Neuralyst™ Demonstration Guide

This is a demonstration version of Neuralyst™ for Microsoft® Excel™.
This document is available both as a Windows Help file for use on-line, and as a Windows Write document suitable for printing.
To order a copy of the full Neuralyst product, see the order form near the end of this document.
The use of this demonstration software is covered by a license agreement that can be found both in this document and in the file README.TXT which should accompany this demonstration. Please read and comply with the terms of this agreement.

Introduction

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.

Installing the Demonstration Version of Neuralyst

About this Document

This document 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 document 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.

Neuralyst Demonstration Distribution

The Neuralyst Demonstration distribution you have received should contain the following items, either collected on a single floppy or compressed into a single PKZIP archive:

1. On-line Neuralyst Demonstration Guide: `neurdemo.wri` and `neurdemo.hlp` {`Demo Guide`}
2. Neuralyst Demonstration Distribution files
3. Neuralyst products Order Form: `orderfrm.wri`

For your convenience, the Neuralyst demonstration distribution disk is not copy-protected. You may make and freely re-distribute additional copies of the disk or archive file, install the software and use it as described in this guide subject to the 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.

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.

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.

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.

Un-install

The Neuralyst demonstration may be un-installed by simply deleting all of the files in the `C:\NEURDEMO` directory {`Neuralyst Demo` folder}. On Windows, the program manager group and its associated icons may also be deleted. Installation of this demonstration version of Neuralyst makes no other changes to your Windows environment.

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 Windows installation procedure described in the next section. If you have a Macintosh follow the Macintosh installation procedure described later.

Windows Installation Procedure

Before installing the Neuralyst Demonstration, 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`.
If you have received a PKZIP archive file of the distribution, simply unzip the archive into a temporary directory or floppy, and substitute the name of that directory for `A:` in the `A:INSTALL.EXE` command.
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:\NEURDEMO`. If your hard drive is not `C`, then edit the entry to show the proper drive name. It is recommended that `\NEURDEMO` 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:\NEURDEMO` 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 for help with any problems you may encounter during installation.

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

Macintosh Installation Procedure

Before installing the Neuralyst Demonstration, 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 Demonstration` to show the disk contents, which will be a folder labeled `Neuralyst Demo`. Select this folder and drag it to your hard disk to copy the contents of the folder. The `Neuralyst Demo` folder may be copied to any area of your hard disk.

