Successful Trading Using Artificial Intelligence

By Steve Ward and Marge Sherald

“Let your systems learn the wisdom of age and experience.”®

Table of Contents

Preface

We developed the trading techniques described in this book over a period of 20 years while assisting customers who own the artificial intelligence software developed by Ward Systems Group: NeuroShell 2 and NeuroShell Trader Professional. However, the techniques may be applied to any trading program that includes a genetic algorithm optimizer and neural network predictions.

Successful Trading with AI: Fact or Fiction?

"Apart from the U.S. Department of Defense, the financial services industry has invested more money in neural network research than any other industry or government body," according to a 1993 book edited by Robert R. Trippi and Efraim Turban[1]. Has the investment paid off?

Many say the late 1980's and early 1990's were a time of exaggerated claims for the technology that never quite panned out. Some traders who continued to use the technology didn't explicitly name it in their client presentations, preferring to use such terms as "statistical modeling techniques". Other traders publicly attribute their success to a mastery of artificial intelligence (AI) techniques.

For example, Andre Archambault, BA, JD, MBA, is the Director of Quantitative Strategies at Standard and Poor's Investment Services, and he is also the developer and genius behind Standard and Poor's extremely popular Neural Fair Value (NFV) System for ranking stocks and building portfolios. NFV, which uses neural nets made with NeuroShell, is among the more popular features of Standard & Poor's MarketScope, an electronic investment intelligence service for financial advisors, brokers, and money managers with approximately 77,000 subscribers worldwide.

His group is representative of similar departments at many of the Fortune 100 companies. However, don't think you need to have an entire department of Ph.D.'s to successfully use artificial intelligence techniques in trading. There are many successful individual traders who base their trading decisions on artificial intelligence models. They share some common traits: analytical thinking, persistence, and a willingness to experiment. They know there are no models that are correct every day but they can improve the number of winners by using neural network and genetic algorithm techniques. Use this book as a guideline for finding your own success.

Learn by Example

There are entire books that describe in detail the nuts and bolts of genetic algorithm optimization and neural network architectures, two of the primary components of AI. This isn't one of them. This book is more like a "how to" guide.

We'll begin with different methods for optimizing trading strategies with AI, including techniques that will generate better results in less time. We'll show you how to let a genetic algorithm play a small but potentially profitable role in managing your stock portfolio.

The neural network section will begin with a discussion of unoptimized predictions. Advanced techniques include how to optimize the buy/sell rules for predictions, select the best indicators, and create predictions based on a basket of stocks. Once you've created nets that are successful in their own right, we'll demonstrate how to combine multiple neural nets into hybrid models that can function as your own team of traders.

We'll end with building day trading models that specialize in trading at a certain time during the day (or night).

Trading Rules versus Neural Networks

When you first begin building a trading model, you might be confused about whether to start with trading rules or a neural network. A good rule of thumb is to use rules when you have some idea about how to construct a rule such as buy when the RSI is low and sell when the RSI is high. (You can let the genetic algorithm optimizer find the best values for low and high for the stock or futures contract you're modeling.) If you believe that a group of indicators are somehow indicative of a change in price for a stock or other trading instrument but you don't have any rules in mind, use those indicators in a neural network to make a prediction.

Optimization

Traditional Crossover Trading Rules

In this example we start building traditional models. Traditional models are those built with rules for buying and selling. We start with a type of system traders have used for years, the unoptimized moving average crossover. The theory is that when a short moving average crosses under a long one, you are starting a down trend, and when it crosses over the longer one, you are most likely starting into an up trend.

General Motors stock was getting hit badly starting in 2002, but the crossover system in this chart picked up the downtrends, as well as some brief uptrends, and made money. Crossover systems don't always pick up a new trend at the exact moment it starts, but if the trends last for a while, the system can be profitable. There are companies selling systems not much more sophisticated than this one for thousands of dollars!

In the chart displayed in Figure 1 we have inserted a 5 day and a 10 day moving average and labeled the short one Buy and the long one Sell, with appropriate colors. Whichever color is on top shows the current position you should be in.

