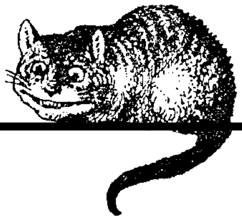
CHESHIRE
ENGINEERING
CORPORATION



Neuralyst™

Version 1.4

User's Guide

(October 1994)

Neural Network Technology

for Microsoft® Excel™

Neuralyst™ User's Guide
by Yin Shih

Editor: Shal Farley

Printing History:

October 1990:	v1.1 Edition
March 1991:	v1.2 Edition
January 1993:	v1.3 Edition
October 1994:	v1.4 Edition
March 1995:	v1.4 Edition (errata)
March 2001:	v1.4 Edition (PDF)
October 2001:	v1.4 Edition (errata)

Disclaimer Notice

Neuralyst is no substitute for real thinking or common sense. The user should always review and check the results of Neuralyst's processing and evaluate it against known references and standards.

The use of Neuralyst for investment, speculation, gambling, or other similar or related purposes is at the user's risk. Results generated by Neuralyst are dependent on past information and there is no guarantee that future results can be forecast or predicted by Neuralyst. Trading in stocks, commodities and other securities or any form of speculation or gambling is inherently risky and may result in loss.

Warranty and Limitation of Liability

Cheshire Engineering Corporation (*Cheshire*) warrants the diskettes on which the software is distributed and the documentation to be free from defects in materials and workmanship for a period of ninety (90) days from the date of purchase. Cheshire will replace any defective diskette or documentation returned to Cheshire during the warranty period. Replacement is the exclusive remedy for any such defects and Cheshire shall have no liability for any other damage.

Cheshire disclaims all other warranties, expressed or implied, including but not limited to implied warranties of merchantability and fitness for any particular purpose.

In no event shall Cheshire be held liable for any damages whatsoever, including without limitation, damages resulting from financial loss, business interruption, loss of information or data, or any other incidental or consequential damages resulting from the use of Neuralyst, even if Cheshire has been advised of the possibility of such damages.

Cheshire Engineering Corporation
650 Sierra Madre Villa Avenue, Suite 201
Pasadena, CA 91107
U.S.A.
+1 626 351 5493   Info@CheshireEng.com

Contents

Chapter 1 Introduction

1.1 Introduction for New Users

Congratulations! You are about to experience a new dimension in spreadsheet processing. Neuralyst adds an unprecedented capability to Excel spreadsheets, providing for open-ended analysis of spreadsheet data and the recognition of associations between data you may have thought to be unrelated.

Up to now spreadsheets (and computers) have provided powerful analysis capabilities, but they were limited by people's ability to envision relationships and express them in spreadsheet models (and computer programs). If a relationship was not realized, then it did not appear on the spreadsheet and resulting tables or charts. Even if a relationship was suspected, it may have been difficult to prove or analyze the effects.

Neuralyst extends the capabilities of spreadsheets in this area using the technology of *neural networks*, also called *neural nets*. Neural networks are simulations of collections of model biological neurons. These are not simulations of real neurons in that they do not model the biology, chemistry, or physics of a real neuron. They do model several aspects of the information combining and pattern recognition behavior of real neurons in a simple yet meaningful way.

This neural modeling has shown incredible capability for emulation, analysis, prediction, and association. Neural networks can be used in a variety of powerful ways: to learn and reproduce rules or operations from given examples; to analyze and generalize from sample facts and make predictions from these; or to memorize characteristics and features of given data and to match or make associations from new data to the old data. Neural networks can be used to make strict yes-no decisions or used to produce more critical, finely-valued judgments.

In this version of Neuralyst, neural network technology is combined with genetic optimization technology to allow you to develop the optimal neural networks to solve your modeling problems. Genetic optimization uses an evolution-like process to refine and enhance the structure of a neural network until it can model your problem in the most efficient way.

Cheshire has integrated neural network technology and genetic optimization technology with spreadsheet technology, combining a comfortable user interface and powerful data management capabilities with this new approach to data processing. All this is available to you now in Neuralyst.

1.2 Changes and Improvements in V1.4

There are many improvements and new features in Neuralyst V1.4. First, there are some slight changes in the format of the Neuralyst Working Area. However, all old V1.x worksheets will be recognized by V1.4 and they can be automatically updated to the format of the new Working Area without any loss of information.

V1.4 now requires Excel 4.0 at a minimum. It works with Excel 5.0, but that is not a requirement. V1.4 now supports both the U.S. and international versions of Excel with a single collection of macro sheets.

Several improvements have been made in the **Config** menu that enhance your ability to manage Neuralyst’s interpretation of data. A new command, **Set Mode Flag Column**, has been added as a replacement to **Set TestFlag Column**. Check Section 8.1.6 for details.

As a companion to **Set Mode Flag Column**, the command **Set Mode Rows** has also been provided. This command provides three options to allow Neuralyst to set certain rows of the Mode Flag column to control the minimum range of data, maximum range of data, and to define symbolic names for data. Check Section 8.1.7 for details.

The final addition to the **Config** menu is the **Edit Mode Lists** command. This command is also part of the new data management features and allows you to set or edit symbolic names for data and set minimum or maximum range limits. Check Section 8.1.10 for details.

There have been several enhancements in the **Neural** menu. **Set Enhanced Parameters** has been modified with some significant improvements. But, the real star of V1.4 is the addition of two commands to support genetic optimization technology.

In **Set Enhanced Parameters**, the Activation Function control panel has been enhanced, and two new control panels Calculation Method and Scaling Margin have been added. Two new functions are now available in the Activation Function control panel, Hyperbolic Tangent and Augmented Ratio of Squares. The Calculation Method control panel now allows you to select Fixed Point or Floating Point computation of neural networks. This allows you to trade off high speed against high precision. Finally, the Scaling Margin control panel allows you to adjust the headroom available in the data rescaling algorithms used by Neuralyst, thus allowing you more control of how your data is represented. Check Section 8.2.6 for details on all of the above.

Two new **Neural** menu commands are the keys to unlocking the new genetic optimization technology now integrated into Neuralyst, **Set Genetic Parameters** and **Run Genetic Supervisor**. These commands allow initialization and control of the Neuralyst Genetic Supervisor. See Chapter 7 for a better understanding of how genetic technology works and Sections 8.2.7 and 8.2.4 for details on the commands.

Finally, the connection weight limit has been increased from 10,920 to more than 131,000. This will allow neural networks more than ten times as complex to be developed.

Overall, we believe that we have provided a combination of enhancements that should improve the neural network training process, allow for more sophisticated neural networks to be built, and improved ease of use. We hope you agree.

These changes and additions were the results of many comments received from current users. While not all suggestions were implemented in this version, we believe we have listened carefully and selected a worthwhile set for implementation. Please let us know how we did — and start sending in suggestions for the next version!

Chapter 2
Installing Neuralyst

2.1 About this Manual

This manual is used for both the Microsoft Windows and Apple Macintosh versions of Neuralyst (including Win32 and Power Macintosh). Generally, Neuralyst operations in either version are indistinguishable, except for file names, path versus folder names, and other features that are dependent on the behavior of the respective operating systems.

If you are experienced with Windows or the Macintosh, then the spirit of the directions for usage and operation given in this manual can be taken and applied to either system.

However, to minimize confusion, we have generally provided instructions or filenames for each version. These computer specific items are identified in one of two ways. If there is material of just a few words in length, then the Windows version is in plain text and the Macintosh version is enclosed immediately after in curly braces, i.e. {}. If there is material that is more extensive, then it is presented as alternative sections or subsections (Windows version first, then Macintosh version), as appropriate.

The screen snapshots shown in this manual are all taken from the Windows version of Neuralyst. If you have the Macintosh version of Neuralyst, the windows, control boxes, scroll bars, and so on will appear somewhat different, but the worksheet data should appear the same.

2.2 Neuralyst Distribution

The Neuralyst distribution you have received should contain the following items:

1. Neuralyst User's Guide
2. Neuralyst Distribution Disk
3. Software License Agreement
4. User Registration Form

If you do not have all the items listed then please contact Cheshire Customer Service immediately. See Appendix C in this guide for the recommended call-in procedure.

For your convenience, the Neuralyst distribution disk is not copy-protected. You may make an archival copy of the disk, install the software and use it as described in this guide and with the single-use restrictions specified in the Software License Agreement. Please observe these restrictions. Cheshire has made a considerable investment in the development of the Neuralyst program and violations of the copyright and software license are not trivial matters.

Each Neuralyst distribution disk has a label with a unique serial number printed on it. Protect this serial number and do not allow anyone to copy it or the software. The serial number is registered to you as the original purchaser or by your submission of the User Registration Form if you did not receive Neuralyst directly from Cheshire. This serial number will be required to validate customer support access and future software update privileges.

2.3 System Requirements

Neuralyst requires certain hardware and software configurations to run properly.

A math co-processor is not necessary. For most of its calculations Neuralyst defaults to using highly optimized fixed-point arithmetic operations because in most computer systems they operate faster than the comparable operations using floating-point. If desired, the user can select floating-point operation for Neuralyst instead of fixed-point. A math co-processor will not significantly increase the performance of Neuralyst's computations using fixed-point, but it will increase the performance of Neuralyst's computations if floating-point is selected and it will increase the performance of Excel's computations.

2.3.1 Windows System Requirements

The Windows version of Neuralyst has been written to work on IBM personal computers (PC/XT, PC/AT, and PS/2) and compatibles. Neuralyst also requires Microsoft Windows Version 3.1 and Microsoft Excel Version 4.0 (or higher versions) in order to run. In general any system capable of running Windows and Excel will be able to support Neuralyst.

Neuralyst places a heavy computational load on a system. Therefore, higher performance systems, while not necessary, will reduce the processing (and corresponding waiting) time needed for Neuralyst to analyze a problem. In general, Neuralyst will perform best on 386, 486, and Pentium systems, while high-speed 80286 systems will probably work acceptably.

A math co-processor (80287, 80387 or 80487) is optional.

2.3.2 Macintosh System Requirements

The Macintosh version of Neuralyst has been written to work on Apple Macintosh computers. Neuralyst also requires Microsoft Excel Version 4.0 (or higher versions) in order to run. In general any Macintosh capable of running Excel will be able to support Neuralyst.

Neuralyst places a heavy computational load on a system. Therefore, higher performance systems, while not necessary, will reduce the processing (and corresponding waiting) time needed for Neuralyst to analyze a problem. In general, Neuralyst will perform best on 68030, 68040, or PowerPC systems. A 68020 will probably work acceptably. 68000 systems will work, but the performance will be marginal.

A math co-processor (68881 or 68882) is optional.

2.4 Installation Procedure

Neuralyst is integrated with Excel. This User's Guide assumes you are familiar with Excel and have all the materials and documentation necessary to run Excel. You should be familiar with the operations and commands available in Excel before proceeding to install and use Neuralyst. Cheshire Customer Service will not be able to answer any questions involving Excel set-up or operations.

If you have a PC follow the installation procedure described in Section 2.4.1. If you have a Macintosh follow the installation procedure described in Section 2.4.2.

2.4.1 Windows Installation Procedure

If you are reinstalling Neuralyst or installing a new version over an old version, then you need to make sure that you have exited Windows at least once since the last time you ran Neuralyst, otherwise Windows will try and run the old version of Neuralyst when you start Neuralyst after the install. Once this has been done, then you may proceed with the installation as described.

If you are installing Neuralyst for the first time, make sure that Windows and Excel are installed on your computer. If this is not so, then follow the respective installation procedures for each program as described by the vendor. After this has been done, you might need to make sure that Excel is registered in the Windows `WIN.INI` file so that Windows will be able to find Excel regardless of which directory is currently active. This will have been done automatically by the Excel installation, if it was done normally.

Turn on your PC and allow it to go through its normal boot procedure. If the boot procedure does not start Windows immediately, do so now — you must be in Windows to install Neuralyst correctly. Once your PC has concluded its boot sequence and Windows is loaded, insert the Neuralyst distribution disk in drive `A`.

From the Program Manager's **File** menu choose the **Run** command to run the install program. Enter the text:

```
A:INSTALL.EXE
```

and confirm **OK** to activate the install program. If your floppy is not drive `A`, then change the drive name to the correct name first, for example, change the text to `B:INSTALL.EXE` if you use drive `B`.

The installation program will now run. It will prompt you for the drive and directory that will be the target for the installation. This is defaulted to `C:\NEURLYST`. If your hard drive is not `C`, then edit the entry to show the proper drive name. It is recommended that `\NEURLYST` be used as the destination directory. Confirm **OK** when you have set the destination.

The installation program will now move the required Neuralyst program files and the example Neuralyst worksheet files to the `C:\NEURLYST` directory and register the Neuralyst program icon with Program Manager. The Neuralyst icon will appear in the Windows Applications Program Group. If there is no Windows Applications Program Group, then the install program will create a new Program Group called Neuralyst.

When the installation is complete, the installation program will exit and return you to Windows. If there are any errors in the installation process the program will exit, but an error report will be generated first and require your acknowledgment before the exit. In the case of error, you will have to correct the problem before trying the installation procedure again. See Appendix A for help with any problems you may encounter during installation.

See Appendix G for instructions on how to install the supplementary Trader's Macro Library. If you have purchased an optional library from Cheshire to integrate with Neuralyst, see the instructions included with each optional library for any special installation instructions.

Once the installation procedure is complete, you may use Neuralyst as described in the tutorial session shown in Chapter 4. But first, let's check to see if Neuralyst is working. Proceed to Section 2.5.

2.4.2 Macintosh Installation Procedure

If you are installing Neuralyst for the first time, make sure that Excel is installed on your computer. If this is not so, then follow the installation procedure for Excel as described by Microsoft. With this done, you are ready to proceed with the Neuralyst installation.

Turn on your Macintosh and allow it to go through its normal boot procedure. Once your Macintosh has concluded its boot sequence, insert the Neuralyst distribution disk in a floppy disk drive. Double-click on the disk icon labeled `Neuralyst Distribution` to show the disk contents, which will be a folder labeled Neuralyst. Select this folder and drag it to your hard disk to copy the contents of the folder. The `Neuralyst` folder may be copied to any area of your hard disk.

See Appendix G for instructions on how to install the supplementary Trader's Macro Library. If you have purchased an optional library from Cheshire to integrate with Neuralyst, see the instructions included with each optional library for any special installation instructions.

Once the installation procedure is complete, you may use Neuralyst as described in the tutorial session shown in Chapter 4. But first, let's check to see if Neuralyst is working. Proceed to the next section.

2.5 The First Neuralyst Session

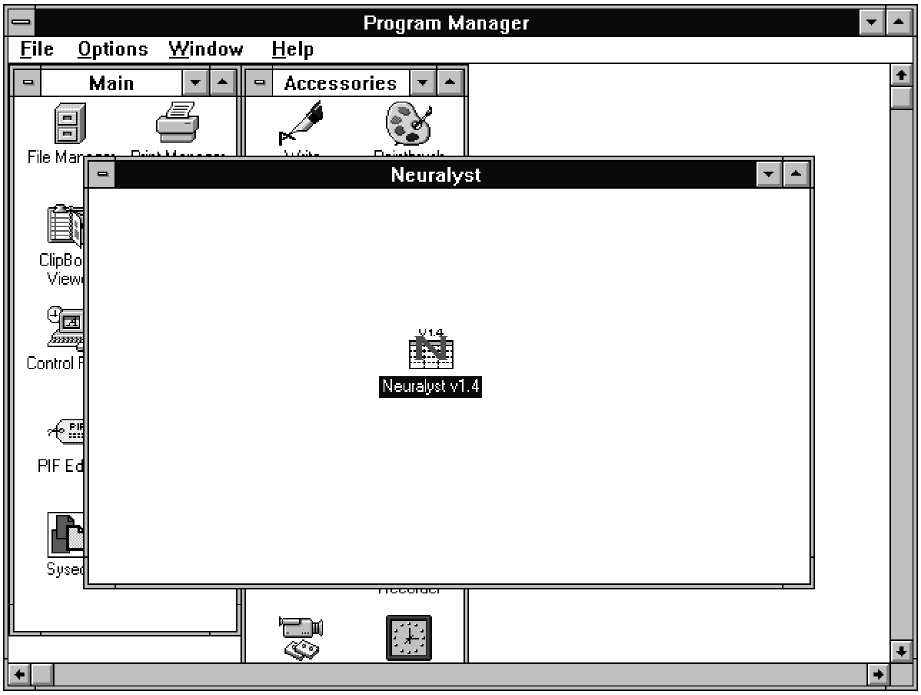


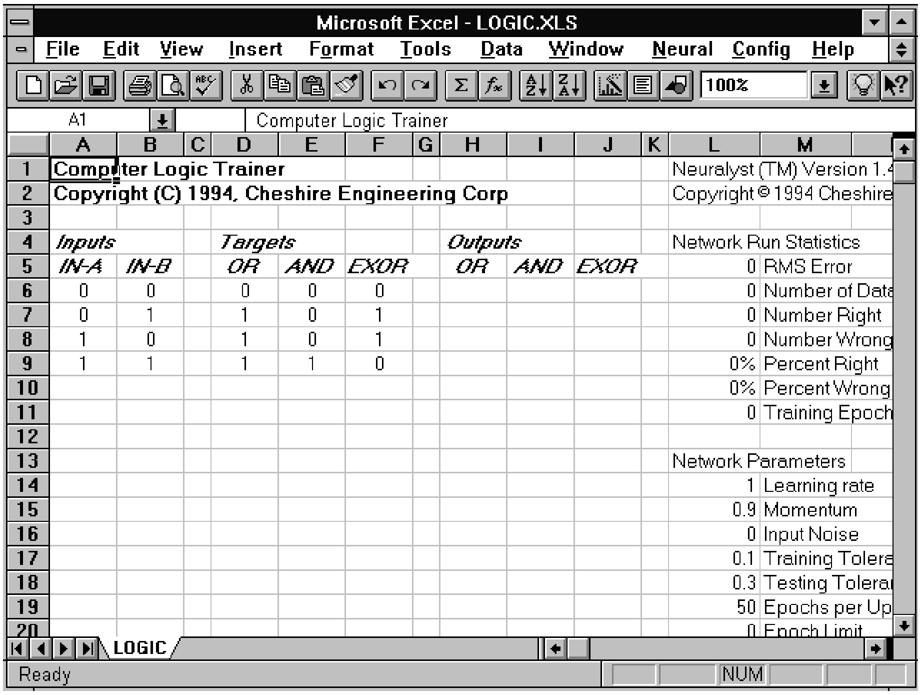
Figure 2-1 Neuralyst Icon *Windows version*

Let’s check Neuralyst out. If you are running from a PC, start from the Windows Program Manager and find the Neuralyst icon. Double-click on the Neuralyst icon to start execution. {If you are running from a Macintosh, start from the Neuralyst folder. Double-click on the Excel macro file named Neuralyst to start execution — NOT the Neuralyst Lib icon.}

First Excel and then Neuralyst will be loaded. You will see the title screens for each appear in sequence. When both have appeared, Neuralyst is ready to run. The environment with which you will interact is primarily Excel’s, but there will be new commands that relate to Neuralyst operations.

From the **File** menu choose the **Open** command. Find the Neuralyst directory and move to that, if you are not already in it. There will be a number of example Neuralyst worksheets listed. Find the one named `LOGIC.XLS` {Logic} and load it by double-clicking on it.

Excel will now open a window that appears as shown in Figure 2-2.


Microsoft Excel - LOGIC.XLS
File Edit View Insert Format Tools Data Window Neural Config Help
100%
A1
Computer Logic Trainer
LOGIC
Ready
NUM

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Computer Logic Trainer											Neuralyst (TM) Version 1.4	
2	Copyright (C) 1994, Cheshire Engineering Corp											Copyright © 1994 Cheshire	
3													
4	*Inputs*			*Targets*				*Outputs*				Network Run Statistics	
5	*IN-A*	*IN-B*		*OR*	*AND*	*EXOR*		*OR*	*AND*	*EXOR*		0	RMS Error
6	0	0		0	0	0						0	Number of Data
7	0	1		1	0	1						0	Number Right
8	1	0		1	0	1						0	Number Wrong
9	1	1		1	1	0						0%	Percent Right
10												0%	Percent Wrong
11												0	Training Epoch
12													
13												Network Parameters	
14												1	Learning rate
15												0.9	Momentum
16												0	Input Noise
17												0.1	Training Tolera
18												0.3	Testing Tolerai
19												50	Epochs per Up
20												0	Epoch Limit

Figure 2-2 LOGIC.XLS {Logic} Example

`LOGIC.XLS` {Logic} defines three of the most basic operations for computer logic circuits. In computer design, conditions or events are represented by *binary* values, 0 or 1, indicating off/on, false/true, or absence/presence of these conditions or events. Most of the time computer circuits operate with two or more conditions or events as inputs and use these to form a new condition or event that will be used as an input in a succeeding operation.

For example, on row **7** of the example, In-A has the value 0 and In-B has the value 1. Under the Targets area are three columns, **D**, **E**, and **F**. These represent three different rules, OR, AND, and EXOR. OR means “either or both inputs must 1 for the output to be 1”, AND means “both inputs must be 1 for the output to be 1” and EXOR means “either input, but not both, must be 1 for the output to be 1”. In the case of row **7**, the OR rule produces 1, the AND rule produces 0 and the EXOR rule produces 1. Each of the four rows, **6**, **7**, **8**, and **9** represent a different set of possible input combinations. With four combinations and three rules, there are a total of 12 possible input-output combinations.

The Inputs and Targets areas represent the information that is known and to which the neural network will be applied. The next area, designated Outputs, shows all 0's. These are the current outputs of the neural network, which have no correlation to the values in the Targets area prior to the application of the neural network.

You will notice two new menu items for Excel in addition to the example worksheet. These are the **Neural** and **Config** menus; which are the operating interface to Neuralyst. These will be discussed in much greater depth later.

*[If you have a Macintosh with a small screen, then Neuralyst will name its menus **N.** and **C.**, rather than **Neural** and **Config**, to save space. If you have a Macintosh, System 7, and a small screen, the normal Neuralyst interface will not work well; see Appendix A for instructions on how to manage this configuration.]*

For now, pull down the **Neural** menu and note the first line, **Reload Network**. Move the pointer to the line and release the mouse button to activate the command. Excel will show an hourglass {watch} cursor for a moment and status messages will flash across the bottom in the Status Display area. The neural network has now been loaded to the initial configuration saved on the worksheet.

[If you normally leave the Excel Calculation mode set to Manual, be sure to set it to Automatic when working with Neuralyst. Some initialization operations and the statistical information Neuralyst reports to you while executing will be incorrect if Automatic Calculation is not set.]

Pull down the **Neural** menu again. This time activate the **Train Network** command. This sets the neural network to work. As time progresses, the cell **L5**, labeled RMS Error, will be highlighted and you will see its value steadily decreasing. The lower this value, the better the neural network has learned the characteristics of the data presented to it. After some time (depending on the speed of your processor) Neuralyst will stop operations and the Outputs area will be updated. The values now shown represent the outputs of the neural network and will match the corresponding values in the Targets area.

Neuralyst is working! It has learned all three rules for these input combinations and is now able to duplicate the rules presented to it.

Let's take a moment to understand neural networks better and what this example means before we try out more examples. For now, use **Exit** {**Quit**} from the **File** menu to exit Excel, but don't save the changes when asked.

Chapter 3
Basic Concepts for Neural Networks

3.1 Real Neurons

Let’s start by taking a look at a biological neuron. Figure 3-1 shows such a neuron.

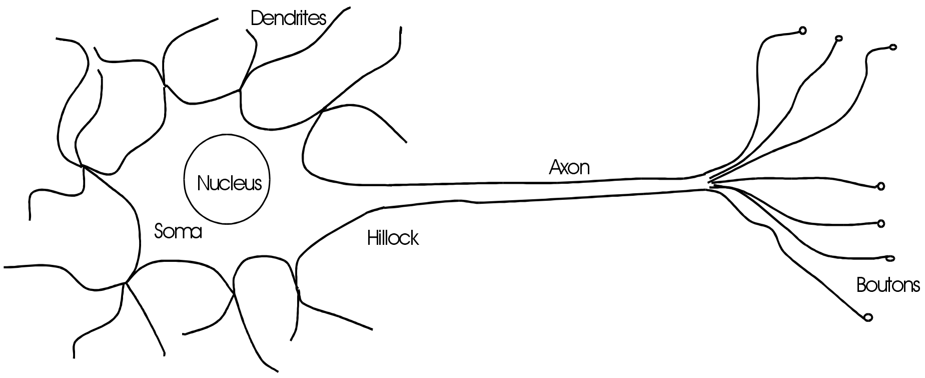


Figure 3-1 A Biological Neuron

A neuron operates by receiving signals from other neurons through connections, called *synapses*. The combination of these signals, in excess of a certain *threshold* or *activation* level, will result in the neuron *firing*, that is sending a signal on to other neurons connected to it. Some signals act as *excitations* and others as *inhibitions* to a neuron firing. *What we call thinking is believed to be the collective effect of the presence or absence of firings in the pattern of synaptic connections between neurons.*

This sounds very simplistic until we recognize that there are approximately one hundred billion (100,000,000,000) neurons each connected to as many as one thousand (1,000) others in the human brain. The massive number of neurons and the complexity of their interconnections results in a “thinking machine”, your brain.

Each neuron has a body, called the *soma*. The soma is much like the body of any other cell. It contains the cell nucleus, various bio-chemical factories and other components that support ongoing activity.

Surrounding the soma are *dendrites*. The dendrites are receptors for signals generated by other neurons. These signals may be excitatory or inhibitory. All signals present at the dendrites of a neuron are combined and the result will determine whether or not that neuron will fire.

If a neuron fires, an electrical impulse is generated. This impulse starts at the base, called the *hillock*, of a long cellular extension, called the *axon*, and proceeds down the axon to its ends.

The end of the axon is actually split into multiple ends, called the *boutons*. The boutons are connected to the dendrites of other neurons and the resulting interconnections are the previously discussed synapses. (Actually, the boutons do not touch the dendrites; there is a small gap between them.) If a neuron has fired, the electrical impulse that has been generated stimulates the boutons and results in electrochemical activity which transmits the signal across the synapses to the receiving dendrites.

At rest, the neuron maintains an electrical potential of about 40-60 millivolts. When a neuron fires, an electrical impulse is created which is the result of a change in potential to about 90-100 millivolts. This impulse travels between 0.5 to 100 meters per second and lasts for about 1 millisecond. Once a neuron fires, it must rest for several milliseconds before it can fire again. In some circumstances, the repetition rate may be as fast as 100 times per second, equivalent to 10 milliseconds per firing.

Compare this to a very fast electronic computer whose signals travel at about 200,000,000 meters per second (speed of light in a wire is 2/3 of that in free air), whose impulses last for 10 nanoseconds and may repeat such an impulse immediately in each succeeding 10 nanoseconds continuously. Electronic computers have at least a 2,000,000 times advantage in signal transmission speed and 1,000,000 times advantage in signal repetition rate.

It is clear that if signal speed or rate were the sole criteria for processing performance, electronic computers would win hands down. What the human brain lacks in these, it makes up in numbers of elements and interconnection complexity between those elements. This difference in structure manifests itself in at least one important way; the human brain is not as quick as an electronic computer at arithmetic, but it is many times faster and hugely more capable at recognition of patterns and perception of relationships.

The human brain differs in another, extremely important, respect beyond speed; it is capable of "self-programming" or adaptation in response to changing external stimuli. In other words, it can learn. The brain has developed ways for neurons to change their response to new stimulus patterns so that similar events may affect future responses. In particular, the sensitivity to new patterns seems more extensive in proportion to their importance to survival or if they are reinforced by repetition.

3.2 Neural Network Structure

Neural networks are models of biological neural structures. The starting point for most neural networks is a model neuron, as in Figure 3-2. This neuron consists of multiple inputs and a single output. Each input is modified by a *weight*, which multiplies with the input value. The neuron will combine these weighted inputs and, with reference to a threshold value and activation function, use these to determine its output. This behavior follows closely our understanding of how real neurons work.

While there is a fair understanding of how an individual neuron works, there is still a great deal of research and mostly conjecture regarding the way neurons organize themselves and the mechanisms used by arrays of neurons to adapt their behavior to external stimuli.

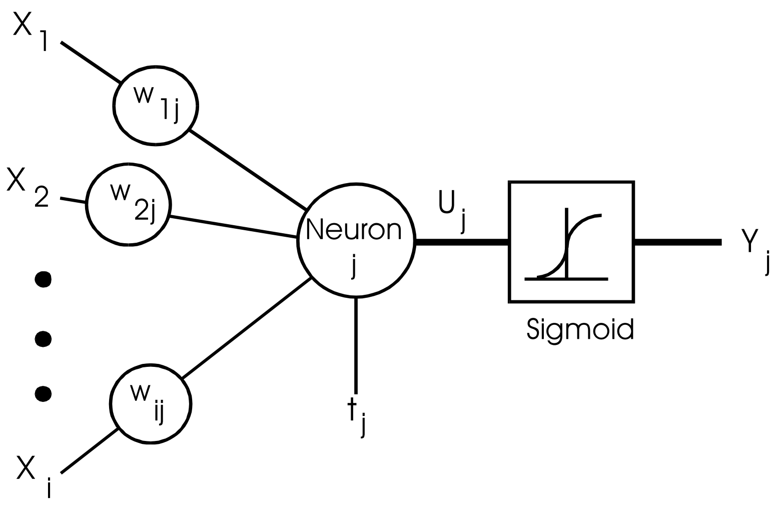


Figure 3-2 A Model Neuron

There are a large number of experimental neural network structures currently in use reflecting this state of continuing research.

In our case, we will only describe the structure, mathematics and behavior of that structure known as the *backpropagation network*. This is the most prevalent and generalized neural network currently in use. If the reader is interested in finding out more about neural networks or other networks, please refer to the material listed in Appendix F.

To build a backpropagation network, proceed in the following fashion. First, take a number of neurons and array them to form a *layer*. A layer has all its inputs connected to either a preceding layer or the inputs from the external world, but not both within the same layer. A layer has all its outputs connected to either a succeeding layer or the outputs to the external world, but not both within the same layer.

Next, multiple layers are then arrayed one succeeding the other so that there is an input layer, multiple intermediate layers and finally an output layer, as in Figure 3-3. Intermediate layers, that is those that have no inputs or outputs to the external world, are called *hidden layers*. Backpropagation neural networks are usually *fully connected*. This means that each neuron is connected to every output from the preceding layer or one input from the external world

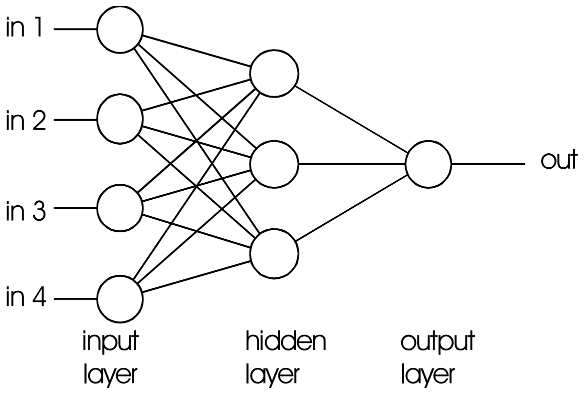


Figure 3-3 Backpropagation Network

if the neuron is in the first layer and, correspondingly, each neuron has its output connected to every neuron in the succeeding layer.

Generally, the input layer is considered a distributor of the signals from the external world. Hidden layers are considered to be categorizers or feature detectors of such signals. The output layer is considered a collector of the features detected and producer of the response. While this view of the neural network may be helpful in conceptualizing the functions of the layers, you should not take this model too literally as the functions described may not be so specific or localized.

With this picture of how a neural network is constructed, we can now proceed to describe the operation of the network in a meaningful fashion.

3.3 Neural Network Operation

The output of each neuron is a function of its inputs. In particular, the output of the ***j***th neuron in any layer is described by two sets of equations:

$$U_j = \sum (X_i * w_{ij}) \qquad \text{[Eqn 3-1]}$$

and

$$Y_j = F_{th} \, (U_j + t_j) \qquad \text{[Eqn 3-2]}$$

For every neuron, ***j***, in a layer, each of the ***i*** inputs, $\boldsymbol{X}_{\mathrm{i}}$, to that layer is multiplied by a previously established weight, $\boldsymbol{w}_{\mathrm{ij}}$. These are all summed together, resulting in the internal value of this operation, $\boldsymbol{U}_{\mathrm{j}}$. This value is then biased by a previously established threshold value, $\boldsymbol{t}_{\mathrm{j}}$, and sent through an activation function, $\boldsymbol{F}_{\mathrm{th}}$. This activation function is usually the sigmoid function, which has an input to output mapping as shown in Figure 3-4. The resulting output, $\boldsymbol{Y}_{\mathrm{j}}$, is an input to the next layer or it is a response of the neural network if it is the last layer. Neuralyst allows other threshold functions to be used in place of the sigmoid described here. See Section 6.6 for details.

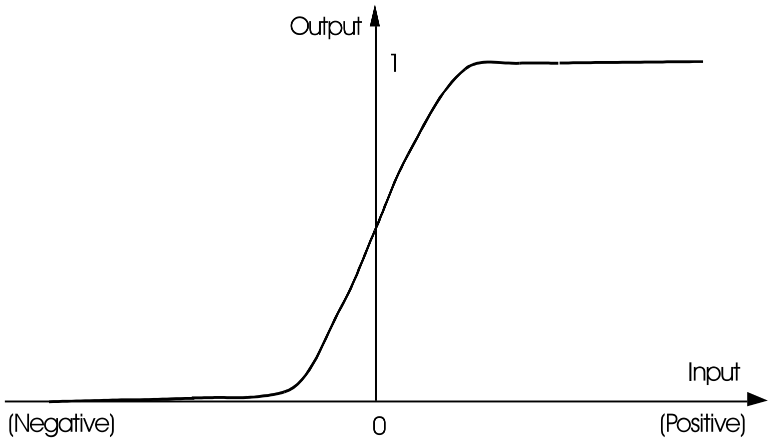


Figure 3-4 Sigmoid Function

In essence, Equation 3-1 implements the combination operation of the neuron and Equation 3-2 implements the firing of the neuron.

From these equations, a predetermined set of weights, a predetermined set of threshold values and a description of the network structure (that is the number of layers and the number of neurons in each layer), it is possible to compute the response of the neural network to any set of inputs. And this is just how Neuralyst goes about producing the response. But how does it learn?

3.4 Neural Network Learning

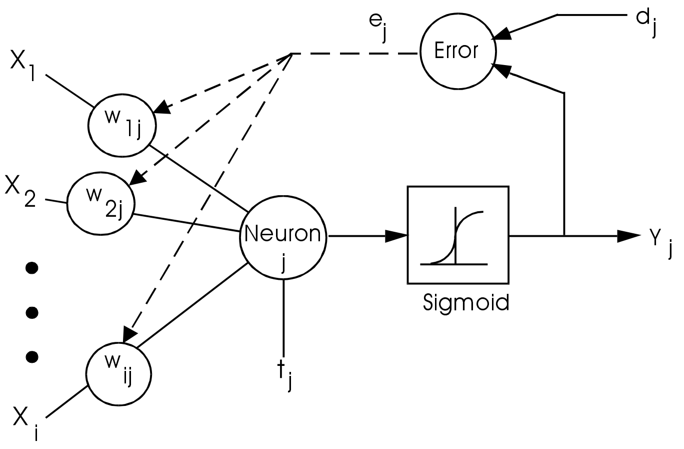


Figure 3-5 Neuron Weight Adjustment

Learning in a neural network is called *training*. Like training in athletics, training in a neural network requires a coach, someone that describes to the neural network what it should have produced as a response. From the difference between the desired response and the actual response, the *error* is determined and a portion of it is propagated backward through the network. At each neuron in the network the error is used to adjust the weights and threshold values of the neuron, so that the next time, the error in the network response will be less for the same inputs.

This corrective procedure is called *backpropagation* (hence the name of the neural network) and it is applied continuously and repetitively for each set of inputs and corresponding set of outputs produced in response to the inputs. This procedure continues so long as the individual or total errors in the responses exceed a specified level or until there are no measurable errors. At this point, the neural network has learned the training material and you can stop the training process and use the neural network to produce responses to new input data.

[There is some heavier going in the next few paragraphs. Skip ahead if you don't need to understand all the details of neural network learning.]

Backpropagation starts at the output layer with the following equations:

$$w_{ij} = w'_{ij} + LR * e_j * X_i \qquad \text{[Eqn 3-3]}$$

and

$$e_j = Y_j * (1-Y_j) * (d_j-Y_j) \qquad \text{[Eqn 3-4]}$$

For the ***i***th input of the ***j***th neuron in the output layer, the weight $\boldsymbol{w}_{ij}$ is adjusted by adding to the previous weight value, $\boldsymbol{w'}_{ij}$, a term determined by the product of a *learning rate*, ***LR***, an error term, $\boldsymbol{e}_j$, and the value of the ***i***th input, $\boldsymbol{X}_i$. The error term, $\boldsymbol{e}_j$, for the jth neuron is determined by the product of the actual output, $\boldsymbol{Y}_j$, its complement, $\boldsymbol{1 - Y}_j$, and the difference between the desired output, $\boldsymbol{d}_j$, and the actual output.

Once the error terms are computed and weights are adjusted for the output layer, the values are recorded and the next layer back is adjusted. The same weight adjustment process, determined by Equation 3-3, is followed, but the error term is generated by a slightly modified version of Equation 3-4. This modification is:

$$e_j = Y_j * (1-Y_j) * \sum (e_k * w'_{jk}) \qquad \text{[Eqn 3-5]}$$

In this version, the difference between the desired output and the actual output is replaced by the sum of the error terms for each neuron, ***k***, in the layer immediately succeeding the layer being processed (remember, we are going backwards through the layers so these terms have already been computed) times the respective pre-adjustment weights.

The learning rate, ***LR***, applies a greater or lesser portion of the respective adjustment to the old weight. If the factor is set to a large value, then the neural network may learn more quickly, but if there is a large variability in the input set then the network may not learn very well or at all. In real terms, setting the learning rate to a large value is analogous to giving a child a spanking, but that is inappropriate and counter-productive to learning if the offense is so simple as forgetting to tie their shoelaces. Usually, it is better to set the factor to a small value and edge it upward if the learning rate seems slow.

In many cases, it is useful to use a revised weight adjustment process. This is described by the equation:

$$w_{ij} = w'_{ij} + (1-M)*LR*e_j*X_i + M*(w'_{ij} - w''_{ij}) \qquad \text{[Eqn 3-6]}$$

This is similar to Equation 3-3, with a *momentum* factor, ***M***, the previous weight, $\boldsymbol{w'_{ij}}$, and the next to previous weight, $\boldsymbol{w''_{ij}}$, included in the last term. This extra term allows for momentum in weight adjustment. Momentum basically allows a change to the weights to persist for a number of adjustment cycles. The magnitude of the persistence is controlled by the momentum factor. If the momentum factor is set to 0, then the equation reduces to that of Equation 3-3. If the momentum factor is increased from 0, then increasingly greater persistence of previous adjustments is allowed in modifying the current adjustment. This can improve the learning rate in some situations, by helping to smooth out unusual conditions in the training set.

[Okay, that's the end of the equations. You can relax again.]

As you train the network, the total error, that is the sum of the errors over all the training sets, will become smaller and smaller. Once the network reduces the total error to the limit set, training may stop. You may then apply the network, using the weights and thresholds as trained.

It is a good idea to set aside some subset of all the inputs available and reserve them for *testing* the trained network. By comparing the output of a trained network on these test sets to the outputs you know to be correct, you can gain greater confidence in the validity of the training. If you are satisfied at this point, then the neural network is ready for *running*.

Usually, no backpropagation takes place in this running mode as was done in the training mode. This is because there is often no way to be immediately certain of the desired response. If there were, there would be no need for the processing capabilities of the neural network! Instead, as the validity of the neural network outputs or predictions are verified or contradicted over time, you will either be satisfied with the existing performance or determine a need for new training. In this case, the additional input sets collected since the last training session may be used to extend and improve the training data.

Now that we have an understanding of how a neural network functions and what it may accomplish, let's get into a more detailed Neuralyst session.

Chapter 4

A Neuralyst Tutorial

4.1 Starting Neuralyst

To start Neuralyst, the PC should first be running Windows. Once you have Windows active, Neuralyst may be started in one of two ways. First, you may start by double-clicking on the Neuralyst icon in Windows Program Manager. This will cause Excel to run followed immediately by the loading of the Neuralyst package. Second, you may start from within Excel: from the **File** menu choose the **Open** command, move to the Neuralyst directory and select `NEURLYST.XLM` to cause the Neuralyst package to load.

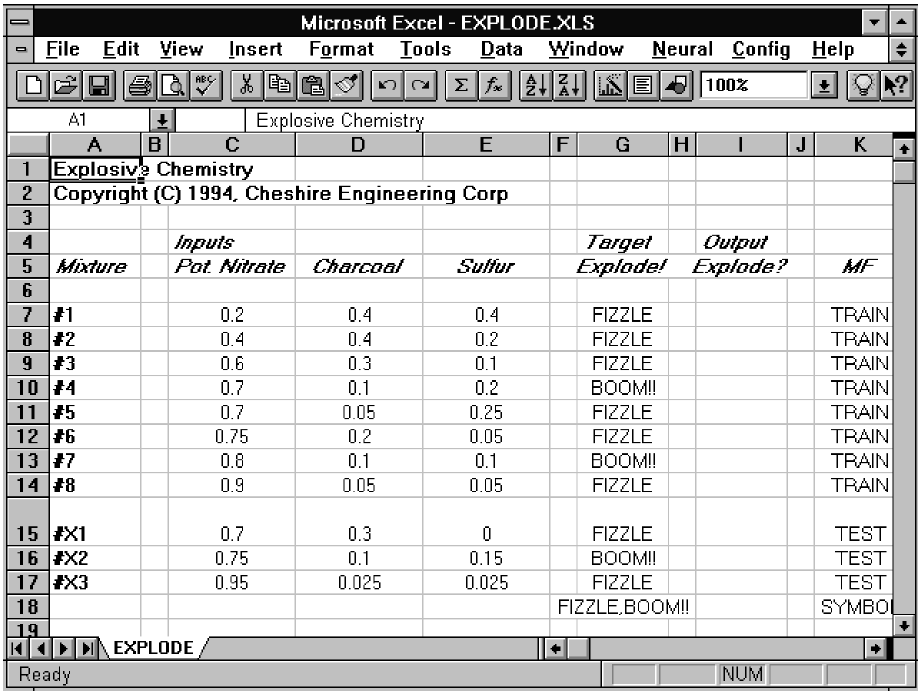
[For Windows, only one instance of Excel with Neuralyst should be running at one time. If you start Neuralyst twice, the two instances may interfere with each other. See Appendix A for more information.]