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

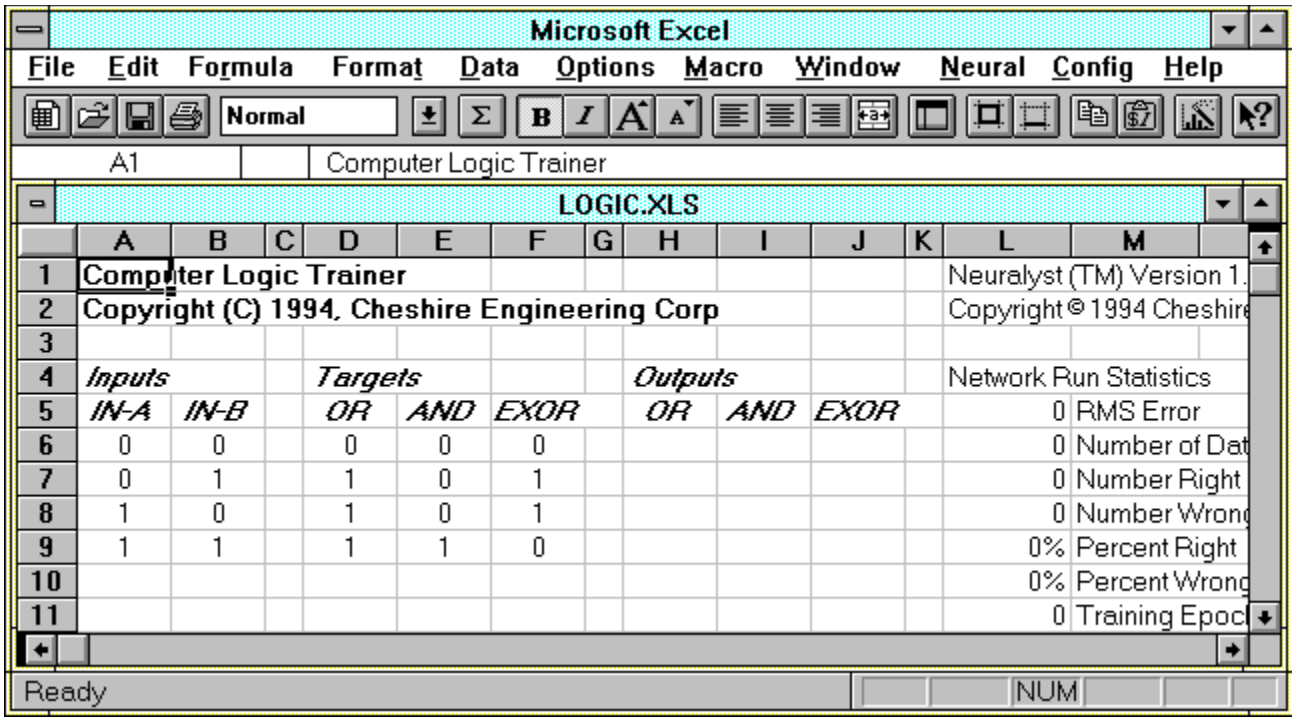
The First Neuralyst Session

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 containing the worksheet.



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.

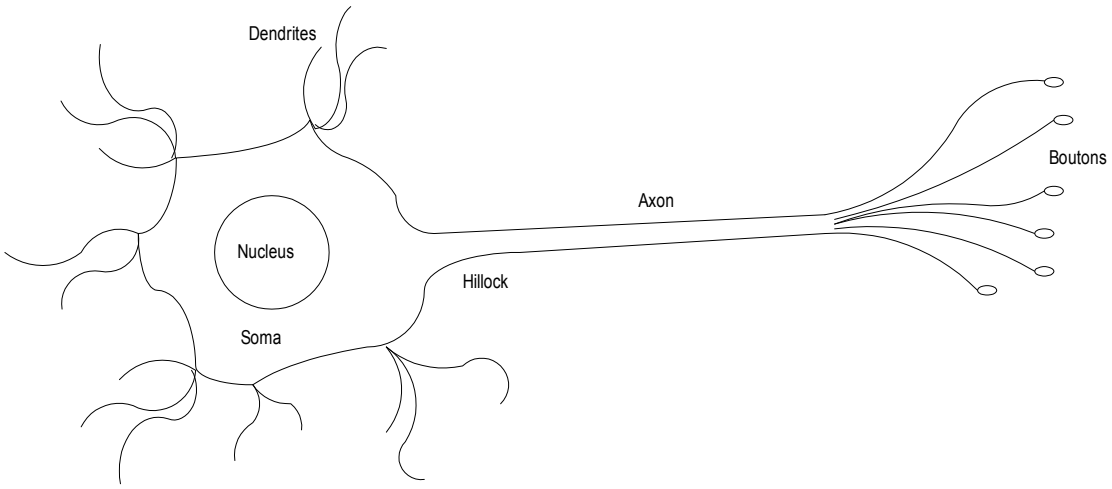
[If you have a Macintosh with a small screen, then Neuralyst will name its menus **N.** and **C.**, rather than **Neural** and **Config**, to save space. If you have a Macintosh, System 7, and a small screen, the normal Neuralyst interface will not work well; an alternate interface is available to handle this circumstance, contact Cheshire for more details 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 demonstration version of Neuralyst will show a message that says "This demonstration version of Neuralyst cannot load a pre-trained neural network. Reset all learning performed so far?" at the end of the Reload Network command. Move the pointer to the "Yes" button and click the mouse button. This message appears any time you reload a network configuration that may include trained network weights. The neural network has now been loaded with the configuration saved on the worksheet and an initial set of random weight values.
[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.

Basic Concepts for Neural Networks

Real Neurons

Let's start by taking a look at a biological neuron. The following figure shows a sketch of such a 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.