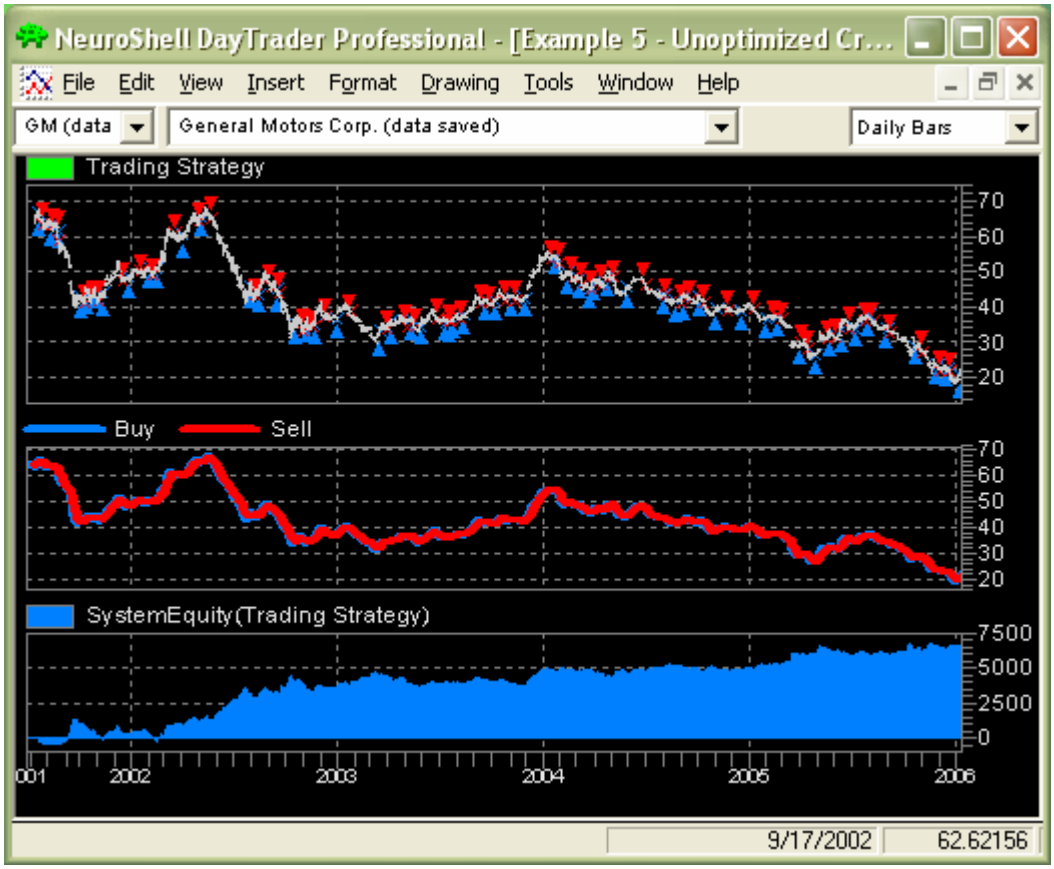


Figure 1: Traditional Trading Rules. This trading strategy uses a 5 and 10 day moving average crossover to generate buy/sell signals.

On the Trading Strategy subgraph in Figure 1 you can see the signals, blue triangle for long entry and red triangle for short entry (sell short) after a daily bar has ended. You will see the fill points (blue and red Xs) on the next morning's open. This Trading Strategy is built as a reversal system. When you exit a long position you immediately enter a short position and vice versa.

Reversals - You can actually accomplish reversals in a dropping market even with the stock uptick rule. Let's say you want a 100 share position. When you start out long, you go long 200 shares and short 100 shares. That's net 100 long. When you want to go short, you sell 200 shares. That's net short 100. When you want to go long again, you buy 200 shares, etc.

Optimization of Trading Rules

After reading about a traditional crossover trading system, you may be wondering if the crossover rules that we picked could be improved upon. That's where a genetic algorithm optimizer comes into play.

Compared to traditional optimizers which examine every possible solution, genetic algorithms are *much faster as the number of variables grows*. While the genetic algorithm searches throughout the entire range of possible solutions, it does so using some shortcuts based on evolutionary techniques much like selective breeding. When you consider the large number of variables involved in testing most trading systems, that time saving is essential to your success.

For more details on Genetic Algorithms, see Appendix A.

Using the NeuroShell Trader Professional, we can optimize the moving average periods to see if we can find periods that do better than the periods of 5 and 10.

When you optimize you can and should set appropriate ranges for each variable. The genetic algorithm optimizer will keep the variables within those ranges when it optimizes. Unlike traditional "exhaustive search" optimizers, you do not need an "increment" for each variable, because genetic algorithms do not search in increments.

In the example displayed in Figure 2, we wanted to make sure that the averaging periods optimize to the same values on the short entry as they did on the long entry. In other words, we didn't want the long entry rule to be a 3 period average crossing over a 10 period one, and the short rule to be a 6 period average crossing over a 12 period one. Not that there's anything wrong with that! It is just

that it would offend most people's sense of symmetry. Anyway, to accomplish the symmetry, we gave each range a name in the long rule so that the same name could be referenced in the short rule. The ranges were set from 2 to 20 in Figure 3.