{Neuralyst may be started in one of two ways on the Macintosh. First, you may start by double-clicking on the Excel macro file named Neuralyst. This will cause Excel to run followed immediately by the loading of the Neuralyst package. Second, you may start from within Excel: from the **File** menu choose the **Open** command, move to the Neuralyst folder, and select Neuralyst to cause the Neuralyst package to load.}

Once Excel is running with the Neuralyst package loaded, from the **File** menu you may choose the **New** command to create a new file or the **Open** command to select the file to use.

Let's go ahead and open the file `EXPLODE.XLS` {`Explode`} in the Neuralyst directory. You will see a window similar to that shown in Figure 4-1.

Note the two new menu items that have been added with the loading of the Neuralyst package. These two new menus are **Neural** and **Config**. These menus interface to Neuralyst and allow you to define the problem and control the operations of Neuralyst.


Microsoft Excel - EXPLODE.XLS
File Edit View Insert Format Tools Data Window Neural Config Help
100%
A1
Explosive Chemistry
EXPLODE
Ready
NUM

	A	B	C	D	E	F	G	H	I	J	K
1	Explosive Chemistry										
2	Copyright (C) 1994, Cheshire Engineering Corp										
3											
4			*Inputs*				*Target*		*Output*		
5	*Mixture*		*Pot. Nitrate*	*Charcoal*	*Sulfur*		*Explode!*		*Explode?*		*MF*
6											
7	#1		0.2	0.4	0.4		FIZZLE				TRAIN
8	#2		0.4	0.4	0.2		FIZZLE				TRAIN
9	#3		0.6	0.3	0.1		FIZZLE				TRAIN
10	#4		0.7	0.1	0.2		BOOM!!				TRAIN
11	#5		0.7	0.05	0.25		FIZZLE				TRAIN
12	#6		0.75	0.2	0.05		FIZZLE				TRAIN
13	#7		0.8	0.1	0.1		BOOM!!				TRAIN
14	#8		0.9	0.05	0.05		FIZZLE				TRAIN
15	#X1		0.7	0.3	0		FIZZLE				TEST
16	#X2		0.75	0.1	0.15		BOOM!!				TEST
17	#X3		0.95	0.025	0.025		FIZZLE				TEST
18						FIZZLE,BOOM!!					SYMBOI
19											

Figure 4-1 EXPLODE.XLS {Explode}

Now, let's take a look at the worksheet. EXPLODE.XLS {Explode} is a representation of a series of chemistry experiments. A young chemist has compounded mixtures #1-8 using the proportions of Potassium Nitrate, Charcoal, and Sulfur shown. For each of these mixtures he has determined whether or not that mixture will explode. This is summarized in the first eight rows of the Explode! column. For example, row **9** shows that a mixture of 60% Potassium Nitrate, 30% Charcoal, and 10% Sulfur did not explode (the value in the column Explode! is FIZZLE), while row **10** shows that a mixture of 70% Potassium Nitrate, 10% Charcoal, and 20% Sulfur did explode (the value in the Explode! column is BOOM!!).

[The chemistry rules in the world of EXPLODE.XLS {Explode} do not correspond to the real world. If you really want to learn how to make gunpowder, you will have to find the formula elsewhere!]

However, his lab manager has gotten a little tired of paying for all the broken glassware and has asked him to cut the experiments short, before he has had a chance to test compounds #X1-X3. Does he have enough information from the previous experiments to make an analysis of whether the proportions of the three chemicals in these three mixtures will result in an explosion?

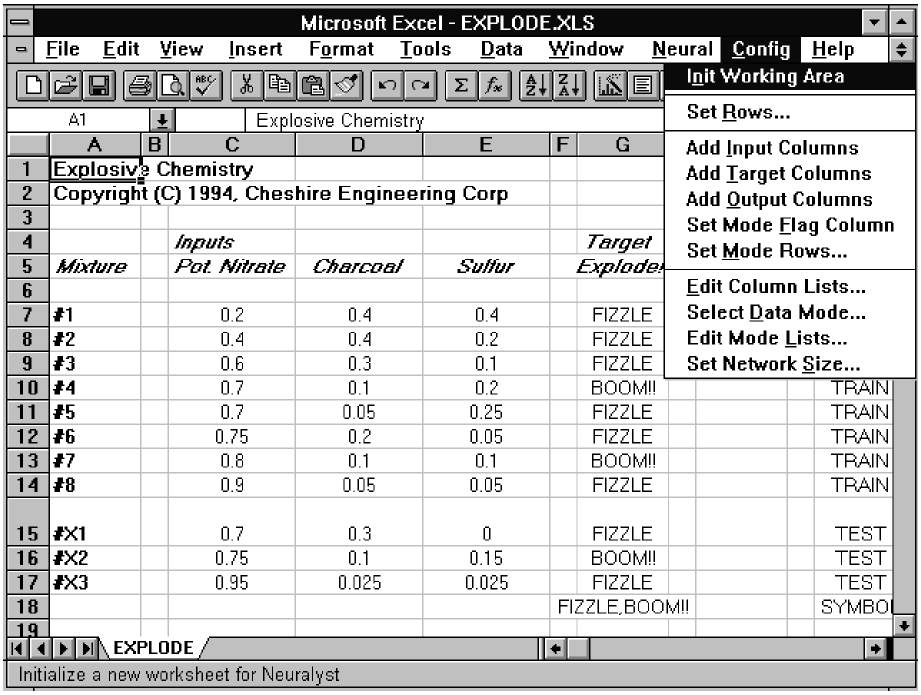
Neuralyst can be used to help this young chemist analyze his results.

4.2 Configuring the Neural Network

Neuralyst cannot work with a completely unorganized conglomeration of facts. For a problem to be presented to Neuralyst, it should be organized as *instances*, *examples* or *cases*, consisting of related data or facts including known or expected results and goals.

Within this context, Neuralyst expects that a certain number of rows, of all the rows of the worksheet, will represent each instance of a problem. If that number is 1, then each row constitutes a separate instance. If that number is greater than 1, then that many rows are taken together to constitute each separate instance. In the case of EXPLODE.XLS {Explode}, the data has been organized into individual rows. Each of rows **7** through **17** is used to describe a different mixture in this series of experiments. The first eight, rows **7** through **14**, represent instances whose results are known; the next three, rows **15** through **17**, represent instances that we wish to predict after neural network training on the first eight. The last row with data, row **18**, is a special row which is used to define the valid symbols, <FIZZLE, BOOM!!>, for the Explode! column.

Neuralyst also expects that the columns of a worksheet individually represent different facts, goals, or predictions for each instance of a problem. In this worksheet the first column, column **A**, is descriptive and identifies each instance. The next three columns, **C**, **D**, and **E** (blank columns are OK), represent facts that describe this particular instance. The next column, **G**, also represents a fact, whether or not the mixture exploded, but in this case it also represents the known result. The next column, **I**, will be used by Neuralyst to present its outputs or predictions. The final column, **K**, is used primarily to distinguish between those instances with known results to be used in training the neural network and those instances which have been saved as tests or those instances without known results for which a prediction is desired. It also identifies those rows which are used for special purposes.



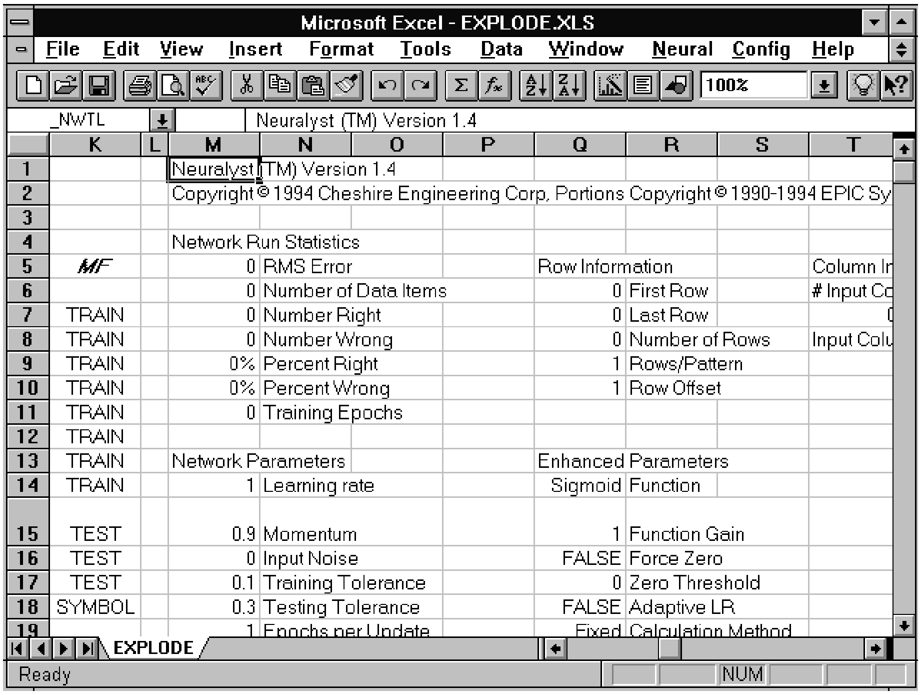
	A	B	C	D	E	F	G				
1	Explosive Chemistry										
2	Copyright (C) 1994, Cheshire Engineering Corp										
3											
4			*Inputs*				*Target*				
5	*Mixture*		*Pot. Nitrate*	*Charcoal*	*Sulfur*		*Explode*				
6											
7	#1		0.2	0.4	0.4		FIZZLE				
8	#2		0.4	0.4	0.2		FIZZLE				
9	#3		0.6	0.3	0.1		FIZZLE				
10	#4		0.7	0.1	0.2		BOOM!!				TRAIN
11	#5		0.7	0.05	0.25		FIZZLE				TRAIN
12	#6		0.75	0.2	0.05		FIZZLE				TRAIN
13	#7		0.8	0.1	0.1		BOOM!!				TRAIN
14	#8		0.9	0.05	0.05		FIZZLE				TRAIN
15	#X1		0.7	0.3	0		FIZZLE				TEST
16	#X2		0.75	0.1	0.15		BOOM!!				TEST
17	#X3		0.95	0.025	0.025		FIZZLE				TEST
18						FIZZLE,BOOM!!					SYMBO
19											

Initialize a new worksheet for Neuralyst

Figure 4-2 The Neuralyst Config Menu

The **Config** menu contains the interface to the commands that allow you to define a problem. Figure 4-2 shows the commands available from the menu.

The **Config** menu is organized in the same sequence as a problem should generally be defined for Neuralyst. The first item is **Init Working Area**. This will establish the area on the worksheet that is reserved for Neuralyst's parameters and data. This area should be the first cell that is to the right or below all the other data in the worksheet. No other data should be entered in or beyond the defined area as it may result in the incorrect operation of Neuralyst or it may be deleted as a result of one of Neuralyst's internal actions. In this example, move your cursor to cell **M1**, select it, pull the **Config** menu down and select the **Init Working Area** command. Neuralyst will ask you to confirm the initialization request; go ahead and click **OK**. The cursor will now show as an hourglass {watch} for a moment as Neuralyst initializes the Working Area. The results should appear as in Figure 4-3.


Microsoft Excel - EXPLODE.XLS
File Edit View Insert Format Tools Data Window Neural Config Help
100%
_NWTL
Neuralyst (TM) Version 1.4
EXPLODE
Ready
NUM

	K	L	M	N	O	P	Q	R	S	T
1			Neuralyst (TM) Version 1.4							
2			Copyright © 1994 Cheshire Engineering Corp, Portions Copyright © 1990-1994 EPIC Sy							
3										
4			Network Run Statistics							
5	MF		0	RMS Error			Row Information			Column In
6			0	Number of Data Items			0	First Row		# Input Co
7	TRAIN		0	Number Right			0	Last Row		0
8	TRAIN		0	Number Wrong			0	Number of Rows		Input Colu
9	TRAIN		0%	Percent Right			1	Rows/Pattern		
10	TRAIN		0%	Percent Wrong			1	Row Offset		
11	TRAIN		0	Training Epochs						
12	TRAIN									
13	TRAIN		Network Parameters				Enhanced	Parameters		
14	TRAIN		1	Learning rate			Sigmoid	Function		
15	TEST		0.9	Momentum			1	Function Gain		
16	TEST		0	Input Noise			FALSE	Force Zero		
17	TEST		0.1	Training Tolerance			0	Zero Threshold		
18	SYMBOL		0.3	Testing Tolerance			FALSE	Adaptive LR		
19			1	Epochs per Update			Fixed	Calculation Method		

Figure 4-3 Portion of Neuralyst Working Area

Now the range of the input instances needs to be defined. This is the purpose of the **Set Rows** command. Scroll the window to show column **A** and select the cells **A7** through **A18**. Now pull down the **Config** menu and select the **Set Rows** command. This will tell Neuralyst which rows it is to use as inputs to the neural network. The column used for this purpose is not critical: it may or may not be one of the Input columns to be defined next. A dialog box will appear after the selection has been accepted to let you set the number of rows per pattern and the number of rows to offset between patterns. Leave both at their default values of 1 for this example and confirm with OK.

The Input columns containing the facts for each instance are defined through the **Add Input Columns** command. The columns to be set as Input columns may either be selected one at a time or several columns may be selected as an extended selection. To do this, point to the column headings and select the three columns, **C** through **E**. Now pull down the **Config** menu again and select the **Add Input Columns** command. This will tell Neuralyst that these columns contain the input facts. Each row of these columns, as previously defined by **Set Rows**, being another instance to be presented to the neural network.

The known result or consequence of these facts is defined for the neural network through the **Add Target Columns** command. Select column **G** and then select this command on the **Config** menu. Similarly, the neural network's output or prediction for these facts after processing is defined through the **Add Output Columns** command. Select column **I** and then select this command on the **Config** menu. Multiple columns may be set as Target columns or Output columns, but this feature is not necessary for this example.

The final column to be defined is the Mode Flag column. When presenting a problem to Neuralyst, there needs to be some way to distinguish between those facts which are to be used for training purposes and either those facts which have known results but which are reserved to test the success of the training or those facts for which no results are known and for which the neural network prediction is desired. The Mode Flag column makes that distinction. Rows that have this column set to TRAIN will be treated as training sets. Rows that have this column set to TEST will be treated as "testing" sets. In the cases where the Target column values are defined (results are known), then these may be considered test rows. In the cases where the Target column values are not defined (results are not known), then these may be considered as facts requiring neural network prediction. To define the Mode Flag column, select column **K** and select the command **Set Mode Flag Column** from the **Config** menu.

The Mode Flag column is also used to indicate a number of special rows that are useful to modify the behavior of Neuralyst. There are three additional rows or row types that can be designated, these are: SYMBOL, MIN, and MAX. A row designated by SYMBOL is interpreted by Neuralyst to contain definitions of symbols for Input or Target columns. A row designated by MIN is interpreted by Neuralyst to contain definitions of the minimum range limit for Input, Target, or Output, columns. A row designated by MAX is interpreted by Neuralyst to contain definitions of the maximum range limit for Input or Target columns. If MIN or MAX rows are set, the limits entered for each target column are applied to the corresponding Output column. To enter row **18** as a SYMBOL row, select row **18** and select the command **Set Mode Rows** from the **Config** menu. When the dialog box appears, select the **Set Symbol Row** option and confirm **OK**.

The **Edit Column Lists** command allows you to view the column labels entered under each column type and to change them if desired. Select **Edit Column Lists** from the **Config** menu to see what this looks like. You can try making some changes, but be sure to restore the labels to their original values and confirm **OK** when you are done.

The **Select Data Mode** command allows you to classify the data for your problem between a training set and a testing set. This is done through a number of options which will set the Mode Flag column as TRAIN or TEST for you to indicate training or testing.

The **Edit Mode Lists** command allows you to view the settings of SYMBOL, MIN, and MAX fields for each designated column and to change them if desired. Select column **G** and then select **Edit Mode Lists** from the **Config** menu to see what this looks like. You can try making some changes, but be sure to restore the labels to their original values and confirm **OK** when you are done.

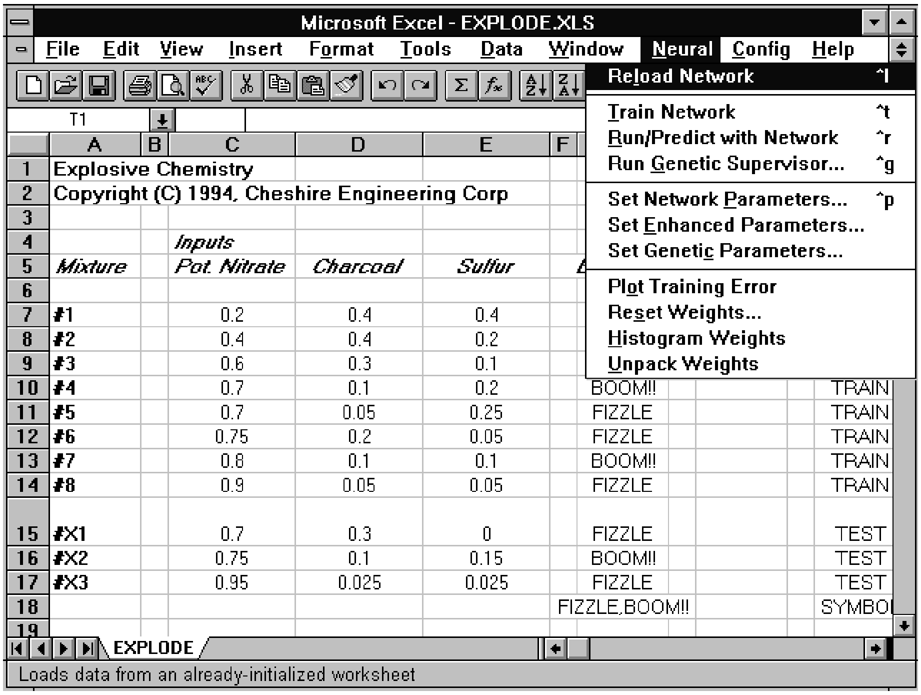
Once the problem data has been defined, the neural network structure needs to be defined. To do this, select the **Set Network Size** command from the **Config** menu. A dialog box will appear requesting you to input the number of layers for the neural network. The default is 2, but in this instance you should enter 3 and then confirm **OK**. Another dialog box will now appear. This dialog box shows the input and output layers with a fixed number of neurons determined by the number of Input Columns and Target columns specified. In this case the input layer is layer 1 which has three neurons and the output layer is layer 3 which has one neuron. There is an additional layer, layer 2, which starts with a default of one neuron. Change this to 3 and confirm **OK**. The hourglass {watch} cursor will appear for a moment as Neuralyst builds the neural network, then Neuralyst will display the Network Weights, as they have been initialized in the Working Area.

This completes the configuration process, but before you proceed to running Neuralyst, you should take a moment to step back and review the worksheet as it appears now. This will give you a general sense of how a Neuralyst worksheet is organized. First there is a Descriptive Area, which is usually located above or to the left of the Problem Definition Area. This includes column titles, information about the worksheet and other descriptive text. Then there is the Problem Definition Area, within which there will be a set of Input columns which represent facts to be presented, a set of Target or goal columns which represent known results, a set of Output columns for the neural network to present its outputs or predictions and a Mode Flag column which allows you to specify training instances, testing or prediction instances, and special definition rows used by Neuralyst. Finally, there is a Neuralyst Working Area which Neuralyst uses for its operational purposes. The Working Area will usually be the rightmost or bottommost part of the worksheet.

4.3 Running the Neural Network

The Neural menu provides the commands that allows you to control the operation of the neural network.

The first command in the menu, **Reload Network**, allows a saved Neuralyst worksheet to be reloaded. The next three, **Train Network**, **Run/Predict with Network**, and **Run Genetic Supervisor**, determine the operating mode of the neural network. The next, **Set Network Parameters**, allows you to set parameters which control the operation of the neural network. **Set Enhanced Parameters** allows you to enhance the behavior of the neural network from the standard backpropagation neural


Microsoft Excel - EXPLODE.XLS
File Edit View Insert Format Tools Data Window Neural Config Help
Reload Network ^l
Train Network ^t
Run/Predict with Network ^r
Run Genetic Supervisor... ^g
Set Network Parameters... ^p
Set Enhanced Parameters...
Set Genetic Parameters...
Plot Training Error
Reset Weights...
Histogram Weights
Unpack Weights
T1
EXPLODE

	A	B	C	D	E	F			
1	Explosive Chemistry								
2	Copyright (C) 1994, Cheshire Engineering Corp								
3									
4			*Inputs*						
5	*Mixture*		*Pot. Nitrate*	*Charcoal*	*Sulfur*				
6									
7	#1		0.2	0.4	0.4				
8	#2		0.4	0.4	0.2				
9	#3		0.6	0.3	0.1				
10	#4		0.7	0.1	0.2		BOOM!!		TRAIN
11	#5		0.7	0.05	0.25		FIZZLE		TRAIN
12	#6		0.75	0.2	0.05		FIZZLE		TRAIN
13	#7		0.8	0.1	0.1		BOOM!!		TRAIN
14	#8		0.9	0.05	0.05		FIZZLE		TRAIN
15	#X1		0.7	0.3	0		FIZZLE		TEST
16	#X2		0.75	0.1	0.15		BOOM!!		TEST
17	#X3		0.95	0.025	0.025		FIZZLE		TEST
18						FIZZLE,BOOM!!			SYMBOL

Loads data from an already-initialized worksheet

Figure 4-4 The Neuralyst Neural Menu

network described in Chapter 3. **Set Genetic Parameters** allows you to set parameters which control the Genetic training supervisor which can be used to optimize the definition and configuration of the neural network. The **Plot Training Error** command allows you to view the progress of training. The next three, **Reset Weights**, **Histogram Weights**, and **Unpack Weights**, allow you to set and view the neural network weights, providing access to the representations of learning the neural network has undergone.

[If you have installed the Trader's Macro Library or other supplementary libraries, they will appear as additional commands at the end of the Neural menu.]

For now, we can start by setting a parameter and then proceed to train the network. Select the **Set Network Parameters** command from the **Neural** menu. A dialog box will appear showing several parameters, **Learning Rate**, **Momentum**, **Input Noise**, **Training Tolerance**, **Testing Tolerance**, **Epochs per Update**,

Epoch Limit, **Time Limit**, and **Error Limit**. Each of these will have a default value.

Learning Rate determines the magnitude of the correction term applied to adjust each neuron's weights when training. Learning Rate must be positive, is adjusted in the range of 0 to 1 and has a default value of 1. Large values of Learning Rate will cause the network to train more quickly, but too large a value may cause the training to be unstable and no learning will occur.

Momentum determines the "lifetime" of a correction term as the training process takes place. Momentum must be greater than or equal to 0 but less than 1 and has a default of 0.9. Values of Momentum closer to 1 will cause the neural network to retain more of the impact of previous corrections to the current corrections. Values of Momentum close to 0 will allow mostly or only the current corrective term to have an effect. Momentum helps to smooth out the training process so that no single aberrant instance can force learning in an undesirable direction.

Input Noise provides a slight random variation to each input value for every training epoch. As training occurs, this has the effect of preventing the neural network from learning the exact input values. Ideally, this will prevent overtraining and improve the generalization process. Input Noise has a range of 0 to 1, but small values of Input Noise are generally the most useful. Input Noise represents a percentage of the range for an input, for example a value of 0.1, or 10%, means that a noise level of up to 10% of the input range will be applied. Input Noise is set to 0 as a default.

Training Tolerance defines the percentage error allowed in comparing the neural network output to the target value to be scored as "Right" during the training process. The Training Tolerance should be between 0 and 1 and has 0.1, or 10%, as a default. The Training Tolerance value has no effect on the learning algorithm. However, when Neuralyst finds 100% Right, as defined by Training Tolerance, it will automatically stop training.

Testing Tolerance is similar to Training Tolerance, but it is applied to the neural network outputs and the target values only for the test data (as defined by the value of the Mode Flag column). Neural network output and test target values are scored as "Right" if they are within the Testing Tolerance. Otherwise they are scored as “Wrong”. The Testing Tolerance should be between 0 and 1 and has 0.3, or 30%, as a default. Testing Tolerance may be set to the same limit as Training Tolerance, but is often set to a less restrictive value since prediction is usually less exact than training.

Epochs per Update allows you to control the number of epochs between updates of the neural network results which are displayed in the Network Run Statistics block in the worksheet. An epoch is one complete processing run through all defined training cases. Epochs per Update has a default value of 1. However larger values will mean less frequent communication between the neural network and the worksheet, reducing overall training time.

Epoch Limit sets a maximum number of training epochs the neural network will undergo in those situations where you wish to control the number of training epochs rather than setting a Training Tolerance. Epoch Limit has a default value of 0, which means that there is no limit set.

Time Limit sets a maximum amount of time that the training of the neural network will undergo. This is useful if Neuralyst may be left unattended during the training process. Time Limit has a default value of 0, which means that no limit is set.

Error Limit sets a limit for an increase in the training error. Generally a neural network will steadily reduce the training error. If too much training occurs or if the neural network has insufficient capacity or an inappropriate configuration for the problem, then it is possible for the training error to increase. Error Limit allows these conditions to terminate training. Error Limit has a default value of 0, which means that no limit is set.

When one or more limits are set, the first limit that occurs, or if no limit occurs, the achievement of training within the Training Tolerance will terminate training.

Only Epochs per Update should be changed in this example. Enter 50, indicating 50 epochs between worksheet updates, for Epochs per Update. Confirm **OK**.

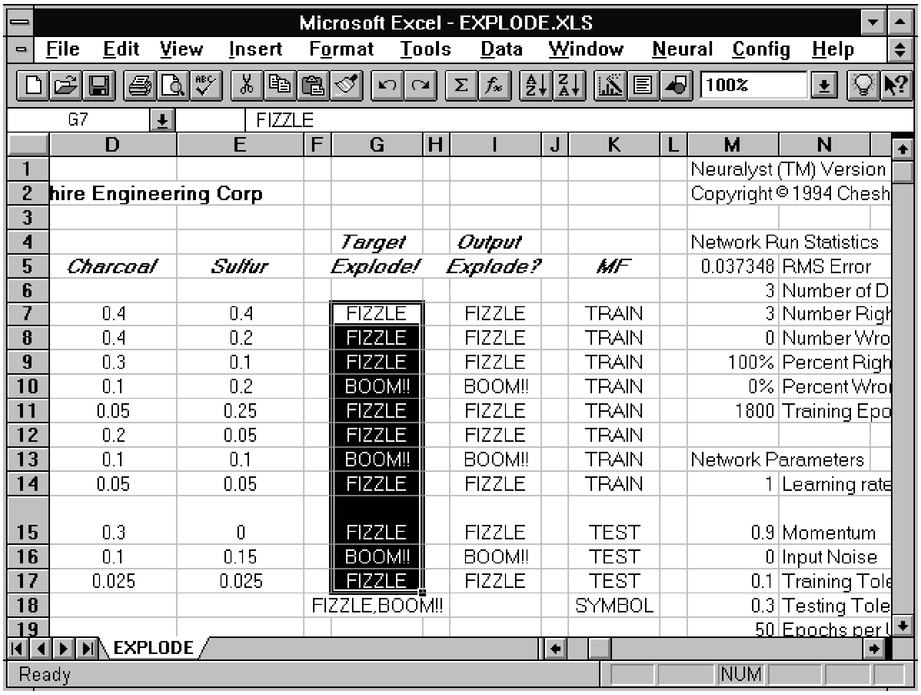
Finally, select the **Train Network** command from the **Neural** menu. If the configuration steps have been followed properly, the window will shift so that the Network Run Statistics block is visible and the first cell in that block, RMS Error, will be changing quickly. As training progresses, the RMS Error will be decreasing and the scores in Right and Percent Right will be increasing.

While the neural network is training, you can cause training to pause by typing the **Esc** {**Esc** or **cmd-.**} key. Try it. When the **Esc** {**Esc** or **cmd-.**} key is typed, Neuralyst will stop at the next update point and save its current work and after a moment allow you to access the worksheet. Training can be resumed by selecting the **Train Network** command from the Neural menu again. Do that now. After a few minutes (depending on the speed of your computer), the Right and Percent Right scores will be 8 and 100%, respectively. The neural network will have been trained.

When Neuralyst concludes training the neural network, it will place the current values of the neural network output in the Output column, **I**. The values in this column should match those in column **G**, the targets used to train the neural network. When the Target column is symbolic, then the Output column is filled with symbols that correspond most closely to the actual numeric values that are processed by the neural network. When the Target column is numeric, then the Output column is filled with the actual numeric values. In the numeric case, the effect of Training Tolerance is more visible as the variation from the target values can be calculated.

With the neural network trained, Neuralyst is now ready to run the neural network to predict the outcomes of experiments X1 through X3. The “known” results (based on a fictitious formula that models the behavior of all the experiments in the `EXPLODE.XLS` {`Explode`} world) have been entered in the last three lines of the Explode! column, but when Neuralyst runs the neural network in predictive mode it will not look at these in making its prediction. These results will only be used in scoring the success of the neural network.

To test the neural network, select **Run/Predict with Network** from the **Neural** menu. The results of the run will be placed in the last three rows of the Explode? column, corresponding to X1 through X3, and the scoring will now be updated to reflect the results of this test run rather than the previous training runs. The Right and Percent Right scores will be 3 and 100%, reflecting the testing of these three additional experiments and the comparison of the test outputs against the known values.



G7 FIZZLE

	D	E	F	G	H	I	J	K	L	M	N
1										Neuralyst (TM) Version	
2	hire Engineering Corp									Copyright © 1994 Chesh	
3											
4				*Target*		*Output*				Network Run Statistics	
5	*Charcoal*	*Sulfur*		*Explode!*		*Explode?*		*MF*		0.037348	RMS Error
6										3	Number of D
7	0.4	0.4		FIZZLE		FIZZLE		TRAIN		3	Number Righ
8	0.4	0.2		FIZZLE		FIZZLE		TRAIN		0	Number Wro
9	0.3	0.1		FIZZLE		FIZZLE		TRAIN		100%	Percent Righ
10	0.1	0.2		BOOM!!		BOOM!!		TRAIN		0%	Percent Wro
11	0.05	0.25		FIZZLE		FIZZLE		TRAIN		1800	Training Epo
12	0.2	0.05		FIZZLE		FIZZLE		TRAIN			
13	0.1	0.1		BOOM!!		BOOM!!		TRAIN		Network Parameters	
14	0.05	0.05		FIZZLE		FIZZLE		TRAIN		1	Learning rate
15	0.3	0		FIZZLE		FIZZLE		TEST		0.9	Momentum
16	0.1	0.15		BOOM!!		BOOM!!		TEST		0	Input Noise
17	0.025	0.025		FIZZLE		FIZZLE		TEST		0.1	Training Tole
18			FIZZLE,BOOM!!					SYMBOL		0.3	Testing Tole
19										50	Epochs per U

Figure 4-5 Final Screen for EXPLODE.XLS {Explode}

4.4 Finishing Up

That's it! We've just configured and run a neural network, had it learn some of the rules of chemistry of the `EXPLODE.XLS` {`Explode`} world, and been able to use it to predict the outcome of additional experiments that the neural network had not seen before!

To save this work, this Neuralyst worksheet can be saved like any other Excel worksheet: from the **File** menu choose the **Save** or **Save As** command. The **Exit** {**Quit**} command (also in the **File** menu) can be used to exit; Neuralyst will unload along with Excel.

Chapter 5

Learning More About Neuralyst

Neural networks are really very simple in concept. However, like their biological counterparts, this simplicity is the foundation for highly complex behavior and sophisticated capabilities.

In chapter 4 the basic capabilities and operation of Neuralyst were discussed. This chapter focuses on several additional facets of Neuralyst and neural network behavior. The examples in this chapter illustrate these points as well as demonstrating a broad, though by no means complete, range of possible applications. At the start of each section, load the example indicated and explore it while reading the discussion. But don't stop there, experiment and see what happens!

The first six examples are based on idealized scenarios (much like `EXPLODE.XLS` {`Explode`}) with fairly simple rules of behavior. This has been done to allow the principles of behavior of neural networks to be demonstrated with fairly small data sets. In general, real data sets will be "noisier". That is, they will not have values that conform perfectly to a hidden model; instead the values will tend to vary around some "true" value with such variations being small to large depending on the circumstances. In order for neural networks to perceive the structure that may exist underlying such variations, more data must generally be presented and more time be spent in training.

In the next two examples, we will depart from the idealized situations and move on to real world data. Both of these will deal with investment analysis. The first is based on fundamental analysis, that is the forecasting of a stock or commodity's future price movements from data relating to a company's revenues, earnings, debt, equity, rates of returns, dividends, and so on. The second is based on technical analysis, that is the prediction of a stock or commodity's future price movements from past price movements. Fundamental analysis is generally considered a long-term approach, with forecasts ranging from many months to a few years. Technical analysis is generally considered a short-term approach, with predictions ranging from a few minutes to many weeks.

Finally, the last example provides a demonstration of Neuralyst's two-dimensional analysis and pattern matching capabilities through the recognition of patterns and shapes. Neuralyst's multi-dimensional analysis capabilities can be applied to a variety of applications, including: image processing, character recognition, and so on.

At the conclusion of the discussion and examples, you will have a much better understanding of the capabilities and limitations of neural networks and how to go about preparing a problem for Neuralyst to analyze.

5.1 Parity Generator — PARITY.XLS {Parity}

`PARITY.XLS` {`Parity`} contains a slightly more sophisticated example of a computer logic operation than shown in the first Neuralyst example, `LOGIC.XLS` {`Logic`}. Like that example, `PARITY.XLS` {`Parity`} works with the binary representation, 0's and 1's, of computer data and is a key function in computer operations.

`PARITY.XLS` {`Parity`} demonstrates the operation of *parity* generation for computer data. You may be aware that many computers, including PC's and Macintosh's, use the parity check operation to verify data in computer memory (you may have seen the message "`PARITY CHECK`" followed by a hung computer when the check fails on a PC).

Remember that computer data is stored in groups of eight bits, known as a *byte*. For each byte, the computer's parity circuits count the number of 1's present in the byte. If the count is odd, then there is odd parity; if the count is even, then there is even parity. When the byte is written, the parity is saved in a ninth bit, known as the parity bit. When the byte is read, the parity of the byte is checked against the previously saved parity bit. If there has been no change while the data resided in memory, then the parity as read will be the same as the parity when written. If there is a difference, then one of the bits must have changed (a 0 changing to 1 or a 1 changing to 0 will change the number of 1's and thereby disturb the parity) and the data has been corrupted. When this occurs, the computer stops since it is safer to stop operations than to try and continue with bad data that may result in additional problems.

In the PARITY.XLS {Parity} example, there are only four bits (sometimes known as a *nibble*) of input, rather than the eight bits present in a full byte. These four bits can represent any number from 0 to 15 and those are the values listed in the example. (If all eight bits had been used, the example would range from 0 to 255 — too many data lines for a simple demonstration.)

There are two Target columns, indicating the parity of the nibble, even or odd (zero 1's being counted as even). There are also two Output columns reserved for the neural network results. To run this example:

1. **Init Working Area** — starting at **N1**
2. **Set Rows** — **6** through **21**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns** — **C**, **D**, **E**, and **F**
4. **Add Target Columns** — **H** and **I**
5. **Add Output Columns** — **K** and **L**
6. **Set Network Size** — 3 Layers, 8 Hidden Neurons
7. **Set Network Parameters** — Training Tolerance to 0.2, Epochs per Update to 20

[If the steps described in this "short" form are not clear to you, please review Chapter 4, which goes through the entire process of configuring a network in great detail.]

With this configuration, you can start training. While this problem is training, pay particular attention to the RMS Error value. Normally, successful training is indicated by a steady decrease in the RMS Error value. If nothing seems to be happening after a while, try stopping the training and use the **Plot Training Error** command to look at the training progress. Resume training and look again after a while. You may notice that there sometimes periods when the error value doesn't seem to make much progress (it may even lose ground for a bit) and then there are other periods when the error value is reduced at a steady rate or even jumps downward. After some time Neuralyst will stop and the Output columns will match the Target columns; Neuralyst has learned to generate the parity of any four bit value!

As it turns out, the data in this problem has a characteristic that is particularly hard for neural networks. That is, very similar inputs lead to very different outputs. For every value in this example, there is another value that is different in only one input bit, yet has the opposite output value! Despite this, the neural network was able to learn the data. Still, this kind of characteristic in the input data can lead to some long training sessions if the problem data, training parameters, or network size are poorly set.

Even worse than data that is structured like this is *contradictory* data. That is data that has two or more input sets that match while they have very different outputs; for example, having the binary representation of 2 be even parity and later on having another instance where the binary representation of 2 is now odd parity. Such contradictions must be removed as the neural network generally cannot resolve these unless the error tolerance is set so loosely that the outputs are often useless.

Now, let's go back to the behavior we mentioned. Those periods, when error reduction seemed to make little progress, are known as *learning plateaus*. When this phenomenon occurs, it is generally believed that the neural network is undergoing a generalization process and developing internal representations of relevant characteristics of the data. In fact when these plateaus occur, the neural network is probably learning the most, even though the error value is changing the least!

Conversely, when the error value is making rapid progress in reduction, this generally means that the neurons have already sorted themselves out and the corrections are being applied with maximum effect to each neuron.

Sometimes it takes multiple plateaus, wherein new or additional distinctions or generalizations are made each time, followed by another phase of rapid error reduction, before the neural network is able to complete its learning.

Watch for this kind of behavior. This will help you in understanding what is happening with the neural network and may give an indication of whether the network has been properly sized for the problem.

To see what happens with marginally sized networks, rerun the problem by giving the **Set Network Size** command again. This time, set 3 layers and 4 hidden neurons. (You will be warned that this will cause any learning done so far to be forgotten; click on **OK**.) Then start training. What happens? Repeat this a few times. You will find that sometimes the neural network trains properly and other times it seems to reach a permanent plateau at some point, with Neuralyst continuing to run since it is not able to achieve the required error tolerance. There exists a solution for 4 neurons, but the neural network is not always able to find the solution!

The neural network training process can be thought of as an exploration of the weight space, all the different possible combinations of weight values, until a weight set is found that produces the desired targets. For this particular problem, with this particular neural network configuration, there exists *local minima* in the weight space. These are points in the weight space that “trap” the neural network; the backpropagation algorithm not being able to move out of that region to find the correct weight set. When local minima exist, one solution is to try increasing the number of neurons until the neural network is able to train consistently.

Another experiment to try is to change one or more lines of data so that contradictory cases are presented. (Anytime you change input or target data values, you must give the **Reload Network** command so that Neuralyst is aware that there have been changes and that it must pick them up.) Also, try different settings of the Training Parameters and Network Size on this problem, with and without contradictory data. Observe the learning behavior in each case.

Important Points:

> Neural networks have a difficult time learning when inputs having small distinctions between them require outputs with large distinctions between them.
>
> Neural networks cannot learn properly from contradictory data.
>
> Neural networks often experience learning plateaus; these are probably phases of neural network development during which distinctions and generalizations are made.
>
> Neural networks must have sufficient network capacity (size) to learn.

5.2 Paper-Rock-Scissors Game — PAPER.XLS {Paper Game}

PAPER.XLS {Paper Game} contains an example that is actually structurally very similar to PARITY.XLS {Parity}. Like that example, PAPER.XLS {Paper Game} works with the representations <PAPER, ROCK, SCISSORS> and <BONNIE, TIE, CHRIS> of possible input values and outcomes. As with PARITY.XLS {Parity}, the goal is to learn the rules of the game, and the rules define distinct and sometimes opposite outcomes for various changes in inputs.

Note that <PAPER, ROCK, SCISSORS> and <BONNIE, TIE, CHRIS> are ternary, that is three-valued, inputs and outputs. This example demonstrates two things, the symbolic capabilities of Neuralyst and the additional multi-valued capabilities of Neuralyst.

There are two Input columns that are set to the three possible choices of the two players Bonnie and Chris, that is Paper, Rock, or Scissors. There is a Target columns indicating who wins, or if it was a tie. There are also a matching Output columns reserved for the neural network results. To run this example:

1. **Init Working Area** — starting at **J1**
2. **Set Rows** — **6** through **15**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns** — **A**, and **B**

4. **Add Target Columns — D**
5. **Add Output Columns — F**
6. **Set Mode Flag Column — H**
7. **Set Mode Rows — 15** Set Symbol Row
8. **Set Network Size** — 3 Layers, 8 Hidden Neurons
9. **Set Network Parameters** — Training Tolerance to 0.2, Epochs per Update to 20

With this configuration, you can start training. After some time Neuralyst will stop and the Output column will match the Target column; Neuralyst has learned the rules of the Paper-Rock-Scissors game!

The comments and discussion for the `PARITY.XLS` {`Parity`} example are also relevant here and you can make many of the same experiments to learn more about how neural networks behave with different kinds of data.