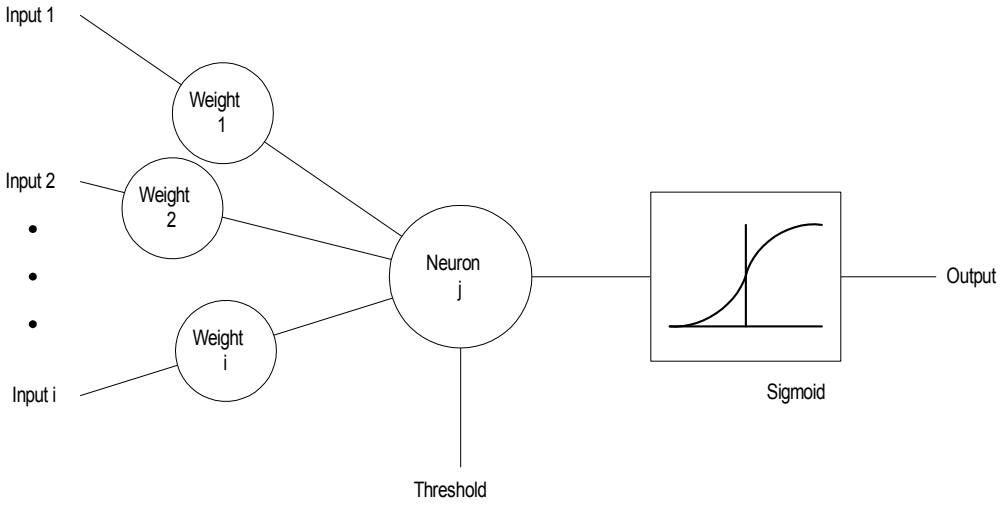
Compare this to a very fast electronic computer whose signals travel at about 200,000,000 meters per second (speed of light in a wire is 2/3 of that in free air), whose impulses last for 10 nanoseconds and may repeat such an impulse immediately in each succeeding 10 nanoseconds continuously. Electronic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

Neural Network Structure

Neural networks are models of biological neural structures. The starting point for most neural networks is a model neuron, as in the figure below. 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.

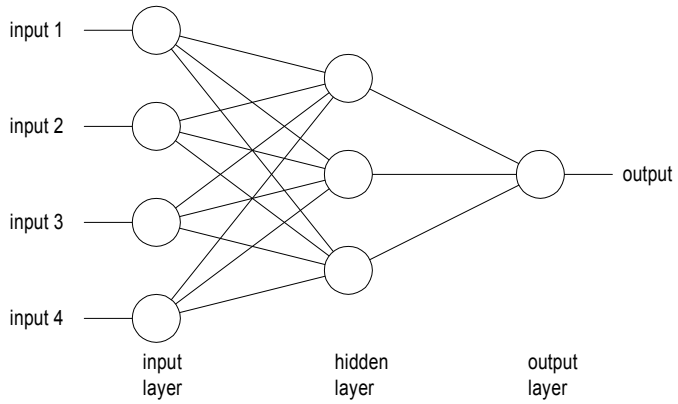


While there is a fair understanding of how an individual neuron works, there is still a great deal of research and mostly conjecture regarding the way neurons organize themselves and the mechanisms used by arrays of neurons to adapt their behavior to external stimuli. There are a large number of experimental neural network structures currently in use reflecting this state of continuing research.

In our case, we will only describe the structure 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 contact Cheshire for additional references.

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 the following figure. 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 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.

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.

A Neuralyst Tutorial

This section shows a complete example of a working neural network built with Neuralyst. It is intended primarily to teach the skills and concepts required to operate Neuralyst and construct representations of problems suitable for treatment by Neuralyst. A later section will describe a variety of additional examples that further explore the capabilities of a neural network for problem solving.

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.

[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.]

{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.

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.
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.

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.
The **Config** menu contains the interface to the commands that allow you to define a problem. 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.

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 four 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 Lists** 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.

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 network. **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.
[The Genetic training supervisor is disabled in the demonstration version.]
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.

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.

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 the previous section the basic capabilities and operation of Neuralyst were discussed. This section focuses on several additional facets of Neuralyst and neural network behavior. The examples in this section 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!

Overview of the Examples

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 last 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.

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.

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 the earlier tutorial section, 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.

Paper-Rock-Scissors — 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.

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.

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 mug book, 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 mug book, based on four physical characteristics: sex, age, coloring, and weight of eight known criminals in Midtown, USA. 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). 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

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.

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.

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.

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 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.

Ordering Neuralyst

Information

Now that you've tried out the demo version, purchase Neuralyst, and you will get:

* Full Neuralyst software distribution:
- Allows trained neural networks to be saved and reloaded
- Data Capacity up to 255 columns and over 6500 rows
- Neural networks with over 130,000 connections
-

Genetic Supervisor to optimize neural networks enabled

- Trader's Macro Library (technical indicator package) enabled

* Neuralyst User's Guide (over 220 pages including tutorial sections)

* Phone access to Cheshire Engineering Customer Support

* 30-day money-back guarantee (less shipping and handling)

The price of Neuralyst is $195 per package. Shipping and handling, sales tax, and import duties, where applicable, are extra.

