------------
```

8.9 Value.h

```
//---------------------------------------------------------------------------
// Slider box for taking a value.
//---------------------------------------------------------------------------
#ifndef ValueH
#define ValueH
//---------------------------------------------------------------------------
#include <vcl\System.hpp>
#include <vcl\Windows.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TValueBox : public TForm
{
__published:
        TButton *OKBtn;
        TButton *CancelBtn;
  TBevel *Bevel;
  TLabel *CurrentVal;
  TTrackBar *TrackBar;
  void __fastcall TrackBarChange(TObject *Sender);
private:
  // For temporarily writing text.
  AnsiString holding;
public:
  // Constructor.
        virtual __fastcall TValueBox(TComponent* AOwner);
  // Local variables storing current values.
  double increment;
  double value;
};
//---------------------------------------------------------------------------
extern PACKAGE TValueBox *ValueBox;
//---------------------------------------------------------------------------
#endif
```

8.10 Value.cpp

```
//---------------------------------------------------------------------------
// Module: TValueBox
// Purpose: Take a value within defined limits using a slider.
// The only definition here is the code for updating the position.
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Value.h"
#include "Data.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TValueBox *ValueBox;
//---------------------------------------------------------------------------
// Object constructor.
__fastcall TValueBox::TValueBox(TComponent* AOwner)
        : TForm(AOwner)
{ // Initial update (clears uninitialised text).
  TrackBarChange(ValueBox);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
// Displays the current value selected by the slider.
// Not tested.
void __fastcall TValueBox::TrackBarChange(TObject *Sender)
{
  // Clear variable.
  holding="";
  // Calculate new value and update: ensure nnData exists first.
  if (nnData!=NULL) value=nnData->rq_Min+(((double)TrackBar->Position)*increment);
  else value=DEF_MIN_REQUEST+(((double)TrackBar->Position)*increment);
  holding.sprintf("Use value: %0.3f", value);
  // Update text.
  CurrentVal->SetTextBuf(holding.c_str());
}
//---------------------------------------------------------------------------
```