Important Points:

Using symbolic representations provides a more natural way to express certain types of problems.

Neural networks can deal with a variety of multi-valued inputs and outputs.

5.3 Sine Wave — SINE.XLS {Sine}

`SINE.XLS` {`Sine`} contains an example of how Neuralyst can match and predict values for a complex mathematical function with just a few data points. In contrast to the computer logic operation that was shown in the `PARITY.XLS` {`Parity`} example, `SINE.XLS` {`Sine`} uses continuous real numbers rather than the binary representation, 0's and 1's, of computer data.

`SINE.XLS` {`Sine`} demonstrates the operation of *interpolation* on mathematical data. Many complex operations can be approximated by a mathematical function; for a given input value, there is a corresponding output value. For real world behavior, it is common to have a few sampled or measured values of the function, but not a complete description or every value of the function. From just a few samples, Neuralyst can often interpolate the values of the function that were not previously known.

The sine function is a familiar mathematical function from high-school trigonometry. It is an important function because it is used in every area of science and engineering. It is an interesting function because it is known as a *transcendental* function. Transcendental functions are harder to describe or generate than most functions that are familiar from high-school algebra. In particular, the values of the sine function are normally derived by computing an infinite series of terms. The input to a sine function is any real value, though in the example training is limited to 0 to 6.28, or 2π, and the output is any real value from -1 to 1.

In the `SINE.XLS` {`Sine`} example, there is only one Input and one Target column. There is also a corresponding Output column to match the Target column. To run this example:

1. **Init Working Area** — starting at **L1**
2. **Set Rows — 5** through **140**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns — A**
4. **Add Target Columns — B**
5. **Add Output Columns — C**
6. **Set Mode Flag Column — D**
7. **Set Network Size** — 3 Layers, 3 Hidden Neurons
8. **Set Network Parameters** — Momentum to 0,
   Training Tolerance to 0.04,
   Epochs per Update to 50

With this configuration, you can start training. After a few minutes Neuralyst will be done training. In the example, some well-spaced samples for a single cycle of a sine wave are selected and used for training. The remaining points are used for testing and plotted against an exact sine wave for comparison. After training is complete do a Run/Predict to fill in the previously unknown points. Look at the resulting comparison chart. Neuralyst has generalized the shape of a sine wave from just a few sample points!

The interpolation is off by only a few percent at the worst points and at many points is almost exact. The function can be made more exact with more training time, more neurons or more data points used for training. More training time allows the neural network more time to adjust its weights to a better solution. More neurons allows the neural network more capacity to develop a model of the sine wave. More data points gives the neural network more information to constrain the approximation of the sine wave at those points that are changing rapidly and are far away from an input training case. Try experimenting with which of the three variations is most successful in generating a more exact interpolation. Also observe that there is a limit to how exact the neural network can be.

In addition to the above experiments, Neuralyst allows you to control two parameters, Calculation Method and Scaling Margin in the **Set Enhanced Parameters** dialog box, which can also affect speed of training, precision and accuracy. Try training with Calculation Method set to Floating Point versus the default Fixed Point method. Also try adjusting the Scaling Margin from 10% to 50%. While adjusting those two parameters, try tightening Training Tolerance to 0.03, 0.02 ore even 0.01 (that is, 3%, 2% or even 1%).

Important Points:

> Neural networks can approximate and interpolate continuously valued functions with relatively few training points.
>
> Despite the capabilities available with relatively few training cases, more training cases will generally provide better training.
>
> The choice of Calculation Method can affect the training of certain types of problems.

The setting of Scaling Margin can also affect the training of certain types of problems.

5.4 Criminal Mugbook — MUGBOOK.XLS {Mug Book}

So far there have been three examples, LOGIC.XLS {Logic}, PARITY.XLS {Parity} and PAPER.XLS {Paper Game}, which have demonstrated a neural network's capabilities to learn and reproduce rules from examples of those rules and two examples, EXPLODE.XLS {EXPLODE} and SINE.XLS {SINE} which have demonstrated a neural network's capabilities to generalize and predict from known facts. MUGBOOK.XLS {Mug Book} will demonstrate an example of how neural networks can also be used for *pattern matching* or as an *associative memory*.

You are probably familiar with the concept of a mugbook, a book of photos used by the police to help witnesses match the physical characteristics of known criminals against the features of a suspect the witness has seen. In some cases, there are problems in the identification process since the witness is not completely sure of the match due to the uncertainty of their memory, poor visibility at the time of the crime or changes in outward characteristics, for example, weight gain, shorter hair length or deliberate disguise.

MUGBOOK.XLS {Mug Book} is a simplified example of a mugbook, based on four physical characteristics: sex, age, coloring, and weight of eight known criminals in Midtown, U.S.A.. Most of these characteristics have been converted to a symbolic value, using the translation shown under each column of the worksheet. For example, age has been broken into decades, coloring has been segmented into three groupings: light, medium, and dark, and so on. Each of the eight criminals has also been assigned an ID number from 1 to 8.

The Target and Output columns are set up as eight separate indicators, each one representing a different ID. A 1 under an ID indicates that the criminal with that ID is completely identified, a 0 under an ID indicates that criminal is completely rejected. For a solid ID, all indicators should be 0 except for one that contains 1. There are two reasons why the outputs have been organized in this fashion. Let's discuss these for a moment.

First, while neural networks can make fine distinctions, there is a limit to the number of distinctions, that is different output values, that a single neuron can meaningfully take on. This can result in self-deception if you are not careful. You can train the neural network to produce the actual output values to match the target values (within the Training Tolerance), no matter how fine the distinction, and so believe that the neural network has made these distinctions. But when the neural network is run, these distinctions will not be successfully reproduced.

There is no hard rule as to how many distinctions can be made successfully by an output, but numbers beyond 4 to 8 are generally difficult. In this case, with 8 ID's to match, we have established a separate output neuron for each ID.

The second reason anticipates the conclusion of the demonstration to a certain extent. Once the network has been trained on the known criminals, we will present the characteristics of an unknown to try and match against the known ones. The use of separate outputs for each ID allows the neural network to use the full output range to indicate the quality of the match for the known characteristics of each ID against the characteristics of the unknown person.

To clarify this some more, if an unknown had some of the characteristics of ID 3 and some of the characteristics of ID 5, the only way a single output could express this would be by presenting 4. You would have no way of distinguishing this output from an actual match with ID 4 or partial matches between two or more ID's that averaged 4. With separate outputs, the outputs for ID 3 and ID 5 could each present a fractional value, indicating a partial match, without any confusion.

To run this example:

1. **Init Working Area** — starting at **AC1**
2. **Set Rows** — **9** through **18**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns** — **D**, **E**, **F**, and **G**

4. **Add Target Columns — I** through **P**
5. **Add Output Columns** — **R** through **Y**
6. **Set Mode Flag Column — AA**
7. **Set Mode Rows — 18**, Set Symbol Row
8. **Set Network Size** — 3 Layers, 6 Hidden Neurons
9. **Set Network Parameters** — Training Tolerance to 0.2, Epochs per Update to 20

With this configuration, you can start training. After a short time Neuralyst will be done. Neuralyst has learned the distinguishing characteristics of the eight known criminals! Any suspect with exactly the same characteristics as one of these known criminals will immediately produce a match. This capability is similar to the rule reproduction capability already demonstrated before.

However, this is somewhat more powerful than may be obvious from this simplified example. The reason is that this capability can be extended to hundreds of characteristics and thousands of criminals (or any other search objects). This is the same function performed by conventional computer databases. However, computer designers know that searching and matching in large databases are among the most time consuming database operations. On the other hand, a neural network trained to the same data as a large database could “retrieve” a match with just one processing operation!

A more sophisticated capability than this will soon be apparent. Select **Run/Predict with Network** from the **Neural** menu to run the network. The characteristics of Mr. X are processed and the eight outputs in that row now have fractional values in them. These fractional values represent the neural network’s assessment of how closely Mr. X matches characteristics of the known criminals. The neural network is able to generate indications for the closest matches even though it didn’t find an exact match!

The strongest outputs are likely to be for ID 5 and ID 7, with other outputs perhaps showing a response. While there is some information in the relative values of the matches, these values should not be taken at first inspection as exact measures of closeness or probability. There are three reasons for this.

First, remember that the Training Tolerance was set to 0.2 or 20% of the output range, thus we can't expect predictions to 5% when we trained to tolerances of 20%. However, there is a danger to training too strictly. It is possible that a neural network tries so hard to match the exact values in training that it loses its generalizations. This is called *overtraining*. This is particularly likely to occur when there are too many hidden layer neurons, too few training samples and too much training time. In this case, what happens is that the neural network has so much capacity in relation to the data it must learn, that it can afford to match the outputs rather than to generalize. In essence, it is easier for the neural network to build an internal "crib sheet" rather than understand the structure of the data!

Second, we don't know exactly what characteristics the neural network has determined are relevant (this is particularly important with small sample sets, as in these examples, where there will be fewer or no samples to contradict bad generalizations) and there is often no way to find out without experimenting with the neural network. Theoretically it should be possible to understand the model developed by the neural network through a detailed examination and understanding of the weights, but in practice these values and relationships are often too complex for this to be attempted.

Third, for complex systems, the weights that a neural network uses to begin training (which are randomly assigned with each new configuration) can determine which one of a few (or many) possible solutions is actually found. This is why rerunning some of these problems with a new set of weights may result in slightly different solutions. The fact that the results are sometimes slightly different doesn't mean that the neural net is giving incorrect answers. Instead, it means that the data presented to the neural network admits of more than one solution.

Having considered these limitations, in the context of this example and with our current understanding of this neural network's behavior, it is best to say that the neural network considers ID 5 and ID 7 to be strong candidates, while the other ID's with weaker outputs are weaker candidates, without placing too much emphasis on exactly how much stronger or weaker these candidates are with respect to each other.

However, you can experiment with this neural network's behavior. When you have learned enough, perhaps you could say more. Try decreasing the Training Tolerance by steps of 0.05 from 0.4 to 0.1, training on the known criminals and running on the unknown after each change. You may want to try using the User Set Randomization option with **Reset Weights** for these experiments in order to start from a standard set of initial weight values (see Section 8.2.9). What happens to the values of the outputs? Try increasing the number of hidden layer neurons in the neural network and retraining. What happens when the unknown is matched now? Try adding more criminals with characteristics spaced evenly from each other and those already in the training set. Does the neural network do a better job of finding matches?

Important Points:

Neural network outputs should not be designed to make many fine distinctions.

Several separated neural network outputs can convey more information than a few combined outputs.

Setting the Training Tolerance more tightly than is necessary may interfere with generalization within the neural network.

Too much network capacity (size) and excessive training time may let the neural network "crib" rather than learn.

Neural networks require comprehensive, well-sampled training data in order to develop good generalizations

5.5 Credit Rater — EZCREDIT.XLS {EZ Credit}

EZCREDIT.XLS {EZ Credit} provides another demonstration of neural network pattern matching. EZCREDIT.XLS {EZ Credit} is similar in concept and structure to MUGBOOK.XLS {Mug Book}, but it provides an example of how the input types may also be categorized by indicators using binary values, in a similar fashion to the outputs in MUGBOOK.XLS {Mug Book}.

In this demonstration the Credit Approval Manager for EasyCredit Corporation has set up a database containing individuals distinguished by four characteristics: Income, Credit Experience, current Debt Burden, and prior Bankruptcy status. These characteristics are then matched to the actual credit history of the individuals EasyCredit has compiled from working records. EasyCredit expects that once the neural network is taught on its database of current clients, it will be able to use the neural network to rate new applicants.

Since the information contained in the credit records is varied, some numerical, some categorical, and some yes/no types, the manager has chosen to break each input type into one or more classifications or subdivisions that she feels are meaningful without being too fine. For each input type, the valid classification will be indicated by a 1, while the other classifications will be indicated by a 0. For example, for Income, she has chosen three classes: 0-30, 30-60, and 60+. She knows that these income ranges tend to define breaks where people have moderate, good, and excellent, ability to repay loans, respectively.

It is also possible that more than one class may be valid. For example, in the case of Credit Experience, she has identified the three most meaningful classifications as those people who have no credit cards, those with a department store credit card or those with a major bank credit card. Since a person can have both store credit cards and bank credit cards, a 1 could be entered for each subdivision, if appropriate.

Each of the four major credit characteristics have been classified in this way and entered into the worksheet for 16 customers. For these customers, the credit risk, represented by the payment history actually experienced by the company, is listed as Low, Medium, or High. A test case, Joe Applicant, is shown in the last row.

To run this example:

1. **Init Working Area** — starting at **AA1**
2. **Set Rows — 8** through **24**, 1 Row/Pattern, 1 Row/Shift

3. **Add Input Columns — C**, **D**, **E**, **G**, **H**, **I**, **K**, **L**, **M**, and **O** (Note the omission of columns **F**, **J**, and **N**! Perform this operation as four separate Add Input Columns — C,D,E then G,H,I then K,L,M then O.)

4. **Add Target Columns — Q**, **R**, and **S**

5. **Add Output Columns — U**, **V**, and **W**

6. **Set Mode Flag Column — Y**

7. **Set Network Size** — 3 Layers, 4 Hidden Neurons

8. **Set Network Parameters** — Training Tolerance to 0.2,
   Testing Tolerance to 0.4,
   Epochs per Update to 10

With this configuration, use the **Train Network** command to begin training. Neuralyst will stop after a short time. At this point it has taken the credit records of EasyCredit's past customers and established from this database the characteristics that contribute to credit rating and whether each characteristic does so positively or negatively!

When it is done, use the **Run/Predict with Network** command to evaluate the prospects of Joe Applicant. You will find that Neuralyst predicts Joe will most likely be a Medium credit risk, though he has a few characteristics of a High credit risk. This matches the Medium credit risk rating given to him by our (hidden) scenario rules.

There is nothing preventing finer subdivisions. Income could be broken down into increasing increments of ten thousand. Bank credit cards could be expanded to separately indicate Visa, MasterCard, or American Express. It is quite possible that the neural network will be able to make finer judgments with this additional information. The disadvantage to much finer subdivisions is the cost of maintaining them when establishing or updating the database and the additional computation time for the neural network with more inputs to consider. Your experience and judgment should guide the process.

Important Points

Separated or categorized input values can be used to convey information to the neural network in a more efficient way.

Too many categories can be burdensome to maintain and cost additional computation time needlessly.

Your experience and judgment should guide the process to make the most meaningful distinctions.

5.6 Marketing Analyzer — FIZZY.XLS {Fizzy Cola}

The demonstrations discussed so far have shown how Neuralyst can be used for rule reproduction, generalization, prediction, pattern matching, and association. `FIZZY.XLS {Fizzy Cola}` will demonstrate one way in which neural networks can be used to analyze data.

In `FIZZY.XLS {Fizzy Cola}`, we meet the Vice President for Sales of Fizzy-Cola. The Fizzy V.P. has divided the National market into eight regions and assigned each region to a manager that reports to him. As a great believer in decentralized management, he has allowed each Regional Manager to allocate their advertising budget independently of the others. The Fizzy V.P. is also scrupulously fair as he has made sure that each region has an equivalent amount of advertising money to spend in proportion to their population base.

Advertising money can be spent in four basic ways: In-store promotions (for example, store displays, price discounting), direct mail (of coupons or other promotional offers to homes), print media (newspaper or magazine advertisements), and radio/TV (commercials). When he reviews the results for the current quarter, he discovers that each Regional Manager has developed a unique allocation of advertising dollars for these four primary categories. He also determines that the sales growth in each region has varied greatly.

Of course, it would be possible to encourage the other regions to duplicate the budget allocation developed by the Regional Manager with the best sales results, but the Fizzy V.P. would like to find out if an even more successful allocation can be developed using the information contained in the current quarterly report.

The Fizzy V.P. has entered the report data into a worksheet. The budget data has been listed by advertising category and region. Since the different regions are not all exactly the same size, he has eliminated population and other base factors by listing expenditures in dollars per 1000 capita instead of total dollars and sales as percentage growth rather than total dollars.

In addition to the standard data rows, there are five more rows. The first four rows of these will be used to “probe” the neural network, once Neuralyst has learned the relationships between the different advertising budgets and each region’s sales performance. The probing is done by taking each category in turn, and setting it to the maximum value known for that category while setting the others to the minimum values known for those categories. In this case, there are four advertising categories, so there are four rows set up for probing. Click on the cells in the range **C14** to **F17** to see the Excel formulas used to generate the maximum or minimum values.

In each one of these cases, the probe will maximize one of the neural network’s inputs, while minimizing all the others. In this way, we can try and quantify the response of a neural network to individual inputs.

The Fizzy V.P. will use the information garnered from this probing to develop a new budget allocation, which we will test for him by entering into the last row.

To run this example:

1. **Init Working Area** — starting at **N1**
2. **Set Rows — 6** through **17**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns — C**, **D**, **E**, and **F**
4. **Add Target Columns — H**
5. **Add Output Columns — J**
6. **Set Mode Flag Column — L**
7. **Set Network Size** — 3 Layers, 4 Hidden Neurons
8. **Set Network Parameters** — Epochs per Update to 10

Train the neural network with this configuration. Neuralyst will be active for a short time and then complete its training. At this point, Neuralyst has discovered the underlying relationships between Fizzy-Cola's advertising expenditures and sales performance!

Once the neural network is trained, run the neural network so that the probe rows will be evaluated. When that is complete, you will see the results of the probe in cells **J14** through **J17**. The greatest output occurs for Radio/TV, second is In-store, third is Print Media, and last is Direct Mail.

Test this result by placing the values 5, 1, 2, and 10 in the cells **C18** to **F18**, respectively, of the Test Budget row. **H18** has already been programmed with the formula used to model the sales performance in the other cases of this scenario. You will find that the resulting predicted sales growth of 26.75% is 0.5% higher than the best previous case, the Northwest region. The Fizzy V.P. has achieved his goal of bettering the prior sales performance using data analysis from Neuralyst.

The technique shown here can be very useful; however, you should always test the results for sensibility before using them as it is possible to go astray.

First, it is important to minimize the number of effects that are being analyzed, so that the effect of each factor can be seen more clearly. If several factors are changing at the same time, then it will be difficult to untangle the knot of inter-relationships. One way to do this is to use ratios and percentages rather than absolute numbers. Another is to hold parameters constant where possible. In the example here, the ratio, dollars per capita, was used as input, the sales growth, as a percentage, was used as the output and the total of advertising dollars for each population unit was constant.

If the total advertising dollars in proportion to the population base had not been constant, would that make the data impossible to analyze? No. It would mean more probe cases would be needed, in this case with varying totals so the response of the model to different total amounts could be measured. Try experimenting with this case.

Second, this case is simplified in that all the inputs contributed positively to the output result. In most cases, some of the inputs will contribute negatively, that is the more the input is increased the more the output is reduced. This case needs to be distinguished from the simpler case where the input has little effect on the output. Recognizing these distinctions is important to a correct analysis.

Additionally, there are times when two or more inputs may interact with each other. These cases cannot be detected with probe cases that only have one input set to the maximum. An example of this can be seen in the LOGIC.XLS {Logic} or PARITY.XLS {Parity} demonstrations. In either of those worksheets, the presence of one 1 on an input results in a 1 on the output, yet two 1's results in a 0 — not a 2. If you suspect that this may be occurring, then probe cases where two or more inputs are set to their maximum values can be used.

Finally, in more complex cases, it may be useful to use probe cases where one input varies by fixed increments, for example, 10% of the input range per case, while the other inputs are held constant, usually at the minimums. This will result in a response curve, which can be plotted with Excel's charting capabilities. Each input can be probed in this way, resulting in a family of response curves that can be studied to determine the characteristics of the internal model developed by the neural network.

Important Points

> Methodical probing of the neural network can lead to a successful analysis of the input data and its underlying relationships.
>
> The number of parameters being measured should be minimized to ease the difficulty of interpreting results.
>
> Positive, negative and inter-related effects should all be considered and probe cases created to test for them if appropriate.
>
> In some cases, generating response curves for each input may be useful in achieving a successful analysis.

5.7 Fundamental Stock Analysis – AMETEK.XLS {Ametek}

In AMETEK.XLS {Ametek}, fundamental stock data (real world!) for the last twenty years for a smaller (annual revenues about $800 Million) New York Stock Exchange listed stock, Ametek (ticker symbol AME), have been entered into the worksheet. This kind of data is readily available from a variety of sources. Two popular ones are the Standard & Poors stock data sheets and the Value Line Investment Survey stock reviews.

The data is primarily organized on a per share basis. These are: sales revenue per share (Sls/Sh), cash flow per share (CF/Sh), earnings per share (Ern/Sh), dividends per share (Div/Sh), capital spending per share (Cap$/Sh), book value per share (BV/Sh), average price to earnings ratio for the year (Avg P/E), relative price to earnings ratio for the year compared to the overall market (Rel P/E), dividend yield (Div %), and the average price per share for the year (Avg $/Sh). (If you do not understand the terms used here please consult a stock investment book to learn the significance of these and other fundamental measures.)

While we could apply Neuralyst to this data directly, it would not be the most effective way to present the data to the neural network. Remember that neural networks work better if they are not required to make many fine distinctions in the input values. While we don't know if the distinctions between the values in this example will be critical, it is obvious that the values for many of the inputs take a different value for each row. Thus we should assume that each of these could be important to a successful forecast.

In order to satisfy the need to present the full range of the data to the neural network while also resolving the need to minimize the number of distinct values, we can present the differences between values. Thus a 1 point change in an input value with a full range of, for example, 10 to 25 would only represent a 6% change presented in this way. However, a 1 point change might represent 50% of the maximum change from year to year when presented in the context of differences between values.

There is another problem. Not all 1 point changes are equal! For example, a 1 point change from a base of 10 is more significant than a 1 point change from a base of 25. The first represents a 10% change, while the second represents a 4% change. One way to resolve this discrepancy is to take the logarithms of the input values. Logarithms have the property that a given percentage change in an input value, regardless of the starting point of the input value, will always be represented by the same change in logarithmic value. Thus, a 50% increase, whether starting from 10 or from 25, would always be represented by a change of 0.41 in the natural logarithm (it doesn't matter whether common or natural logarithms are used as long as the usage is consistent).

However, taking differences or logarithms may not make sense if the relationship between instances is not structured in time or some other dependent fashion. For example, taking differences between instances in MUGBOOK.XLS {Mug Book} would not make sense. This is because there is no reason to expect any relationship or special order between the processing of one criminal and the next criminal.

In those problems where there is a structured relationship, such as time, between instances, examples, or cases, these two methods are individually applicable. They can also be combined by taking the differences of the logarithms of the input values.

In order to implement the techniques just described, the differences of the logarithms of each input value from the previous input value have been computed in a new area just below the original area. (Note that the differences of the logs of two values is the same as the log of the ratio of those values. Though we describe it as differences, the formulas programmed are expressed in the ratio form since only one log is computed rather than two in this form.) Since each row in this area requires two rows from the original area so it can be computed, the first year, 1974, can no longer be shown.

Once the new area has been set up, we need to establish the training targets. In this case, we take advantage of future knowledge. Basically, we are setting the neural network to find any relationships that may exist that can be correlated to, or used to forecast, what will happen in the succeeding year during the training process. When the training process has ended, the future knowledge will no longer be available - but by then the relationships that could forecast that future may have been uncovered.

The Buy training target is established by "peeking" ahead to the next year and checking if the stock price has risen by at least 20% from the current year. If it has, then that is deemed a positive movement and the stock should be purchased in the current year, indicated by a BUY in the Buy column. Click on the cell **N29** (or similar cell in column **N**) to see the formula used to generate this target.

The Sell training target is generated in a similar fashion, but in its case if the stock has not at least retained its current price, then that is deemed a negative movement and the stock should be sold in the current year, indicated by a SELL in the Sell column. Click on the cell **O29** (or similar cell in column **O**) to see the formula used to generate this target.

(Note that the last row has no targets for training or testing. Since we cannot really look into the "future", except in hindsight, the number of rows that we "peek" ahead determines the number of rows that must be left blank at the end of the Target columns.)

To run this example:

1. **Init Working Area** — starting at **V1**
2. **Set Rows** — **29** through **48**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns** — **C** through **L**
4. **Add Target Columns** — **N** and **O**
5. **Add Output Columns** — **Q** and **R**
6. **Set Mode Flag Column** — **T**
7. **Set Mode Rows** — **48**, Set Symbol Row
8. **Set Network Size** — 3 Layers, 6 Hidden Neurons
9. **Set Network Parameters** — Epochs per Update to 10

Train the neural network on the data with this configuration. After some time Neuralyst will stop training. Has Neuralyst uncovered relationships that can be used to forecast the price performance of Ametek stock? Try running the neural network to make a forecast for the most recent year, 1994. The forecast will likely be to Sell Ametek stock for 1994, given 1993's data. As of the publication date of this manual, that may or may not have been a good forecast. This is because the current price of Ametek is up 20% primarily due to a 20% stock repurchase that occurred in 1994. This is a reminder that a neural network cannot predict events for which no training or modeling has been done.

In fact, this forecast, while it is useful for this demonstration, should not be relied upon at this point, independent of the probable outcome of a single prediction. In this case, there are only 17 data sets available to train the neural network. For real-world data, particularly data that is as noted for its “noise” content as stock data is, much more training data should be presented before the forecasts of the neural network should be considered in any serious way.

This can be done by going backward and presenting data from more previous years than shown in this example. Unfortunately, this approach may be difficult to implement for at least two reasons. First, data much older than this is not as readily available. Second, economic, competitive and other structural conditions often change significantly over such a long duration. In the case of Ametek, for example, it was a much different company in the 1950’s and 1960’s than it has been in the 1970’s and 1980’s. In order for the neural network to identify the relationships between input factors while these underlying conditions are changing, even more data must be presented so enough cases representative of their effects can be seen by the neural network. After a certain point, this can become an impossible task. As an example, events such as the aforementioned stock repurchase are rare or unique events which have few precedents.

Another approach that is more likely to be successful is to present data from a large number of companies during the same time period. This would hold certain implicit factors constant, for example general economic climate, interest rates, credit availability, inflation, and so on, allowing the neural network to measure the factors that lead to relative differences in performance. Two specific variations on this approach would be: 1) to train the neural network on companies that are in the same industry or produce the same product, or 2) to train the neural network across the spectrum of companies that are structured in similar ways, for example conglomerates or highly-leveraged companies.

Important Points:

Taking differences between input values is a useful technique for improving the ability of the neural network to interpret the data.

Taking logarithms of input values is another useful technique for improving the ability of the neural network to interpret the data.

Combining the two techniques of differences and logarithms is also useful.

For either or both of these techniques to work there should be a structured relationship, such as time, between instances, examples or cases.

Training a neural network for forecasting usually requires some use of “future knowledge”.

Be sure there is enough training data and the neural network is tested thoroughly before you rely on its predictions.

Be sure the training data you use does not contain more underlying conditional variations than you want the neural network to consider.

5.8 Technical Stock Analysis — DJIA.XLS {DJIA}

In DJIA.XLS {DJIA} price data on a weekly basis for the Dow Jones Average of 30 Industrial stocks from the beginning of January 1993 through September 1994 have been entered into the worksheet. This data consists of the highest price for the week, the lowest price for the week, the closing price on the week and the total volume of stocks traded in the market. This kind of data, whether on a monthly, weekly, daily, hourly, or even minute-by-minute basis is the starting point of most technical analysis methods.

We will use another set of techniques, as distinguished from the differences-of-logs form used in AMETEK.XLS {Ametek}, to pre-process raw price data into more meaningful forms. These will include a differences of inputs over time and moving averages. (Some of the pre-processed columns in this example were actually generated by the accompanying package Trader's Macro Library. See Appendix G for a discussion of how to load and use this macro toolbox for technical investment analysis.)

In the first pre-processed input column, the Close of the current work is detrended by taking the difference from the previous week, this is labeled Delta Close. The second pre-processed input column is generated by taking the ratio of the current Close to a previous Close as a percentage, this is labeled ROC or Rate of Change. The third pre-processed input column is generated by taking the difference of the Close of the current week from the Close five weeks ago. This difference over long periods of time is called "Momentum" by technical investment analysts. Finally, the last pre-processed input column is filled with the difference between a five week Moving Average of the Close for each week and a three week Moving Average of the Close for each week. This difference of Moving Averages of different periods is known as a Moving Average Oscillator by technical investment analysts. (Notice that the ROC, Momentum and Moving Average cannot be computed until a number of weeks of data are available. This will result in the first five rows being skipped when we give the **Set Rows** command.)

In setting up the targets for training, we will make use of future knowledge as in AMETEK.XLS {Ametek}. Our Buy and Sell targets will be determined by the future value of the Close column. If the Close for the next week will be higher than the Close for this week, then the Buy target will be set for this week. If the Close for the next week will be lower than the Close for this week, then the Sell target will be set for this week.

(As in AMETEK.XLS {Ametek}, the last row has no targets for training or testing. Since we cannot look into the "future", except in hindsight, the number of rows that we "peek" ahead determines the number of rows that must be left blank at the end of the Target columns.)

We will use a new technique in this neural network example, that of *iterated data windows*. So far in these examples, we have only set the neural network to train on one row of the worksheet at a time. For time based problems, this corresponds to making predictions while looking at data from just one point in time. In fact, there is every reason to believe that the relationships between values at different points in time are also significant to successful prediction. To do this we will set the number of Rows per Pattern in the problem definition to be 5. This corresponds to looking at the most recent 5 weeks of price data in every prediction. With this setting, Neuralyst will present 5 weeks at a time to the neural network, stepping one week each time.

To run this example:

1. **Init Working Area — starting at U1**
2. **Set Rows — 11** through **97**, 5 Rows/Pattern, 1 Row/Shift
3. **Add Input Columns — H** through **K**
4. **Add Target Columns — M** and **N**
5. **Add Output Columns — P** and **Q**
6. **Set Mode Flag Column — S**
7. **Set Mode Rows — 97**, Set Symbol Row
8. **Set Network Size** — 3 Layers, 12 Hidden Neurons
9. **Set Network Parameters** — Training Tolerance to 0.2,
   Testing Tolerance to 0.4,
   Epochs per Update to 10

Train the neural network with this configuration. After a short time Neuralyst will stop training. Has Neuralyst learned to predict the price performance of the Dow Jones Industrial Average from price data alone? Try running the neural network on the remaining data and compare the results to the targets. How did it do?

Most likely the matches are good, perhaps 60-70%, but not as high as we are used to from prior examples. Technical analysis of stock (or commodity) prices is a difficult problem and generally any predictive results with an accuracy better than 50% could provide an important edge in a trading situation.

Further, as with AMETEK.XLS {Ametek}, the number of weeks of data provided in this example is comparatively small when the high noise factor in stock prices is considered. In actual use, much more data and more extensive use of the kind of pre-processing we did with ROC, Momentum, and Moving Average Oscillator could help in establishing more sophisticated neural network models for stock or commodity trading.

Most important though is the complexity of the system being modeled by the neural network. If a system has a well-defined structure with clear relationships between data, then a neural network will do well when modeling it. If a system is completely unordered and random, or its relationships are much weaker than the noise that is present, then a neural network will not be able to model it successfully.

Important Points:

> Multi-row data windows are an important technique to present time structured (or any other inter-related) data to a neural network.
>
> Developing and providing intermediate processing results can help the neural network build more sophisticated internal models.
>
> Predictive accuracy is related to the complexity of the system being modeled and the amount of "noise" present in the system. In the worst case, random systems cannot be modeled by neural networks.

5.9 Shape Recognizer — SQUARE.XLS {Square}

SQUARE.XLS {Square} will demonstrate the use of Neuralyst's two-dimensional capabilities for analysis and pattern matching through the recognition of squares as distinguished from other shapes. This is a simplified, but highly relevant example, derived from the problem of recognizing shapes in applications such as robotic vision, satellite image processing, or optical character recognition.

In these and other similar applications, image data is organized as a rectangular grid of values, where each grid position is called a *pixel*, or picture element. A photograph or other image may be *scanned* (another term is *digitized*) and converted to pixels by imposing this grid over the image and converting the amount of light that is present at each grid position into binary, discrete or continuous values. The greater the number of pixels, the greater the scanning resolution, and the more accurate a representation the scanned image will be of the original image.

For medium resolution image processing, it would be common to see grids of 100 by 100 to 250 by 250 and with pixels having 16 to 256 discrete values. For high resolution image processing, grids of 500 by 500 to 2000 by 2000 and pixels having 1000 to 4 billion (essentially continuous) discrete values are possible. The resolutions of your computer monitor or TV screen are representative of the high-end of medium resolution or the low-end of high resolution displays.

In SQUARE.XLS {Square} pattern data has been entered into the worksheet as symbolic values in an input grid that is 6 by 6; obviously low resolution, but the principles remain the same. This data consists of alternate examples of squares in various positions in the field, and assorted odd-shaped, but otherwise regular, “blobs”. The neural network will be trained to symbolic outputs, indicating whether the shape is a square or is a blob.

In order to train a neural net on image data we will need to have it accept inputs in a two-dimensional form. We will do this by expanding on the iterated data window technique used in DJIA.XLS {DJIA}, by increasing the value of Rows to Shift per Pattern in the Set Rows command from 1 so that the number of Rows to Shift equals the number of Rows per Pattern. This technique may be called *Stepped data windows*.

This example will also make use of *input noise*. Input Noise is one of the parameters that can be set by **Set Network Parameters**. Setting moderate values of Input Noise provides for slight variations in the value of training data between each training epoch. Moderate values of input noise encourage the neural network in the development of generalizations. Large values of Input Noise are usually not useful.

To run this example:

1. **Init Working Area** — starting at **R1**
2. **Set Rows** — **6** through **90**, 6 Rows/Pattern, 6 Rows/Shift
3. **Add Input Columns** — **C** through **H**

4. **Add Target Columns — J** and **K**
5. **Add Output Columns — M** and **N**
6. **Set Mode Flag Column — P**
7. **Set Mode Rows — 90**, Set Symbol Row
8. **Set Network Size** — 4 Layers,
   36 neurons on Layer 2,
   9 neurons on Layer 3
9. **Set Network Parameters —** Input Noise to 0.3,
   Epochs per Update to 5

Train the neural network with this configuration. After a short time, Neuralyst will stop training. Try testing the neural network on the additional cases given. You will find that Neuralyst has learned to distinguish squares from other shapes!

Try seeing what happens without Input Noise. Train with Input Noise reset to 0 a few times. While Neuralyst will still distinguish most shapes, it will not train as consistently nor perform as reliably. Try increasing Input Noise to larger values, up to 1. How well does the neural network train or perform under these conditions?

In addition to encouraging the neural network to develop generalizations, moderate values of Input Noise may also be used help keep a neural network from getting trapped in local minima (see Section 5.1).

You have probably noticed that this example uses a four layer network. Try multiple runs with only three layers (36 hidden layer neurons). You will find that Neuralyst gets the right answers most of the time, but not as often as with the four layer network suggested. Extra layers can improve a neural network's generalization capability, particularly when there are complex features in the data.

The technique we used in this example, iterated or stepped data windows, is primarily used for conserving data space and retaining data in formats familiar to the user. The neural network itself is not given any information about these organizational details when it is being trained or tested. Neuralyst presents all data items, regardless of their position in time or space, consistently to the neurons of a neural network so that any relationships between different neurons are developed as part of the training process.

As a matter of fact, this example could have been presented so that each grid of 6 by 6 was presented as a single row of 36 values with the same results. Similarly, three-dimensional and higher-order data structures can also be presented to Neuralyst by remapping them into a one- or two-dimensional format.

Important Points:

Use stepped data windows to build neural networks to analyze two-dimensional problems.

Use moderate values of input noise to improve training consistency and encourage generalization by the neural network.

Use moderate values of input noise also to help keep neural networks from getting trapped in local minima during training.

Adding more neural network layers can improve analysis capabilities when complicated relationships are present in the data.

Multi-dimensional data structures can be presented in many ways to a neural network; the neural network will develop the necessary structural relationships.

Chapter 6
Advanced Neuralyst Topics

6.1 Input and Output Value Ranges

If you have read Chapter 3 carefully, you will be aware that neural network outputs can only range from 0 to 1. If you examine the sigmoid that is the default activation function for every neuron in the neural network, you can see the reason for this is a mathematical consequence of the way the sigmoid function works. In fact, a neural network also works best when its inputs range from 0 to 1, though the limitation here is not as absolute as for the outputs. While the examples we've shown used many inputs and outputs that complied with this range restriction, there were also instances where that wasn't true and Neuralyst still worked. In fact, if none of the values held to the restricted range, Neuralyst would still have worked.

So how does Neuralyst deal with these unrestricted ranges when neural networks work best within the restricted range of 0 to 1? Basically, Neuralyst pre-processes all input and output data and performs offset and scaling operations to match the actual ranges to the range 0 to 1. For example, if an Input column has values that range from 3 to 11, first 3 is subtracted so that the new range is 0 to 8, then each value in the new input range is multiplied by 0.125 (the reciprocal of 8) so that all values now fit in the range 0 to 1. In the case of outputs, the reverse process takes place.

In all cases, you are able to use numeric values that make sense or are convenient to your problem. Neuralyst will adjust the data in the input or output direction so the neural network is able to operate within its best value ranges.

Neuralyst also offers a variation of this scaling process through the Scaling Margin parameter in **Set Enhanced Parameters**. Scaling Margin causes the input and output data to be mapped into a smaller range than 0 to 1. A setting of 0.1, or 10%, in the Scaling Margin parameter causes an extra 10% to be reserved, 5% each at the bottom and top of the range, so the input and output data are now mapped into the range 0.05 to 0.95.

The allocation of Scaling Margin serves at least two functions. First, there are some problems, particularly those where the inputs and outputs are linear and continuously variable, that are better modeled when the activation function is restricted to output in the linear portion of its transfer function than the saturated, or non-linear, portion of its transfer function. Second, there are instances when new input or output values may be expected to be above or below those used to train or test the neural network model. Reserving some headroom allows some continued utility to the neural network model under these conditions, as long as the variant input or output values are not excessive.

Neuralyst also offers another extension to data representation by allowing the definition of symbolic input and output values. By defining a Symbol List for each Input or Target column, the Symbols are taken in order and interpreted for you. For example, if the Symbol List is defined as <RED, GREEN, BLUE>, then the range is divided into three equal subranges of 0 to 0.33, 0.34 to 0.66, and 0.67 to 1. An input value that is set to BLUE will be entered as 0.83, the center value of the 0.67 to 1 subrange. An Output value that is 0.95 will be found to be in the BLUE subrange and reported as BLUE. All the details of mapping Symbols to subranges and interpreting values to Symbols are all managed by Neuralyst for you.

A final issue regarding input and output value ranges are the defined minimum and maximum values for Input and Target columns. Each time that a neural network model is trained, the minimum and maximum values for each column define the range for that column that must be mapped into the neuron input or output range of 0 to 1, or subrange reduced by the Scaling Margin. If new data is later added to the neural network model that has a lower minimum or higher maximum in that column then that will cause a shift in all the values previously used to train the network. For any significant shift in the minimum or maximum values, this will invalidate the training of the neural network.

The MIN scale row and MAX scale row mechanism allows the actual minimum or maximum values at the time of training to be recorded and used to fix the scaling range to the correct range that was used for training. The Scaling Margin parameter will extend the set range by the designated amount allowing values that fall within the headroom region to be accepted, even if new values are added which exceed the stated range. If the MIN scale row and MAX scale row are set and values fall outside the Scaling Margin modified range, then the neural network will reject those values.

6.2 Setting Network Size

Setting the optimum network size for a problem is an intriguing and sometimes difficult process. The number of inputs and outputs are automatically defined by the structure of the problem. The number of hidden layers and number of neurons in each hidden layer are left to you (within the limits of Neuralyst's capabilities).

It has been shown that an arbitrarily large three layer (that is one hidden layer) backpropagation network can approximate just about any real-world mathematical function to an arbitrary degree of accuracy. If this is true, then why does Neuralyst allow as many as six layers (that is four hidden layers)? This is because in the three layer case the number of neurons in the single hidden layer may have to be so large that the neural network is impractical or impossible to implement. Multiple hidden layers allow greater flexibility for generalization (internal organization and model development) during the learning process and can significantly reduce the need for large numbers of neurons.

So how do we determine the number of hidden layers? There are no real rules regarding this, only rules of thumb. Generally, the more complex the problem and the more inter-related you perceive the problem characteristics to be, the more layers will be needed. It is best to start with three or four layers. Add additional layers only as the training proves to be difficult or impossible or as the predictive ability of the neural network proves unsatisfactory.

How do we determine the number of neurons in each hidden layer? Again, there are no real rules regarding this, just rules of thumb. Generally, it is best to have a pyramidal shape, that is have the greatest number of neurons in the initial layers and have fewer neurons in the later layers. A good working range for the number of neurons in each layer is to have a number from mid-way between the previous and succeeding layers to twice the number of the preceding layer. For example, if there were 12 neurons in the previous layer and 3 neurons in the succeeding layer, then a working range for the neurons in the intermediate layer would be about 6 to 24.

Often setting network size is a process of adjustment and iteration. You need enough layers and neurons for characteristics to be identified and generalizations to be made, but too many layers and neurons will be costly in computation time.

A methodical process that is often successful is to increase the size of the network (layers and neurons) in large jumps and look for successful training, then decrease the network size incrementally and observe the predictive accuracy.

The Neuralyst Genetic Supervisor can be used to automate the neural network optimization process using genetic technology. See Chapter 7 for a further discussion.

6.3 Learning Rate, Momentum, and Training Tolerance

The settings of Learning Rate and Momentum control the way in which the error is used to correct the weights in the neural network for each training case. Some settings may cause surprising behavior.

