Tutorial Example One

The best way to explain how to use NeuroShell 2 is to build a neural network application. Therefore what follows is a NeuroShell 2 tutorial on how to create a network which will predict the electricity cost per day in a home. The problem file named Electric is in the set of NeuroShell 2 examples available on the distribution diskette, so you can follow along on the computer as you read this tutorial.

The Problem

Sean DuPree is the sales manager of a new development of "smart" homes that have built-in computer systems to monitor security and energy usage, turn lights off and on, etc. As sales manager, he has lived in one of the homes since 1992. Since the real estate market in his area has become more competitive, he wants to create a neural network which will help him market the houses in the development.

The Outputs

Electric rates are relatively expensive in his area and he believes his "smart" homes can substantially save the owners money. He decides to create a network which will predict the cost of electricity per day in the homes he is selling.

The Inputs

Once he decides what he wants to predict, Sean has to decide which variables need to be considered when making that prediction. He believes that several factors can influence a home's electric cost each day, including the average daily temperature, the number of people in the home, the residents' habits of leaving lights and appliances turned on, etc. He wants to keep his problem simple for his first attempt to build a neural network application, so he decides he will use what he considers to be the most important variable, the average daily temperature, to predict the cost of electricity per day in his home.

Note that prior to deciding to use a neural network, Sean had considered regression analysis, but believed one factor would make it difficult to construct a linear model: he uses a wood stove when the temperature is between 32 and 45 degrees, so the daily electric cost is not linearly related to the temperature.

Running an Example Program

Once Sean decides what he wants to predict and which variables he is going to use to make that prediction, he is ready to use NeuroShell 2.

You can follow along with his example by using the mouse to double click on the NeuroShell 2 Program Group that is displayed in the Windows Program Manager. Double click on the NeuroShell 2 "brain" icon. Select the NeuroShell 2 File Menu and choose the Open Option. Several list boxes are displayed which include the names of subdirectories and files. Double click the mouse on the NeuroShell 2 Examples subdirectory to display a list of files with a .DSC suffix. Single click the mouse on the file "ELECTRIC.DSC" to open the Tutorial Example One. Click "OK" to load the file.

The NeuroShell 2 Main Menu is displayed with icons for the Beginners, Advanced, and Runtime modules. Proceed to the Beginner's Neural Network system by double clicking the "tricycle" icon.

The usual procedure for creating a NeuroShell 2 application in either the Beginners or the Advanced system is to follow the icons from left to right, top to bottom.

<u>Data Entry (Datagrid)</u>

Since Sean is going to enter our data directly into NeuroShell 2, skip the first icon, which is File

Import. Double click on the Data Entry icon, and a spreadsheet-like data entry form is displayed.

Note that a warning appears on the spreadsheet which says the Datagrid should only be used for small problems. Sean's problem is considered a small problem so the Datagrid is fine for this example. We'll tell you how to work with larger files by using your usual spreadsheet program at the end of this tutorial.

First, Sean decides that he wants to include the names of the variables as column headings with his data. He types a 1 in the edit box for "Number of row with variable names." He types a 2 in the edit box for "First row containing actual training data."

When Sean opens a new problem, the Datagrid displays the "dummy" column names Name 1, Name 2, and Name 3 in row 1and 0 values in rows 2 and 3 beneath the column names.

In row 1 he types in the names "Avg Temp", and "Cost/Day". Beginning with row 2, he types in data for the past year. He includes a daily temperature reading for 1992, sorted from the lowest to highest temperature, as well as each day's cost. Each day's data is entered on a separate row. His final spreadsheet contains 1 row of labels and 365 rows of data. He uses the File menu's save option to save the data under the name "ELECTRIC.PAT". (This file has been created for you.)

Sean also wants to create another set of data which he will use to test the trained network. While still in the Datagrid Module, he selects New from the File Menu. Unlike the first time Sean opened the Datagrid, a spreadsheet is displayed without any data entered in the cells. (This file, called ELECTRIC.PRO, has also been created.)

When you initially open the Datagrid Module and a spreadsheet is displayed, the "dummy" column names and data are displayed, which means that the Datagrid has already been formatted so that row 1 cells are text type and rows 2, 3, etc. cells are numbered with decimal places. If you open a second Datagrid sheet from the Datagrid File Menu, this dummy information will not be displayed and you will have to use the Datagrid Format Menu to set row 1 as text type if you want to enter and display column names.

Sean clicks on row 1 to blacken the entire row. He then selects the Format Menu , Data Type option, so he can select Text type which will allow him to enter column names. He selects Left Aligned text to match his other spreadsheet.

