Tutorial Example Two

The best way to explain how NeuroShell 2 is utilized to build a practical neural network is by example, we believe. Therefore what follows is a NeuroShell 2 tutorial in the form of an explanation which takes you through the major steps of building an actual neural network problem. The problem file is in the set of NeuroShell 2 examples, so you can even follow along on the computer as you read this tutorial.

You may also want to refer to **Tutorial Example One** for information on how to create a network which predicts the dollar value of an electric bill.

We have chosen **a stock market prediction example because it is very representative of many types of other prediction problems,** including sales predictions. Read through this even if market and futures predictions aren't your interest.

The Problem

In the example we put ourselves in the place of a stock or options trader who wants to build a network that will predict tomorrow's New York Stock Exchange Index. We realize that price prediction of this nature is difficult and that we will never achieve 100% accuracy, but we also know that an excellent modeling tool like NeuroShell 2 will increase our odds of making profitable trades if it predicts better than we humans can predict and accurately enough to cover overhead of making the trades, etc. We have a great deal of historical data from which we can show the network's correct predictions, using our hindsight.

Note: This is an example only and does not purport to be an accurate predictor of the market. Users who have the most successful predictive systems of any kind are those who can apply expertise in the problem domain to choosing exactly what to predict and what variables to use for the model.

Decide Upon the Outputs

The first thing we must decide is what exactly we want to predict. Some of the multitude of possibilities are:

1. **Tomorrow's NYSE index price**
2. **The NYSE index price next week or next month**
3. **Whether or not the index price will move above or below a given range in a given period of time**
4. **Whether or not we are in a buy, sell, or hold situation**

Upon careful thought we realize that #2 and #3 above are much the same since in order to determine whether to buy, sell, or hold, most people will probably first want to know whether the stock will be moving above or below predetermined levels. Lets say the level is +10 points for buy, -10 points for sell, and between the two for hold. Let us further decide that the given period of time is three weeks. If we decide to predict either of those, we conclude that we would want three Boolean (true/false) outputs as follows:

Output #1 - True if the index will move up 10 points from current level in 3 weeks (BUY); otherwise False

Output #2 - True if the index will move down 10 points from current level in 3 weeks (SELL); otherwise False

Output #3 - True if the index will not move 10 points in either direction in 3 weeks (HOLD); otherwise False

Output Strength

If we decide that the outputs coming from the network will be either 0 for False or 1 for True, then we have to decide what do when the outputs have only partial strength, say .65. We decide for the moment that anything .5 or over will be considered a 1, and any value below .5 will be considered a 0. Later we can observe what happens with live data and adjust this threshold of .5 to reduce false positives (by raising it) or reduce false negatives (by lowering it).

Perhaps you noticed that we might even get away with only outputs 1 and 2 above, since if neither of those is true, we must be in a HOLD situation, right?. Perhaps not. The network may not be able to classify the current situation at all and both outputs could be below the .5 threshold we decided upon. At least with the three outputs we would have a definite HOLD signal.

This network, solving possibilities 3 and 4 above has a better chance of succeeding than one to solve possibilities 1 and 2, because it is obviously easier to predict that the market will rise 10 points than to predict exactly how many points it will rise. But we decide nevertheless to do the harder problem of predicting price.

To predict a price we need only one output. We will be reading the strength of the one output not to determine how true it is as before, but we will read the output's strength as the price we want to predict. In other words, the network's output will not be a fuzzy true/false signal (fuzzy means it can have values in between, like .75 which could mean probably true). Instead we will take that .75 and consider it to be a price. You'll see how that works in a minute.

Should we predict a day, week, or month in advance? We know that daily data is noisy and that the index only moves a little each day. Both of these factors make daily prices harder to predict, but we are day traders so we take the hard route.