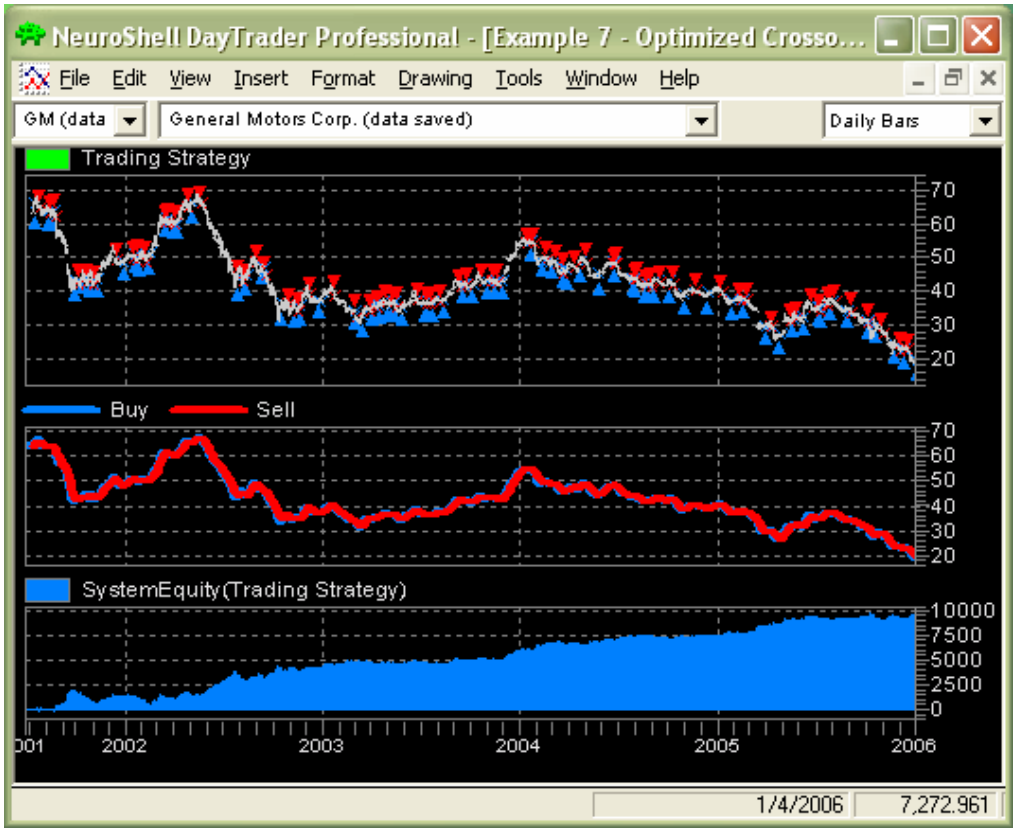


Figure 2: Optimized Trading Rules. The genetic algorithm found that a trading strategy with a 2 and 9 day moving average crossover generated additional profit.

The genetic algorithm tests the values it comes up with by computing a “fitness” or “objective” function. When building a trading system, the fitness function could be something such as "maximize return on account" or “maximize #winners - #losers”. Other suggestions for fitness functions include maximize net profit, maximize return on account times equity curve correction, minimize max drawdown, maximize average bar profit, maximize the ratio of gross profit/loss, maximize the Sharpe Ratio by trade, and so on.

For the GM model, the optimizer determined that the averaging periods of 2 and 9 were best over the time period in the chart.