Sean enters the column names "Avg Temp", and "Cost/Day" in the Datagrid. Sean selects 46 days from his 1993 data which cover a range of temperatures. He enters the temperature and cost per day for each of the 46 days. He uses the File Menu's Save option to save the file as ELECTRIC.PRO.

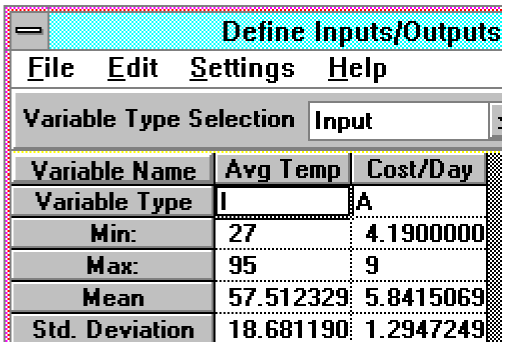
Define Inputs and Outputs

Next, Sean needs to inform NeuroShell 2 which of the columns are inputs and which are the actual outputs. Double click on the Define Inputs/Outputs module. The module displays all of the column names. You need to make an entry in the Variable Type row for each column. Use the mouse to select one of the options from the list box: Inputs, Actual Outputs, or Blank (not used). Once you have made a selection, click on the cell in the Variable Type row beneath the column to mark your selection.

Sean decides to designate the Avg. Temp column as an input, I, and the Cost/Day column as the column which the network is trying to predict (actual output, A).

The next step is to enter the minimum and maximum values of each variable in the minimum and maximum rows . Since neural networks require variables to be scaled into the range 0 to 1 or -1 to

1, the network needs to know the variable's real value range. You may use this module to enter a minimum and maximum value for each variable that is to be included in the network, or you can compute the range automatically from your data by selecting the Compute mins/maxes option. Selecting this option also calculates the mean and standard deviation for each variable.

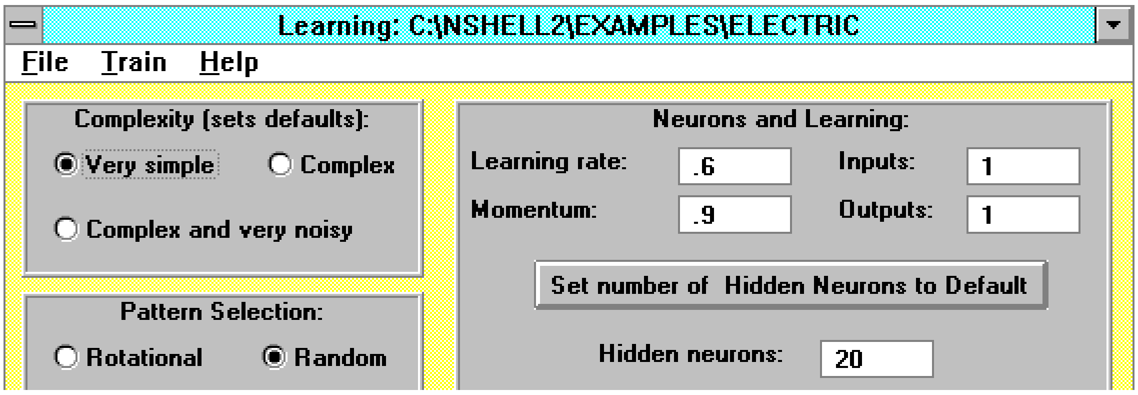


Variable Type Selection: Input

Variable Name	Avg Temp	Cost/Day
Variable Type	I	A
Min:	27	4.1900000
Max:	95	9
Mean	57.512329	5.8415069
Std. Deviation	18.681190	1.2947249

In general, use a range that is tight around your data. (You may want to specify minimum and maximum values that are slightly above and below values in the data file to allow a wider range for future predictions, or you may want to select values smaller than existing values to eliminate outliers that may affect the network's precision. Refer to **Define Inputs and Outputs** for details.) If you fail to set the minimum and maximum values tightly around your data, the network may lose its ability to spot small differences in the data.

Sean proceeds to train the network by double clicking the learning icon to bring up the training module. When it comes up it should already have sensed the number of inputs and outputs from the .MMX file (created in the Define Inputs/Outputs module). However there are still several parameters Sean needs to set before he can start training. First he has to specify the problem complexity by clicking on one of the radio buttons on the top left of the screen. Although this is not a "toy" problem of the type many neural network inventors and writers like to use for testing purposes, such as the XOR problem, Sean decides that his data is very simple. He uses the mouse to click on the Very Simple button. Notice that the learning rate and momentum factors are automatically set to .6 and .9.