Decide Upon the Inputs

We must now decide what inputs the network will need to predict tomorrow's NYSE index. Obviously today's price is a starting point, so we will use that. To some extent we are limited by the data we have at hand from which to choose inputs. We decide to use many of the technical indicators we have:

NYSE high, low, and close
SPX high, low, and close (SPX = S&P 500 index)
NYSE advances, declines, and unchanged
Advances, declines, unchanged, and total volume
NYSE new highs and new lows

In order to show the network trends in the data we will also want to feed the network yesterday's price, the price the day before that, etc. Also, the average of the NYSE index for the last few days will help, we think, and some other technical indicators.

We can use a spreadsheet to produce these inputs, but we choose to utilize the **Market Indicators Package**, a NeuroShell 2 option to do so. If you don't have this package, you can probably write spreadsheet macros to do them yourself, or perhaps you already have another indicator package. By the way, these indicator packages are useful for any type of time series data, not just stock market data. The indicators have already been applied for you in the example, but we have only applied a few very simple ones and have not done any study to see which might work the best, since this is only an example.

Develop Case Histories

We have in hand a spreadsheet file containing the last 20 years of NYSE index prices, as well as the

other inputs we will use (NYSE20.WK1). This is a clip from the top of that spreadsheet:

DATE	NYSE high	NYSE low	NYSE close	SPX high	SPX low	SPX close
870102	141.01	139.95	141.01	246.45	242.17	246.45
870105	144.39	142.15	144.39	252.57	246.45	252.19
870106	145.39	144.6	144.81	253.99	252.14	252.78
870107	146.43	144.91	146.43	255.72	252.65	255.33
870108	147.55	146.35	147.55	257.28	254.97	257.28

Each column is an input to the network and every row is a potential historical pattern. We notice that in the period 1988 to near the end of 1991 the NYSE index close varied in the range about 136 to 217, but before that the index was much lower. In order to use patterns for days before 1988 the values of the prices and probably many of the input indicators should be normalized to current levels. See **Market Predictions** for details. Therefore, we will only use 4 years of data, since there are plenty of patterns here anyway. We create another spreadsheet (NYSE.WK1) with only these years. As previously mentioned, we have used the NeuroShell 2 **Market Indicator Package** to create lags, moving averages, and some regressions, but we do not explain that process here.

In order to show the network tomorrow's index price, we created another column. To create this column all we need to do is to copy the NYSE close and move it UP 1 row in the new column. We placed an asterisk in the same column in the final row where a value was vacated. The NeuroShell 2 learning algorithms will not try to learn rows which have an * in an input or output column. The Indicator Package also will do this for you, but it has already been done in the NYSE sample files.

Using NeuroShell 2

After starting NeuroShell 2 we use the File Menu to choose the name of our problem by selecting the description file NYSE.DSC (or we could choose any of the other NYSE files, or simply type NYSE). These files are in c:\nshell2\examples, assuming you installed NeuroShell 2 in c:\nshell2. Then the name of the problem becomes NYSE. Proceed to the **Beginner's Neural Network System** by double clicking the "tricycle" icon.

File Import

Following the icons from left to right, top to bottom we are immediately reminded that the spreadsheet file should be imported. NeuroShell 2 internal pattern files use a spreadsheet format, the .WK1 format. All major spreadsheets can save files in .WK1 format, so we have previously saved our spreadsheet file as NYSE.WK1.

Even though the internal format is .WK1, we still need to import it in order to store within it information about where the data actually begins and which row will contain descriptive column (variable) labels. The spreadsheet import program does this very quickly, so we select that icon after double clicking the **File Import** icon. That action will start a separate "module" to actually perform the import. From the module screen we record in the appropriate text boxes the label row (which is row 1) and the first data row (which is 2). Then from the Import menu we begin the importing, creating the primary NeuroShell 2 pattern file in the process, NYSE.PAT. (Refer to **NeuroShell 2 Files** for an explanation of all of the file types used in NeuroShell 2.) Exit the module when the import action is complete, then close the file import screen.