When Learning Rate is set to high values (close to 1), there is some possibility that you will see unstable behavior. This is often evidenced as wildly varying values of RMS Error (remember that neural networks also undergo learning plateaus which may show small increases of RMS Error for brief periods; we are not talking about these). As the Learning Rate is set lower the possibility of unstable behavior is reduced. Generally, when Learning Rate is set to 0.25 or less, you will not see any unstable behavior. However, lower values of Learning Rate will result in longer training times.

A more aggressive, though workable, approach is to set the Learning Rate high and to decrease it if you see any unstable behavior. This is essentially what happens when you use the Adaptive Learning Rate mode in the **Set Enhanced Parameters** command. A more conservative approach is to start with Learning Rate low and to try increasing it if training is taking too long.

The higher the value of Momentum, the greater the percentage of previous errors is applied to weight adjustment in each training case. For example, when Momentum is set at 0.5, then 50% of the weight adjustment will be due to the current error and 50% will be the weight adjustment applied in the previous case.

This means that any weight adjustment due to the error from one training case will have a continuing effect with an exponential decay. For example, consider a training case *t*; with Momentum set to 50%, the weight adjustment will be portioned as just described. In the next training case, 50% of the weight adjustment will be due to the error from *t+1*, 25% will be due to the error from *t* and the remaining 25% will be due to the previous cases. In the third training case, 50% of the weight adjustment will be due to *t+2*, 25% will be due to *t+1* 12.5% will be due to *t* and 12.5% from previous cases. As each case is considered, then the effect on weight adjustment of each previous case is halved (or reduced in proportion as determined by the Momentum setting if it is not 0.5).

In the special case when Momentum is set exactly at 1, then 100% of the previous error is used for weight adjustment. In the very first training instance, there are no previous weight adjustments, thus the first weight adjustment will be 0. Since the next weight adjustment takes 100% of the current weight adjustment, that will also be 0. This continues, regardless of how many training instances are considered. This is why Momentum must be less than 1.

In the special case when Momentum is set exactly at 0, then 0% of the previous error is used. Therefore, each weight adjustment is applied as 100% of the weight adjustment due to the current training instance;

no previous values are included in the adjustment. This is a workable setting.

Setting Momentum to higher values helps to smooth out the training process and prevent unusual cases from throwing the training off track, while continuously correcting for consistent errors. Setting Momentum lower may be appropriate for data which is more regular and smoother or data for which the relationships to be learned are relatively simple. It is generally necessary to experiment with different values of Momentum to find an appropriate value for a particular problem.

The Neuralyst Genetic Supervisor can be used to automate the Learning Rate and Momentum optimization process using genetic technology. See Chapter 7 for a further discussion.

The setting of Training Tolerance is used primarily to determine how much training the neural network undergoes. When Training Tolerance is set to a number, for example 0.4, Neuralyst will continue training the neural network until the output of the neural network for every training case is within 40% of the target range for every target value. Thus, if the target range has values from 100 to 300, then 40% of the target range is 80, or 0.4 x 200 (300-100). Thus every output value must be within 80 of the target value before Neuralyst will stop training.

It is not usually useful to have the Training Tolerance set to greater than 0.5. In the case of binary (0 or 1) values, Training Tolerance set to 0.5 means that the outputs must be less than 0.5 to be considered a 0 and greater than 0.5 to be considered a 1. While this is sufficient for binary values, and proportionately tighter Training Tolerances would be appropriate for multi-valued output cases, there are times when you may want to set Training Tolerance tighter than strictly needed to distinguish between the number of distinct outputs or to be within some error allowance for continuous outputs.

Generally, neural network output accuracy on testing or running data will be less than accuracy on training data. So, if a neural network were trained to a Training Tolerance of 0.5, outputs, for example, that should be 1's would range from 0.5 to 1. If it were then run on testing data, the outputs that should be 1's might now range from 0.3 to 1. This would result in some outputs being treated as 0 when they should have been 1. Training to a tighter Training Tolerance, for example 0.4 or 0.3, helps alleviate these marginal conditions, in effect providing a “guardband” between values that should be distinct or an extra margin for error for values that are continuous.

However, setting Training Tolerance too tightly (close to 0) may create additional problems. The first problem is a matter of practicality. Tight values result in longer training times. The second problem is more subtle. Since training time is increased, the opportunity for overtraining is also increased.

Overtraining occurs as the neural network is forced to match the target values more exactly. During this process it devotes capacity to learning the exact values of the training data rather than to generalization from the training data. If insufficient capacity is available, then the Training Tolerance is not achievable and the neural network will never stop training. If sufficient capacity is available, then the Training Tolerance will be achieved, but testing or running accuracy will suffer.

It is interesting to observe the behavior of the RMS Error generated by the neural network on training data versus testing data as the neural network trains. This can be done by setting the Error Limit value in **Set Network Parameters** to a moderate value, for example 0.1, training the neural network, and then performing a **Plot Training Error**. Typically you will see a reduction in the RMS Error plot for the training data, with the rate of reduction changing over time. If the neural network begins to overtrain, you may eventually see it as a long plateau in the RMS Error plot for training data; in some cases, the plot may begin rising. However, you will generally see it much earlier as an increase or rise in the RMS Error plot for testing data.

The Error Limit setting and the **Plot Training Error** command are powerful tools for detecting the onset of overtraining; though at the expense of increased processing time.

6.4 Learning, Weights, and Multiple Solutions

Learning takes place in neural networks through a weight adjustment process that is driven by the error developed by the neural network in each training instance. In a sense, you can visualize each weight set as defining a point on a globe (using weights as latitude and longitude on the globe). A neural network weight set that produces the desired outputs then corresponds to a special location on that globe. The training process is then similar to the game of “hot-or-cold”; as each training instance is presented, the neural network may be told that it is “cold” (or “hot”), in which case it moves a lot (or a little) across the globe by adjusting its weights a large (or a small) amount. As the neural network gets closer to the right location, it will get “hotter” until it moves just enough to find the spot.

The weight adjustment process for a given weight of each neuron is dependent on the prior value of the weight, the actual output, the desired output and the related input for that neuron (see Section 3.4). If the weights for all neurons were set to the same initial value, then all neurons having the same output value and the same input values in a training instance would have those weights adjusted by the same amount (since the weight, input, output and desired output would be the same). This is not the most desirable behavior and under some circumstances can lead to a failure to learn (like going around in circles). By setting the initial weights of a neural network to random values, this undesirable behavior is avoided. However, this randomization process raises another issue.

It is possible that more than one weight set will satisfy the input data and training constraints (like looking for cities with population over 1 million that have English as the primary language — New York, Los Angeles, London, and others would fit). In such cases, the initial value of the random weight set (starting point) could change the solution (city) the neural network finds each time. It is possible that copying a randomized weight set, saving it, and copying it back every time you wished to restart training could minimize this problem. However, reducing, extending or even changing the sequencing of the training data could change the solution found as well.

The existence of these multiple solutions may be disturbing to you since they can manifest themselves as different predictions or behavior by the neural network. However, within the constraints of the problem presented, any of these are perfectly satisfactory solutions, so the neural network has no reason to prefer one over the other. If you have such a preference, then you need to provide more training data.

Setting Training Tolerance to high values also has the effect of creating multiple solutions artificially. Setting any Training Tolerance is analogous to drawing a circle around the special location (or locations) that is desired. Any given point in that circle would be a satisfactory solution. If the Training Tolerance is low enough (circle is small), most weight sets (corresponding to points within the circle) would behave in much the same way. If the Training Tolerance is high enough (circle is large), some weight sets may behave differently than others.

6.5 Some Causes of Poor Results

If a neural network is difficult to train, tests poorly, or if its real-world performance doesn't match its training and testing performance, what can be done? The answer to this question lies primarily in having sufficient experience with neural networks to understand their capabilities and limits. The more neural networks you design and use, the better you will become at getting the best results from them. In many ways this is as much an art as a science. Until that experience is gained, here are some considerations.

First, the neural network needs to be properly sized for the problem. As discussed before (see Section 6.2), the number of layers and number of neurons are under your control: too small a network and the neural network may not be able to learn the problem; too large a network and the neural network takes longer than necessary to train. You need to observe the training behavior and test results to make these judgments.

Second, also as discussed before (see Section 6.3), the neural network may be overtrained. One way, already mentioned, to avoid overtraining is to set a looser Training Tolerance (so that training stops sooner). A second way is to set one of the cutoff limits so that

Neuralyst will stop training after a certain number of epochs, after a certain amount of time, or if RMS Error starts increasing.

Another technique to prevent overtraining, particularly with a limited training set, is to modify the training data from presentation to presentation so that some *noise* is present. That is, vary the training data slightly with successive presentations so that the network doesn't recognize the exact values presented but rather the general pattern. This can be done by setting Input Noise to small values. You must choose the amount of noise added so that it is sufficient to encourage generalization without overwhelming the underlying relationships.

Another way you can address this is with more training data, enough so that the neural network is forced into generalizations rather than specifics. When preparing more data, be aware that some underlying factors may not be constant and may have changed over the extended training set (or even in the original training set); this is particularly relevant for time-based data. If increasing the data set also increases the amount of variation of such factors, then there needs to be enough cases representative of each combination of factors for the neural network to be able to distinguish between these. For example, it may be very relevant to a fundamental analysis based model of a given stock that the most current rise in sales was due to war-driven revenues; however, without one or more additional examples of this effect from other wars, the neural network may not be able to learn the reason for the rise.

When the decision is made to increase the amount of data used for training after experiencing poor performance, another decision must also be made. Should the training be done incrementally, that is, resume training with the existing weight set while adding the new data, or should the training be restarted from a completely "blank" neural network? The answer to this really depends on the problem and its relationships.

If there is only one weight set that the neural network can develop to satisfy the relationships inherent in the input data and targets, then the incremental training and the blank-slate retraining will both achieve the same general solution. If there are multiple weight sets that the neural network can develop, due to the existence of multiple sets of relationships that are consistent with the data or due to a loose setting of Training Tolerance which might allow several solutions that are correct within the tolerance range, then the weight set actually developed by the neural network will depend on how the training was done.

While the weight set developed by the neural network in any case will satisfy the data presented and constraints established during training, it is possible that some other weight set, if it exists, might be a better predictor during testing or running (see Section 6.4). Different neural network weight sets (and thereby different predictive behavior) may develop when trained incrementally as compared to training from a blank-slate. This is often an indicator that the Training Tolerance has been set high enough that multiple solutions are acceptable. This may also be an indication that one or more factors or relationships changed over the initial training period versus the incremental periods. This may not be a problem unless you don't have enough data representative of these changing factors or relationships (see the next paragraph). Usually, the only way to verify the existence of this phenomenon in your problem is to try training with different increments and observing the resulting predictive behavior.

In some cases, results may be poor because you haven't chosen a complete set of data representative of the conditions that are relevant. This was alluded to earlier in terms of the number of cases in the training data being sufficient to be representative. But this also applies with regard to the different input types that you choose to present. For example, if you were to design a neural network for predicting customer activity in a department store, you would probably find that it would be more accurate if some inputs indicated Federal and local school holidays, though your first attempt to build the model might only have considered the day of the week.

In fact, the success of a neural network depends to a large extent on your ability to find the relevant types of input data and the manner of their presentation. It is possible to present all the data types which you perceive to have any relevance. In this case, the neural network will, if it has enough capacity, learn which data types are really relevant and which it can ignore in making a prediction. However, this raises the cost of obtaining and maintaining the data for training the neural network. Further, it requires greater computer resources in memory space and computing time in order to process the increased neural network size. Realistically, you must select just a few data types that you think are most relevant and include or exclude a few types over time as you measure the performance of the neural network.

In many cases, input data can be improved through restatement or intermediate processing. A neural network can develop many internal operations and correlations on its own; however, it makes no sense to train the neural network to perform, for example, a moving average, when Excel can compute it more exactly and more easily. Performing these intermediate computations can reduce the training time, free neurons for generalizations and, in many cases, facilitate the process of generalization. (Software programmers are familiar with this last phenomenon; the choice of data structures and programming languages makes some operations much more or much less difficult to implement even though they can all be expressed ultimately.)

The neural network targets deserve just as much thought as the inputs. In some cases, for example, rule reproduction, the targets are fairly obvious and easy to state. In other cases, particularly with regard to predictive models, the targets will require some thought in order to achieve the best statement. (Oracles throughout human history have been notorious for requiring very careful statement of the questions put to them!)

For example, the target for a technical analysis based stock model could be stated in many ways: a future price level, a future price change, a future price movement (up versus down), the number of days to an up or down move, or the volume of shares to be traded prior to an up or down move. Which of these is the best target depends not only on what you want to achieve but also on the kind of data available to be presented to the neural network. A neural network cannot make any oracular predictions (despite the comment in the last paragraph), it can only develop predictions from relationships that are present in the data put to it.

These are just a few points of consideration if you find yourself having a difficult time solving your problem with a neural network. While there are many other points that could be made, this will help you over a large number of tough spots.

6.6 Experimenting with Enhanced Neural Networks

So far our discussion has focused on the standard backpropagation neural network, as described in Chapter 3. Neuralyst also provides several ways to modify the standard backpropagation neural network. These can be grouped into the categories of selectable neuron activation function, the ability to force weights to zero if they become “insignificant”, and the ability to adapt learning rate dynamically in proportion to the error of the neural network output. Let’s see how these enhanced functions can help you to develop a better neural network.

Neuron activation functions cause a “decision” to be generated from a neuron. Thus changing the neuron activation function or its activation parameters will change the nature and characteristic behavior of the decision process. Six basic activation functions are available in Neuralyst: Augmented Ratio, Gaussian, Hyperbolic, Linear, Sigmoid, and Step. The behavior of all of these functions, except the Step function, can be further adjusted through a Function Gain parameter.

The Hyperbolic, Linear and Sigmoid functions are similar to each other. They all generate inhibitory outputs for negative inputs and excitatory outputs for positive inputs.

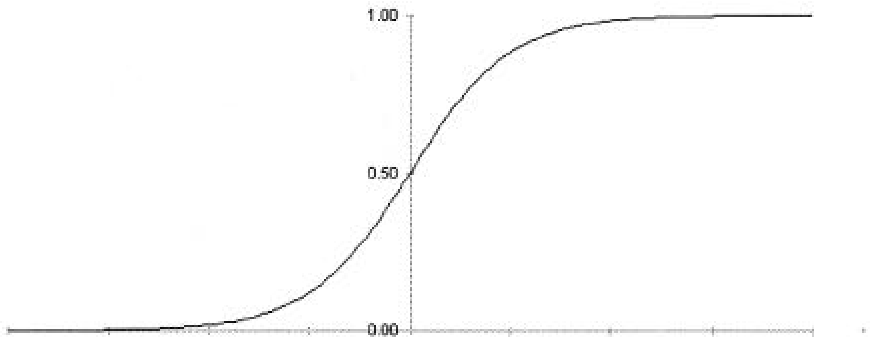


Figure 6-6 Sigmoid Function

The Sigmoid function typically has a narrow region about zero wherein the output will be roughly proportional to the input, but outside that region the Sigmoid function will limit to full inhibition or full excitation. Thus a Sigmoid function is a switch with an intermediate range where it can be discriminating.

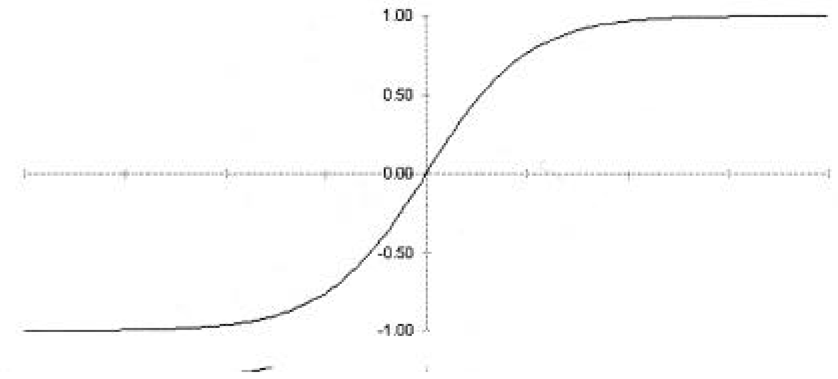


Figure 6-8 Hyperbolic Function

The Hyperbolic function is shaped exactly like the Sigmoid function, but it ranges from -1 to +1 rather than 0 to 1. Thus it has the interesting property that there is inhibition near 0, but values at either extreme will be excited to full levels, but in opposite sense. A Hyperbolic function is also a switch with an intermediate range where it can be discriminating.

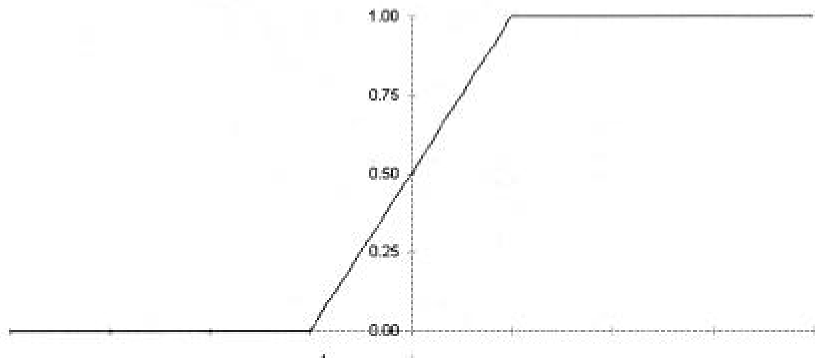


Figure 6-7 Linear Function

The Linear function always generates outputs which are proportional to the inputs, up to the level of full output. A subtle but important distinction is that the Linear function is stepped, at the point when it transitions from proportional output to full output, whereas the Hyperbolic and Sigmoid function are always smooth, that is differentiable. Differentiability of the neuron activation function is an important factor in getting consistent backpropagation training behavior.

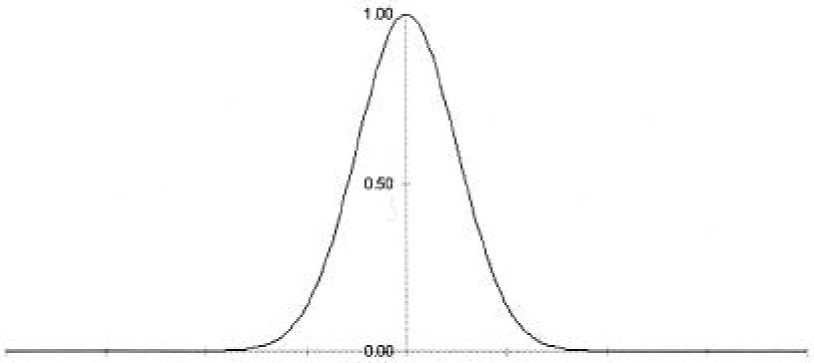


Figure 6-9 Gaussian Function

The Gaussian function is an interesting variation on the other functions. It is derived from the equation which generates a normal probability distribution and it has a central peak and low tails at both ends. This results in excitation close to zero inputs and inhibition as the inputs vary more significantly from zero.

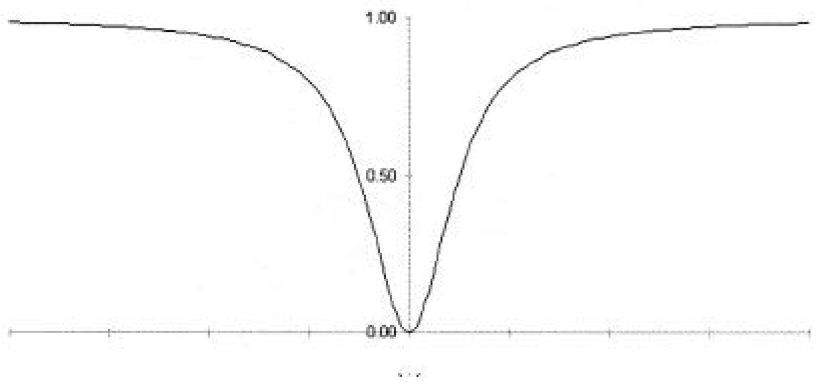


Figure 6-10 Augmented Ratio Function

The Augmented Ratio function is an upside-down version of the Gaussian function. It is defined by the equation

$$y = \frac{x^2}{1 + x^2}$$

which is also known as the augmented ratio of squares. It has a central valley reaching 0 and high tails at both ends. This results in inhibition near zero and excitation as the inputs vary significantly from zero. The Gaussian function and Augmented Ratio are both smooth (differentiable).

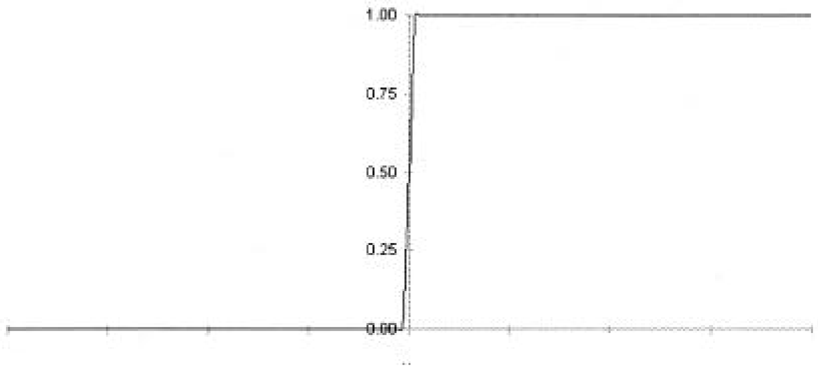


Figure 6-11 Step Function

The last function, Step, is not generally very useful, but it can be interesting to observe its behavior for simple problems. It was actually the original activation function used in early neural networks called Perceptrons. The Step function basically converts any negative input into a fully inhibitive output and any positive input into a fully excitatory output. This has the behavior of a switch; there is no fine discrimination. It did not take long to reach the limit of capabilities of Perceptrons and neural networks progressed beyond them.

Generally the Sigmoid function is far and away the most useful and the Step function is the least useful. But an examination of the characteristics of your own data and the desired decision function behavior may lead you to try a non-Sigmoid function. Experiment!

Frequently inputs or connections will have a low contributory effect on the output of a neural network. When this happens, the backpropagation algorithm will change the weights over time to be close to zero. However, there will still be some instances of excitation which will cause the weight to jitter around zero. When this happens, it increases the noise or uncertainty associated with an output. Neuralyst has the option to select a threshold and to force weights which fall below this threshold to be kept at a zero value. Forcing a weight to zero is equivalent to breaking a connection in a neural network.

It is usually best to set this mode after some initial training. Otherwise it would be possible for weights which start close to zero, but which would have been adjusted to a respectable value, to be forced to zero unnecessarily. Another approach is to complete training, then do one epoch of training with this mode set. This can help to clear the neural network of meaningless connections. When this mode is set, it is also frequently interesting to perform an **Unpack Weights** command and use the resulting display to correlate the zero weights with your input data.

It is important to be judicious in your use of this mode. It is possible that some critical analysis may hinge on the resulting fine shadings or distinctions generated by these low value weights! Always test your results and evaluate the quality of predictions before and after setting this mode.

In some instances, it may be difficult to find a setting of Learning Rate that will allow learning to occur but which doesn't take an inordinate amount of training time. This can happen because large settings of

Learning Rate cause too much adjustment to the weights during each training epoch, causing the neural network to jitter around the right area, while small settings of Learning Rate make too little progress during each training epoch.

Neuralyst provides an option to enable an Adaptive Learning Rate to address these situations. With this mode set, the Learning Rate parameter is ineffective. Instead, Neuralyst will set the Learning Rate to be proportional to the RMS Error generated during a training epoch. Thus when a neural network is far away from being correctly trained, the RMS Error will be high, and the Adaptive Learning Rate will be at a maximum. As the RMS Error is reduced, the Adaptive Learning Rate will be reduced proportionately. When the RMS Error is very small and the neural network is on the verge of completing training, the Adaptive Learning Rate will be at a minimum level still able to achieve effective learning.

Note that it is not the best solution to enable the Adaptive Learning Rate mode in all cases, since there are still many instances when a large setting for Learning Rate will train a neural network perfectly well. Setting the Adaptive Learning Rate in these instances will still achieve correct and possibly even better training, but it will take longer to complete the training, since the Learning Rate will be smaller than could be useful.

6.7 Excel Charts and Neuralyst

The use of Excel's built in charting functions can greatly enhance the utility of Neuralyst. Neuralyst provides the **Histogram Weights** and **Plot Training Error** commands (see Sections 8.2.8 and 8.2.10), but Excel charts can be used with Neuralyst in many other ways.

One use is for the graphic presentation of your data. Thus, stock price data can be shown as a High-Low-Close price chart, marketing data can be shown as pie or bar charts, technical data can be shown as line or scatter plots, and so on. Input data, target data and neural network outputs can be shown as separate lines or regions, line extensions or on their own charts.

A second use is to help you visualize the match between targets and outputs after training. This is particularly useful with regard to different settings of Training Tolerance. Comparison plots of targets versus outputs can show you how well the neural network may be learning and also let you see whether or not tighter Training Tolerances may be needed.

Charts can also aid in visualizing the behavior of your neural networks. One technique, discussed in the example `FIZZY.XLS` {`Fizzy Cola`} (see Section 5.6), is to generate a response curve for individual inputs to the neural network. This is done by holding all inputs except one constant and then varying that one evenly across its input range. The resulting response curves generated in this way can give you a picture of how each input affects the outputs both in direction and sensitivity.

Another technique is to plot the accuracy achieved on test data for an increasing number of training epochs (by decreasing Training Tolerance — see Section 6.3). If this technique is combined with trials of different numbers of network layers and changing numbers of hidden layer neurons, then it can be helpful in visually identifying the optimum neural network configuration for your problem.

Take advantage of Excel's plotting capabilities. They can help improve your presentation of the data as well as improve your understanding of a given neural network and its behavior.

Chapter 7
Genetic Optimization of Neural Networks

7.1 Genetic Technology

A new feature of Neuralyst is the inclusion of a Genetic Supervisor for enhancing the development of a given neural network model. Normally, a user of Neuralyst must experiment with the characteristics of a desired neural network model, adjusting the number and types of Input columns, the neural network configuration, and the neural network parameters in order to determine the characteristics which best produce successful predictions with the minimum amount of error or training time or both.

The Genetic Supervisor will optimize the Input column set and training parameters for a Neuralyst neural network model so that subsequent usage or adaptation of the neural network model will perform well as a successful predictor with a minimum amount of training. This is done in an automated fashion, but at the expense of a lengthy run for moderate size problems.

The Genetic Supervisor uses a special type of optimization technology known as *genetic algorithms*. Just as neural networks are models of biological neural networks that have the properties of adaptation and inference; genetic algorithms are models of a selective evolutionary process which develops superior entities from a population of entities. The genetic algorithm used in Neuralyst attempts to select the best subset of data from that provided as input, configure the best neural network that will train with the data, and adjust the various parameters that control the neural network to optimum points.

While there are many methods for determining the optimum solution to well-defined problems; there are very few methods for finding optimal solutions to unstructured, poorly understood, or partially unknown problems. Neural networks and genetic optimization are two of these limited set of methods.

7.2 Operation of the Genetic Supervisor

Although genetic algorithms are directly modeled after biological systems and behavior, the specific terms used in these two disciplines are different. The Neuralyst Genetic Supervisor uses terms from the literature of genetic algorithms to describe its parameters and behavior. Due to the close relationship to biology, the language and terms used to describe genetic algorithms can be defined in terms of their correspondance to genetics terms.

In biology, specific genetic terms are used to express the desired meaning. The *chromosomes* of a biological genetic system correspond to *strings* in an artificial genetic system. Chromosomes are composed of *genes* which take on values called *alleles*. Examples of genes are those for eye color or height; examples of alleles are blue and tall. The corresponding terms in artificial genetic systems are *features* and *values*. For example, in the genetic system in Neuralyst two features are input column name and learning rate; these features may take the values A and 0.85, respectively. The entire genetic package of chromosomes is called a *genotype* corresponding to the entire package of strings called a *structure*. In the Neuralyst genetic system, there are three strings, one for Input columns, one for network configuration, and one for network parameters; the combination of all three representing a structure which can define a neural network configuration. The expression of a genotype as a biological entity is called a *phenotype*, which corresponds to the expression of a structure as a *candidate solution*.

All the values of a structure represent the neural network characteristics that uniquely define a candidate solution in the space of possible solutions. The Neuralyst Genetic Supervisor evolves successor populations, or *generations*, from a limited population of initial candidate solutions. It does this by treating the inclusion or exclusion of each column of data in the full Input column set as features; the number of layers and the number of neurons per layers as features; and the control parameters of each neural network as features.

These features are then varied in each new generation with the resulting structure evaluated in terms of neural network *fitness*. Each structure in the generation is evaluated and judged by either the lowest RMS Error achieved after a fixed number of epochs or by the number of epochs taken to achieve a minimum point in RMS Error. These two measures represent neural networks that train to minimal error or neural networks which train with minimal epochs. These criteria can be applied to the set of training cases or the set of test cases.

If the structure representing a neural network successfully meets the fitness criteria selected, then the values of its features will be retained and bred with other structures. For each generation, the Genetic Supervisor generates a population of structures in one of two ways. All the structures of an initial generation and a certain number of structures in subsequent generations are created with features set to random values constrained within specified limits. Subsequent generations are created by cross-breeding the strings of successful structures or occasional mutations of randomly selected features of successful structures. Some or many of the weakest structures may be culled, these are replaced with new structures.

Through this evolution-like process, an optimal neural network can be developed. Note, however, that this process requires the training of many versions of the neural network to determine an optimal one; for neural network models that have large network configurations or have large data sets this can be a lengthy process — but so can biological evolution!

7.3 Structure Strings and Features

Each candidate solution is represented by a structure composed of three strings. The strings represent the selection of Input columns, the neural network configuration, and the neural network parameters.

7.3.1 Input Column Selection

Each Input column that is selected as part of a neural network model is included because the user believes that the data is correlated, positively or negatively, to the desired output. However, the data may or may not be in fact useful in predicting the output data. Further, even if useful, the data may be redundant with respect to other data already present. Thus the Genetic Supervisor treats each column of data as an individual candidate for exclusion or inclusion.

Only the set of Input columns is considered for optimization as it is presumed that a specified Target column and its corresponding output column are always a necessary part of the model. In the few cases where targets and outputs are redundant, then you will need to restrict the targets and outputs for best performance.

The Input column set is represented internally by a string of features which specifies the exclusion or inclusion of the Input columns for the structure. There is a feature for every possible column that can be included or excluded. There are two possible values for each feature, inclusion or exclusion. The user may select a nominal percentage from 1 to 100% which represents the average rate of inclusion. If the user wants to force all data to be included, then the user can set 100%. The default setting is 75%.

A percentage less than 100% does not mean that 100% of the Input columns will never be included; it means that achieving 100% inclusion will require cross-breeding and mutation to reach that level. Similarly, specifying 100% does not mean that there will never be Input column sets that are less than 100%; it means that achieving less than 100% inclusion will occur through mutation and subsequent cross-breeding.

7.3.2 Neural Network Configuration

The first and last layer of each neural network is automatically determined by the number of inputs and outputs defined by the candidate solution representing a neural network model. The count of included values in the Input column string represents the number of inputs. The number of outputs remains constant and is determined by the base neural network model. Within those constraints, the neural network can have from two to six layers and a wide range of variations in number of neurons per layer.

The neural network configuration is represented by a five feature string, where the first feature represents the total number of layers for the neural network and each subsequent feature represents the number of neurons in the corresponding hidden layer. An implicit rule is that if a lower layer feature has a value of zero neurons, then higher layer features must also be zero. The user can constrain the maximum number of layers and the maximum number of neurons per layer. The default maximum settings are 4 layers, that is an input layer, 2 hidden layers, and an output layer; with 30 neurons in the first hidden layer and 10 neurons in the second hidden layer.

7.3.3 Neural Network Parameters

The training behavior of a neural network is controlled predominantly by the user settable parameters for Learning Rate, Momentum, and Input Noise. The Learning Rate applies a greater or lesser amount of correction as a result of the error derived from a given case. The Momentum averages the current error to a greater or lesser extent with previous errors. The Input Noise factor causes the data to be perturbed from a slight extent to none to simulate real world noise or perturbations. These settings can affect the rate of training and the robustness of the neural network on test cases. The neural network training parameter features are represented by a three feature string, where each feature of the string represents the value of a particular parameter. The Learning Rate and Momentum can vary from almost zero to one, while the Input Noise can vary from 0 to 0.1. The user can constrain the minimum value of Learning Rate, the maximum value of Momentum, and the maximum value of Input Noise. The default values are a minimum of 0.5 for Learning Rate, a maximum of 1 for Momentum, and a maximum of 0.03 for Input Noise.

7.4 Population Management

There are two controls which determine the management of structure population. The first is total population Pool Size. The second is population replacement Pool Mode.

7.4.1 Population Pool Size

The population Pool Size controls the number of structures created or evolved in each generation. The number of structures remains constant, but the management of the structures is controlled by the population Pool Mode control.

Setting a large population pool provides a richer number of structures to test and evaluate to generate an optimal neural network. However, large pools will take greater amounts of time to test and therefore search. Setting a small pool has the reverse effect, each generation completes more quickly, but the pool is sparser and may take a longer amount of time to reach an optimal network.

Current research indicates that if processing resources are limited, corresponding to limited processing time, then the best strategy is to select a limited pool of 3-5 structures and to weed out the weakest structure at each generation. Conversely, current research also seems to indicate that if processing resources are unlimited, corresponding to unlimited processing time, then the best strategy is to pick as large a pool as possible. The default population Pool Size is set to 3.

7.4.2 Population Management Mode

The population Pool Mode control is a switch which sets one of three modes: Closed Pool, Immigration, or Emigration. These three modes represent different population replacement strategies from generation to generation.

With Closed Pool set, then the existing population pool will be evolved with no new structures ever introduced except through cross-breeding and mutation. With Immigration set, the population pool will be evolved and each generation entirely new structures will replace the weakest structures of the current pool with the remaining structures being cross-bred and mutated. With Emigration set, each generation the best members of a current pool will be emigrated to an entirely new population and cross-bred with the new population.

Closed Pool is similar to inbreeding in that weak structures are perpetuated through the transfer of their strings. For very small pools this is probably not useful; but for large pools, this may work fine. Immigration corresponds to the most versatile population management mode in that weak structures are continuously culled. It should work well for small or large pools. Emigration may be best for small pools, as it allows new structures to be tested rapidly and compared to the best current structures. In the case of large pools, emigration may also work well, but the characteristics of the best structures may be rapidly diluted. The default population management Pool Mode is Immigration.

7.5 Genetic Operators

Each time a population has been fully evaluated, each structure is ranked by its fitness. The structures are then evolved to generate a new population with each structure deriving its features from the previous generation such that the most fit structures of the previous generation have a higher chance of passing on their features in proportion to their ranking.

There are two genetic operations the user can control to determine the mechanism for passing features to the succeeding generation. These genetic operators are cross-breeding and mutation.

7.5.1 Cross-breeding

The primary genetic operator is cross-breeding which is determined by the Crossovers setting. Crossovers determines the frequency of intermingling of features on the same string to create new structures. Thus a setting of 1 means that two strings are crossed over at one point; a setting of 2 means that two strings are crossed over at two points, etc. The maximum setting for crossover frequency is 10.

As an example of cross-breeding, if Input columns **A**, **B**, **C**, and **D**, are selected for the base neural network model; then the string

X = <1,0,1,1> means that only Input columns **A**, **C**, and **D**, are included in this structure. If a second string, Y = <0,1,1,0>, from another structure is cross-bred with the first string, such that X is read first then crossed over at the mid-point to Y, then the resulting string for a new structure would be <1,0,1,0>, representing Input columns **A** and **C**.

Note that only the Input column string will actually be likely to see a high frequency of crossovers as the neural network configuration and neural network parameter strings are too short to intermingle more than once or twice. The default setting for Crossovers is 1.

7.5.2 Mutation

The secondary genetic operator for creating new structures is mutation. With mutation, structures and features are chosen at random, then randomly changed to new values. The Mutation Rate parameter sets the percentage of structures which will undergo a mutation rather than a crossover to create the new population.

As an example of mutation, if Input columns **A**, **B**, **C**, and **D**, are selected for the base neural network model; then the string <1,0,1,1> means that only Input columns **A**, **C**, and **D**, are included in this structure. If mutation is selected for this structure, and the third feature is mutated, then the new string would be <1,0,0,1>, representing Input columns **A** and **D**.

Mutations and cross-breeding are never mixed. A structure is either cross-bred or mutated. The maximum setting for Mutation Rate is 100%. The default setting for Mutation Rate is 10%.

7.6 Fitness Criteria

The evaluation of each structure is based on training or testing while comparing the best RMS Error level achieved or the least number of epochs. There are four modes which can be set: Train Epochs, Train Error, Test Epochs, and Test Error.

Train Error finds the structure with the least RMS Error when training each candidate solution up to the number of epochs specified in the Fitness Limit; it operates on training data only. Train Epochs finds the structure with the least epochs to achieve the RMS Error level set in Fitness Limit; it operates on training data only.

Test Error finds the structure with the least RMS Error in the test set data when training each candidate solution up to the number of epochs specified in the Fitness Limit; it operates on training data and testing data. Test Epochs finds the structure with the least epochs to achieve the RMS Error level of the test set data as set in Fitness Limit; it operates on training data and testing data.

The default Fitness Criteria mode is Train Error with Fitness Limit set to 100 epochs. This will optimize RMS Error while training each candidate solution to an epoch limit of 100.

7.7 Genetic Supervisor Tutorial

In Chapter 5, we trained a neural network to predict the possible investment potential of a NYSE stock, Ametek. We will now use that example to show you how to use the Genetic Supervisor.

Let's recap the example: AMETEK.XLS {Ametek} contains fundamental stock data. The data is primarily organized on a per share basis. These are: sales revenue per share (Sls/Sh), cash flow per share (CF/Sh), earnings per share (Ern/Sh), dividends per share (Div/Sh), capital spending per share (Cap$/Sh), book value per share (BV/Sh), average price to earnings ratio for the year (Avg. P/E), relative price to earnings ration for the year compared to the overall market (Rel P/E), dividend yield (Div %), and the average price per share for the year (Avg $/Sh).

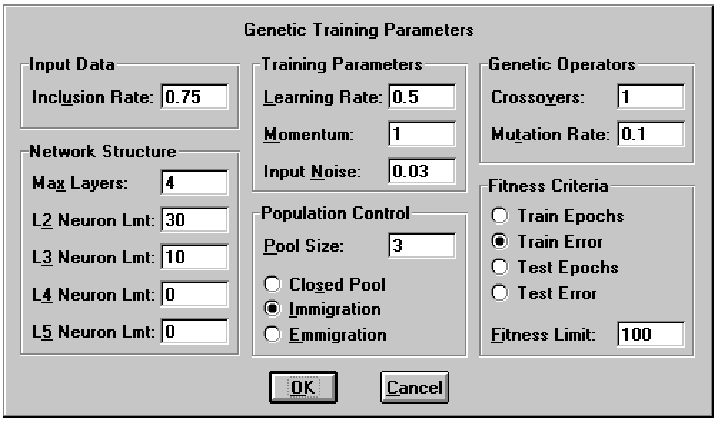
First set up the example in the same way as you did previously.

To run this example:

1. **Init Working Area** — starting at **V1**
2. **Set Rows** — **29** through **48**, 1 Row/Pattern, 1 Row/Shift
3. **Add Input Columns** — **C** through **L**

4. **Add Target Columns — N** and **O**
5. **Add Output Columns — Q** and **R**
6. **Set Mode Flag Column — T**
7. **Set Mode Rows — 48**, Set Symbol Row
8. **Set Network Size** — 3 Layers, 6 Hidden Neurons
9. **Set Network Parameters** — Epochs per Update to 10

At this point the neural network configuration is set exactly as it was the first time. Now we will allow Neuralyst to develop a more optimal neural network using genetic optimization technology. Select the **Set Genetic Parameters** command from the **Neural** menu. This will cause the Genetic Parameters dialog box to appear.


Genetic Training Parameters

Input Data
Inclusion Rate: 0.75

Network Structure
Max Layers: 4
L2 Neuron Lmt: 30
L3 Neuron Lmt: 10
L4 Neuron Lmt: 0
L5 Neuron Lmt: 0

Training Parameters
Learning Rate: 0.5
Momentum: 1
Input Noise: 0.03

Population Control
Pool Size: 3
○ Closed Pool
● Immigration
○ Emmigration

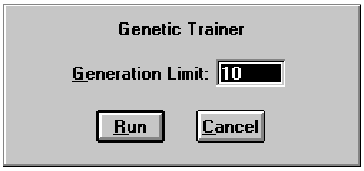
Genetic Operators
Crossovers: 1
Mutation Rate: 0.1

Fitness Criteria
○ Train Epochs
● Train Error
○ Test Epochs
○ Test Error
Fitness Limit: 100

Figure 7-1 Genetic Parameters Dialog Box

The various fields have default values which are good starting points for genetic optimization. With the exception of Pool Size, we will leave them set to their defaults for this example, but you should review the description of the parameters in Section 8.2.7 and experiment with their functions. For now, change the Pool Size setting to 10. Confirm **OK** to accept the settings. This will result in a message asking you to confirm that you want to initialize the Genetic Supervisor state. Confirm **OK**. The Genetic Supervisor is now ready to run.

Select the **Run Genetic Supervisor** command from the **Neural** menu. This will cause a dialog box to appear with an entry field to limit the generation count.