Learning: C:\NSHELL2\EXAMPLES\ELECTRIC
File Train Help
Set number of Hidden Neurons to Default

Complexity (sets defaults): [✓ Very simple] Complex Complex and very noisy

Pattern Selection: Rotational [✓ Random]

Neurons and Learning:

Learning rate:	.6	Inputs:	1
Momentum:	.9	Outputs:	1

Hidden neurons: 20

Next the number of hidden neurons has to be set. Clicking the default button computes the NeuroShell 2 default number which is a good place to start for future problems. The default number of hidden neurons for a 3 layer network is computed with the following formula: # of hidden neurons = 1/2 (Inputs + Outputs) + the square root of the number of patterns in the .TRN file if it exists; otherwise the .PAT file.

Sean decides to use random presentation of his data because it is sorted from low to high temperature and he wants the network to be able to predict well on all temperatures.

Sean sets the **Calibration** interval to 0 for his initial training session, although this is a powerful feature that should be used for most problems.

It's time to begin training by selecting Start Training from the Train menu. The training set statistics are updated every epoch (one complete training pass through the data).

The main training statistic is the internal average "error factor". You cannot calculate or reproduce this average error factor exactly yourself on the training set, nor is it in any way useful for you to do so. However, it is the mean over all patterns of the squared error of all outputs computed within NeuroShell's internal interval. The value of the number is not useful in itself. It is useful during training to see if the network is improving, because it gets lower as the network improves.

When should training be stopped? Sean notes that after 1000 epochs (an epoch is an entire pass through all of the 365 training patterns) the minimum average error is equal to approximately .0017 and that the events since the minimum average error are equal to approximately 400. The network does not seem to be making additional progress towards a lower minimum average error. To stop learning he selects Interrupt Training from the Train menu. (The advanced networks have features that will automatically stop training for you.)

Apply to File

Sean wants to see if his network is producing good results. He double clicks the "Apply to File" icon, and selects Start Processing from the Run Menu. By default the check boxes for Compute R squared, "Include actuals in .OUT file," and "Include in .OUT file actuals minus network outputs" are turned on.

The Apply Module defaults to processing the .PAT file, which is the first set of data that Sean entered. Each day's temperature (the input) is processed through the trained network and it computes a cost per day for electricity. Statistics which measure the accuracy of the trained network are displayed on the screen.

The trained network produces an R squared value of .9539 for the .PAT file. He records the R squared statistic to compare this network with other ones he might create later.

R squared, the coefficient of multiple determination, is a statistical indicator usually applied to multiple regression analysis. It compares the accuracy of the model to the accuracy of a trivial benchmark model wherein the prediction is simply the mean of all of the samples. A perfect fit would result in an R squared value of 1, a good fit near 1, and a very poor fit near 0. If your neural model predictions are worse than you could predict by simply using the mean of your sample case outputs, the R squared value will be 0.

Now Sean views the .OUT file which shows the actual cost per day that he entered in the file, the network's predicted outputs, and the differences. You may optionally want to "attach" the output file to the original pattern file in order to view the original pattern file with the network's predictions. The "Attach Outputs" module will do this for you, creating a new .OUT file in the process. Click on the "Attach Outputs" icon to use this module. Since you probably want to view the network's results beside the input variables, click on the Attach File Side by Side icon. (The correct file names should be displayed as the default settings. ELECTRIC.PAT is Sean's original file and ELECTRIC.OUT contains the network's answers.)

Examine Data

Sean is now ready to look at the .OUT file. He can do this with our Datagrid by double clicking on the "Examine Data" icon.

The Datagrid is not a commercial grade spreadsheet and is in fact somewhat slow loading large files. If you have a very fast computer this may be all right; otherwise use your spreadsheet.