At this point we would apply technical indicators to the pattern file. Since you may not have purchased the Market Indicator option, we should restore the backed-up pattern file which has the indicators already applied for you. This is OK because the import was only an exercise for you; the

pattern file was correct before the import. Go to the Utility Menu on the main window and select Restore Backed Up Pattern File. Now the pattern file is as it was before you imported the .WK1 file, and it has indicators applied.

Define Inputs and Outputs

Next we need to inform the system which of the columns are inputs and which are the actual outputs. The Define Inputs/Outputs module will do this. Double click its icon to bring it up. This module will display all of the column names with several rows below them where we can document for NeuroShell 2 which are the inputs (I) and which are the actual outputs (A). Two of the rows are for supplying the minimum and maximum values of each variable. Fortunately there is a menu item for computing these mins and maxes. When this module is finished it creates a min/max file (.MMX) to hold the information we just gave it.

Test Set Extract

One more thing is required before we actually start training. We need to extract a "test set" from the pattern file if we want to utilize one of NeuroShell 2's most powerful features: Calibration. Calibration optimizes training by assuring that just the right amount of training is applied to the pattern file and no more. The right amount is defined as the amount necessary to minimize the mean of the error factors on the test set. Therefore it is important that the test set resemble as closely as possible the what the network is likely to encounter in future live predictions. Double click the appropriate icon to extract a test set.

There are many ways a representative test set can be chosen. One of the most popular is simply a random extraction of patterns from the pattern file (about 10% to 40% is usually good.) Assume that we plan for our network to make good predictions for at least a month. Then the last month for which we have data might also make a reasonable test set. If the data is seasonal and we want to predict next month (June, say), then a test set composed of the last 3 Junes' data might be more representative. For our purposes we conclude that we will extract 10 percent of the patterns for a test set, leaving the remainder as the "Training set." Select the radio button that performs this task, and set N = 10. Then do the extraction. The result should be that two additional problem files have been created: the training set (.TRN) and the test set (.TST). The union of these two (i.e., all their patterns together) is the original pattern file (.PAT). When using Calibration, both the .TRN and .TST files are used during training. When not using Calibration only the .PAT file is used. If there is no .TRN file, the .PAT file will be used in both cases.

Learning

Now we are finally ready to start training. Double click the learning icon to bring up the training module. The **Beginner's Learning** module utilizes only one of the 16 major architectures that NeuroShell 2 uses, but it is not necessarily the least capable. On the contrary, most successful neural network models in use today utilize this architecture, called a three layer Backpropagation paradigm.

When the module comes up it should already have sensed the number of inputs and outputs from the .MMX file. However there are still a couple of pieces of information we need to specify before we can start training. First with the radio buttons on the left we will specify the type of problem we have. This is not a simple or "toy" problem of the type many neural network inventors and writers like to use for testing purposes. The XOR problem is an example of an extremely simple problem. However, our data is daily and probably noisy, so we'll select complex and noisy. Notice that this action has set the learning rate and momentum factors to .05 and .5.

Now the number of hidden neurons has to be set. Actually, since we have already trained this problem, the number we used (40) should still be showing. Clicking the default button sets the default number which is a good place to start for future problems. Also for complex problems,

random pattern selection is usually (but not always) better than rotational.

The proper setting of the factors we set in the paragraph above are part of the art of neural networks (it is not really a science yet, we believe, and to date there are no absolute formulae or even extremely successful rules of thumb for setting them.) In the final analysis, neither our defaults nor our suggestions should determine how these factors are set. Your experience with your particular problem is the determining factor: use whatever works the best!

Moving down the training window, we need to set **Calibration on**. This is accomplished by setting the Calibration interval to something higher than zero. We have good experience with 200, but like the other factors, try several values over time and see what works the best. We will choose 50 for this exercise, since we want maximum accuracy.

Every 50 training events, the mean error factor for the test set will be computed, and the network will be saved whenever one produces a lower average error. Make sure that Save on "Best Test Set" is also selected. If we weren't using Calibration, we would want to save on the best training set, since we would not be trying to optimize training on an independent test set.