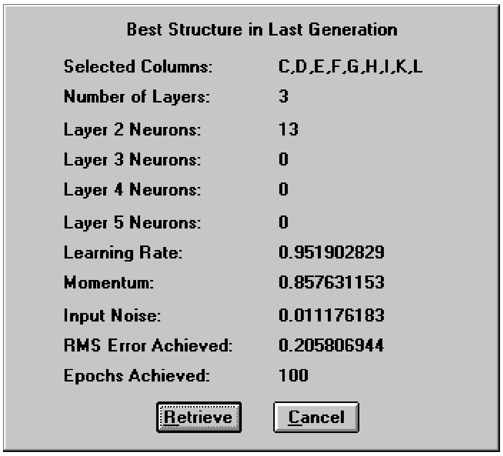


Generation Limit: 10

Figure 7-2 Genetic Trainer Dialog Box

The Genetic Supervisor will stop at the Generation Count which matches the setting. The default setting is Generation Count 10. Confirm **OK** to accept the settings. The Genetic Supervisor will now begin training and evaluating multiple neural network configurations to find the one which can provide the least RMS Error for a given number of training epochs. Note that this is only one evaluation condition that is possible, there are three others which emphasize the fewest training epochs for a given RMS Error and the same two conditions but applied to the testing set rather than the training set.

The Genetic Supervisor will report the Generation Count, the Structure Count, and the Least RMS Error or Least Epochs for the most recent completed generation. This will continue for several minutes until the Generation Count reaches the limit set in the **Run Genetic Supervisor** dialog box. At this time, the best structure identified in the current generation, after 10 generations of optimization, will be reported in a status display dialog box.


Best Structure in Last Generation

Selected Columns:	C,D,E,F,G,H,I,K,L
Number of Layers:	3
Layer 2 Neurons:	13
Layer 3 Neurons:	0
Layer 4 Neurons:	0
Layer 5 Neurons:	0
Learning Rate:	0.951902829
Momentum:	0.857631153
Input Noise:	0.011176183
RMS Error Achieved:	0.205806944
Epochs Achieved:	100

Figure 7-3 Best Structure Dialog Box

You have the option of retrieving the structure that is displayed or not doing so. If you chose not to retrieve the structure, you could resume the Genetic Supervisor at the stopped point by executing **Run Genetic Supervisor** again and increasing the generation limit to a higher Generation Count. This would proceed with additional generations of genetic optimization up to the new limit. At stopping points, you can also change the Pool Mode, Genetic Operators, or Fitness Criteria, without forcing a reinitialization of the Genetic Supervisor. However, changing most neural network configuration settings or the structure definitions in the Genetic Parameters dialog box would invalidate the optimization done to that point and require a reinitialization.

In this case, go ahead and confirm the retrieval with **Retrieve**. This will cause the structure that was shown to you to overwrite the Neuralyst Working Area with settings that duplicate the structure. The worksheet now contains a better optimized neural network. The settings which define the more optimal neural network are saved, but not the connection weights. So to try out the new neural network configuration, train and run the network as before by performing a **Train Network** command and then a **Run/Predict with Network**.

7.8 Operating Techniques

The Neuralyst Genetic Supervisor is a powerful tool, but as mentioned before, it can require lengthy optimization runs to achieve the best results. There are some techniques that are worth following when operating the Genetic Supervisor to get the most out of it.

When using a Fitness Criteria of Train Epochs or Test Epochs, which means that epoch count is optimized in reference to a set RMS Error level, not all neural network configurations will be able to achieve the set RMS Error level. These neural networks will continue training unless stopped by some other means. The external effect is that the Genetic Supervisor becomes “stuck” on a single structure. Remember that only the parameters specified in the Genetic Parameters dialog box are varied by the Genetic Supervisor, the remaining parameters that are settable in the Network Parameters and Enhanced Parameters dialog box are effective where they make sense. In particular, the three training cutoffs of Epoch Limit, Time Limit, and Error Limit are effective even under the Genetic Supervisor. Generally, it is best to set a Time Limit, with a value approximately one and a half to two times the amount of time that a trial run takes to achieve the set RMS level.

When using the Genetic Supervisor on a neural network model that has relatively few Input columns with a small Pool Size setting and an Inclusion Rate of notably less than 1, it is often possible for an initial generation to be created where no structure has all the Input columns specified, or where all Input columns are specified but the other structure characteristics cause the candidate solution to be culled. As a result, it can take many generations for cross-breeding or mutation to bring the Input column set back up to a full count for consideration. Under these conditions it is best to set Inclusion Rate to 1 or almost 1 and allow mutation to eliminate columns or to increase the Pool Size so the full Input column set is more likely to receive immediate evaluation.

When observing the Genetic Supervisor, it often happens the reported Least RMS Error or Least Epochs will increase in a current generation from the preceding one. These variations are not unusual and occur because the weight sets are initialized randomly. This may result in the best structures having slight variations in their reported results from generation to generation even though a single structure remains the best candidate solution.

Also, while observing the Genetic Supervisor, it often happens that the reported Least RMS Error or Least Epochs will attain an effective plateau after a few generations, subject to the minor variations mentioned previously. It is possible that this means that the Genetic Supervisor has attained an optimal solution. However, it also happens that after attaining a plateau for many generations, a sudden mutation can cause a marked improvement and a new, improved performance level. There is no clear way to predict the difference. This is very similar to biological evolution.

Generally, if the Genetic Supervisor has attained a plateau for a number of generations, the majority of structures will have become bred to be like the best structure. The propagation of the best structure in this fashion will generally take fewer generations than the set Pool Size if the Pool Mode is set to Closed Pool or Immigration. That is, if the Pool Size is set to 10, then the majority of structures will become similar in fewer than 10 generations. At this point, the Genetic Supervisor will attain better results only if Immigration is set or if there is a mutation. When this happens, it is possible to stop the Genetic Supervisor and change the settings of the Pool Mode, Genetic Operators or Fitness Criteria; then restart the Genetic Supervisor with the effect of stressing and evolving the population in a different way, with possibly beneficial results.

As a final suggestion, the Genetic Supervisor can deliver very useful results even with short runs. This can be achieved by setting the Genetic Supervisor to a large pool with only one or two iterations. This is an automated way to test a random space of neural network configurations for the best performer. If the neural network is moderately sized, the selected neural network will often be close to an optimal network.

Chapter 8

Neuralyst Operations Reference

8.1 Neural Network Configuration Menu

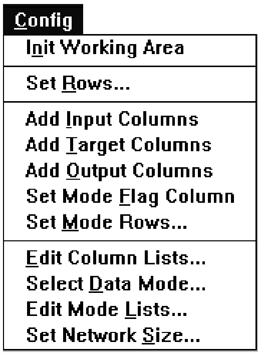


Figure 8-1 Config Menu

The **Config** menu provides you with access to the commands necessary to define the structure of a problem and the associated neural network to Neuralyst. The commands are ordered in the same sequence that you would generally proceed in defining the problem to Neuralyst, though this is not a requirement in all situations (in particular the various Add/Edit Column commands may be used in any order).

8.1.1 Init Working Area

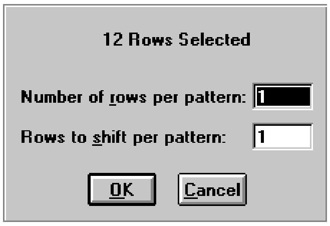
Function: **Init Working Area** is used to reserve the area that Neuralyst will use for its operations.

Usage: To use **Init Working Area**, select the cell that is at the upper left corner of the area to be reserved and execute the command. All cells to the right of the specified cell and all cells below the specified cell will be included in the Working Area. It is usually convenient to designate the Working Area starting with the first cell at the top of the sheet and to the right of other data.

Effect: **Init Working Area** will first confirm the intent to overwrite the region. If this is confirmed it will clear the region. Starting from a clear region, it will build its Working Area by placing the Title Block, Statistics Block, Parameter Block, Row Description Block, Column Description Block, Network Description Block, and Network Weights Block in this region (see Section 8.3 for a description of these data blocks). These blocks will be initialized to default values.

Warnings: **Init Working Area** must be executed for a new sheet before any other configuration operations can occur. Any data that is in the Working Area prior to the execution of the command will be deleted; therefore Neuralyst will always ask for confirmation prior to executing the command. Any data that is placed in the Working Area after the execution of the command may cause incorrect operation of Neuralyst or be changed by Neuralyst depending on the actual position within the Working Area. Executing **Init Working Area** a second time in the same or different location will result in the old area being ignored and the new area being set up and used.

8.1.2 Set Rows


12 Rows Selected

Number of rows per pattern: 1

Rows to shift per pattern: 1

Figure 8-2 Set Rows Dialog Box

Function: **Set Rows** is used to designate the rows that will be input to the neural network by Neuralyst.

Usage: To use **Set Rows**, select the rows that will be input and execute the command. After the region is set, a dialog box will show the number of rows contained in the selected region and will then request the number of rows per pattern (other appropriate terms might be instance or example) and the number of rows to shift per pattern.

When a worksheet is modified by inserting or deleting one or more rows so that the range of rows is no longer the same as that previously configured for Neuralyst, then a **Set Rows** on the new range should be performed.

Effect: The first and last rows in the selected region will be retained as the lower and upper bounds, respectively, for training, testing, or running patterns with the neural network. For example, when Rows per Pattern is set to 3, (while Rows to Shift is retained at its default value of 1) the window is three rows high. The window will step down one row for each pattern, this means that there is a two row overlap with each previous pattern.

To change the overlap, Rows to Shift may be changed. Increasing it to 2 in the example would mean the window would step down two rows for each pattern, creating a one row overlap with each previous pattern. Increasing it to 3 in the example, would mean the window would step down three rows for each pattern, resulting in no overlap between patterns. Setting the Rows per Pattern and Rows to Shift to equal values is how two-dimensional input patterns are defined.

Warnings: It will not matter which column or columns are in the selected region used for **Set Rows**; only the row numbers will be used. Only one contiguous region may be selected; discontinuous extended regions will not be handled properly. Selection and execution of **Set Rows** on a second region will change the row numbers to those of the new region.

There may be blank rows in the selected region; blank rows will always be ignored and skipped. Note that this includes the operation of Rows per Pattern and Rows to Shift per Pattern; blank rows will not be counted in these either.

In addition, partially blank rows or rows with non-numeric data at the beginning or end of the selected region will be skipped. Data in partially blank rows or rows with non-numeric data that are interspersed with valid rows in the selected region will be treated as 0's in the case of blanks or cause an error in the case of non-numeric data.

8.1.3 Add Input Columns

Function: **Add Input Columns** is used to designate those columns that will be used as inputs to the neural network for training, testing, or running patterns.

Usage: To use **Add Input Columns**, select one or more columns that contain components of the input pattern and execute the command. **Add Input Columns** may be repeated; each repetition will add those columns selected to the list of previously selected columns.

Effect: The columns selected will be noted and used as components of the input patterns to the neural network. Only those cells of the columns selected in the range established by **Set Rows** will be used.

Warnings: It will not matter which rows in the column are used to select the column; only the columns themselves will be set. The rows used will be determined by the selection passed to **Set Rows**; each column will use the same rows. All cells in the region to be used for input patterns must contain valid numeric or symbolic values; if there are any invalid values (other than blank rows) in the region determined by the intersection of rows as set by **Set Rows** and columns as set by **Add Input Columns**, then there will be an error.

8.1.4 Add Target Columns

Function: **Add Target Columns** is used to designate those columns that will be used as targets (goals or known outputs) to the neural network for training or testing input patterns.

Usage: To use **Add Target Columns**, select one or more columns that contain components of the known output pattern and execute the command. **Add Target Columns** may be repeated; each repetition will add those columns selected to the list of previously selected columns.

Effect: The columns selected will be noted and used as components of the target patterns for the neural network to compare with its own outputs. Only those cells of the columns selected in the range established by **Set Rows** will be used.

Warnings: It will not matter which rows in the column are used to select the column; only the columns themselves will be set. The rows used will be determined by the selection passed to **Set Rows**; each column will use the same rows. All cells in the region to be used for target patterns must contain valid numeric or symbolic values; if there are any invalid values (other than blank rows) in the region determined by the intersection of rows as set by **Set Rows** and columns as set by **Add Target Columns**, then there will be an error. The number of Target columns should match the number of Output columns, otherwise there could be an error.

8.1.5 Add Output Columns

Function: **Add Output Columns** is used to designate those columns that will be used as outputs for the neural network after training, testing, or running input patterns.

Usage: To use **Add Output Columns**, select one or more columns that are available to hold the output pattern and execute the command. **Add Output Columns** may be repeated; each repetition will add those columns selected to the list of previously selected columns.

Effect: The columns selected will be noted and used as components of the output patterns from the neural network after it processes the input patterns. Only those cells of the columns selected in the range established by **Set Rows** will be used.

Warnings: It will not matter which rows in the column are used to select the column; only the columns themselves will be set. The rows used will be determined by the selection passed to **Set Rows**; each column will use the same rows. All cells in the region to be used for output patterns will be overwritten by the neural network outputs; if there are any values in the region determined by the intersection of rows as set by **Set Rows** and columns as set by **Add Output Columns**, then these will be overwritten. The number of Target columns should match the number of Output columns, otherwise there could be an error.

8.1.6 Set Mode Flag Column

Function: **Set Mode Flag Column** is used to designate the column whose values will be used to switch Neuralyst from training mode to testing or running mode on a pattern by pattern basis. The column also holds special Neuralyst Mode Flags to indicate a MIN, MAX, or SYMBOL, rows.

Usage: To use **Set Mode Flag Column**, select the column that contains the flags designating whether the pattern on that row is to be used as a training pattern, as a testing/running pattern, or special Mode Flags, and execute the command.

Effect: The column selected will be noted and for each pattern window, as determined by the rows selected by **Set Rows** and the pattern window parameter, Rows per Pattern, Neuralyst will check the value of the cell in the Mode Flag column on the same row as the last row of the pattern window. If the value is TRAIN the pattern will be used for training and if the value is TEST the pattern will be used for testing or running. If the value is blank, then it is treated as TRAIN. Testing is distinguished from running only by the presence of values on the relevant rows of the Target columns. If the value is MIN, MAX, or SYMBOL, the pattern will be skipped for training, testing or running purposes; but will be interpreted appropriately otherwise.

Warnings: It will not matter which rows in the column are used to select the column; only the column itself will be set. The rows used will be determined by the selection passed to **Set Rows**. Only the text values, TRAIN, TEST, MIN, MAX, SYMBOL, or blank cells should be in the Mode Flag Column; other values may cause an error. Only one Mode Flag Column may be set; if another column is selected and **Set Mode Flag Column** is executed, then the new column will replace the old column. If a Mode Flag Column is not set then both **Train Network**, **Run/Predict with Network**, and **Run Genetic Supervisor**, will use all the defined rows. Only one instance each of MIN, MAX and SYMBOL may be present in the Mode Flag column. Additional instances may cause errors.

8.1.7 Set Mode Rows

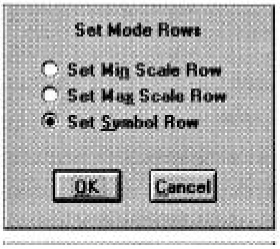


Figure 8-3 Set Mode Rows Dialog Box

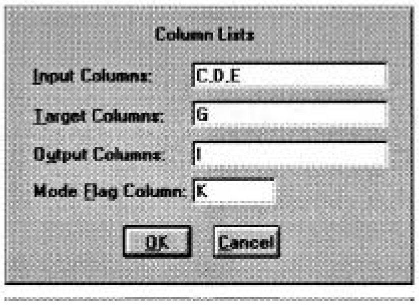
Function: **Set Mode Rows** is used to designate and optionally initialize the rows which will be identified with special Mode Flags which will indicate to Neuralyst that those rows have special functions.

Usage: To use **Set Mode Rows**, select the row that contains the Mode Flags designating whether the data on that row is to be used as a MIN, MAX, or SYMBOL, row and execute the command.

Effect: The row selected will be identified with a Mode Flag of MIN, MAX, or SYMBOL, in accordance with the type selected from the dialog box. Neuralyst will then treat the information set in those rows in accordance with the Mode Flag set. Each column of a MIN row can be initialized at the time of the command automatically with the scanned minimum numeric values, or later by using **Edit Mode Lists** with scanned or user defined numeric or symbolic values, which will be taken by Neuralyst to be the minimum value defined value for that row. Each column of a MAX row can be initialized at the time of the command automatically with the scanned maximum numeric values, or later by using **Edit Mode Lists** with scanned or user defined numeric or symbolic values, which will be taken by Neuralyst to be the maximum defined value for that row, subject to additional headroom added by Scaling Margin. Each column of a SYMBOL row can be filled in later by using **Edit Mode Lists** with a Symbol List defining the valid Symbols for that row.

Warnings: It will not matter which columns in the row are used to select the row; the row will be set with the selected Mode Flag and the column used to identify the row will be determined by the column selection passed to **Set Mode Flag Column**. Only one row can be selected of each Mode Flag type. If a mode row has been previously set of the type selected in the dialog box, then **Set Mode Rows** will overwrite the previous Mode Flag. The command will automatically initialize the Min or Max fields with the respective Min or Max numeric values from the currently defined Input or Target columns, if feasible and confirmed. Only the Mode Flag values, MIN, MAX, or SYMBOL, will be entered by **Set Mode Rows**. The only other Mode Flag values that are legal for the Mode Flag Column are TRAIN, TEST or blank; other values may cause an error. Before executing a **Set Mode Rows** command, a Mode Flag Column must have been set first with **Set Mode Flag Column**.

8.1.8 Edit Column Lists


Column Lists

Input Columns: C,D,E

Target Columns: G

Output Columns: I

Mode Flag Column: K

Figure 8-4 Edit Column Lists Dialog Box

Function: **Edit Column Lists** allows you to make additions or deletions to existing settings for Input, Target, Output and Mode Flag columns. Note that this is the only way to delete columns from the lists.

Usage: To use **Edit Column Lists** execute the command. A dialog box will appear listing the columns of each type. These lists may be edited with mouse and keyboard operations as in any Windows data entry box. The format for column lists is a series of one or two letter column names separated by spaces or commas. When complete you confirm **OK** to accept the changes or **Cancel** to abort the changes.

When a worksheet is modified by inserting or deleting a column so that some of the columns previously configured for Neuralyst no longer have the same column names, then **Edit Column Lists** may be used to revise the column lists to reflect these changes. Without **Edit Column Lists**, it would be necessary to perform an **Init Working Area** and rebuild the configuration.

Effect: When the command is executed, Neuralyst will put up the Edit Column Lists Dialog Box listing the column information entered to that point by previous **Add Column** commands. Adding new columns to any list will not be treated any differently than if the column was added by an **Add Column** command. Deleting a column already on the list will cause Neuralyst to “forget” that the column was ever entered. Once you have completed editing and confirmed the changes with **OK**, Neuralyst will enter the revised data into its column lists.

Warnings: All considerations listed for the various **Add Column** commands apply for this method of column deletion or addition. In addition, changing the column lists, other than the Mode Flag Column, will require the **Set Network Size** command to be re-executed for the new set of columns, thus forgetting any learning that has been done. In this case Neuralyst will first ask you to confirm your intentions.

8.1.9 Select Data Mode

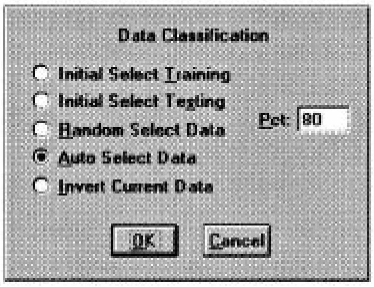


Figure 8-5 Data Classification Dialog Box

Function: Select Data Mode is used to automate the classification of data between training data and testing data.

Usage: To use **Select Data Mode**, make sure that all rows and columns are first defined, then execute the command. A dialog box will appear asking for one of five options, Initial Select Training, Initial Select Testing, Random Select Data, Auto Select Data, and Invert Current Data. Select the desired option and confirm with **OK**. The operation can be aborted with **Cancel**.

Effect: When one of the five options: Initial Select Training, Initial Select Testing, Random Select Data, Auto Select Data, and Invert Current Data, is selected, the Mode Flag column for each row associated with a pattern window will be set to TRAIN or TEST, corresponding to the classification of the data in the pattern window associated with that row as training data or testing data. A percent setting is associated with Initial Select Training, Initial Select Testing, Random Select Data, and Auto Select Data.

Initial Select Training will set the initial sequence of Mode Flag column entries to TRAIN, indicating training, up to the percentage threshold. The remaining entries will be set to TEST. For example, if there are 50 rows of training data and the setting is 80%, then the first 40 rows will be defined as training data, with the remainder defined as testing data. Initial Select Testing performs a similar function, but the initial rows are set to TEST, corresponding to testing data, and the remaining rows are set to TRAIN.

Random Select Data will randomly assign rows to be TRAIN or TEST, with the percent setting governing the approximate percentage of data to be set as training data. For example, a setting of 90% will generate a random classification with approximately 90% of the data classified as training data.

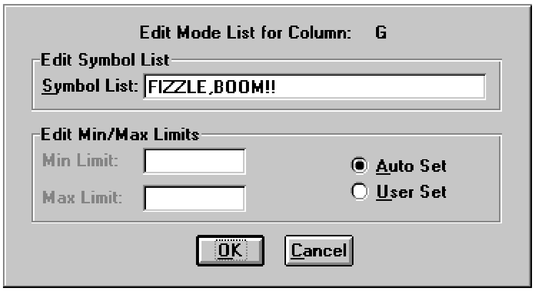
Auto Select Data will use a heuristic method to assign rows to be TRAIN or TEST, with the percent setting governing the approximate percentage of data to be set as training data. The method will scan for those rows which contain extremes or large variations of values in each column and include additional rows with data that may be suitable as possible, depending on the percent setting.

Invert Current Mode will scan any pre-existing classifications and invert them. For example, a sequence of 50 rows that had the first 40 set to TRAIN and the last ten set to TEST, would be inverted so that the first 40 were set to TEST and the last ten were set to TRAIN. If the Mode Flag column entry for a row is not filled in, then the setting will become TEST since a non-existent entry corresponds to an implied TRAIN.

Warnings: **Select Data Mode** offers some of the more useful operations on the classification of data between training and testing. However, its options may not always be suitable. When none of the **Select Data Mode** options are suitable or desirable, it is always possible to enter the classification directly in the Mode Flag column.

Auto Select Data is currently implemented as an Excel macro sequence and could take large amounts of time to classify larger data sets.

8.1.10 Edit Mode Lists


Edit Mode List for Column: G

Edit Symbol List

Symbol List: FIZZLE,BOOM!!

Edit Min/Max Limits

Min Limit:

Max Limit:

[✓ Auto Set] User Set

Figure 8-6 Edit Mode Lists Dialog Box

Function: **Edit Mode Lists** is used to enter or change the Symbol List, Min limit, or Max limit, for an Input or Target column. The Symbol List, Min limit, or Max limit, entered for a Target column is applied to an Output column.

Usage: To use **Edit Mode Lists**, select the column that is to have a Symbol List, Min limit, or Max limit, created or updated and execute the command. A dialog box will appear with fields for a Symbol List, Min limit, or Max limit, present or grayed out depending on the settings of the MIN, MAX, or SYMBOL, mode rows. Each field, if present, is clear if the the field has not been defined previously, or is initialized to the old settings if they have been defined previously. Enter or edit the settings in numeric or symbolic form, while maintaining consistency between the fields. The Symbol List, Min limit, or Max limit, can be accepted with **OK** or aborted with **Cancel**.

Effect: If symbolic data is entered or edited in the Symbol List field and the User Set option is enabled, then the Symbol List and the symbols defined for Min and Max, if Min and Max fields are present, are entered in the respective SYMBOL, MIN, and MAX, fields of the column selected. If symbolic data is entered or edited in the Symbol List field and the Auto Set option is enabled, then the Symbol List, and the first symbol and last symbol of the Symbol List, if Min and Max fields are present, are entered in the respective SYMBOL, MIN, and MAX, fields of the column selected. If the Symbol List is left blank and the User Set option is enabled, then the Min and Max values, if Min and Max fields are present, are entered in the respective MIN and MAX fields of the column selected. If the Symbol List is left blank and the Auto Set option is enabled, then the Min and Max values scanned from the designated row range of the designate column, if Min and Max fields are present, are entered in the respective MIN and MAX fields of the column selected. Any entries in the Min and Max fields, whether numeric or symbolic, are ignored in Auto Set mode.

The Symbols in the Symbol List define the valid symbolic values to be used as Inputs, Targets or Outputs, depending on the column type. The Symbols in the Symbol List will be assigned values that are increasing in proportion to their position in the list. Thus for the list <RED, GREEN, BLUE>, RED will be assigned the lowest value, GREEN will be assigned an intermediate value, and BLUE will be assigned the highest value. The same Symbol in different Symbol Lists are treated independently and are not assigned the same value. Thus the two lists, <RED, GREEN, BLUE> and <GREEN, BLUE, VIOLET>, have common Symbols, GREEN and BLUE, but they are not equal between Symbol Lists.

Warnings: It will not matter which rows in the column are used to select the column; only the fields defined by the intersection of the selected column and the defined Mode Flag rows will be set. The Mode Flag column and the Mode Flag rows must have been set previously with the **Set Mode Flag Column** and **Set Mode Rows** commands. The entries must be consistent, that is all symbolic or all numeric, not mixed symbolic and numeric.

The symbols of a Symbol List can be any alphanumeric combination separated by commas. Leading white space, space or tab, in a symbol is ignored, but trailing white space is distinctive. The symbols defined in the Min and Max fields, if present, must match one of the defined Symbols and the Min symbol must precede the Max symbol in the Symbol List. Most neural networks are not able to accurately resolve more than 5-10 Symbols for an Input or Target. There is a limit of 255 characters in a Symbol List.

8.1.11 Set Network Size

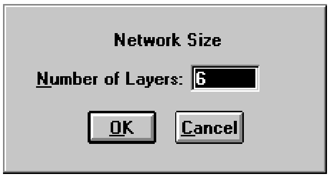
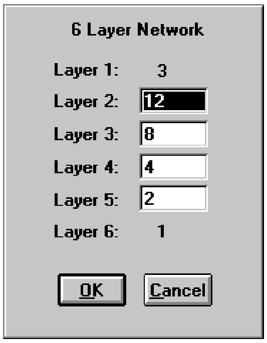


Figure 8-7 Network Size Dialog Box

Function: **Set Network Size** is used to define the structure of the neural network used in the problem, in particular the number of hidden layers and the number of neurons per layer.

Usage: To use **Set Network Size**, make sure that all rows and columns are first defined, then execute the command. A dialog box will appear asking for the Number of Layers in the neural network. There is a default minimum of 2 layers, the input and output layers. To add one or more hidden layers, you would change the entry and confirm **OK**. Another dialog box will then appear allowing you to define the number of neurons in each hidden layer which can be confirmed with **OK**. Changes can be aborted at either step with **Cancel**.


6 Layer Network

Layer 1: 3
Layer 2: 12
Layer 3: 8
Layer 4: 4
Layer 5: 2
Layer 6: 1

Figure 8-8 Network Description Dialog Box

Effect: When the first dialog box, Neural Network Configuration, is presented, you can specify the total number of layers in the neural network. Since the input and output layers are required layers, the number entered is 2 plus the number of desired hidden layers; for example, 4 would specify the default layers plus 2 hidden layers. The Network Description Dialog Box will then allow the specification of neurons for each layer. The input and output layer neurons are fixed by the number of Input columns and Output columns specified previously. Each hidden layer must have at least one neuron, but the upper limit is defined by a combination of memory availability and processing time you are willing to devote to the problem. Practical experience suggests using a range from a number midway between the preceding and succeeding layers to double the number of neurons in the preceding layer. For example, in a 3 layer network with 12 inputs and 2 outputs, a workable range is from 6 to 24 neurons for the one hidden layer. Once the network configuration has been confirmed, Neuralyst will build the neural network and initialize the starting weights for the neural network.

Warnings: **Set Network Size** is usually the last configuration step performed; in particular all columns must be defined before it can be executed. **Set Network Size** must also be executed any time there is a change in the number of columns in any column lists. The network configuration may be changed through another **Set Network Size** at any time, but when this is done, the previous weights, representing any learning that has taken place to that point, will be overwritten and so forgotten. In that situation, Neuralyst will warn you that learning will be forgotten, allowing you to confirm your intentions.

8.2 Neural Network Operations Menu

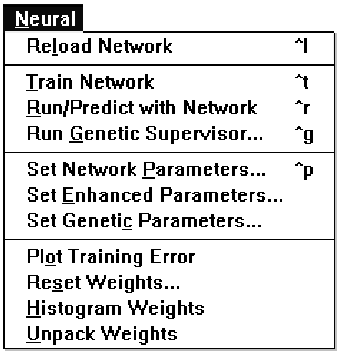

Neural
Reload Network ^l
Train Network ^t
Run/Predict with Network ^r
Run Genetic Supervisor... ^g
Set Network Parameters... ^p
Set Enhanced Parameters...
Set Genetic Parameters...
Plot Training Error
Reset Weights...
Histogram Weights
Unpack Weights

Figure 8-9 Neural Menu

The **Neural** menu provides you with access to the commands necessary to control the operation of Neuralyst once a problem structure and corresponding neural network have been defined.

8.2.1 Reload Network

Function: **Reload Network** allows saved Neuralyst worksheets to be reloaded. It also allows switching between two or more Neuralyst worksheets that may be open at the same time.

Usage: **Open** a Neuralyst worksheet or bring an opened Neuralyst worksheet to the front and make it the active worksheet, then execute **Reload Network**. The worksheet will now be ready to run Neuralyst.

Effect: **Reload Network** will cause the configuration information and network weight values saved in the Working Area of a Neuralyst worksheet to be loaded. Once a Neuralyst worksheet is loaded the configuration may be changed or the neural network can be trained, tested or run.

Warnings: Neuralyst remembers which worksheet was last loaded and will require that worksheet be the active worksheet when it runs; this helps prevent mistakes and confusion when multiple Neuralyst worksheets are active. If you are not sure which worksheet is active or if you are not sure that a change you have made to the network configuration has taken effect, it doesn't hurt to do a **Reload Network**. If you change any data in the Problem Definition Area, even if the network configuration has not changed, a **Reload Network** should be done since Neuralyst keeps a copy of all data and needs to be informed so it can refresh its copy.

8.2.2 Train Network

Function: **Train Network** causes Neuralyst to use the set of defined training patterns as inputs to the neural network, backpropagating the resulting errors (differences between actual outputs and targets) to adjust the network weights.

Usage: To use **Train Network**, once the pattern and network configurations have been defined with the **Config** menu or a saved network has been reloaded, execute the command. Once training is started, it may be halted and Neuralyst returned to command mode by using the **Esc** {**Esc** or **cmd-.**} key.

Effect: **Train Network** will cause Neuralyst to sequence through the set of training patterns by shifting the defined pattern window through the defined data. As the pattern window is shifted, Neuralyst will submit the pattern to the neural network for training if the Mode Flag is TRAIN for that pattern window. The sequence will recycle to the beginning of the defined data once the end of the defined data is reached. Each complete pass through the data is called a training epoch. After the number of epochs specified in the Epochs per Update of the **Set Network Parameters** command, the Statistics Block is updated.

This process will continue until you cause it to halt by typing the **Esc** {**Esc** or **cmd-.**} key, until the neural network achieves 100% output accuracy with respect to the defined Training Tolerance, or reaches the Epoch Limit specified with the **Set Network Parameters** command.

Warnings: Once training begins, the network weights will be modified from whatever values were saved previously. The Output columns will only be valid for those rows that correspond to TRAIN entries in the Mode Flag column or for all rows if no Mode Flag Column has been set.

8.2.3 Run/Predict with Network

Function: **Run/Predict with Network** causes Neuralyst to use the set of defined testing or running patterns as inputs to the neural network. No backpropagation takes place so the network weights are not modified.

Usage: To use **Run/Predict with Network**, once the pattern and network configurations have been defined with the **Config** menu or a saved network has been reloaded, execute the command. Testing or running will proceed through the end of the defined data.

Effect: **Run/Predict with Network** will cause Neuralyst to sequence through the set of testing/running patterns by stepping the defined pattern window through the defined data. As each pattern window is stepped, Neuralyst will submit the pattern to the neural network for testing or running if the Mode Flag is TEST for that pattern window. The sequence will stop once the end of the defined data is reached. For testing data, the output accuracy with respect to the defined Testing Tolerance is reported in the Statistics Block. For running data (no comparison to targets), the Statistics Block will change but the statistics are not meaningful.

Warnings: Testing or running does not modify the network weights. The Output columns will only be valid for those rows that correspond to TEST entries in the Mode Flag column or for all rows if there is no Mode Flag Column set.

8.2.4 Run Genetic Supervisor

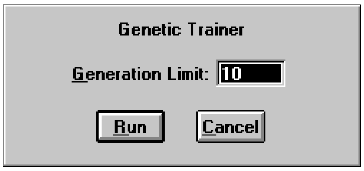
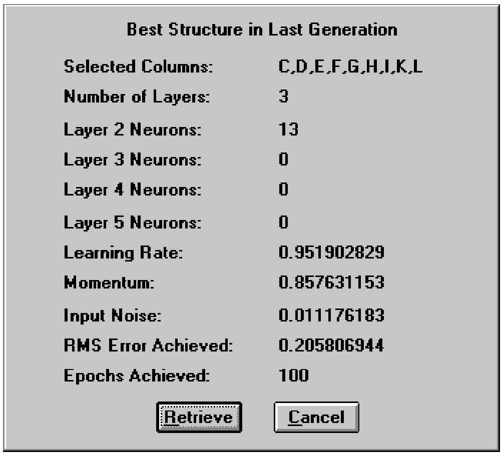


Figure 8-10 Genetic Trainer Dialog Box

Function: **Run Genetic Supervisor** causes Neuralyst to use the parameters subject to variation as defined in **Set Genetic Parameters** combined with the definition of the neural network model and iterate through a selection of candidate solutions to identify an optimal neural network.

Usage: To use **Run Genetic Supervisor**, first perform all the steps necessary to define a neural network to the stage ready for training; then select the desired parameters to control the genetic optimization process in **Set Genetic Parameters**; finally select **Run Genetic Supervisor** in the **Neural** menu and execute the command. A dialog box will appear allowing the specification of the number of generations to iterate for the genetic optimization process. Enter the number of desired generations and start the process with **OK** or abort with **Cancel**. At the conclusion of the desired number of generations, the best candidate solution will be reported and it may be retrieved to overwrite the existing neural network model by accepting it with **Retrieve** or rejecting it with **Cancel**.


Best Structure in Last Generation

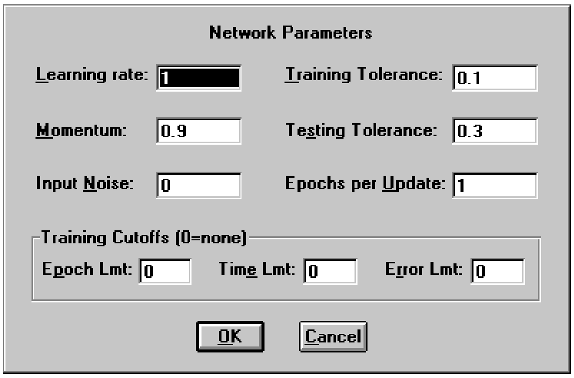
Selected Columns:	C,D,E,F,G,H,I,K,L
Number of Layers:	3
Layer 2 Neurons:	13
Layer 3 Neurons:	0
Layer 4 Neurons:	0
Layer 5 Neurons:	0
Learning Rate:	0.951902829
Momentum:	0.857631153
Input Noise:	0.011176183
RMS Error Achieved:	0.205806944
Epochs Achieved:	100

Figure 8-11 Best Structure

Effect: **Run Genetic Supervisor** will cause Neuralyst to make a copy of the Input and Target data. It will then create a population of structures according to the parameters set in **Set Genetic Parameters** and evaluate them against the Fitness Criteria and Fitness Limit set. The population will be evolved over the number of generations selected. The current generation, current structure, and best fitness values for the current generation are reported regularly. At the designated generation, the best candidate solution will be reported and it may be used to replace the current neural network model.

Warnings: **Run Genetic Supervisor** makes a copy of all Input and Target data as well as keeping its own data structures. As a result, using the Genetic Supervisor will approximately double the amount of memory that is normally used by Neuralyst. The Genetic Supervisor has to keep track of both the original neural network model and numerous variants; as a consequence there are many operations that can be performed which will invalidate its copy of the various states it is attempting to track, resulting in a need to reinitialize its state. Its state will always be valid as long as only **Set Genetic Parameter** settings for Population Mode, Genetic Operators, or Fitness Criteria are adjusted; most other commands in the **Config** menu, **Set Network Parameters**, or **Set Enhanced Parameters** will result in a reinitialization being needed. The genetic optimization process will stop because it is interrupted or because it reaches the designated generation. A repetition of the **Run Genetic Supervisor** command will allow the genetic optimization process to continue from its last point if it was interrupted or to a higher generation if the generation setting is increased; so long as only the previously indicated acceptable operations are performed between **Run Genetic Supervisor** commands. The Genetic Supervisor will overwrite the original neural network model if the best candidate solution is retrieved; if the original neural network model is still desired, a copy of the worksheet should be made and saved before **Run Genetic Supervisor** is executed.

8.2.5 Set Network Parameters


Network Parameters

Learning rate: 1
Training Tolerance: 0.1
Momentum: 0.9
Testing Tolerance: 0.3
Input Noise: 0
Epochs per Update: 1

Training Cutoffs (0=none)
Epoch Lmt: 0
Time Lmt: 0
Error Lmt: 0

Figure 8-12 Network Parameters Dialog Box

Function: **Set Network Parameters** allows various parameters affecting the network training, testing, and computation process to be adjusted.

Usage: To use **Set Network Parameters**, execute the command. A dialog box, Network Parameters, showing the current values of nine control parameters, Learning Rate, Momentum, Input Noise, Training Tolerance, Testing Tolerance, Epochs per Update, Epoch Limit, Time Limit, and Error Limit, will be displayed. These values may be changed and confirmed with **OK** or aborted with **Cancel**.

Effect: **Set Network Parameters** will allow you to change one or more of nine parameters, Learning Rate, Momentum, Input Noise, Training Tolerance, Testing Tolerance, Epochs per Update, Epoch Limit, Time Limit, and Error Limit, that affect the training process.

Learning Rate determines the magnitude of the correction term applied to adjust each neuron's weights when training. Learning Rate corresponds to the variable ***LR*** in Equation 3-3. Learning Rate must be positive, is adjusted in the range of 0 to 1 and has a default value of 1. Large values of Learning Rate will cause the network to train more quickly, but too large a value may cause the training to be unstable and no learning will occur.

Momentum determines the "lifetime" of a correction term as the training process takes place. Momentum corresponds to the variable ***M*** in Equation 3-6. Momentum must be greater than or equal to 0 but less than 1 and has a default of 0.9. Values of Momentum closer to 1 will cause the neural network to retain more of the impact of previous corrections to the current corrections. Values of Momentum close to 0 will allow mostly or only the current corrective term to have an effect. Momentum helps to smooth out the training process so that no single aberrant instance can force learning in an undesirable direction.

The Input Noise parameter enables the addition of noise during the training process. Input Noise has a value of 0 as a default, meaning no noise is added. Input Noise should be set between 0 and 1, meaning 0% to 100% of the input range will be set as a noise range. For example, if the input ranges from 0 to 15 and the Input Noise is set to 0.1, then random values ranging from 0 to 1.5, 10% of 15, will be added to or subtracted from each input value. Input Noise is only in effect when training.

The Training Tolerance and Testing Tolerance parameters define the percentage error allowed in comparing the neural network output to the target value to be scored as "Right" during training and testing, respectively. The Tolerance parameters should be between 0 and 1, with Training Tolerance having 0.1, or 10%, as a default and Testing Tolerance having 0.3, or 30%, as a default. The Tolerance parameters are defined as a percentage of the range between the highest and lowest values in the Target columns. For example, if values in the

Target columns ranged from 100 to 300, a Tolerance of 0.2, corresponding to 20%, would allow errors of 20% of 200 (300 minus 100) or 40. The Training Tolerance value has no effect on the learning algorithm. However, when Neuralyst finds 100% Right, as defined by Training Tolerance, it will automatically stop training. The Testing Tolerance value is used only for scoring during testing or running.

The Epochs per Update parameter allows you to control the number of epochs between updates of the neural network results which are displayed in the Network Run Statistics block in the worksheet. Epochs per Update has a default value of 1. However larger values will mean less frequent communication between the neural network and the worksheet, reducing overall training time.

The Epoch Limit parameter sets a maximum number of training epochs the neural network will undergo in those situations where you wish to control the number of training epochs rather than setting a Training Tolerance. Epoch Limit has a default value of 0, which means that there is no limit set.

The Time Limit parameter sets a maximum number of hours (enter minutes as a decimal fraction of an hour, for example 0.25 would be ¼ hour or 15 minutes) to continue training. Time Limit has a default value of 0 which means that there is no limit set.

The Error Limit parameter sets a maximum increase allowed, in the RMS Error values of the training data or the testing data, from the lowest achieved value of either. Under normal circumstances, a neural network that is training properly will steadily decrease the RMS Errors of each set of data. However, if the neural network exceeds its capacity or starts experiencing some effects of overtraining, some epochs may increase the RMS Error of either or both sets of data. Setting an Error Limit value will stop the training process if the RMS Error of either set increases, over any number of epochs, more than the amount set. Error Limit has a default value of 0 which means that there is no limit set.

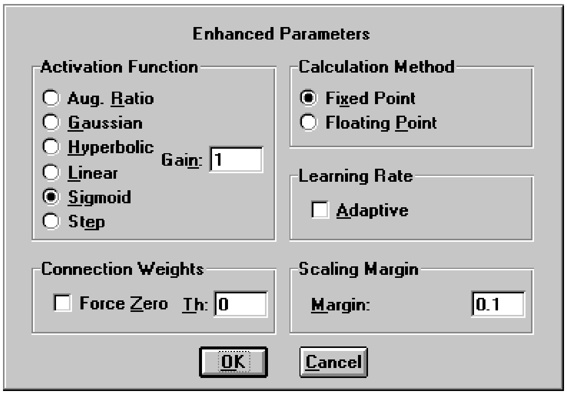
When more than one of Epoch Limit, Time Limit, or Error Limit, is set, whichever limit is reached first will terminate the training process. When no limit is set or if limits are set but the limit condition is never reached, the normal termination condition occurs when the neural network outputs meet the Training Tolerance criteria for all training data.