For prompt delivery call (818)351-0209 or fax (818)351-8645 and order with your credit card. Or if you prefer fill out and mail the form below with your payment to:

Cheshire Engineering Corporation
650 Sierra Madre Villa, Suite 201
Pasadena, CA 91107

Neuralyst is available for Apple Macintosh or PC's running Microsoft Windows. Neuralyst works with Excel 4.0/5.0 or higher and System 7.x or higher on Macintosh or Windows 3.1 or higher on PC's.*

* Neuralyst is a trademark of EPIC Systems Corporation licensed to Cheshire Engineering Corporation. Apple and Macintosh are trademarks of Apple Computer Corporation. Microsoft, Excel and Windows are trademarks of Microsoft Corporation.

Order Form

Name_____

Company_____

Address_____

City _____State/Country _____ Postal Code_____

Phone _____ (required for credit card orders)

_____ Macintosh Neuralyst Packages @ $195 each =

_____

_____ Windows Neuralyst Packages @ $195 each =

_____

Specify disk type: ____5-1/4"; ____3-1/2"

plus Sales Tax (CA residents only) =

_____

plus Shipping and Handling per package =

_____

US

Priority Mail: ___$5

Canada/Mexico Air: ___$8; Other Foreign Air: ___$15

US Express: ___$20; Foreign Express: ___$50

Total (all items above) =

_____

_____ Check or Money Order (enclosed)

_____ Credit Card _____ Visa

_____ M/C

_____ Exp Date

_____ Credit Card Number

_____ Signature (required)

Legal Notices

Neuralyst™ Demonstration Guide by Yin Shih, Edited by Shal Farley and Ross Berteig.
Copyright © 1994 Cheshire Engineering Corporation. All Rights Reserved.
The information in this document is subject to change without notice and should not be construed as a commitment by Cheshire Engineering Corporation. Cheshire Engineering Corporation assumes no responsibility for any errors that may appear in this document.
The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of that license (see page).

Trademark Notices

The Epic logo and Neuralyst™ are trademarks of Epic Systems Corporation licensed to Cheshire Engineering Corporation.
Apple® and Macintosh™ are trademarks of Apple Computer Corporation. IBM®, PC/XT™, PS/2™, and PC/DOS™ are trademarks of IBM Corporation. Microsoft®, Windows™, and Excel™ are trademarks of Microsoft Corporation.

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.

License Agreement

NEURALYST FOR EXCEL DEMONSTRATION
EVALUATION LICENSE INFORMATION

This section contains license information for users of the Neuralyst for Excel Demonstration (Neuralyst Demo Software). Neuralyst Demo Software may be freely used and distributed, subject to the terms listed here. Your use of the Neuralyst Demo Software implies acceptance of the terms of this license.

This license is governed by the laws of the State of California.

DEMO USE LICENSE

Neuralyst Demo Software, which includes all the programs and documentation listed in the Packing List Section of the related README.TXT file, are NOT public domain programs. They are Copyrighted Material by Cheshire Engineering Corporation and others. All rights are reserved.
This software and the accompanying documentation are protected by US. Copyright Laws and by International Treaty. Any violation of the terms of this license or the Copyright Law will be prosecuted to the full extent of the law.
You are hereby granted a limited license to use this software for evaluation purposes only. You may not use, copy, rent, lease, sell, network, transmit, modify, adapt, translate, decompile, disassemble, or otherwise reverse engineer, create derivative works or transfer this program except as provided in this agreement. Any such unauthorized use shall result in immediate and automatic termination of this license.
All rights not expressly granted here are reserved.

LIMITED DISTRIBUTION LICENSE

Individuals and Company Users are granted permission to copy the evaluation version of Neuralyst Demo Software for their own evaluation, or the evaluation of other individuals, ONLY when the following conditions are met.
The Neuralyst Demo Software is defined as containing ALL the material listed in the Packing List Section of the related README.TXT file. If any of the files listed in README.TXT are missing, or README.TXT itself is missing, distribution is forbidden. Please contact us to obtain a complete package suitable for distribution.
The Neuralyst Demo Software, including all related programs, documentation and data files, cannot be modified in any way and must be distributed as a complete package.
No price or other compensation may be charged for the distribution of the Neuralyst Demo Software other than the actual cost of mailing or electronic transmission.
The Neuralyst Demo Software cannot be sold as part of any other package, nor may it be

included in any commercial software packaging offer without the express written consent of Cheshire Engineering Corporation.
Under no circumstances do the terms of distribution listed here apply to other Neuralyst products that are not part of the Neuralyst Demo Software. Other Neuralyst products are also Copyrighted Material and licenses for their use are available for purchase from Cheshire Engineering Corporation.
Cheshire Engineering Corporation650 Sierra Madre Villa Avenue, Suite 201Pasadena, CA 91107USA+1 818 351 0209 +1 818 351 8645 FAX