It's time to select training from the menu. You may not see anything for a short period until 50 training events have been completed, whereupon the test set statistics will be updated. The training set statistics are updated every epoch (one complete training pass through the data.)

Error Factor

The main training statistic is the internal average "error factor," i.e., the error computed after the numbers are scaled by NeuroShell 2. You cannot calculate or reproduce this average error factor exactly yourself, 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. This number is not useful for its value itself. It is useful during training to see if the network is improving, because it gets lower as the network improves. As the network learns the training set better, the average error for the training set gets lower, and eventually makes very slow downward progress. As the network does better on the test set, the test set error decreases, and then usually starts increasing after awhile. This is because the network is getting worse on the test set even though it is getting better on the training set. Calibration saves the network when the average test set error is at its lowest point.

When should training be stopped? Since we are using Calibration, we should let it be our guide. When no better network has been found in quite awhile (20,000 events is usually sufficient) we can select interrupt training from the menu, especially if we notice that new test set average errors are climbing generally or at least not close to the lowest that has been found. (The advanced networks have features that will automatically stop for you.)

Apply Network

Once you have decided to interrupt training and exit the training module, you can now apply the network you have just created to the pattern file. In other words, feed every pattern through the network and record the network's outputs on another file (the .OUT) file. You do this by double clicking the "**Apply to File**" icon, and executing the module that it starts up. All you should have to do is start the processing from the Run menu. The entire pattern file will be passed through the network and statistics to show you how accurate the network is will be generated on the screen. You can also apply the module to the .TRN or .TST files if you desire. Record the R squared statistic to compare this network with other ones you might do later. Do not confuse our R squared statistic with the r squared statistic in statistic books or other packages! However, R squared is in advanced statistics books.

Attach Output File

Now you can view the .OUT file which shows the actual outputs, the network's predicted outputs, and the differences. You may optionally want to "attach" the output file to the original pattern file to get all outputs together with the inputs as well. The **Attach Outputs** module will do this for you, creating a net .OUT file in the process. Execute the Attach module, which should default to the correct file names.

Examine Data

Now you are ready to look at the .OUT file. You can do this with our Datagrid with the **Examine Data** icon, but you must remember 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. You can call one of the three popular spreadsheet programs from the main Utility menu. All you have to do is select the .OUT file to view. You can also change NeuroShell 2 so that it always calls your spreadsheet instead of the Datagrid (see the Options Menu).

Advanced Neural Networks

You may want to evaluate your new stock model in some way other than using R squared. Let's suppose you decide that the answers aren't good enough. What should you do? The easiest thing to do is to try other network architectures from the **Advanced Neural Networks** module. The recurrent networks, for example, are excellent for time series data. A recurrent net will require consecutive patterns in both the training and test sets, however, so you will have to extract differently. Recurrent nets may work better without the added indicators. Sometimes good results are obtained with only one input!

You may get somewhat better results from those, but in reality what you really need to do is probably one of the following, listed in order of likelihood:

1. 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. In our case we may have overlooked some excellent technical variable. The **Market Indicator Package** has many to choose from, and indicators can even be made from indicators. You may want to collect some fundamental variables, or even some market sentiment variables. Even your personal evaluation of the day's news might be another predictor you can use. Perhaps you have your own indicators, or you want to use some from other programs. NeuroShell 2 can "stand on the shoulders of giants" and predict even from the opinions of other experts.

You also may want to rearrange some existing inputs into ratios, e.g., advances/declines. 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.

2. Reevaluate what it is you want to predict. Some things are easier than others. You may get more accuracy predicting the change in price for tomorrow, since the range from min to max is smaller. You may also want to predict relative highs and relative lows.

3. Collect a better set of historical patterns, or a more representative test set. Make sure your variables are normalized if necessary. In the 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.

4. **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.