Warnings: It is important to observe the limitations on the range of parameter values for each parameter, though Neuralyst will check for valid values. It is possible to set Learning Rate, Momentum, and Input Noise parameters to values that cause little or no learning to take place; you should observe the change in RMS Error to be sure that it is being reduced at a satisfactory rate.

The time measured in the Time Limit parameter is actual neural network computation time — not wall-clock or waiting time. Thus parameter settings, for example low values of Epochs per Update, which result in heavy I/O, without computation, will appear to run for longer than the set Time Limit.

The RMS Error of the testing data is evaluated every training epoch if Error Limit is set to non-zero values. To skip this evaluation cost, particularly with large neural networks that take longer to train, set Error Limit to zero. With a zero setting, testing data will not be evaluated, saving substantial processing time. However, this also means that the RMS Error plot of the testing data will not be available in the **Plot Training Error** command.

8.2.6 Set Enhanced Parameters


Enhanced Parameters

Activation Function: Aug. Ratio, Gaussian, Hyperbolic, Linear, [✓ Sigmoid], Step — Gain: 1

Calculation Method: [✓ Fixed Point], Floating Point

Learning Rate: ☐ Adaptive

Connection Weights: ☐ Force Zero — Th: 0

Scaling Margin: Margin: 0.1

Figure 8-13 Enhanced Parameters Dialog Box

Function: **Set Enhanced Parameters** allows several parameters controlling special functions and modes, that can modify the behavior of the neural network, to be selected.

Usage: To use **Set Enhanced Parameters**, execute the command. A dialog box, Enhanced Parameters, showing the current values of five parameters, Function, Function Gain, Force Zero, Zero Threshold, and Adaptive Learning Rate, will be displayed. These values may be changed and confirmed with **OK** or aborted with **Cancel**.

Effect: **Set Enhanced Parameters** will allow you to change one or more of five parameters, Function, Function Gain, Force Zero, Zero Threshold, and Adaptive Learning Rate, that change the behavior of the neural network.

The Function parameter allows a change of the neuron activation function described in Section 3.3 and Section 6.6. The functions that are available are: Gaussian, Linear, Sigmoid, Step, Hyperbolic and Augmented Ratio. The Gaussian function is a smooth, differentiable function which transforms values within a range close to 0 to +1, very large negative or positive values to 0, and transforms intermediate values with varying proportion. The Linear function provides a continuous response multiplying input values by a constant scaling factor until the limits of 0 or +1 are reached. The Sigmoid function is also a smooth, differentiable function, which forces large negative or large positive values to 0 or +1, respectively, but transforms intermediate values with varying proportion. The Step function is a sharp switching function, all negative inputs are forced to 0 and all positive inputs are forced to +1. The Hyperbolic function is another smooth, differentiable function, which forces large negative or large positive values to -1 or +1, respectively, but transforms intermediate values with varying proportion. The Hyperbolic function is actually the Hyperbolic Tangent function and is very similar to the Sigmoid function, except it is negative-going. The Augmented Ratio function is a smooth, differentiable function which transforms values within a range close to 0 to 0, very large negative or positive values to 1, and transforms intermediate values with varying proportion. The Augmented Ratio function is similar to the Gaussian function, but inverted. Thus selecting different neuron functions can significantly change the behavior of the neuron.

The Function Gain parameter allows a change in the scaling or width of the selected Function. Increasing Function Gain will narrow the central region in the Gaussian function or steepen the slope of the Linear function and the Sigmoid function. Decreasing the Function Gain, correspondingly, will broaden the central region in the Gaussian Function or make the slope of the Linear or Sigmoid functions shallower. With a Function Gain setting of 1, all three functions will cover the normal input range of the neural network smoothly. The Function Gain applies to all of the Activation Functions except the Step function.

The Force Zero parameter enables a special mode which scans the weights after every training epoch and sets those weights which are close to 0 to be 0. The Zero Threshold parameter selects the threshold below which weight values are set to 0. Thus, a Zero Threshold value of 0.01 with Force Zero enabled will cause a weight value of 0.009 to be made to be 0, while a value of 0.15 will be left unchanged.

The Calculation Method parameter allows a selection of Fixed Point calculation or Floating Point calculation. Fixed Point mode is a very fast and memory efficient technique, allowing speeds up to 2-3x a computer equipped with a math co-processor using Floating Point calculations. It is inherently less precise, but is satisfactory for the majority of neural network problems that require precision to only 2-3 places. Floating Point mode is more precise and has higher resolution, but requires a math co-processor — not present on many 386, 486SX, 486SL, 68030, and 68LC040 processors — more memory, and more processing time. Fixed Point mode was the native mode of earlier versions of Neuralyst; but now either mode can be selected depending on the circumstances.

The Adaptive Learning Rate parameter enables a special mode which evaluates the RMS Error after every training epoch and causes a revision of the Learning Rate for the next training epoch. If the RMS Error is high, then a high Learning Rate will be set, as the RMS Error is reduced, the Learning Rate will be reduced correspondingly. The net effect is to speed up the training process when the neural network is far away from the correct weights, but to slow it down as it gets closer.

The Scaling Margin parameter adds additional headroom, as a percentage of range, to the rescaling computations used by Neuralyst in preparing data for the neural network or interpreting data from the neural network. All Input, Target, and Output data actually processed by the neural network must be in the range 0 to 1. As a consequence, all data presented or interpreted is rescaled from the actual worksheet values to values in the range of 0 to 1. Setting Scaling Margin above 0, increases the amount of headroom or margin, decreasing the range onto which an Input, Target, or Output, is mapped. As an example, if the Scaling Margin is set to 0.1, or 10%, then 10% margin is allowed and split between the high end and the low end of the range, resulting in a mapping to the range 0.05 to 0.95 rather than 0 to 1. This margin can facilitate precision in problems that have continuously variable data or that may exceed the current values in the future by a small amount. Scaling Margin modifies the Min or Max limits that may be set for an input or target by the designated amount.

The options of Input Noise, Force Zero, and Adaptive Learning Rate will typically increase the total number of training epochs it takes to achieve a successfully trained neural network over a “no-frills” approach. However, when used judiciously, the quality of training - and the corresponding quality of predictions — of the neural network can be notably improved. Fixed Point calculation mode generally provides the fastest training; however, for continuously variable inputs or outputs requiring high resolution, Floating Point calculation mode can be more precise. In addition, on high performance Pentiums or PowerPCs, Floating Point approaches Fixed Point speed. Scaling Margin is another control which is useful for problems that have continuously variable inputs or outputs to allow higher precision.

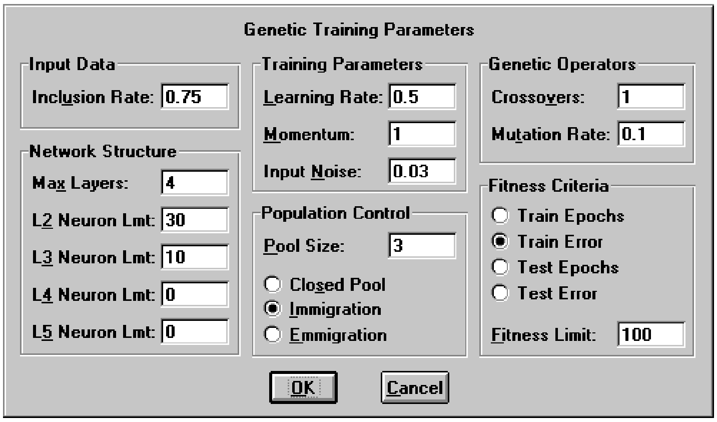
Warnings: Selecting neuron activation functions aside from the Sigmoid will typically change the behavior of the neural network significantly. The Hyperbolic function is probably the most useful after the Sigmoid. However, the Hyperbolic function is only effective in Floating Point calculation mode. The Step function has limited utility and is particularly difficult to train and has been included mostly for experimental purposes for readers interested in duplicating the work of early researchers on neural networks. The Linear, Gaussian, and Augmented Ratio functions may or may not train well, depending on the characteristics of the training data.

While there are no limits on the settings of the Function Gain parameter or the Zero Threshold parameter, extreme values will prevent training. Typically values of Function Gain should not exceed the range 1/10 to 10 and the Zero Threshold parameter should not exceed a level of 0.5.

Selecting Force Zero and Adaptive Learning Rate (as well as Input Noise in the **Set Network Parameters** command) will typically increase the total number of training epochs it takes to achieve a successfully trained neural network over a "no-frills" approach.

Setting Scaling Margin to allow headroom for future values that exceed the current data range is a marginal practice. Since, in principle, the neural network has not been trained on data outside the current range, it cannot be expected to accurately predict data outside the current range, even with headroom. In practice, it can be effective for certain neural network models.

8.2.7 Set Genetic Parameters


Genetic Training Parameters

Input Data
Inclusion Rate: 0.75

Network Structure
Max Layers: 4
L2 Neuron Lmt: 30
L3 Neuron Lmt: 10
L4 Neuron Lmt: 0
L5 Neuron Lmt: 0

Training Parameters
Learning Rate: 0.5
Momentum: 1
Input Noise: 0.03

Population Control
Pool Size: 3
Closed Pool
[✓ Immigration]
Emmigration

Genetic Operators
Crossovers: 1
Mutation Rate: 0.1

Fitness Criteria
Train Epochs
[✓ Train Error]
Test Epochs
Test Error
Fitness Limit: 100

Figure 8-14 Genetic Parameters Dialog Box

Function: **Set Genetic Parameters** allows several parameters controlling the creation of structure populations, the manner in which they are evolved, and the way they are evaluated, to be selected.

Usage: To use **Set Genetic Parameters**, execute the command. A dialog box, Genetic Parameters, showing the current values of the 15 parameters, Inclusion Rate, Max Layers, four Layer **n** Neuron Limits, Learning Rate, Momentum, Input Noise, Pool Size, Pool Mode, Crossovers, Mutation Rate, Fitness Criteria, and Fitness Limit, will be displayed. These values may be changed and confirmed with **OK** or aborted with **Cancel**.

Effect: **Set Genetic Parameters** will allow you to change one or more of the fifteen parameters, Inclusion Rate, Max Layers, four Layer **n** Neuron Limits, Learning Rate, Momentum, Input Noise, Pool Size, Pool Mode, Crossovers, Mutation Rate, Fitness Criteria, and Fitness Limit, that bound the limits of the genetic optimization process and change the behavior of the Genetic Supervisor.

The Input column Inclusion Rate controls the average rate of Input column inclusion in structure initialization. The user may select a nominal percentage from 1 to 100% which represents the average rate of inclusion. If the user wants to force all Input columns to be included, then the user can set 100%. The default setting is 75%.

The first and last layer of each neural network is automatically determined by the number of inputs and outputs defined for the original neural network model. Within those constraints, the optimized neural network can have from two to six layers and a wide range of variations in number of neurons per layer. The user can constrain the maximum number of layers and the maximum number of neurons per hidden layer. An implicit rule is that if a lower hidden layer is zero, then higher hidden layers must be zero. The default maximum settings are 4 layers, an input layer, 2 hidden layers, and an output layer; with 30 neurons in the first hidden layer and 10 neurons in the second hidden layer.

There are three network parameters which can be optimized by the Genetic Supervisor: Learning Rate, Momentum, and Input Noise. Learning Rate and Momentum can vary from almost 0 to 1, while the Input Noise can vary from 0 to 0.1. The user can constrain the minimum value of Learning Rate, the maximum value of Momentum, and the maximum value of Input Noise. The default values are a minimum of 0.5 for Learning Rate, a maximum of 1 for Momentum, and a maximum of 0.03 for Input Noise.

There are two controls which determine the management of populations. The first is a control which sets the total population size. Changing this sets the number of structures initialized or evaluated in each generation. The second is a control which chooses among the three modes Closed Pool, Immigration, or Emigration. With Closed Pool set, then the initial population pool will only be cross-bred or mutated with no new structures initialized. With Immigration set, the population pool will be evolved and every generation new structures will replace the weakest structures of the existing pool. With Emigration set, the population pool will be evolved and every generation the best structure will be emigrated to an entirely new population. The default settings are for population size to be 3 and Immigration to be enabled.

There are two settings the user can control to determine the mechanism for genetic optimization. These settings are Crossovers and Mutation Rate. Cross-breeding is controlled by the Crossovers setting, which determines the amount of intermingling of features on the same string to create new structures. A setting of 1 means that two strings are crossed over at one point; a setting of 2 means that two strings are crossed over at two points. The maximum setting for Crossovers is 10 and the default is 1. The secondary mechanism for creating new structures is mutation. With mutation, structures are chosen at random, then features are randomly changed to new values. The mutation rate sets the percentage of structures which will undergo a mutation rather than a cross-breeding to create the new population. The maximum setting for mutation is 100%. Mutation is always in place of cross-breeding; the two never occur at the same time on a single structure. The default setting for mutation rate is 10%.

The evaluation of each structure is based training or testing while comparing the best RMS Error level achieved or the least number of epochs. There are four modes which can be set: Train Epochs, Train Error, Test Epochs, and Test Error. Train Error finds the structure with the least RMS Error while holding epochs to the set Fitness Limit. Train Epochs finds the structure with the least epochs to achieve the RMS Error level set in Fitness Limit. These two operate on training data only. Test Error finds the structure with the least

RMS Error in the test set data while holding epochs to the Fitness Limit. Test Epochs finds the structure with the least epochs to achieve the RMS Error level of the test set data as set in Fitness Limit. These two measure RMS Error by evaluating the testing data. The default Fitness Criteria mode is Train Error with Fitness Limit set to 100 epochs. This will optimize RMS Error while training each candidate solution to an epoch limit of 100.

Warnings: Setting a low value of Inclusion Rate and a small Pool Size may result in very few Input columns being evaluated. The Neuron Limits for hidden layers in excess of those specified by the Max Layers settings must be 0. A very high Mutation Rate means that relatively little cross-breeding occurs, since mutation supersedes cross-breeding. Train Epoch and Test Epoch optimize for least epochs, so the Fitness Limit represents the reference RMS Error settings and must be between 0 and 1 for these Fitness Criteria. Train Error and Test Error optimize for least RMS Error, so the Fitness Limit represents the reference epoch count and must be an integer greater than 1 for these Fitness Criteria.

Many **Config** menu, **Set Network Parameters**, and **Set Enhanced Parameters**, commands will invalidate the Genetic Supervisor state and require another **Set Genetic Parameters** command to reinitialize the Genetic Supervisor. This will cause genetic optimization results to be lost.

8.2.8 Plot Training Error

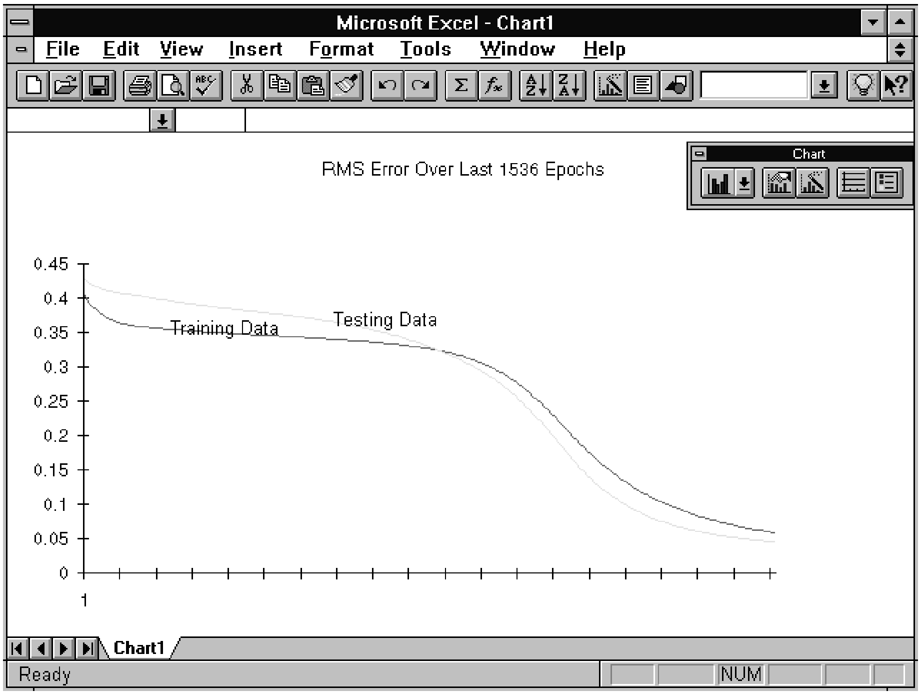


Figure 8-15 Training Error Chart

Function: **Plot Training Error** will cause a line plot of the RMS Error of the training data and the testing data over the recent training epochs to be displayed in an Excel chart.

Usage: To use **Plot Training Error** execute the command. Neuralyst will then transfer the set of values of RMS Errors of the training data and the testing data resulting from training to that point and create a new chart window containing line plots of the data.

Effect: **Plot Training Error** will display plots of the convergence of the RMS Error values over the past training epochs of the neural network. One plot will indicate the convergence of the neural network on achieving learning of the training data. The second plot will show the success of the neural network on the testing data while training is occurring. The second plot will only be available if there is testing data and if the Error Limit (in **Set Network Parameters**) has been set to be non-zero. This command is executed when you want to get an impression of how much progress has been made and how successful the neural network has been at learning during training.

Ideal plots of the RMS Error of the training data will show a steep descent early on with progress in further reductions of RMS Error, though not as steep as initially, as training progresses. If the RMS Error attains a long term plateau, then it typically indicates that the neural network has achieved as much learning as it can. If the RMS Error starts increasing, it often indicates that the neural network is becoming severely overtrained. Along with the training data, the testing data will also be plotted (if settings permit). The RMS Error plot of the testing data can be used to measure the point at which the neural network training was most successful on testing data. Overtraining can often be seen earlier on this plot than for training data.

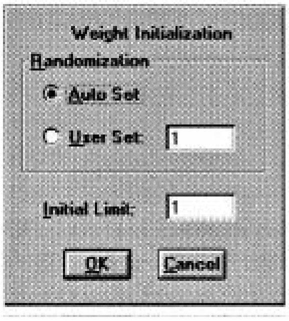
Warnings: The line plots generated by **Plot Training Error** are a snapshot of the RMS Error behavior up to the time the command was executed. Subsequent training will not be reflected in the chart. To view the effects of additional training, another **Plot Training Error** command must be given. In this event, new line plots are created, without effect on the previous line plots, allowing comparison of the effects of additional training epochs.

At approximately 250 epochs, and every doubling of epochs thereafter, the line plots will be rescaled — so plots at 100 epochs will have points for every epoch, but plots at 300 epochs will have only 150 points per plot (a point for every two epochs). Thus the scale will be different, though the shape and information will be consistent.

The training error history is not retained between reloads of neural network data. Thus if you have a partially trained neural network, save your work, reload the next day, train some more and then do a **Plot Training Error** command, the plot will only show the training error behavior over the most recent session. The training error history of the previous session will be lost and no longer available.

You will need to manually delete the line plots when you no longer want them.

8.2.9 Reset Weights



Weight Initialization

Randomization

● Auto Set

○ User Set: 1

Initial Limit: 1

Figure 8-16 Weight Initialization Dialog Box

Function: **Reset Weights** provides for control of the weight initialization process.

Usage: To use **Reset Weights** execute the command. Neuralyst will provide a dialog box allowing for Auto or User Randomization of the initial weight values. The range of the initial weight values may also be set. Changes to the initialization options and the initialization operation itself can be confirmed with an **OK** or else the changes and operation may be aborted with a **Cancel**.

Effect: **Reset Weights** allows the neural network weights to be changed to new random values. (Neural network weights are initialized to random values to allow learning to occur without any established patterns or biases.) Prior to resetting the weights the user may set certain initialization options. These options are Auto Set Randomization, User Set Randomization, and Initial Limit. Once the options are set and the operation is confirmed, the weights are set to new initial values.

Selection of Auto Set will provide for a completely random set of weights. There will be no relationship between the values generated from one instance of Auto Set Randomization to the next or other instances of Auto Set randomization.

The user may select User Set Randomization in lieu of Auto Set Randomization, causing a particular predetermined and repeatable random sequence to be used, based on the value supplied to User Set Randomization. In other words, supplying 1 to the User Set Randomization option will always cause the same random numbers to be used for the weights in a given neural network. Changing the value supplied will result in a new, but equally repeatable, set of random values to be generated. The initial value supplied to the User Set Randomization is 1 and may be changed to any other integer value.

With the Randomization type selected, the user may also set the Initial Limit value. This has a default value of 1, which means that initial weight values will be randomly chosen from the range -1 to +1. Reducing the Initial Limit value to 0.2, would result in initial weight values in the range -0.2 to +0.2. Initial Limit may be set to a number between 0 to 16, though initial values much larger than 1 are not very useful.

This command is executed when you want a fresh start to learning or training of the neural network. Weights are always randomized when set to starting values since fixed or ordered starting values may result in all neurons learning the same characteristic at the same time. Randomization allows individual neurons to become “sensitized” to different characteristics.

Warnings: All previous weights will be overwritten and so all prior learning will be forgotten when **Reset Weights** is executed. Because of this, Neuralyst will first ask you to confirm your intentions.

8.2.10 Histogram Weights

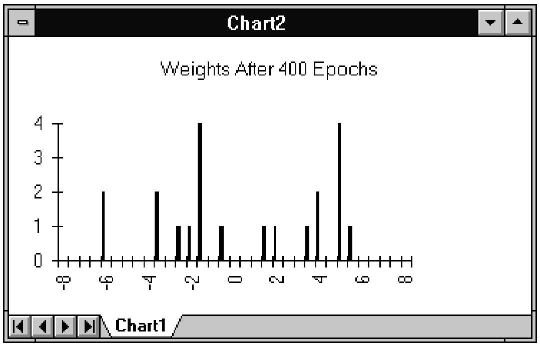


Figure 8-17 Weight Histogram Chart

Function: **Histogram Weights** will cause a histogram of the weights of the defined network to be displayed in an Excel chart.

Usage: To use **Histogram Weights** execute the command. Neuralyst will then read the weights of the neural network at that point and create a new chart window containing a histogram of the weight data.

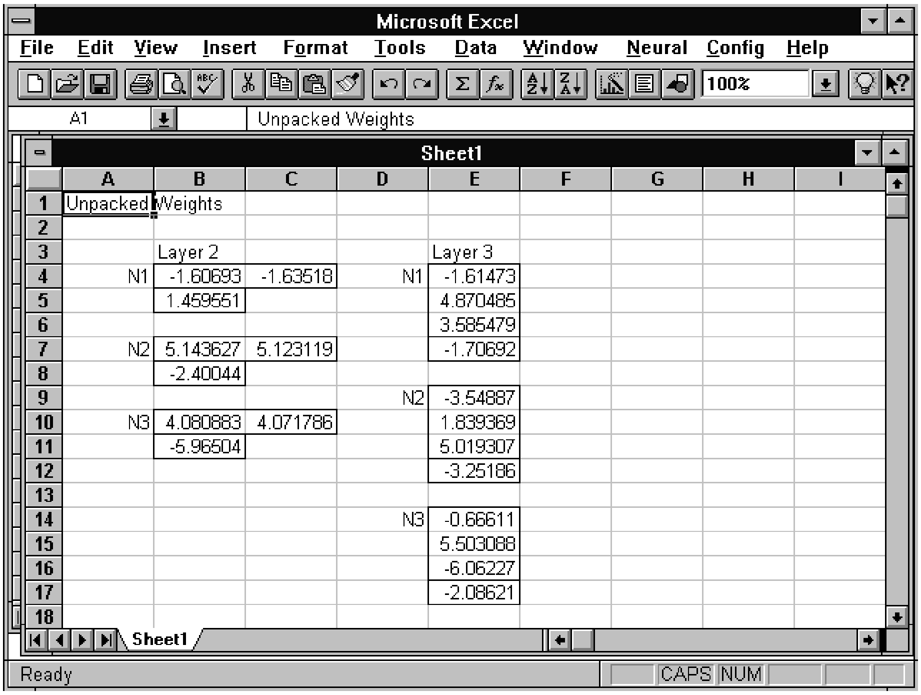
Effect: **Histogram Weights** will display the distribution of weight values of the neural network. This command is executed when you want to get an impression of how much progress has been made and how successful the neural network has been at learning during training.

Early on, the weight values will be distributed in the range determined by the Initial Limit option in the Weight Initialization dialog box. If this is 1, then the histogram would show weight values distributed between -1 and +1. As time progresses, the distribution will spread out, with many values still in the initial range but with several values just outside the range and a few values at the extremes. This kind of distribution is most common in successfully trained neural networks.

If a neural network has been unsuccessful at training, a histogram of the weights will often show a distribution heavily biased to the extremes, values below -8 and above +8. This may indicate a need to set a larger neural network or different network parameters.

Warnings: The histogram generated by **Histogram Weights** is a snapshot of the weights at the time the command was executed. Subsequent training will not be reflected in the chart. To view the effects of additional training, another **Histogram Weights** command must be given. In this event, a new histogram is created, without effect on the previous histogram, allowing comparison of the weight distribution from the two points of training. You will need to manually delete the histograms when you no longer want them.

8.2.11 Unpack Weights


Microsoft Excel
File Edit View Insert Format Tools Data Window Neural Config Help
100%
Sheet1
Ready
CAPS NUM

A1 | Unpacked Weights

	A	B	C	D	E	F	G	H	I
1	Unpacked Weights								
2									
3		Layer 2			Layer 3				
4	N1	-1.60693	-1.63518	N1	-1.61473				
5		1.459551			4.870485				
6					3.585479				
7	N2	5.143627	5.123119		-1.70692				
8		-2.40044							
9				N2	-3.54887				
10	N3	4.080883	4.071786		1.839369				
11		-5.96504			5.019307				
12					-3.25186				
13									
14				N3	-0.66611				
15					5.503088				
16					-6.06227				
17					-2.08621				
18									

Figure 8-18 Unpacked Weights Display

Function: **Unpack Weights** will cause the weights of the defined network to be displayed in a newly created Excel worksheet in a readable form.

Usage: To use **Unpack Weights** execute the command. Neuralyst will then write the unpacked weight values in a new Excel worksheet window.

Effect: **Unpack Weights** will cause a formatted display of the weight values to be written to a newly created Excel worksheet. This display is organized by neural network layers and by neurons in each layer, with each component being so labeled.

The first neural network layer displayed is Layer 2 (Layer 1 “neurons” are the inputs themselves by convention). The weights of each neuron in Layer 2 are shown in multiple columns grouped by neurons. The separate columns reflect the weights applied to each column designated as an input. The rows of the grouped columns reflect the different weights for each row used when multi-row inputs are defined. Thus the pattern window created by the number of inputs and the number of rows per pattern corresponds directly to the values shown in the columns and rows, respectively, for each neuron in the Layer 2 weight display. These weight values are outlined for each neuron. The threshold value for each neuron is outlined separately and terminates the weight values.

Subsequent neural network layers are displayed in a more simplified way with the weights of a neuron shown in single columns grouped by neurons. The first value in each column reflects the weight from the first neuron of the previous layer to that neuron, the second value in each column reflects the weight from the second neuron of the previous layer to that neuron, and so on. The column of weights is outlined for each neuron. The threshold value for each neuron is outlined separately and terminates the weight values.

Warnings: The display of weights generated by **Unpack Weights** is a snapshot of the weights at the time the command was executed. Subsequent training will not be reflected in the display. To view the effects of additional training, another **Unpack Weights** command must be given. In this event, a new worksheet will be created. You will need to manually delete the worksheets when you no longer want them.

8.3 Neuralyst Working Area

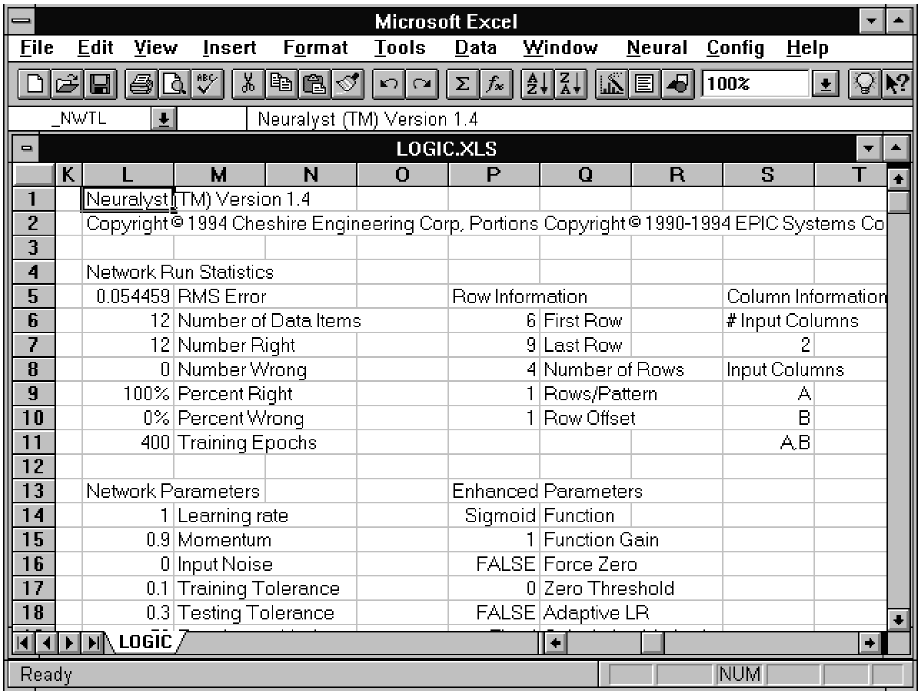
The Neuralyst Working Area is the area set aside for Neuralyst to save its internal data, save its control parameters and display its progress to you.

The Working Area is divided into several distinct areas: the Title Block, the Statistics Block, the Parameter Block, the Row Description Block, the Column Description Block, the Network Description Block, and the Network Weights Block. Also within the Working Area are three areas which are dedicated to the Genetic Supervisor: the Genetic Statistics Block, the Genetic Parameter Block, and the Genetic State Block.

Warning: Some fields in the Working Area depend on Excel calculations to be set to their proper values. If you have the Excel calculation mode set to Manual, instead of Automatic, then the information in these fields will not be properly updated. This may result in improper initialization or incorrect statistical updates while Neuralyst is executing.

It is generally best to have the Excel Calculation mode set to Automatic when working with Neuralyst. However, in some cases, particularly with large amounts of data or formulas, Excel is more responsive if Manual Calculation is set. In such cases, you may keep it set to Manual, switching to Automatic when you execute an operation from the Neuralyst **Config** menu or when actually training, testing, or running.

8.3.1 Title Block


Microsoft Excel
File Edit View Insert Format Tools Data Window Neural Config Help
100%
_NWTL
LOGIC.XLS
LOGIC
Ready
NUM

Neuralyst (TM) Version 1.4

	K	L	M	N	O	P	Q	R	S	T
1		Neuralyst (TM) Version 1.4								
2		Copyright © 1994 Cheshire Engineering Corp, Portions Copyright © 1990-1994 EPIC Systems Co								
3										
4		Network Run Statistics								
5		0.054459	RMS Error			Row Information			Column Information	
6		12	Number of Data Items			6	First Row		# Input Columns	
7		12	Number Right			9	Last Row		2	
8		0	Number Wrong			4	Number of Rows		Input Columns	
9		100%	Percent Right			1	Rows/Pattern		A	
10		0%	Percent Wrong			1	Row Offset		B	
11		400	Training Epochs						A,B	
12										
13		Network Parameters				Enhanced Parameters				
14		1	Learning rate			Sigmoid	Function			
15		0.9	Momentum			1	Function Gain			
16		0	Input Noise			FALSE	Force Zero			
17		0.1	Training Tolerance			0	Zero Threshold			
18		0.3	Testing Tolerance			FALSE	Adaptive LR			

Figure 8-19 Title, Statistics & Parameter Blocks

The Title Block identifies the name of the program and provides the copyright notice for Neuralyst.

The Title Block is also used to carry the version number of Neuralyst that created and maintained the Working Area. This is useful when upgrading to new versions of Neuralyst. When a new version of Neuralyst is loaded with an old Neuralyst worksheet, it will check the version number and update the worksheet if possible or inform you of the need for conversion.

The Title Block also defines the top left corner of the Neuralyst Working Area. Any cell in the same row or higher and in the same column or higher is subject to modification by Neuralyst.

8.3.2 Statistics Block

The Statistics Block allows you to monitor the progress of training of the neural network and shows the success of prediction in test cases. The values in this block are updated for each cycle of the defined data during training or at the end of the defined data during testing or running.

RMS Error shows the *root mean square error* (the square root of the average of the error terms squared, a better measure than a simple average since errors of opposite signs cannot cancel each other) during training or testing between the outputs and the targets; it is not meaningful when running since there are no valid targets. When the neural network is training, the RMS Error should generally decrease at a steady rate, though there may be some short term fluctuations which cause small increases. If there are large fluctuations or the RMS Error is increasing, then some adjustment of Learning Rate or Momentum is needed through the **Set Network Parameters** command.

Number of Data Items represents the total count of cells in the Output columns or Target columns (remember Output columns are paired with Target columns) that are in the Problem Definition Area matching the current operating mode. For example, if there are two Output columns with 6 training rows (TRAIN flag) and 3 testing rows (TEST flag), then executing **Train Network** will cause this count to be 12 (2 columns times 6 rows) or executing **Run/Predict with Network** will cause this count to be 6 (2 columns times 3 rows).

Number Right represents how many of the data items counted in Number of Data Items have matching outputs and targets within the effective Tolerance parameter. Number Wrong represents those that are not within the effective, Training or Testing, Tolerance parameter. Percent Right and Percent Wrong are similar, except representing percentage values instead of counts.

When the network is training successfully, you will see these values change from predominantly wrong or a mix of right and wrong to mostly or all right. When the Number Right matches the Number of Data Items (the Percentage Right is 100%), then training will stop automatically. When the network is testing, the Number Right and Number Wrong have similar meanings but in this case should be treated more as “scores” for the level of predictive success rather than a running indicator of learning progress. When the network is running, these values have no meaning.

The number of training epochs the neural network has undergone is recorded by the value of Training Epochs. In some circumstances, the user may prefer to limit training to a fixed number of epochs; Training Epochs provides this capability. In other circumstances, particularly when the user is trying to measure how well the neural network is learning under changing conditions, Training Epochs provides a reference point for evaluating training performance.

8.3.3 Parameter Block

The Parameter Block consists of two parts. The first part contains the current values of the Network Parameters. They may be changed through the **Set Network Parameters** command. The parameters displayed are Learning Rate, Momentum, Input Noise, Training Tolerance, Testing Tolerance, Epochs per Update, Epoch Limit, Time Limit, and Error Limit. The second part contains the current settings of the Enhanced Parameters. They may be changed through the **Set Enhanced Parameters** command. The parameters displayed are Function, Function Gain, Force Zero, Zero Threshold, and Adaptive Learning Rate.

Learning Rate determines the magnitude of the correction term applied to adjust each neuron’s weights when training. Learning Rate corresponds to the variable ***LR*** in Equation 3-3. Learning Rate must be positive, is usually adjusted in the range of 0 to 1 and has a default value of 1. Large values of Learning Rate will cause the network to train more quickly, but too large a value may cause the training to be unstable and no learning will occur.

Momentum determines the “lifetime” of a correction term as the training process takes place. Momentum corresponds to the variable ***M*** in Equation 3-6. Momentum must be greater than or equal to 0 but less than 1 and has a default of 0.9. Values of Momentum closer to 1 will cause the neural network to retain more of the impact of previous corrections to the current corrections. Values of

Momentum close to 0 will allow mostly or only the current corrective term to have an effect. Momentum helps to smooth out the training process so that no single aberrant instance can force learning in an undesirable direction.

The Input Noise parameter enables the addition of noise during the training process. Input Noise has a value of 0 as a default, meaning no noise is added. Input Noise should be set between 0 and 1, meaning 0% to 100% of the input range will be set as a noise range. For example, if the input ranges from 0 to 15 and the Input Noise is set to 0.1, then random values ranging from 0 to 1.5, 10% of 15, will be added to or subtracted from each input value. Input Noise is only in effect when training.

The Training Tolerance and Testing Tolerance parameters define the percentage error allowed in comparing the neural network output to the target value to be scored as “Right” during training and testing, respectively. The Tolerance parameters should be between 0 and 1; with Training Tolerance having 0.1, or 10%, as a default and Testing Tolerance having 0.3, or 30%, as a default. The Tolerance parameters are defined as a percentage of the range between the highest and lowest values in the Target columns. For example, if values in the Target columns ranged from 100 to 300, a Tolerance of 0.2, corresponding to 20%, would allow errors of 20% of 200 (300 minus 100) or 40. The Training Tolerance value has no effect on the learning algorithm. However, when Neuralyst finds 100% Right, as defined by Training Tolerance, it will automatically stop training. The Testing Tolerance value is used only for scoring during testing or running.

The Epochs per Update parameter allows you to control the number of epochs between updates of the neural network results which are displayed in the Network Run Statistics block in the worksheet. Epochs per Update has a default value of 1. However larger values will mean less frequent communication between the neural network and the worksheet, reducing overall training time.

The Epoch Limit parameter sets a maximum number of training epochs the neural network will undergo in those situations where you wish to control the number of training epochs rather than setting a

Training Tolerance. Epoch Limit has a default value of 0, which means that there is no limit set.

The Time Limit parameter sets a maximum number of hours (enter minutes as a decimal fraction of an hour, for example 0.25 would be ¼ hour or 15 minutes) to continue training. Time Limit has a default value of 0 which means that there is no limit set.

The Error Limit parameter sets a maximum increase in the RMS Error value from training epoch to training epoch. Under normal circumstances, a neural network that is training properly will steadily decrease the RMS Error. However, if the neural network exceeds its capacity or starts experiencing some effects of overtraining, some epochs may increase the RMS Error. Setting an Error Limit value will stop the training process if the RMS Error increases on any given epoch more than the amount set. Error Limit has a default value of 0 which means that there is no limit set.

When more than one of Epoch Limit, Time Limit, or Error Limit, is set, whichever limit is reached first will terminate the training process. When no limit is set or if limits are set but the limit condition is never reached, the normal termination condition occurs when the neural network outputs meet the Training Tolerance criteria for all training data.

The Function parameter allows a change of the neuron activation function described in Section 3.3. The functions that are available are: Gaussian, Linear, Sigmoid, Step, Hyperbolic and Augmented Ratio. The Gaussian function is a smooth, differentiable function which transforms values within a range close to 0 to +1, very large negative or positive values to 0, and transforms intermediate values with varying proportion. The Linear function provides a continuous response multiplying input values by a constant scaling factor until the limits of 0 or +1 are reached. The Sigmoid function is also a smooth, differentiable function, which forces large negative or large positive values to 0 or +1, respectively, but transforms intermediate values with varying proportion. The Step function is a sharp switching function, all negative inputs are forced to 0 and all positive inputs are forced to +1. The Hyperbolic function is another smooth, differentiable function, which forces large negative or large positive values to -1 or +1, respectively, but transforms intermediate values with varying proportion. The Hyperbolic function is actually the Hyperbolic Tangent function and is very similar to the Sigmoid function, except it is negative-going. The Augmented Ratio function is a smooth, differentiable function which transforms values within a range close to 0 to 0, very large negative or positive values to 1, and transforms intermediate values with varying proportion. The Augmented Ratio function is similar to the Gaussian function, but inverted. Thus selecting different neuron functions can significantly change the behavior of the neuron.

The Function Gain parameter allows a change in the scaling or width of the selected Function. Increasing Function Gain will narrow the central region in the Gaussian function or steepen the slope of the Linear function and the Sigmoid function. Decreasing the Function Gain, correspondingly, will broaden the central region in the Gaussian Function or make the slope of the Linear or Sigmoid functions shallower. With a Function Gain setting of 1, all three functions will cover the normal input range of the neural network smoothly. The Function Gain applies to all of the Activation Functions except the Step function.

The Force Zero parameter enables a special mode which scans the weights after every training epoch and sets those weights which are close to 0 to be 0. The Zero Threshold parameter selects the threshold below which weight values are set to 0. Thus, a Zero Threshold value of 0.01 with Force Zero enabled will cause a weight value of 0.009 to be made to be 0, while a value of 0.15 will be left unchanged.

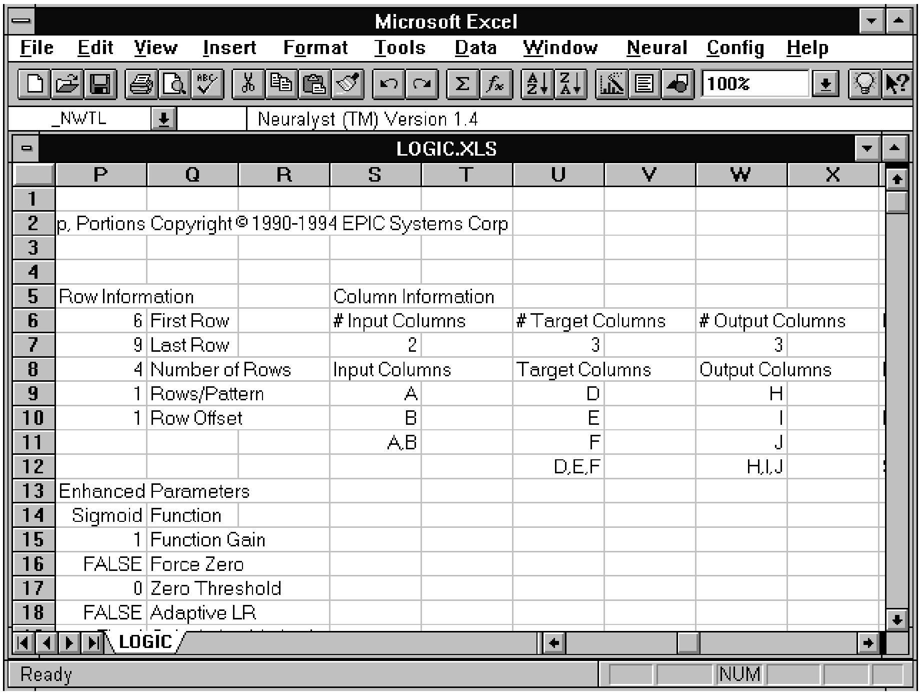
The Adaptive Learning Rate parameter enables a special mode which evaluates the RMS Error after every training epoch and causes a revision of the Learning Rate for the next training epoch. If the RMS Error is high, then a high Learning Rate will be set, as the RMS Error is reduced, the Learning Rate will be reduced correspondingly. The net effect is to speed up the training process when the neural network is far away from the correct weights, but to slow it down as it gets closer.