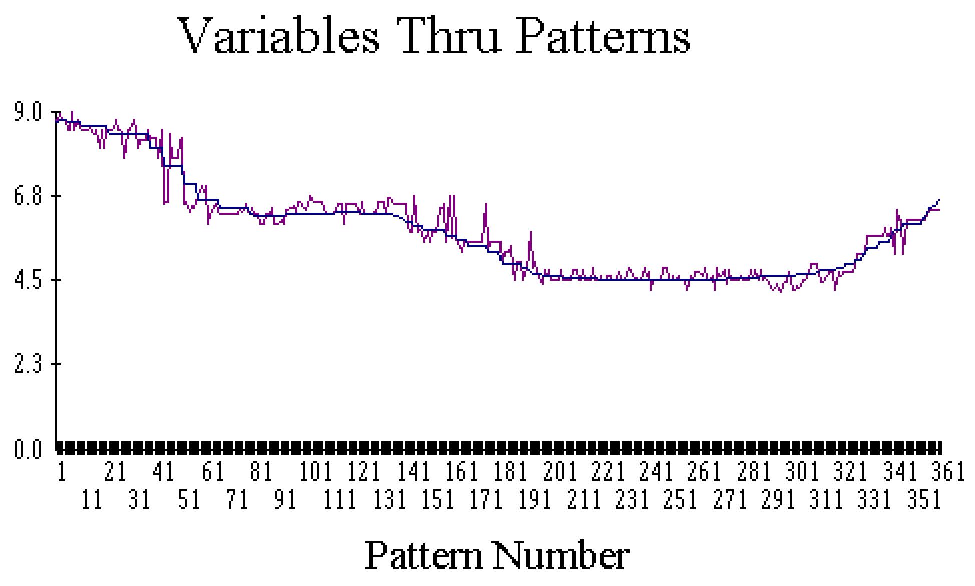
Using Your Usual Spreadsheet Program

You may change NeuroShell 2 so that it always calls your spreadsheet instead of the Datagrid. From the NeuroShell 2 Main menu, select Options, Select Datagrid Program. File directory and list boxes will be displayed. Click on the .EXE program which runs your spreadsheet program and click OK.

Sean also wants to apply the module to the .PRO file. He uses the File Menu to select an alternate pattern file, ELECTRIC.PRO. He then selects Start Processing from the Run Menu. He notes that the network gets an R squared value of about .95 on the .PRO file.

Sean may want to evaluate his electric cost prediction model in some way other than using R squared.

He decides he wants to view the data graphically. He exits the Beginner's System and returns to the NeuroShell 2 Main Menu and selects the Advanced System. Under the Post Network column, he selects the Variable Graphs icon. He clicks on the Graph Variable(s) Across all Patterns icon. He clicks on Actual(1) from the Variable/Column list and then uses the Ctrl key and clicks the mouse on Network(1) to select both columns. He then clicks on the Graph button.



His neural network model appears to closely follow his actual data.

Let's suppose he decides that the answers aren't good enough. What should he do? The easiest thing to do is to try other network architectures from the Advanced Neural Networks (racing bike) icon.

He may get somewhat better results from other architectures, but in reality what he really needs to do is probably one of the following, listed in order of likelihood:

1. **Use Calibration**. NeuroShell 2 uses Calibration to optimize the network by applying the current network to an independent test set during training. (You can automatically create a test set of data by using the **Test Set Extract** module. Calibration finds the optimum network for the

data in the test set (which means that the network is able to generalize well and give good results on new data).

Calibration does this by computing the mean squared error between actual and predicted for all outputs over all patterns. (The mean squared error is the standard statistical technique for determining closeness of fit.) Calibration computes the squared error for each output in a pattern, totals them and then computes the mean of that number over all patterns in the test set.

For Backpropagation networks, the network is saved every time a new minimum average error (or mean squared error) is reached. In order to use Calibration, you need to set the Calibration test interval, which is how often the test set is evaluated. We suggest setting it within the 50 to 200 range. You must also select Save network on best test set.

2. Get variables which are better predictors of what you are trying to predict and/or figure out better ways to represent the ones you have. Sean may have gotten better results by including a variable for number of people in the home. More people directly affects hot water usage for laundry, showers, etc.

If he had a large number of inputs, he might want to think about converting some inputs into ratios. This provides more information with less variables. Remember that neural nets are like people: the more you simplify the inputs they are expected to learn, the easier it is for the network to learn the task. Ratios serve this purpose.

3. Reevaluate what it is you want to predict. Some things are easier than others. You may get more accuracy predicting the percent change in electric price rather than the exact price.

4. Collect a better set of historical patterns, or a more representative test set. Make sure your variables are normalized if necessary. In a stock market example, this means making sure levels of several years ago are adjusted to the same range as today's levels. In the scientific area, normalization can mean many other things. You may call us for assistance. If you don't normalize, then you will have to show the network many more patterns with a corresponding increase in learning time.

5. Try adjusting learning rate, momentum, and hidden neurons and see if better networks result. Try TurboProp, which does not require you to set a learning rate and momentum. It is included in the Advanced System's Design module and works with backpropagation networks.

Note: The algorithms and techniques used in the tutorial example may have changed since the Help File was written. Refer to Program Changes in the index for any changes, which may include other ways of preprocessing data or training. The data used in this problem was created for example purposes and was not based on real electric bills. The name Sean DuPree is fictional and is not based on any person living or dead.