The Calculation Method parameter allows a selection of Fixed Point calculation or Floating Point calculation. Fixed Point mode is a very fast and memory efficient technique, allowing speeds up to 2-3x a computer equipped with a math co-processor using Floating Point calculations. It is inherently less precise, but is satisfactory for the majority of neural network problems that require precision to only 2-3 places. Floating Point mode is more precise and has higher resolution, but requires a math co-processor — not present on many 386, 486SX, 486SL, 68030, and 68LC040 processors — more memory and more processing time. Fixed Point mode was the native mode of earlier versions of Neuralyst; but now either mode can be selected depending on the circumstances.

The Scaling Margin parameter adds additional headroom, as a percentage of range, to the rescaling computations used by Neuralyst in preparing data for the neural network or interpreting data from the neural network. All Input, Target, and Output data actually processed by the neural network must be in the range 0 to 1. As a consequence, all data presented or interpreted is rescaled from the actual worksheet values to values in the range of 0 to 1. Setting Scaling Margin above 0, increases the amount of headroom or margin, decreasing the range onto which an Input, Target, or Output, is mapped. As an example, if the Scaling Margin is set to 0.1, or 10%, then 10% margin is allowed and split between the high end and the low end of the range, resulting in a mapping to the range 0.05 to 0.95 rather than 0 to 1. This margin can facilitate precision in problems that have continuously variable data or that may exceed the current values in the future by a small amount. Scaling Margin modifies the Min or Max limits that may be set for an input or target by the designated amount.

The options of Input Noise, Force Zero, and Adaptive Learning Rate will typically increase the total number of training epochs it takes to achieve a successfully trained neural network over a “no-frills” approach. However, when used judiciously, the quality of training - and the corresponding quality of predictions — of the neural network can be notably improved. Fixed Point calculation mode generally provides the fastest training; however, for continuously variable inputs or outputs requiring high resolution, Floating Point calculation mode can be more precise. In addition, on high performance Pentiums or PowerPCs, Floating Point approaches Fixed Point speed. Scaling Margin is another control which is useful for problems that have continuously variable inputs or outputs to allow higher precision.

8.3.4 Row Description Block


Microsoft Excel
File Edit View Insert Format Tools Data Window Neural Config Help
100%
_NWTL
LOGIC.XLS
LOGIC
Ready
NUM

Neuralyst (TM) Version 1.4

	P	Q	R	S	T	U	V	W	X
1									
2	p, Portions Copyright © 1990-1994 EPIC Systems Corp								
3									
4									
5	Row Information			Column Information					
6	6	First Row		# Input Columns		# Target Columns		# Output Columns	
7	9	Last Row		2		3		3	
8	4	Number of Rows		Input Columns		Target Columns		Output Columns	
9	1	Rows/Pattern		A		D		H	
10	1	Row Offset		B		E		I	
11				A,B		F		J	
12						D,E,F		H,I,J	
13	Enhanced	Parameters							
14	Sigmoid	Function							
15	1	Function Gain							
16	FALSE	Force Zero							
17	0	Zero Threshold							
18	FALSE	Adaptive LR							

Figure 8-20 Row & Column Description Blocks

The Row Description Block contains the row information bounding the defined data and the spacing and offset information describing the pattern window.

The First Row and Last Row values mark the beginning and end of the Problem Definition Area, respectively. The Total Number of Rows represents the rows within and including those bounding rows.

Rows/Pattern describes the number of rows that are included in each pattern presented to the neural network. It can be considered as describing the *height* of the pattern window. Row Offset describes the number of rows to shift between patterns presented to the neural network. It can be considered as describing the step size of the pattern window. A Row Offset equal to Rows/Pattern means that the pattern window is stepped the same number of rows as its height, in other words a series of contiguous but non-overlapping two-dimensional input patterns is defined.

The intersection of the rows set in the Row Description Block and the columns listed in the Column Description Block constitutes the Problem Definition Area.

8.3.5 Column Description Block

The Column Description Block contains the column lists that describe the columns that constitute the defined data and represent each component of the pattern window.

The first three fields of the block, # Input Columns, # Target Columns, and # Output Columns, have counts of the number of columns of each type. Below each count is the actual list of Input Columns, Target Columns, and Output Columns. Each column is listed in its own cell in sequence, but a terminating cell contains all the columns of that type as one string.

The Mode Flag Column? field shows either FALSE, indicating no Mode Flag Column is set, or the column that has been set. If a Mode Flag Column is not set then all rows, as defined by the other parameters, are input to the neural network when either **Train Network** or **Run/Predict with Network** are executed. If a Mode Flag Column is set then the values in the fields are used to switch between training or testing/running modes.

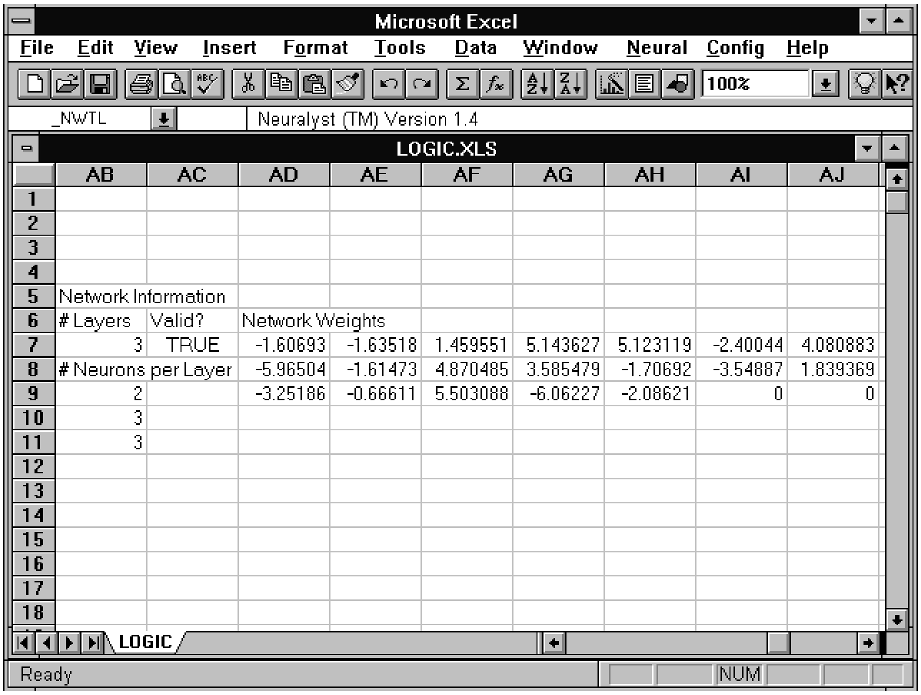
The Min Scale Row? field shows either FALSE, indicating no MIN **scale row is set, or the number of the row that has been set. If no** MIN **scale row is set, then the neural network takes its minimum value for rescaling by searching each column to find the minimum. If a** MIN scale row is set then the values in each column are used as the minimum for rescaling purposes.

The Max Scale Row? field shows either FALSE, indicating no MAX scale row is set, or the number of the row that has been set. If no MAX scale row is set, then the neural network takes its maximum value for rescaling by searching each column to find the maximum. If a MAX scale row is set then the values in each column are used as the maximum for rescaling purposes.

The Symbol Row? field shows either FALSE, indicating no SYMBOL row is set, or the number of the row that has been set. If no SYMBOL row is set, then the neural network expects that all column data will be numeric only. If a SYMBOL row is set then those columns that have a Symbol List defined should contain symbols that are only from that Symbol List, those columns that do not have a Symbol List defined should contain numeric values only.

The intersection of the rows set in the Row Description Block and the columns listed in the Column Description Block constitutes the defined data and the description of the pattern window.

8.3.6 Network Description Block


Microsoft Excel
File Edit View Insert Format Tools Data Window Neural Config Help
100%
LOGIC.XLS
LOGIC
Ready
NUM

_NWTL   Neuralyst (TM) Version 1.4

	AB	AC	AD	AE	AF	AG	AH	AI	AJ
1									
2									
3									
4									
5	Network Information								
6	# Layers	Valid?	Network Weights						
7	3	TRUE	-1.60693	-1.63518	1.459551	5.143627	5.123119	-2.40044	4.080883
8	# Neurons per Layer		-5.96504	-1.61473	4.870485	3.585479	-1.70692	-3.54887	1.839369
9	2		-3.25186	-0.66611	5.503088	-6.06227	-2.08621	0	0
10	3								
11	3								
12									
13									
14									
15									
16									
17									
18									

Figure 8-21 Network Description & Weights Blocks

The Network Description Block contains information describing the structure of the neural network.

# Layers indicates the number of layers in the neural network. Two of the layers are there by default, the input and output layers, and are always part of the count so the minimum value is 2.

# Neurons per Layer lists the number of neurons in each layer in sequential rows. The first row of the field is the number of neurons in the input layer, the last row is the number of neurons in the output layer. These two numbers are fixed and defined by the number of Input columns and Target or Output columns. The hidden layers are variable and the number can be set when defining the neural network structure with **Set Network Size**.

There is also a Valid? flag which indicates whether or not the information in the Network Weight Block is correct or may have been changed and needs to be redefined.

8.3.7 Network Weights Block

The Network Weights Block contains a copy of the weights of the neural network and is kept current as the neural network is trained.

When first initialized, the weights are set to random values and there is no meaning to their values. After the neural network has been trained, the value of the weights represent all the learning the neural network has done to that point. Once a neural network has been trained, you should be careful about accidentally re-initializing or otherwise changing these weights; there is no way to reconstruct the weights without retraining the neural network.

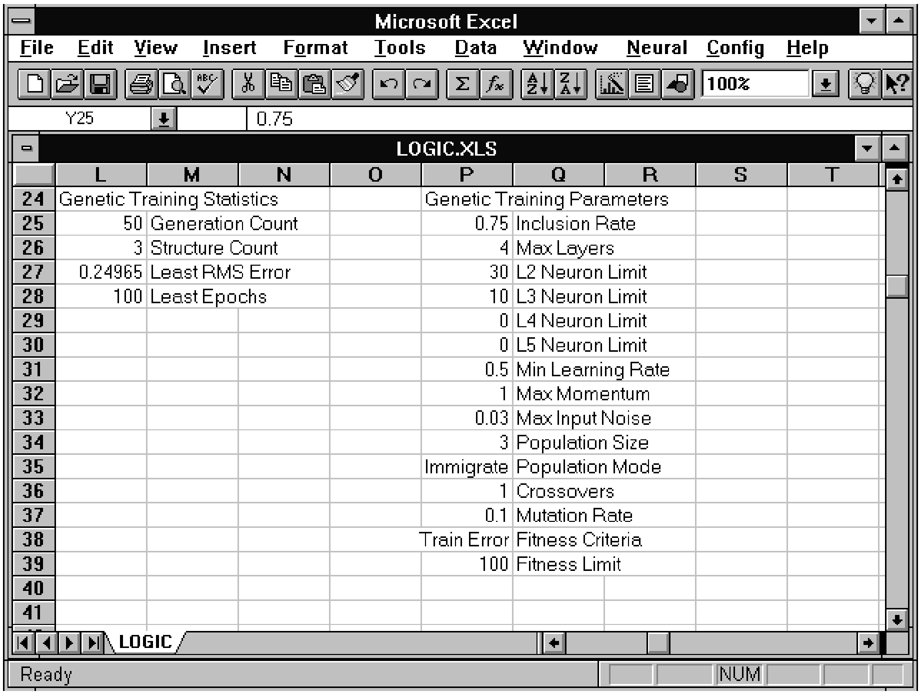
It is possible to make a copy of the current weight set and save it elsewhere on the worksheet. Then if you want restore the saved weight set, copy it back to the Network Weight Block. In order for this to work, the network size must not have changed or the network size must be restored to the same configuration as at the time the weight set was saved.

In some cases, it is possible to analyze or interpret the weights. The **Histogram Weights** and **Unpack Weights** commands provide you with some methods of viewing the weights when attempting this.

The **Histogram Weights** command will display the distribution of weight values of the neural network in a histogram chart (see Section 8.2.10).

The **Unpack Weights** command provides a formatted display of the neural network weights. This display is organized by neural network layers and by neurons in each layer (see Section 8.2.11).

8.3.8 Genetic Statistics Block


Microsoft Excel
File Edit View Insert Format Tools Data Window Neural Config Help
100%
Y25
0.75
LOGIC.XLS
LOGIC
Ready
NUM

	L	M	N	O	P	Q	R	S	T
24	Genetic Training Statistics				Genetic Training Parameters				
25	50	Generation Count			0.75	Inclusion Rate			
26	3	Structure Count			4	Max Layers			
27	0.24965	Least RMS Error			30	L2 Neuron Limit			
28	100	Least Epochs			10	L3 Neuron Limit			
29					0	L4 Neuron Limit			
30					0	L5 Neuron Limit			
31					0.5	Min Learning Rate			
32					1	Max Momentum			
33					0.03	Max Input Noise			
34					3	Population Size			
35					Immigrate	Population Mode			
36					1	Crossovers			
37					0.1	Mutation Rate			
38					Train Error	Fitness Criteria			
39					100	Fitness Limit			
40									
41									

Figure 8-22 Genetic Statistics & Parameters Blocks

The Genetic Statistics Block displays a brief summary of Genetic Supervisor results when it is in operation. There are four fields: Generation Count, Structure Count, Least RMS Error, and Least Epochs. The Structure Count field is updated every time another structure is evaluated. The Generation Count, Least RMS Error, and Least Epoch fields are updated after all the structures of a population are evaluated.

The Generation Count and Structure Count fields maintain numeric counts of the current structure under evaluation. The Generation Count will count up to the generation limit set in **Run Genetic Supervisor**. When this count is reached, genetic optimization will halt. At this time, the best structure may be retrieved, or genetic optimization may be resumed under different Population Mode, Genetic Operator, or Fitness Criteria settings. The Structure Count will count up to the Pool Size set in the Population Mode settings. On evaluating the last structure of a population, the structure that best meets the Fitness Criteria that is set will have its RMS Error and epoch count reported in the Least RMS Error and Least Epochs fields.

8.3.9 Genetic Parameters Block

The Genetic Parameters Block contains fifteen parameters, Inclusion Rate, Max Layers, four Layer **n** Neuron Limits, Learning Rate, Momentum, Input Noise, Pool Size, Pool Mode, Crossovers, Mutation Rate, Fitness Criteria, and Fitness Limit, that bound the limits of the genetic optimization process and change the behavior of the Genetic Supervisor.

The Input column Inclusion Rate controls the average rate of Input column inclusion in structure initialization. The user may select a nominal percentage from 1 to 100% which represents the average rate of inclusion. If the user wants to force all Input columns to be included, then the user can set 100%. The default setting is 75%.

The first and last layer of each neural network is automatically determined by the number of inputs and outputs defined for the original neural network model. Within those constraints, the optimized neural network can have from two to six layers and a wide range of variations in number of neurons per layer. The user can constrain the maximum number of layers and the maximum number of neurons per hidden layer. An implicit rule is that if a lower hidden layer is zero, then higher hidden layer must be zero. The default maximum settings are 4 layers, an input layer, 2 hidden layers, and an output layer; with 30 neurons in the first hidden layer and 10 neurons in the second hidden layer.

There are three network parameters which can be optimized by the Genetic Supervisor: Learning Rate, Momentum, and Input Noise. Learning Rate and Momentum can vary from almost 0 to 1, while the Input Noise can vary from 0 to 0.1. The user can constrain the minimum value of Learning Rate, the maximum value of Momentum, and the maximum value of Input Noise. The default values are 0.5 for Learning Rate, 1 for Momentum, and 0.03 for Input Noise.

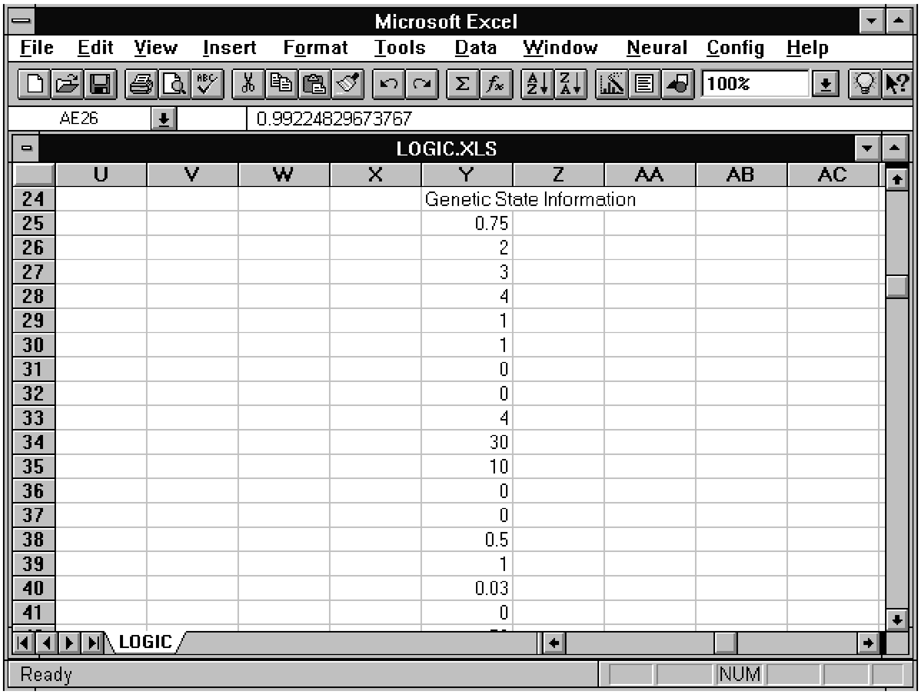
There are two controls which determine the management of populations. The first is a control which sets the total population size. Changing this sets the number of structures initialized or evaluated in each generation. The second is a control which chooses among the three modes: Closed Pool, Immigration, or Emigration. With Closed Pool set, then the initial population pool will only be cross-bred or mutated with no new structures initialized. With Immigration set, the population pool will be evolved and every generation new structures will replace the weakest structures of the existing pool. With Emigration set, the population pool will be evolved and every generation the best structure will be emigrated to an entirely new population. The default settings are for population size to be 3 and Immigration to be enabled.

There are two settings the user can control to determine the mechanism for genetic optimization. These settings are Crossover frequency and Mutation Rate. The primary mechanism is cross-breeding controlled by Crossover frequency, where structures are intermingled to create new structures. The crossover rate determines the amount of intermingling. The maximum setting for crossover rate is 10. The secondary mechanism for creating new structures is mutation. With mutation, structures are chosen at random, then features are randomly changed to new values. The mutation rate sets the percentage of structures which will undergo a mutation rather than a cross-breeding to create the new population. The maximum setting for mutation is 100%. Mutation is always in place of cross-breeding; the two never occur at the same time on a single structure. The default settings are for crossover rate to be 1 and mutation rate to be 10%.

The evaluation of each structure is based training or testing while comparing the best RMS Error level achieved or the least number of epochs. There are four modes which can be set: Train Epochs, Train

Error, Test Epochs, and Test Error. Train Error finds the structure with the least RMS Error while holding epochs to the set Fitness Limit. Train Epochs finds the structure with the least epochs to achieve the RMS Error level set in Fitness Limit. These two operate on training data only. Test Error finds the structure with the least RMS Error in the test set data while holding epochs to the Fitness Limit. Test Epochs finds the structure with the least epochs to achieve the RMS Error level of the test set data as set in Fitness Limit. These two measure RMS Error by evaluating the testing data. The default Fitness Criteria mode is Train Error with Fitness Limit set to 100 epochs. This will optimize RMS Error while training each candidate solution to an epoch limit of 100.

8.3.10 Genetic State Block


Microsoft Excel
File Edit View Insert Format Tools Data Window Neural Config Help
100%
LOGIC.XLS
LOGIC
Ready
NUM

AE26 0.99224829673767

	U	V	W	X	Y	Z	AA	AB	AC
24					Genetic State Information				
25					0.75				
26					2				
27					3				
28					4				
29					1				
30					1				
31					0				
32					0				
33					4				
34					30				
35					10				
36					0				
37					0				
38					0.5				
39					1				
40					0.03				
41					0				

Figure 8-23 Genetic State Block

The Genetic State Block contains information necessary for the Genetic Supervisor to retain intermediate state information. It is not a complete state information area and so it is not sufficient to support a full reload operation between Neuralyst sessions.

There is no command to support interpretation or display of the genetic state information.

Appendix A
Help Me!

Some of the more common difficulties that you may experience have been listed here. If you have a problem, please review this section and see if the solution or answer has been provided. If there is an Error Message associated with your problem, please check Appendix B for a list of Error Messages and their causes. If you still experience difficulties, then please consult with Cheshire Customer Service, as discussed in Appendix C.

A.1 Installation Problems

A.1.1 Windows Problems

I tried to run the install program but I got an error saying “`This program requires Microsoft Windows.`”

> The install software can only be run from within Windows. Start up Windows and then follow the install instructions in Chapter 2 of the Neuralyst User’s Guide.

When I ran the install program I got an error saying “`Unable to copy file to destination directory. Check your free disk space.`”

> The Neuralyst software requires about 300K bytes of free disk space. If the install fails due to lack of space, try specifying a different disk drive than `C:` in the box holding the directory to install to, or else clear some room on your disk and try again.

The install program fails with the message “`Unable to create directory to copy to`.” What’s happening?

> The install program allows you to specify the directory to install the Neuralyst software to. The default is `C:\NEURLYST`, which is fine for most systems. If you get this message it indicates that the directory that you have specified cannot be created. Make sure you have given a drive letter and destination directory that would be legal on your system. If you have no drive `C`, for example, edit the default to a drive appropriate for your system before continuing.
>
> The Neuralyst installer will create the destination directory to hold the files, but it won’t create a whole chain of directories. If you specify `C:\X\Y\NEURLYST` for your destination, `C:\X\Y` must already exist. The install program will then create the `NEURLYST` subdirectory within the `\X\Y` directory. However, it won’t create `\X` and `\X\Y` for you. You would have to do that before running the install program.

I reinstalled the software and then when I ran Neuralyst, I got an error message saying “`Change Disk. Cannot find NEUR14.DLL. Please insert in Drive A:`.” What’s happening?

> This message can only appear in a situation where you are doing a second or later install of Neuralyst. It will not occur on a normal first installation. If you reinstall Neuralyst you must exit Windows *after* the installation and then restart Windows. That will prevent this message from occurring.
>
> The problem occurs because of the way Windows handles program libraries like the `NEUR14.DLL` file included with Neuralyst. If an old version of `NEUR14.DLL` has been running and a new version is installed, Windows gets confused about whether you want the old version or the new one to be running. Restarting Windows clears the old version from memory, eliminating the confusion.

The install program still isn't working for me even after following all this advice. What can I do?

> Please contact Cheshire Customer Service for assistance. We are committed to getting you up and running with Neuralyst as soon as possible!

A.1.2 Macintosh Problems

Something went wrong with my install.

> You should first make sure Excel has been properly installed. Then clear the Neuralyst folder from your hard disk. Re-drag the Neuralyst folder from the installation disk to your hard disk. Be sure to launch Neuralyst from the Excel macro file named Neuralyst, NOT from the Cheshire icon labeled Neuralyst Lib. If you still have problems, call Cheshire Customer Service.

I installed Neuralyst on my small screen Macintosh running with System 7; but when I tried to run Neuralyst, I couldn't find the Neural and Config menus; the N might have been there, but that was all.

> On small screens (those with horizontal resolution of 512 pixels), the combination of the Excel menus and the System 7 Multifinder menu leaves no room for any additional menu items. After installing Neuralyst, you should copy the contents of the folder Small Screen I/F into the Neuralyst folder. This will replace the standard menu interface with a special menu interface that is called on command. See the documentation in the folder for further details.

A.2 Neuralyst Problems

Whenever I give a command I get the error "`Initialize or Reload the worksheet`."

> You have to tell Neuralyst that you want to use a particular worksheet before you can give any Neuralyst commands. If it is a new worksheet that you haven't run Neuralyst on before, use the **Init Working Area** command to set it up. If it's a worksheet that you have used Neuralyst on in the past, use the **Reload Network** command. After one of these two commands is given Neuralyst will know that you want it to use this worksheet for its operations.

I try to give a command but I get an error telling me to "`Select worksheet XYZ first`."

> After you tell Neuralyst which worksheet to use with either the **Init Working Area** or **Reload Network** commands, it remembers that worksheet name. Then when you give another Neuralyst command it checks to make sure that worksheet is still the active one (the one in front of all others). If it isn't, you get this message telling you to bring your worksheet to the front.
>
> If you actually want to start using a different worksheet than the one you were using before, bring that one to the front and give the **Reload Network** command. Then if you want to switch back to the first one, bring it back to the front and do **Reload Network** again. It's always safe to do **Reload Network** in order to make sure that the worksheet in the front is the one that Neuralyst is working with.

When I give the Reload Network command, I get a message saying "`Worksheet XYZ must be initialized`."

> This message means that Neuralyst does not think that your worksheet ever got initialized with the **Init Working Area** command. Neuralyst remembers this fact by creating a named cell in the top left corner of the Working Area, the cell which has the Neuralyst program name and version number. This cell must have the name **_NWTL**.

If you have accidentally deleted that name from your worksheet, you can re-create it by selecting the cell with the Neuralyst program name, and giving Excel's **Define Name** command. Type **_NWTL** in the box labeled **Name:** and hit **OK**. Neuralyst will now be able to find the Working Area on your sheet when you give the **Reload Network** command.

After I set the network size, I got a message saying "`Input Column C is bad.`"

Neuralyst checks your columns to make sure that they hold valid numeric data. This message indicates that your column **C** has some invalid data in the rows that Neuralyst is looking at. Take a look at the Row Information block in the Working Area; it's just to the right of the Network Run Statistics block. Look at the entries for First Row and Last Row to see what range of rows Neuralyst is working with. Then look carefully at your column **C** in that range of rows. Find the non-numeric data and replace it. Blank cells will be treated as zeros, which may be OK for your particular application, but any cells with text or error values will not be acceptable.

I made a change to the values in one of my target columns, but Neuralyst seems to be training to the old value, ignoring the change.

After any change to the values in your Input, Target, or Mode Flag columns, you need to give the **Reload Network** command to force Neuralyst to re-read your columns and pick up the changed data. Neuralyst will also re-read your column data after the **Set Network Size** command, or if you have changed the set of rows and columns in use by giving the **Set Rows** or one of the column commands.

A.2.1 Windows Problems

I ran two Neuralysts at the same time so I could have two different Neuralyst worksheets running, but I am seeing strange behavior and getting bad results.

> You can't run two Neuralysts at the same time. Windows is only able to load one instance of the Neuralyst program library `NEUR14.DLL` at one time. When two instances of Excel running with Neuralyst both try to interface with the single instance of `NEUR14.DLL` that is available, then data transfers and other communications gets confused and errors will result.
>
> If you want to run two or more instances of Excel at the same time, that is OK so long as only one of them is running with Neuralyst. Within a single instance of Excel running with Neuralyst, you may open as many worksheets as Excel will allow and switch between them using **Reload Network** as described in the previous problem.

I got an error message which the manual said represents an internal problem in the Neuralyst software. What should I do?

> The Neuralyst design team has made every effort to provide you with reliable, bug-free software. However, errors may still exist. Should you encounter such a problem, please inform Cheshire Customer Service of the error message and the conditions under which it occurred. We will work with you to resolve the problem as quickly as possible.
>
> Neuralyst keeps a great deal of its data in the Working Area. Any accidental alteration you might make to this data could cause Neuralyst to behave incorrectly or to abort with an error message. Also, much of the Neuralyst functionality resides in a program library which is loaded with Excel. If that library should become corrupted, it will be necessary to exit and then re-start Windows to load a fresh copy of the program. Cheshire Customer Service can provide more information in these situations.

A.2.2 Macintosh Problems

I got an error message which the manual said represents an internal problem in the Neuralyst software. What should I do?

> The Neuralyst design team has made every effort to provide you with reliable, bug-free software. However, errors may still exist. Should you encounter such a problem, please inform Cheshire Customer Service of the error message and the conditions under which it occurred. We will work with you to resolve the problem as quickly as possible.
>
> Neuralyst keeps a great deal of its data in the Working Area. Any accidental alteration you might make to this data could cause Neuralyst to behave incorrectly or to abort with an error message. Also, much of the Neuralyst functionality resides in a program library which is loaded with Excel. If that library should become corrupted, it may be necessary to Restart the Macintosh and re-load Neuralyst and Excel to load a fresh copy of the program. Cheshire Customer Service can provide more information in these situations.

Appendix B
Error Messages

Neuralyst Error Messages have been included here in alphabetical order. If the Error Message you see is not listed here, then it is likely that the message is being given by DOS, Windows, or Excel. In that case, you may have to refer to their respective manuals for help.

```
Bad arg to GenNextPop
Bad arg to Init
Bad arg to InitPop
Bad arg to SetParams
Bad arg to SetEnhanced
Bad arg to SetCutoffs
Bad arg to SetWeights
Bad arg to SetColumn
Bad arg to SetColumn for blanks
Bad arg to TrainNextSchema
```

These messages should never appear; they represent an internal problem in the Neuralyst software. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area.

```
Bad column in SelectDataMode
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that an invalid column was created.

```
Bad column string in GetSchema
Bad selector in GetSchema
```

These messages should never appear; they represent an internal problem in Neuralyst and indicate that an invalid value was passed to the Genetic Supervisor. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area.

```
Bad data in ReloadPop
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that an invalid value was passed to the Genetic Supervisor. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area.

```
Bad parameter in InitSuper
Bad parameter in InitPop
Bad parameter in TrainNextSchema
Bad parameter in GenNextPop
Bad parameter in ReloadPop
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that an invalid value was passed to the Genetic Supervisor. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area.

```
Blank flag column invalid
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that an invalid column was created.

```
Call Init before InitSuper
Call InitSuper before InitPop
Call InitPop before TrainNextSchema
Call InitPop before GenNextPop
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that the neural network was not set up before the Genetic Supervisor was called or that the Genetic Supervisor was not called in the proper sequence.

```
Can't have more than one MIN row
Can't have more than one MAX row
Can't have more than one SYMBOL row
```

These messages will appear if more than one row of that type was marked in the Mode Flag column.

```
Can't set symbols before Init
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that a column was transferred before a network was initialized.

`Changing the activation function or calculation method will reinitialize the network, causing any learning done so far to be forgotten. Proceed?`

This warning appears if you are changing the activation function type or gain or the calculation method when using the **Set Enhanced Parameters** command. It allows you to confirm your intentions.

`Cheshire Engineering Corporation prefers that this file be called NEURLYST.XLM.` *Windows only*

This message appears when Neuralyst is loaded if the `NEURLYST.XLM` file has been copied or renamed.

`Cheshire Engineering Corporation prefers that this file be called Neuralyst.` *Macintosh only*

This message appears when Neuralyst is loaded if the Neuralyst file has been copied or renamed.

```
Do Set Genetic Parameters first.
```

This message appears if an attempt is made to execute **Run Genetic Supervisor** before **Set Genetic Parameters** has initialized the Genetic Supervisor state.

```
Do Set Mode Flag Column first.
```

This message appears if an attempt is made to execute **Set Mode Rows** before **Set Mode Flag Column** has initialized the location of the Mode Flag column.

```
Do Set Rows first.
```

This message appears if an attempt is made to execute **Set Mode Rows** before **Set Rows** has initialized the range of defined rows.

```
Establish input columns before InitSuper.
Establish gap column before InitSuper.
Establish mode flag column before InitSuper.
Establish target columns before InitSuper.
```

This message should never appear; it represents an internal problem in Neuralyst and indicates that the neural network was not set up before the Genetic Supervisor was called.

```
Failed to translate in SetColumnEx.
```

This message appears if the symbol translator failed to identify or parse the symbols present in a column.

```
Generate Min values for currently defined Input and Target
columns?
Generate Max values for currently defined Input and Target
columns?
```

These messages appear after a **Set Mode Rows** command has established a Min or Max row. It allows you to confirm an operation to automatically generate Min or Max values based on data defined previously with **Set Rows**, **Add Input Columns**, and **Add Target Columns**. These values will overwrite any data currently in the Min or Max fields of the applicable columns.

`Genetic Supervisor data may not be compatible with data saved on the worksheet. Do Set Genetic Parameters to reinitialize.`

> This message appears if an attempt is made to execute **Run Genetic Supervisor** and an event has occurred which makes the Genetic Supervisor state inconsistent with the neural network model. Such events include retrieving an optimized neural network configuration from a prior Genetic Supervisor run, as well as the following commands:
>
> **Set Rows**,
> **Add Input Column**,
> **Add Target Column**,
> **Add Output Column**,
> **Set Mode Flag Column**, and
> **Set Network Size**.

`Hyperbolic activation function can only be used with Floating Point calculation mode.`

> This message appears after the **Set Enhanced Parameters** dialog box if the Hyperbolic activation function and Fixed Point calculation method were chosen at the same time.

`Illegal value. All values must be > 0.`

> This message appears after the second **Set Network Size** dialog box if one or more of the layers has been given a number of neurons which is negative or zero.

`Illegal value. Crossovers must be between 0 and 10 and Mutation Rate must be between 0 and 1.`

> This message appears after the **Set Genetic Parameters** dialog box if you have an illegal value for the Crossovers or Mutation Rate fields.

`Illegal value. Gain and Threshold values must be >= 0.`

> This message appears after the **Set Enhanced Parameters** dialog box if you have an illegal value for the function gain or zero threshold fields.

`Illegal value. Input Inclusion Rate must be > 0 and <= 1.`

This message appears after the **Set Genetic Parameters** dialog box if you have an illegal value for the input column Inclusion Rate field.

`Illegal value. Learning Rate and Momentum must be between 0 and 1 and Input Noise must be between 0 and 0.1.`

This message appears after the **Set Genetic Parameters** dialog box if you have an illegal value for Learning Rate, Momentum or Input Noise fields.

`Illegal value. Mode Flag type not recognized.`

This message appears if an invalid entry was made in a Mode Flag field.

`Illegal value. Network structure must be between 2 and 6 layers.`

This message appears after the **Set Genetic Parameters** dialog box if you have an illegal value for the network Max Layers field.

`Illegal value. Number of hidden layer neurons must be > 0 for the layers specified and = 0 for the layers above the number of max layers specified. Remember, hidden layers plus 2 (input and output layers) equals max layers.`

This message appears after the **Set Genetic Parameters** dialog box if you have an illegal value for a Layer n Neuron Limit field.

`Illegal value. Number of rows per pattern must be at least 1 and less than the total number of rows.`

This message appears after the **Set Rows** dialog box if you have an illegal value for the number of rows per pattern.

`Illegal value. % (percent) must be between 0 and 100.`

This message appears after the **Select Data Mode** dialog box if you have an illegal value for the % field.

`Illegal value. Population Pool Size must be > 0 and < 380.`

This message appears after the **Set Genetic Parameters** dialog box if you have an illegal value for the Pool Size field.

`Illegal value. Rows to shift per pattern must be at least 1.`

This message appears after the **Set Rows** dialog box if you have an illegal value for the number of rows to shift.

`Illegal value. The first five values must be in the range from 0 to 1; Epochs per Update must be > 0; and Epoch Limit, Time Limit, and Error Limit must be >= 0.`

This message appears after the dialog box of the **Set Network Parameters** command, if one or more entries is out of the legal range.

`Inconsistent input column count in Genetic Supervisor data.`

This message should never appear; it represents an internal problem in the Neuralyst software. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area.

```
Incorrect index
Incorrect type
Incorrect column size
Incorrect column width
Incorrect weight array size
Incorrect network size
```

These messages should never appear; they represent an internal problem in the Neuralyst software. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area.

`Initial limit must range from 0 to 16.`

This message appears after the **Reset Weights** dialog box if the value for the initial weight limit is out of range.

`Initialize or Reload the worksheet.`

You have given a Neuralyst command before any worksheet has been specified with the **Init Working Area** or **Reload Network** commands. Use one of those commands and then try again.

`Input column` <column name> `is bad.`

This message indicates that Neuralyst has found bad data in the specified column. Check that column and check the rows that have been set (look in the Working Area to see the row limits). Make sure there is only numeric or valid symbolic data in that area.

`Input column list is invalid. Please re-enter.`

This message appears after the **Edit Column Lists** dialog box if Neuralyst wasn't able to interpret the Input column list entry. A legal entry looks like a series of one or two letter column names separated by spaces and/or commas.

`Insufficient memory (`N`).`

N is a number in the range 1 through 11 which is used internally by Neuralyst development teams for debugging. This message means that your system doesn't have enough memory to handle the network and data sizes that you have chosen.

`Insufficient memory in Genetic Supervisor (`N`).`

N is a number in the range 1 through 12 which is used internally by Neuralyst development teams for debugging. This message means that your system doesn't have enough memory to handle the data structures needed to manage the Genetic Supervisor.

`Invalid Column Type`

This message should never appear; it represents an internal problem in Neuralyst and indicates that a column was incorrectly specified.

`Invalid column list, please re-enter.`

This message may appear after the **Edit Column Lists** dialog box if Neuralyst wasn't able to interpret one of the column list entries. A legal entry looks like a series of one or two letter column names separated by spaces and/or commas.

`Library file Neuralyst Lib is bad.` *Macintosh only.*

This message should never appear. It indicates that the library file Neuralyst Lib has been corrupted in some way. Try re-installing the software from the distribution diskette.

`Macro error at cell: NEURLYST.XLM!`<Cell ID> *Windows only.*

This message should never appear; it represents an internal problem in the Neuralyst software. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area. Press the button labeled "Halt" to remove this message.

`Macro error at cell: Neuralyst!`<Cell ID> *Macintosh only.*

This message should never appear; it represents an internal problem in the Neuralyst software. Other causes could include accidental changes made to the data Neuralyst stores in the Working Area. Press the button labeled "Halt" to remove this message.

`Maximum number of rows is 6550.`

Some currently supported versions of Excel allow arrays of only 6550 elements. This limits Neuralyst to allowing only 6550 rows in the **Set Rows** command. Future versions of Excel and/or Neuralyst may overcome this restriction.

`Min or Max symbols must be in the Symbol List.`

This message indicates that a Min and/or Max symbol was defined which was not also defined in the Symbol List.

`Min cannot be >= Max.`

This message indicates that a Min value was entered which was greater than the entered Max value.

`Min symbol must precede Max symbol and Min and Max symbols must be different.`

This message indicates that a Min symbol was entered which has higher or equal precedence to the Max symbol as defined by order of entry in the Symbol List.

`Missing Symbol`

This message indicates that an incorrectly structured symbol list was passed to symbol initialization.

`Mode Flag column field is invalid. Please re-enter.`

This message appears after the **Edit Column Lists** dialog box if Neuralyst wasn't able to interpret the Mode Flag column entry. A legal entry is a one or two letter column name.

`Mode Flag column is bad.`

This message indicates that Neuralyst has found bad data in the Mode Flag column. Check that column and check the rows that have been set (look in the Working Area to see the row limits). Make sure there are only blanks or TRAIN, TEST, MIN, MAX, or SYMBOL values in that area.

`Mode is set to optimize epoch count. Fitness limit represents RMS error and should be between 0 and 1.`

This message appears after the **Set Genetic Parameters** dialog box if the range of the Fitness Limit field does not match the Fitness Criteria selected.

`Mode is set to optimize RMS error. Fitness limit represents epochs and should be an integer >= 1.`

This message appears after the **Set Genetic Parameters** dialog box if the range of the Fitness Limit field does not match the Fitness Criteria selected.

`Network must be completely set up before Genetic Supervisor data can be initialized.`

This message appears after the **Set Genetic Parameters** dialog box if a neural network model has not been set up in all normal respects first.

`No data is available to create a Training Error Plot.`

No training error history is available for plotting. This can occur if: a worksheet has just been initialized with an **Init Working Area** command and no training has occurred yet; a **Set Network Size** or **Reset Weights** command was issued and no training has occurred yet; or if a worksheet has just been reloaded with a **Reload Network** command (there is no training error history saved between reloads).

```
Non-blank value in a "blank" row for input column <name>
Non-blank value in a "blank" row for target column <name>
Non-blank value in a "blank" row for Mode Flag column <name>
```

Neuralyst skips blank rows in your data area. The determination of whether a row is blank or not is done based on values in the first input column. This message indicates that it found a partially blank row, one where the first input column was blank but some other column was not. The message tells you which column had the non-blank value.

`Not a single column array.`

This message should never appear; it represents an internal problem in Neuralyst and indicates that a column was incorrectly specified.

`Not enough room to save weights.  Try reducing the size of your network or relocating the Working Area higher on the worksheet.`

This message appears after a **Set Network Size** command if the space from the Working Area to the bottom of the worksheet has insufficient room to save all the weights.

`Number of layers must be 2-6.`

This message appears after the first **Set Network Size** dialog box if the number of layers entered is outside of the range supported by Neuralyst. Two layers is the minimum and six is the maximum currently supported.

`Number of target columns must equal number of output columns.`

This message appears after the dialog box of the **Edit Column Lists** command, if the number of columns in the Target column list is different from the number in the Output column list.

`Number of weights exceeds Neuralyst limit of 131,008. Try reducing the size of your network, especially the size of the largest layer.`

This warning appears if you are changing the network size using the **Set Network Size** command. The total number of weights in the network is limited to 131,008 in this version of Neuralyst. The number of weights in a network is calculated by taking each layer except the last, and multiplying one plus the number of neurons in that layer times the number of neurons in the next layer, then adding those products. For example, a 4 layer network with sizes of 15, 20, 10, 5 would have 16*20 + 21*10 + 11*5 or 585 weights.

`Only one column may be selected.`

This message appears after the **Edit Mode Lists** command if more than one column was entered as a selection.

`Only one row may be selected.`

This message appears after the **Set Mode Row** command if more than one row was entered as a selection.

`Output column too large.`

This message should never appear; it represents an internal problem in Neuralyst and indicates that a column was incorrectly specified.

`Output column list is invalid. Please re-enter.`

This message appears after the **Edit Column Lists** dialog box if Neuralyst wasn't able to interpret the Output column list entry. A legal entry looks like a series of one or two letter column names separated by spaces and/or commas.

`Randomization setting must be greater than zero.`

This message occurs after the **Reset Weights** dialog box if User Set randomization has been selected, and the User Set value is invalid.

`Reload macro sheet NEURLYST.XLM.` *Windows only.*

This message indicates that something went wrong when Neuralyst originally loaded. Try exiting Excel and re-starting Neuralyst.

`Reload macro sheet Neuralyst.` *Macintosh only.*

This message indicates that something went wrong when Neuralyst originally loaded. Try exiting Excel and re-starting Neuralyst.

`Select worksheet <your worksheet name> first.`

The worksheet which is currently selected is not the one you have specified for Neuralyst. Bring that worksheet to the front, or else use the **Reload Network** command on the front worksheet if you want Neuralyst to change over to using that one.

`Selected column has not been defined as an Input or Target column.`

This message indicates that the column selected for Symbol and/or Min/Max entry with **Edit Mode Lists** is not currently defined as an Input or Target column.

`Selected row not in current row range. Include this row with Set Rows later.`

> This message indicates that the row defined by a **Set Mode Rows** command is not currently in the range of rows defined by the **Set Rows** command. It will not be used until a **Set Rows** is done with the row included in the row range.

`Selection must be at least two rows high.`

> This message appears if you have selected just one row for the **Set Rows** command. Neuralyst requires at least two rows to be specified.

`Set the network size. That will randomize the weights, too.`

> This message appears if you give the **Reset Weights** command at a time when the network size still needs to be specified. What you should do is to give the **Set Network Size** command, which will also randomize the weights.

`Symbol is neither blank nor a string.`

> This message appears if non-blank or non-string data is present in the SYMBOL row.

`Symbol List must have at least two symbols, separated by a comma.`

> This message appears if a Symbol List was entered that has less than two symbols.

`Set up all rows and columns first.`

> This message appears if you give the **Select Data Mode** or **Set Network Size** commands before you have finished setting up the rows and columns. Either the number of rows is zero, or the number of Input or Output columns is zero, or the number of Target columns is not equal to the number of Output columns, or the Mode Flag column has not been set. Fix things with the **Set Rows** and/or the column commands, and try again.

```
Target column <column name> is bad.
```

This message indicates that Neuralyst has found bad data in the specified column. Check that column and check the rows that have been set (look in the Working Area to see the row limits). Make sure there is only numeric data in that area.

```
Target column list is invalid. Please re-enter.
```

This message appears after the **Edit Column Lists** dialog box if Neuralyst wasn't able to interpret the Target column list entry. A legal entry looks like a series of one or two letter column names separated by spaces and/or commas.

```
The network size must be set before training can begin
The network size must be set before running
The network size must be set before histogramming
The network size must be set before unpacking
```

This message appears if you give the **Train Network**, **Run/Predict with Network**, **Histogram Weights**, or **Unpack Weights** commands at a time when the network size still needs to be set. Use the **Set Network Size** command and then try again.

```
This mode row was previously defined. Change to new row?
```

This warning appears if **Set Mode Rows** is used to set a row type that was already set. It allows you to confirm your intentions.

```
This will cause any genetic training state to be lost! Proceed?
```

This warning appears if you are initializing the supervisor state with the **Set Genetic Parameter** command. It allows you to confirm your intentions.

```
This will cause any learning done so far to be forgotten.
Proceed?
```

This warning appears when you give the **Reset Weights** command. It allows you to confirm your intentions.

```
This will clear a large area to the right and down from the cell
you have selected. Are you sure you have chosen an area that it's
OK to clear?
```

This warning appears whenever you give the **Init Working Area** command. It allows you to confirm your intentions.

```
This will overwrite your current network structure and other
parameters. Proceed?
```

This warning appears if you confirm retrieval of an optimized neural network configuration after a **Run Genetic Supervisor** command. It allows you to confirm your intentions.

```
This will randomize the network weights, causing any learning
done so far to be forgotten. Proceed?
```

This warning appears if you are changing the network size using the **Set Network Size** command. It allows you to confirm your intentions.

```
This will require you to change the network size, forgetting any
learning done so far. Proceed with the change? [Proceed with the
column addition?, Proceed with the column changes?]
```

This warning appears when you are about to make a change which will require the network size to change. You are being warned that proceeding with that change will cause any learning done so far to be forgotten. Changes which will lead to this message are: changing the number of rows per pattern in the **Set Rows** command, or changing the set of Input, Output, or Target columns with the column commands.

`This worksheet is set up for an old version of Neuralyst. Should it be updated for the new version?`

This message occurs when you give the **Reload Network** command for a worksheet which was set up under a previous version of Neuralyst. In order for your worksheet to be used with the new version, Neuralyst will automatically update the worksheet, preserving your data. Confirm with OK if you want this to be done; use Cancel if you don't want the worksheet changed (which will mean that it can't be used with this version of Neuralyst).

`Total weights exceeded 131,008 limit.`

This warning appears if you are changing the network size using the **Set Network Size** command. The total number of weights in the network is limited to 131,008 in this version of Neuralyst. The number of weights in a network is calculated by taking each layer except the last, and multiplying one plus the number of neurons in that layer times the number of neurons in the next layer, then adding those products. For example, a 4 layer network with sizes of 15, 20, 10, 5 would have 16*20 + 21*10 + 11*5 or 585 weights.

`Unable to find macro sheet NEURXTRA.XLM [TML.XLM]. Please make sure it is installed in the Neuralyst directory and try again.` *Windows only.*

Neuralyst keeps some of its functions in separate Excel macro sheets which should be in the same directory as `NEURLYST.XLM`. This message indicates that it was not able to find one of these macro sheets there. Check to see that the file named in the message was properly installed into the Neuralyst directory.

`Unable to find macro sheet Neuralyst_Inits [TML]. Please make sure it is installed in the Neuralyst folder and try again.` *Macintosh only.*

> Neuralyst keeps some of its functions in separate Excel macro sheets which should be in the same folder as Neuralyst. This message indicates that it was not able to find one of these macro sheets there. Check to see that the file named in the message was properly installed into the Neuralyst folder.

`Unable to load library file NEUR14.DLL. You must either copy that file into a directory on your search path, or start Excel by using the Neuralyst icon in Program Manager.` *Windows only.*

> Neuralyst relies on the external library file `NEUR14.DLL`. This message appears if it was unable to locate that file. Make sure you start Neuralyst from a directory which includes both the `NEURLYST.XLM` macro sheet and the `NEUR14.DLL` file, which can be done by using the Neuralyst icon in the Program Manager. Alternatively, you can put `NEUR14.DLL` into your Windows home directory or any directory on your search path, which will guarantee that Neuralyst will be able to find it.

`Unable to load library file Neuralyst Lib. Be sure it is in the Neuralyst folder.` *Macintosh only.*

> Neuralyst relies on the external library file Neuralyst Lib. This message appears if it was unable to locate that file. Make sure you start Neuralyst from a folder which includes both the Neuralyst macro sheet and the Neuralyst Lib file.

```
Value greater than Max in input column
Value greater than Max in target column
Value less than Min in input column
Value less than Min in target column
```

> This message indicates that a value in the designated column exceeded the set Min or Max values, modified by Scaling Margin, for that column.

```
Value out of range in input column
Value of of range in target column
```

This message should never appear; it represents an internal error in Neuralyst and indicates that numeric representation of an input, target or output exceeds the range of defined symbols.

```
When optimizing epoch count, it is generally necessary to set a
Time Limit in Set Network Parameters to prevent untrainable
networks from tying up testing. Proceed?
```

This message appears if you give the **Run Genetic Supervisor** command with no Time Limit set and Fitness Criteria set to Train Epochs or Test Epochs. It allows you to confirm your intentions.

`Worksheet` <your worksheet name> `must be initialized.`

This message appears if you give the **Reload Network** command for a worksheet which has never been prepared for Neuralyst with the **Init Working Area** command. Use **Init Working Area** to set up the worksheet for Neuralyst.

```
You must first name this worksheet (by saving it).
```

This message appears if you give the **Init Working Area** command for a worksheet which has never been saved (such as SHEET1). Neuralyst can't work on such a worksheet. Save the worksheet and give it a name and then you will be able to use the **Init Working Area** command.

```
You must make a continuous selection before using this function.
```

This message appears for the **Set Rows** command, or any of the **Add Columns** commands, if you have selected a region that is not continuous before giving the command. Neuralyst can only work with continuous regions for these commands.

Appendix C
Cheshire Customer Service

If you have problems installing or running Neuralyst, please review the procedures and error messages discussed in this guide, particularly Appendices A and B.

If you are still not able to get Neuralyst to work, please note down all information relating to the failure and have it ready at hand. Then call Cheshire Customer Service at (626)351-5493 (or FAX (626)351-8645). Cheshire Customer Service is available Monday to Friday between the hours of 0900 to 1700 Pacific Time.

Due to the highly involved and widely varying nature of individual neural network models, Cheshire Customer Service can only answer questions on general usage, incorrect operation and Neuralyst generated faults and errors. Cheshire Customer Service cannot answer specific questions on developing neural network models nor will it be able to debug neural network models that you may have developed.

If you need help in developing or debugging neural network models then please write Cheshire at:

Cheshire Engineering Corporation
650 Sierra Madre Villa Avenue, Suite 201
Pasadena, CA 91107

Cheshire can provide full tutorial and design services for Neuralyst and development of custom neural network designs. These services are provided independently of its Customer Service operations for Neuralyst.

Appendix D
Neuralyst Specifications

D.1 Windows Specifications

Windows Version

3.1 or later.

Excel Version

4.0 or later.

CPU or Memory Requirements

Any system able to run Windows with Excel. More memory will allow larger neural networks to run.

Maximum Network Size

Greater than 131,000 weights.

Maximum Number of Neural Network Layers

Six.

Maximum Number of Neurons per Layer

Limited by Maximum Network Size.

Maximum Number of Input Columns

Limited only by memory and worksheet size.

Maximum Number of Output Columns

Limited only by memory and worksheet size.

Maximum Number of Rows

6550 per Input or Output Column.

Maximum Number of Rows per Pattern Window

Limited by Maximum Number of Rows.

Neural Network Computation Rate

Sample performance (in *Connections per Second*):

	386DX-40 (est.)	486DX-33	486DX2-66	Pentium-60
Fixed/Small Training	71,000	149,000	262,000	440,000
Fixed/Small Testing	221,000	465,000	836,000	1,363,000
Fixed/Huge Training	63,000	133,000	234,000	370,000
Fixed/Huge Testing	201,000	423,000	749,000	1,147,000
Float/Small Training	22,000	48,000	92,000	166,000
Float/Small Testing	69,000	145,000	269,000	489,000
Float/Huge Training	22,000	47,000	87,000	153,000
Float/Huge Testing	66,000	142,000	265,000	462,000

Connections per second is equivalent to a single weight/input computation for a neuron. Computation rates listed are examples only. Other computers will be slower or faster in proportion to their clock rates, processor type and other system characteristics.

Small applies to fixed-point networks having 10,920 or fewer connections (weights) and to floating-point networks having 4096 or fewer weights; larger networks are *huge*.

D.2 Macintosh Specifications

Maximum Network Size

Greater than 131,000 weights.

Maximum Number of Neural Network Layers

Six.

Maximum Number of Neurons per Layer

Limited by Maximum Network Size.

Maximum Number of Input Columns

Limited only by memory and worksheet size.

Maximum Number of Output Columns

Limited only by memory and worksheet size.

Maximum Number of Rows

6550 per Input or Output Column.

Maximum Number of Rows per Pattern Window

Limited by Maximum Number of Rows.

Excel Version

4.0 or later.

CPU or Memory Requirements

Any system able to run Excel. More memory will allow larger neural networks to run.

Neural Network Computation Rate

Sample performance (in *Connections per Second*):

	68030-33 (est.)	68040-25 (est.)	PowerPC-60 Emulated (est.)	PowerPC-60 Native (est.)
Fixed-point Training	82,000	209,000	104,000	682,000
Fixed-point Testing	256,000	650,000	325,000	2,112,000
Floating-point Training	31,000	79,000	39,000	257,000
Floating-point Testing	91,000	232,000	116,000	757,000

Connections per second is equivalent to a single weight/input computation for a neuron. Computation rates listed are examples only. Other computers will be slower or faster in proportion to their clock rates, processor type, and other system characteristics.

Appendix E
Macro Interface Specifications

E.1 Neuralyst/Excel Macro Interface

Neuralyst can be controlled by Excel macros on Windows or Macintosh. The list of accessible Neuralyst menu items and corresponding macro names are:

Menu Item	Macro Name
Reload Network	`MacroReloadNetwork`
Train Network	`MacroTrainNetwork`
Run/Predict with Network	`MacroRunNetwork`
Plot Training Error	`MacroPlotError`
Histogram Weights	`MacroHistChart`
Unpack Weights	`MacroUnpackWeights`

Note: When using these macro commands, prepend the macro name with “`NEURLYST.XLM!`” on Windows and “`Neuralyst!`” on Macintosh.

Other Neuralyst menu items do not have a correspondence through the macro interface since they use dialog boxes to interface and capture settings and parameters and there is no way to set dialog boxes from macros. However, many menu items can be simulated by writing directly to the Working Area and then using the `MacroReloadNetwork` command to get Neuralyst to accept the new settings.

For example, to change the row limits, write directly to the First Row and or Last Row cells in the Working Area. Let's say the First Row cell is in "R7C16" and you wish to set the value to 12, then use:

```
FORMULA(12, "R7C16")
```

Then give the `MacroReloadNetwork` command. (A more convenient method to reference Working Area cells is described in the next section.)

Most other parameters can be changed by this method. A useful command that is not accessible in this way is the **Set Network Size** command. **Set Network Size** opens a dialog box, fills in values, and then executes a Neuralyst DLL function to generate values to initialize the Working Area. There is no way to force this to happen from the macro interface. Fortunately, this command does not have to be given frequently. It is also not possible to call the Genetic Supervisor from the macro interface.

E.2 Referencing the Working Area

It is not convenient to count cells on each worksheet and generate "RC" references for the Neuralyst Working Area when setting parameters. Fortunately, Neuralyst provides a reference mechanism for dealing with this. The top-left corner cell of the Neuralyst Working Area is referenced by the name _NWTL as follows:

```
SET.NAME("NWTL",TEXTREF("NETSHEET.XLS!_NWTL"))
```

Where `NETSHEET.XLS` is replaced by the name of your worksheet. This sets the local name, NWTL, on your macro sheet to point to the Working Area of your worksheet.

The other cells can be referenced from this location. Here is a list of offsets:

Parameter	Offset
RMS Error	`OFFSET(NWTL,4,0)`
Number of Data Items	`OFFSET(NWTL,5,0)`
Number Right	`OFFSET(NWTL,6,0)`

Parameter	Offset
Number Wrong	OFFSET(NWTL,7,0)
Training Epochs	OFFSET(NWTL,10,0)
Learning Rate	OFFSET(NWTL,13,0)
Momentum	OFFSET(NWTL,14,0)
Input Noise	OFFSET(NWTL,15,0)
Training Tolerance	OFFSET(NWTL,16,0)
Testing Tolerance	OFFSET(NWTL,17,0)
Epochs per Update	OFFSET(NWTL,18,0)
Epoch Limit	OFFSET(NWTL,19,0)
Time Limit	OFFSET(NWTL,20,0)
Error Limit	OFFSET(NWTL,21,0)
First Row	OFFSET(NWTL,5,4)
Last Row	OFFSET(NWTL,6,4)
Number, Of Rows	OFFSET(NWTL,7,4)
Rows/Pattern	OFFSET(NWTL,8,4)
Row, Offset	OFFSET(NWTL,9,4)
Function	OFFSET(NWTL,13,4)
Function Gain	OFFSET(NWTL,14,4)
Force Zero	OFFSET(NWTL,15,4)
FZ Threshold	OFFSET(NWTL,16,4)
Adaptive LR	OFFSET(NWTL,17,4)
Calculation Method	OFFSET(NWTL,18,4)
Scaling Margin	OFFSET(NWTL,19,4)
#Input Columns	OFFSET(NWTL,6,7)
Input Columns (list)	OFFSET(NWTL,8,7)
#Target Columns	OFFSET(NWTL,6,9)
Target Columns (list)	OFFSET(NWTL,8,9)
#Output Columns	OFFSET(NWTL,6,11)
Output Columns (list)	OFFSET(NWTL,8,11)
Mode Flag Column	OFFSET(NWTL,6,13)
Min Scale Row	OFFSET(NWTL,8,13)
Max Scale Row	OFFSET(NWTL,10,13)
Symbol Row	OFFSET(NWTL,12,13)
Generation Count	OFFSET(NWTL,24,0)
Structure Count	OFFSET(NWTL,25,0)
Least RMS Error	OFFSET(NWTL,26,0)
Least Epochs	OFFSET(NWTL,27,0)
Inclusion Rate	OFFSET(NWTL,24,4)
Max Layers	OFFSET(NWTL,25,4)

Parameter	Offset
L2 Neuron Limit	OFFSET(NWTL,26,4)
L3 Neuron Limit	OFFSET(NWTL,27,4)
L4 Neuron Limit	OFFSET(NWTL,28,4)
L5 Neuron Limit	OFFSET(NWTL,29,4)
Min Learning Rate	OFFSET(NWTL,30,4)
Max Momentum	OFFSET(NWTL,31,4)
Max Input Noise	OFFSET(NWTL,32,4)
Population Size	OFFSET(NWTL,33,4)
Population Mode	OFFSET(NWTL,34,4)
Crossovers	OFFSET(NWTL,35,4)
Mutation Rate	OFFSET(NWTL,36,4)
Fitness Criteria	OFFSET(NWTL,37,4)
Fitness Limit	OFFSET(NWTL,38,4)
Genetic State Information	OFFSET(NWTL,24,13)

By using `OFFSET(NWTL,R,C)` in a macro function, a reference will be returned pointing to the desired cell.

E.3 Windows DDE Interface

Neuralyst can also be controlled through external programs via DDE (Dynamic Data Exchange) on Windows. The technique is similar to using DDE to control basic Excel functions. In each case, the DDE Execute function is used. The format of a string sent using an Execute function is always:

[<Command Name>(<Arguments>)]

<Command Name> should be the name of a command to run and <Arguments> should be the arguments, separated by commas, to the command. If there are no arguments, you still need a pair of parentheses after the command name. For example:

```
[BEEP()]

[COLUMN.WIDTH(15)]
```

Neuralyst commands may be executed by using Excel's RUN function. The format is like this:

```
    [RUN("NEURLYST.XLM!<Neuralyst Macro Name>")]
```

<Neuralyst Macro Name> should be replaced by one of the documented commands listed in Section E.1. As with Excel macros, not all Neuralyst commands are accessible through this interface since Neuralyst uses many dialog boxes to interface with the user. However, most functions can be simulated by writing directly to the Working Area and then using MacroReloadNetwork to get Neuralyst to accept the new values. The technique can be used with DDE just as was described above for an Excel macro.

Note that some Neuralyst commands can take a long time, particularly the MacroTrainNetwork command. Excel will not return the DDE "Acknowledge" message until the command has finished. Code which expects to receive the Acknowledge within a short time will therefore not work. However, waiting for the Acknowledge is safer since it guarantees that the command has finished.

E.4 Macintosh Apple Events Interface

It is also possible to control Neuralyst through Excel 4.0's Apple Events Interface on the Macintosh with System 7.0. For details, reference the *Microsoft Excel 4.0 Software Development Kit*.

Appendix F
Reading List & Bibliography

Anderson, J.A. & Rosenfeld, E., (eds); **Neurocomputing: Foundations of Research**; MIT Press; Cambridge, MA; 1988.

Anderson, J.A., Pellionisz, A. & Rosenfeld, E., (eds); **Neurocomputing 2: Directions for Research**; MIT Press; Cambridge, MA; 1990.

Arbib, M.A.; **Brains, Machines, and Mathematics (2nd Ed)**; Springer-Verlag; New York, NY; 1987.

Buckles, B. P. & Petry, F. E.; **Genetic Algorithms**; IEEE Computer Society Press, Los Alamitos, CA; 1992.

Butler, C. & Caudill, M.; **Naturally Intelligent Systems**; MIT Press; Cambridge, MA; 1990.

Caudill, Maureen; **Neural Network Primer**; Miller Freeman Publications; San Francisco, CA; 1990.

Churchland, P.S. & Sejnowski, T.J.; **The Computational Brain**; MIT Press; Cambridge, MA; 1992.

Dayhoff, Judith; **Neural Network Architectures**; Van Nostrand Reinhold; New York, NY; 1990.

Deboeck, Guido (Ed); **Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets**; John Wiley & Sons; New York, NY; 1994.

Defense Advanced Research Projects Agency (DARPA); **DARPA Neural Network Study**; AFCEA International Press; Fairfax, VA; 1988.

Goldberg, David E.; **Genetic Algorithms in Search, Optimization and Machine Learning**; Addison-Wesley; Reading, MA; 1989.

Grossberg, Stephen (Ed); **Neural Networks and Natural Intelligence**; MIT Press; Cambridge, MA; 1988.

Hecht-Nielson, Robert; **Neurocomputing**; Addison-Wesley Publishing Co.; Reading, MA; 1990.

Holland, John H.; **Adaptation in Natural and Artificial Systems**; MIT Press; Cambridge, MA; 1992.

Kosko, Bart; **Neural Networks and Fuzzy Systems**; Prentice-Hall; Englewood Cliffs, NJ; 1992.

Koza, John R.; Genetic Programming: **On the Programming of Computers by Means of Natural Selection**; MIT Press; Cambridge, MA; 1992.

Lau, C. & Widrow, B.; **Special Issue on Neural Networks, I & II**; Proceedings of the IEEE; Vol.78, No. 9 & 10; September & October 1990.

Maren, A., Harston, C., & Pap, R.; **Handbook of Neurocomputing Applications**; Academic Press; San Diego, CA; 1990.

Mehra, P. & Wah, B.W.; **Artificial Neural Networks: Concepts and Theory**; IEEE Computer Society Press; Los Alamitos, CA; 1992.

Minsky, M. & Papert, S.; **Perceptrons: An Introduction to Computational Geometry**; MIT Press; Cambridge, MA; 1969.

McClelland, J.L. & Rumelhart, D.E. (Eds); **Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I & II**; MIT Press; Cambridge, MA; 1986.

McClelland, J.L. & Rumelhart, D.E.; **Explorations in Parallel Distributed Processing: A Handbook of Models, Programs and Exercises**; MIT Press; Cambridge, MA; 1988.

Pao, Yoh-Han; **Adaptive Pattern Recognition and Neural Networks**; Addison-Wesley Publishing Company; Reading, MA; 1989.

Trippi & Turban (Eds); **Neural Networks in Finance and Investing**; Probus Publishing; Chicago, IL; 1993.

Vemeri, V (ed); **Artificial Neural Networks: Theoretical Concepts**; Computer Society Press of the IEEE; Washington, DC; 1988.

AI Expert; Miller Freeman Publications; San Francisco, CA.

(An easy to read magazine on artificial intelligence technology for small computers, with single and continuing article series on neural networks starting from late 1987 to the present.)

IEEE Transactions on Neural Networks; Institute of Electrical and Electronic Engineers, Inc.; New York, NY.

(A technical journal devoted exclusively to current research on neural networks.)

Neural Networks; Pergamon Press; Elmsford, NY.

(The Official Journal of the International Neural Network Society — a technical journal devoted exclusively to current research on neural networks.)

PC-AI; Publications; Phoenix, AZ.

(Another easy to read magazine on artificial intelligence technology for small computers.)

Appendix G
Trader’s Macro Library

G.1 Support for Technical Analysis

One of the more popular applications for Neuralyst is investment analysis; in particular, the kind of investment analysis known as technical analysis as demonstrated in the example `DJIA.XLS` {DJIA}. Cheshire has included a Trader’s Macro Library (TML) specifically designed to help you set up your Neuralyst worksheet to integrate neural network analysis and technical analysis.

[There is a vast literature on technical analysis and we will not attempt to duplicate or summarize that information here; see Section G.6 for some introductory references.]

The goal of technical analysis is to predict the future price movement of equities, futures, or other investment instruments based solely on historical price and volume data. TML recognizes and deals specifically with data designated as one of the following types: Open, High, Low, Close, Volume, and Open Interest. The first four are price-related and the last two are volume-related.

You must obtain this data for the kind of equity, future, or investment that you wish to analyze. This data may be obtained from the business section of your daily paper, computer news services (such as CompuServe or Dow Jones News Retrieval), or specialized investment data services.

Once this data is obtained, it must be entered into an Excel worksheet, organized so that the data are headed by columns labeled by one or more of the names just listed, and with the respective data for each new day appended in succeeding rows. TML can then be used to add and format additional columns that contain technical analysis indicators selected by you and derived from the basic data.

G.2 Enabling and Disabling the Library

G.2.1 Enabling and Disabling in Windows

TML is an optional feature which can be enabled or disabled prior to the execution of Neuralyst and Excel. To use TML the first time, it must be enabled.

In order to enable TML, execute the command:

```
TMLON.EXE
```

from the Neuralyst directory while in DOS or Windows. The next time Neuralyst is loaded, there will be a new command in the **Neural** menu, **Trader's Macro Library**.

In order to disable TML, execute the command:

```
TMLOFF.EXE
```

from the Neuralyst directory while in DOS or Windows. The next time Neuralyst is loaded, the **Trader's Macro Library** command in the **Neural** menu will no longer be present.

Alternatively, renaming, moving, or deleting `TML.XLM` from the Neuralyst directory will disable TML. Restoring it will enable TML.

If you have re-installed Neuralyst or have installed an updated version of Neuralyst, you must give the `TMLON.EXE` command in order for the new version of TML to be enabled.

G.2.2 Enabling and Disabling in Macintosh

TML is an optional feature which can be enabled or disabled prior to the execution of Neuralyst and Excel. To use TML the first time, it must be enabled.

In order to enable TML change the name of the file called `NoTML` to `TML`. The next time Neuralyst is loaded, there will be a new command in the **Neural** menu, **Trader's Macro Library**.

In order to disable TML, change the name of the file called `TML` to `NoTML`. The next time Neuralyst is loaded, the **Trader's Macro Library** command in the **Neural** menu will no longer be present.

If you have re-installed Neuralyst or have installed an updated version of Neuralyst, you must be sure to remove your old `TML` file before you change the name of the new `NoTML` file to `TML`. This will ensure that Neuralyst uses the new version of TML.

G.3 Using the Library

All TML operations are accessed through the **Trader's Macro Library** command in the **Neural** menu.

G.3.1 Initializing the Library

If you use TML, the location of the Neuralyst Working Area is more restricted than usual. When you use TML, the Neuralyst Working Area must be to the right of your data, indicators, targets, outputs, and Mode Flag column. When TML is in operation, it will make use of some of the information kept in the Working Area and store some of its internal data there. Because of this, after you create a new worksheet that contains price data, you should perform an **Init Working Area** command prior to executing **Trader's Macro Library**.

Once the Working Area has been defined, you may add columns for indicators, targets, outputs, and Mode Flag column by using the **Edit Insert** operation from Excel if you need to make additional space. As with Neuralyst, if you insert columns after the column configuration operations have been executed, you will need to use **Edit Column Lists** to revise the column configuration.

TML requires that the data to be operated on and transformed into indicators must be in columns headed by one of the following labels: Open, High, Low, Close, Volume, or Open Interest (OI for Open

Interest also works). The values in such columns must be numeric and positive, but there is no restriction on the magnitude. Price data quoted in points ($^1/_{32}$, $^1/_{16}$, $^1/_8$, and so on) must be converted to decimal representations first.

One or more of these columns may be designated as inputs to TML by selecting the cells containing the labels and then executing **Trader's Macro Library**. The TML dialog box will appear providing a list of TML operations; each operation is preceded by a "pushbutton" indicator. Select the **Set TML Columns** command and confirm **OK**. This will cause TML to record the column or columns and associated names for future operations.

After the columns have been set, select the rows containing price data, excluding the label for the data, and then execute **Trader's Macro Library**. The TML dialog box will appear. Select the **Set TML Rows** command and confirm **OK**. This will cause TML to record the rows containing price data for future operations.

To summarize:

1. Start by creating a new worksheet.

2. Set aside columns to be used for price data and head them with the proper labels.

3. Enter the price data in the respective columns.

4. Set aside as many other columns as may be needed to store the desired number of indicators, targets, outputs, and so on.

5. Select a blank column to the right of this area and perform an **Init Working Area** command.

6. Select the cells containing the labels for the columns containing the price data, call the **Trader's Macro Library**, and perform a **Set TML Columns** command.

7. Select the rows containing the price data, call the **Trader's Macro Library**, and perform a **Set TML Rows** command.

The worksheet is now ready for technical indicator creation by TML.

G.3.2 Creating a Technical Indicator

There are fourteen technical "indicators" supported (strictly speaking Log, Delta, and Log-Delta are not indicators). These supported indicators do not constitute a complete toolbox of technical analysis indicators; however, they are some of the most popular and useful.

All TML indicators use one or more of the price data columns defined previously. Eight of the fourteen indicators supported may derive their inputs from any valid price data column. These indicators are: Log, Delta, Log-Delta, Moving Average, Exponential Average, Moving Average Oscillator, Momentum, and Rate-of-Change (ROC). Six of the indicators supported have fixed inputs (fixed subset of Open, High, Low, Close, and Volume) required by the definition of the indicator. These are: Moving Average Convergence-Divergence (MACD), On Balance Volume, Relative Strength Index (RSI), Stochastic Oscillator, Volatility, and Williams' Ultimate Oscillator.

Some indicators also allow the setting of one or more additional parameters. These are: Moving Average, Exponential Average, Moving Average Oscillator, Momentum, MACD, ROC, RSI, Stochastic Oscillator, Volatility, and Ultimate Oscillator. The parameters are described in the next section.

To create a technical indicator, select the column or columns that will contain the results of the technical indicator computation. Then execute **Trader's Macro Library**. A dialog box will appear showing the available TML operations. Select one of the items from the Indicators section. If the indicator allows an input type to be selected, then a dialog box listing the possible inputs will appear; select one of these. If parameters are required to complete the definition of the indicator, then the dialog box will contain fields allowing these parameters to be set; enter the desired parameter values. If both an input type may be selected and parameters may be set, then the dialog box will have fields allowing both operations to be performed. When the necessary entries have been made, confirm **OK** to complete the operation.

TML will then create the indicator and store both the formulas and the values in the designated column or columns.

G.3.3 Technical Indicators

These indicators are described below.

Log — calculates the logarithm of each value in the column selected and places the results in the designated column.

Delta — calculates the difference between the current value and the previous value in the column selected and places the results in the designated column. The first row is blank since there is no previous value.

Log-Delta — calculates the difference between the logarithms of the current value and the previous value in the column selected and places the results in the designated column. The first row is blank since there is no previous value.

Moving Average — calculates moving average of the current and preceding values for ***n*** rows in the column selected and places the results in the designated column. The first ***n-1*** rows are blank since the moving average is not defined for these rows.

Exponential Average — calculates the exponential average places the results in the designated column. The decay factor is entered as an integer so that, when the formula for the exponential average is created, it is calculated to be the corresponding fractional value which approximates the smoothing characteristics of the moving average value ***n***. The initial value of the exponential average is set to the initial value of the input data; this causes the first ***n-1*** rows to be weighted more than usual by that initial value. Because of this, you may wish to skip the first ***n-1*** rows when performing your analysis.

Moving Average Convergence-Divergence (MACD) — calculates the exponential average of the difference between a fast and slow moving average using the Close column and places the results in the three designated columns. The first column contains the fast exponential average, with period ***f***, of the Close (traditionally a period of 12 days). The second column contains the slow exponential average, with period ***s***, of the Close (traditionally a period of 26 days). The third column contains a smoothed (another fast exponential, traditionally 9 days), with period ***k***, difference of the first two columns. Since all three columns are generated using exponential averages, all rows contain values in each column, however, as with all exponential averages, the initial data is more heavily weighted by early values, so it may be desirable to skip a number of rows, corresponding to the sum of the slow exponential average period and the smoothing period, ***s* + *k***, when performing your analysis.

Moving Average Oscillator — calculates the difference between the moving average of ***m*** rows and ***n*** rows of the column selected and places the results in the designated column. The first ***m-1*** or ***n-1*** (whichever is greater) rows are blank since the Moving Average Oscillator is not defined for these rows.

Momentum — calculates the difference between the current value and the value ***n*** rows previous in the column selected and places the results in the designated column. The first ***n*** rows are blank since Momentum is not defined for these rows.

On Balance Volume (OBV) — calculates the accumulation or distribution of volume by adding the day's volume if it was an up day or subtracting the day's volume if it was a down day from a running total. OBV is calculated using the Close and Volume columns and the results are placed in the designated column. The first row is blank since there is no previous value.

Rate-of-Change (ROC) — calculates the ratio as a percentage of the current value and the value ***n*** rows previous in the column selected and places the results in the designated column. The first ***n*** rows are blank since the ROC is not defined for these rows.

Relative Strength Index (RSI) — calculates the RSI for period ***n*** using the High, Low and Close columns and places the results in the three designated columns. The RSI is an index computed from the ratio of the total points in up movement on closing prices divided by the total points of down movement on closing prices over the period ***n***. The values of the up closes are copied in the first column. The values of the down closes are copied in the second column. The third column contains the computed RSI. The first ***n-1*** rows of the third column are blank since the RSI is not defined for these rows.

Stochastic Oscillator — calculates the two Stochastic Oscillator values %K, of period ***m***, and %D, of period ***n***, using the High, Low and Close columns and places the results in the two designated columns. The %K Stochastic Oscillator is computed by taking the difference of the most recent close from the lowest low of the last ***m*** days divided by the difference between the highest high and the lowest low of the last ***m*** days. The %D Stochastic Oscillator is the ***n*** day moving average of %K. The first ***m-1*** rows of the %K column are blank since the %K Stochastic Oscillator is not defined for these rows. The first ***m+n-1*** rows of the %D column are blank since the %D Stochastic Oscillator is not defined for these rows.

Volatility — calculates the Volatility for period ***n*** using the High, Low and Close columns and places the results in the two designated columns. The Volatility is an index computed from the moving average of the "true range" over the period ***n***. The true range is defined as the greater of the difference between the High and Low of a day, the High of the day and the Close of the previous day, or the Low of the day and the Close of the previous day. The values of the true ranges are in the first column. The moving average of the true ranges are in the second column. The first row of the first column is blank since there is no previous day. The first ***n*** rows of the second column are blank since the Volatility is not defined for these rows.

Williams' Ultimate Oscillator — calculates a weighted three period average using the High, Low, and Close columns and places the results in the three designated columns. The Ultimate Oscillator is an index computed from the difference of the day's Close and the "true low", this difference is then divided by the "true range". The true low is defined as the lesser of the Low of the day or the Close of the previous day. The true range is defined as the greater of the difference between the High and Low of a day, the High of the day and the Close of the previous day, or the Low of the day and the Close of the previous day. Finally, a fast, an intermediate, and a slow period moving averages, with periods ***f***, ***i***, and ***s***, are calculated from this ratio and recombined with weighting inversely proportional to the three periods. The weighting of the fast moving average is ***s/f***, the weighting for the intermediate moving average is ***s/i***, and the weighting of the slow moving average is 1. The first column contains the difference between the Close and the true low. The second column contains the true range. The third column contains the sum of the weighted moving averages. The first row of the first two columns is blank since there is no previous day. The first ***s*** rows of the third column are blank since the slow moving average is not defined for those rows.

G.4 Updating Technical Indicators

Once a technical analysis application has been created and Neuralyst has been trained on the data, it will be necessary to update the price data on a regular, usually daily or weekly, basis.

One more command has been provided in TML to facilitate these operations. **Extend TML Rows** will copy the last data row, including price data, technical indicators, target formulas, output area, and Mode Flag value, into the row below it. The row information in Neuralyst and in TML will be updated to reflect this change. Once you have entered the new price data, the indicators and other formulas will be updated automatically. You can then perform a **Reload Network** to update Neuralyst and train, test, or run with the new data.

G.5 TML Error Messages

`Close and Volume columns must be defined for this function.`

This message appears for the **OBV** (On Balance Volume) indicator. This indicator always works with Close and Volume data. This message indicates that one or both of those columns are not yet defined. Use the **Set TML Columns** command to define the needed data columns.

```
Column "Open" has not been set. Please try again.
Column "High" has not been set. Please try again.
Column "Low" has not been set. Please try again.
Column "Close" has not been set. Please try again.
Column "Volume" has not been set. Please try again.
Column "Open Interest" has not been set. Please try again.
```

You have tried to apply an indicator to a column type which has not been defined with the **Set TML Columns** command. Try applying the indicator to a different column, or give the **Set TML Columns** command again for the column type you want.

```
Column heading "XXX" is not recognized, please try again.
```

This message appears for the **Set TML Columns** command, when one of the column headings you have selected is not in the list of headings that are recognized. Re-check your selected column headings to make sure they correspond to those allowed for this command.

```
High, Low, and Close columns must be defined for this function.
```

This message appears for the **Stochastic Oscillator**, **Volatility**, and **Williams' Ultimate Oscillator** indicators. These indicators needs to have High, Low, and Close data to work with. This message indicates that one or more of those columns are not yet defined. Use the **Set TML Columns** command to define the needed data columns.

```
Interval must be >= 1. Please try again.
```

This message appears for the **Momentum** and **ROC** (Rate-of-change) indicators. It indicates that the value for the Interval is invalid.

```
Multiple-row selection is not allowed here.
```

This message appears for the **Set TML Columns** command. The selection needs to be one row high and to include only the column headings. Make sure your selection is of this format, and try the command again.

```
Number of rows selected < longest period. Please try again.
```

This message appears for the **Williams' Ultimate Oscillator**. There are fewer rows selected than the longest period of the oscillator. No data can be generated under these conditions. Increase the row selection or decrease the longest period.

```
Only a single column may be selected here.
```

This message appears for several of the indicators. Most indicators produce only one column of output, and so only one column should be selected before using them.

```
Period [Periods] must be >= 1. Please try again.
```

Several indicators produce this message. It indicates that one or more values specified for the periods are invalid.

```
Periods must be increasing. Please try again.
```

The **Williams' Ultimate Oscillator** indicator produces this message. It indicates that the sequence of periods set does not have increasing values.

```
Set TML columns first.
Set TML rows first.
```

These messages appear if you try to use a TML indicator before you have done both the **Set TML Columns** and the **Set TML Rows** commands. Make sure both of those commands have been given before trying to use indicators. The TML information is saved in the Working Area, so future uses of the library will re-use any row and column information you have set previously.

```
The Close column must be defined for this function.
```

This message appears for the **MACD** (Moving Average Convergence-Divergence) and **RSI** (Relative Strength Indicator). They always works with Close data. This message indicates that the Close column has not yet been defined. Use the **Set TML Columns** command to define this data column.

`Three columns must be selected here.`

This message appears for the **MACD** (Moving Average Convergence-Divergence), **RSI** (Relative Strength Indicator), and **Williams' Ultimate Oscillator**. These indicators use three columns for output. Make sure your selection is three columns wide and then try again.

`Two columns must be selected here.`

This message appears for the **Stochastic Oscillator** and **Volatility**. These indicators use two columns for output. Make sure your selection is two columns wide and then try again.

`Unimplemented indicator`

This message should not occur and indicates an internal problem with the TML software.

`You must make a continuous selection before using this function.`

This message can appear for any TML command or indicator, if you have selected a region that is not continuous before giving the command. TML can only work with continuous regions for its commands.

G.6 Technical Analysis Literature

Colby, R.W. & Meyers, T.A.; **Encyclopedia of Technical Market Indicators**; Dow Jones-Irwin; Homewood, IL; 1988.

Kaufman, Perry J.; **New Commodity Trading Systems and Methods**; John Wiley & Sons; New York, NY; 1987.

LeBeau, C. & Lucas, D.W.; **Computer Analysis of the Futures Market**; Business One-Irwin; Homewood, IL; 1992.

Murphy, John J.; **Technical Analysis of the Futures Market**; New York Institute of Finance; New York, NY; 1986.

Pring, Martin J.; **Technical Analysis Explained (2nd Ed)**; McGraw-Hill; New York, NY; 1985.

NeuroVe$t Journal; Randall B. Caldwell; Haymarket, VA.

A bi-monthly magazine devoted to neural network techniques and financial applications.

Technical Analysis of Stocks and Commodities; Technical Analysis Inc, Seattle, WA.

A monthly magazine devoted to technical analysis techniques.

Index