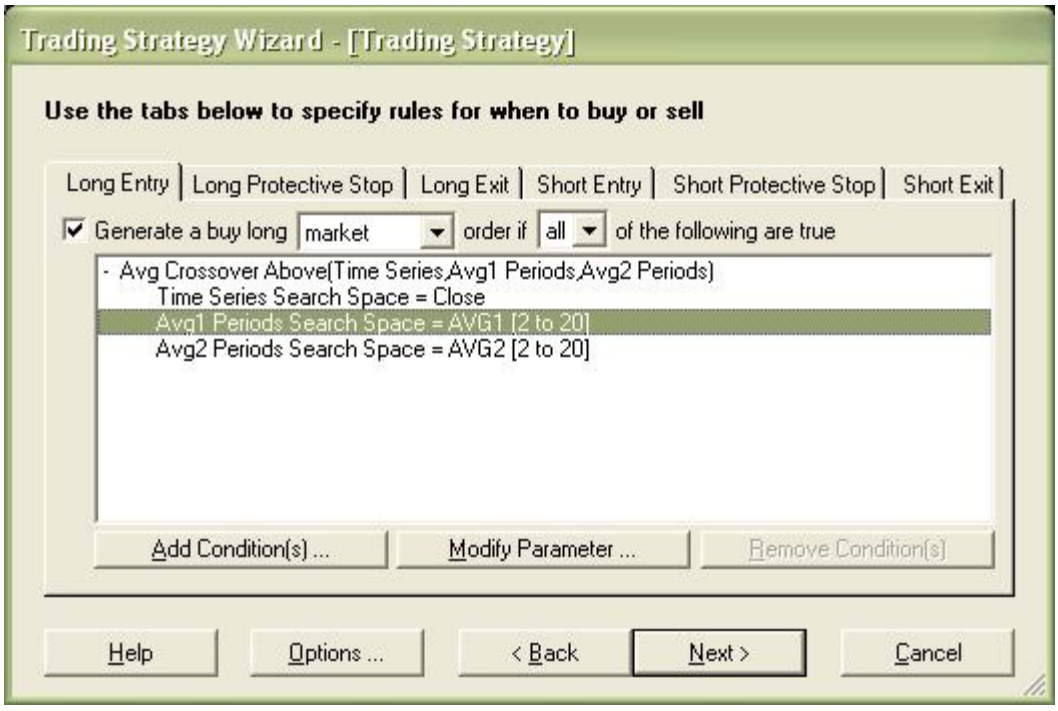


Figure 3. Setting the appropriate range for each part of the trading rule insures the most efficient use of the genetic algorithm optimizer.

Optimization with Evaluation

We're going to examine a modeling problem common to all model building with past data. Any time you optimize there is a possibility that you may "over-fit", which means that you build a model that worked so well in the past that it fit the past noise too, so that it probably will not work well into the future.

Some people use the word "curve fitting to describe the over-fitting phenomenon, but that is a misnomer because all modeling from past data is curve fitting - it is over curve fitting, i.e. or over-fitting - that is the problem.

Over-fitting can occur even if you do not optimize, because in fact when you backtest different strategies to see which works the best, you are in fact optimizing manually! However, machine optimizing increases the odds of over-fitting because it is so much more efficient. **The possibility of over-fitting is reduced by not optimizing, optimizing over plenty of historical data, and/or optimizing as few parameters as possible.**

Test for Over-fitting

One way that has been traditionally used to test if over-fitting has occurred is simply to see how the model holds up on new data. There are a couple of ways that can be done:

1. Of course you could simply watch your model for several weeks or months and "paper trade" it in the traditional sense to see if you would have made money. Few of us want to take the time to do that, however.

2. Another way to do this is to optimize a model and then test it on later data. Optimization will take place on an earlier period of the chart, and the final backtesting (evaluation) using the model will take place immediately after that. The disadvantage of this "hold out" approach is that even if you are satisfied the model held up during evaluation, you essentially have an old model, one that was not built on the latest data (although it was evaluated on the latest data). This approach is demonstrated in Figure 4, which builds a model based on the earliest data in the chart and tests that model on the last year of data in the chart.

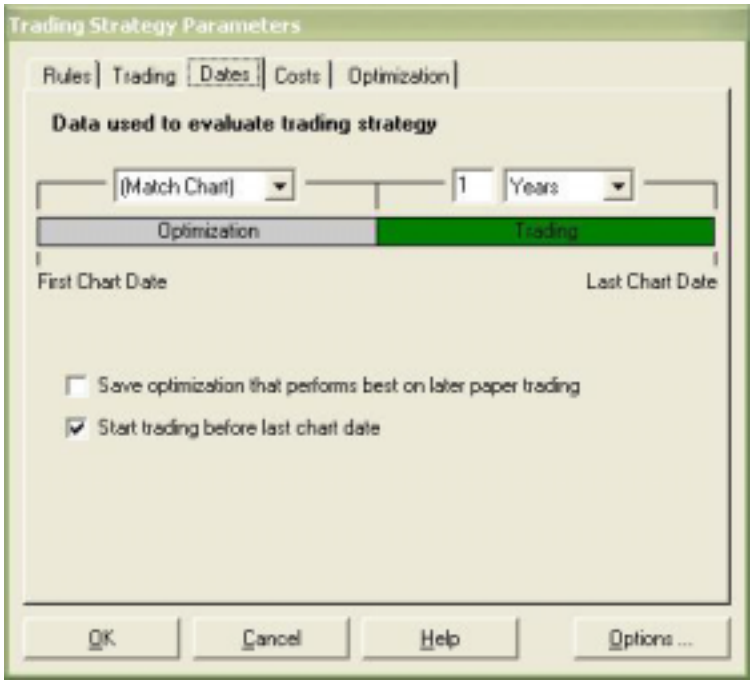


Figure 4. As indicated by the Dates tab, this model will be created on the earliest data in the chart and tested on data from the last year in the chart.

In the chart displayed in Figure 5, we have continued the crossover example. Recall that in the last example we optimized over all the data that was available at the time. In this example, we optimize over all data EXCEPT the data starting in 2005. Then we allowed the backtest on data starting with 2005 (the green line on the Trading Strategy). Note that the trading rules (averaging periods) found prior to 2005 did not hold up well starting with 2005. It's nice to know this in advance! We'll continue to work on that problem in the next example.

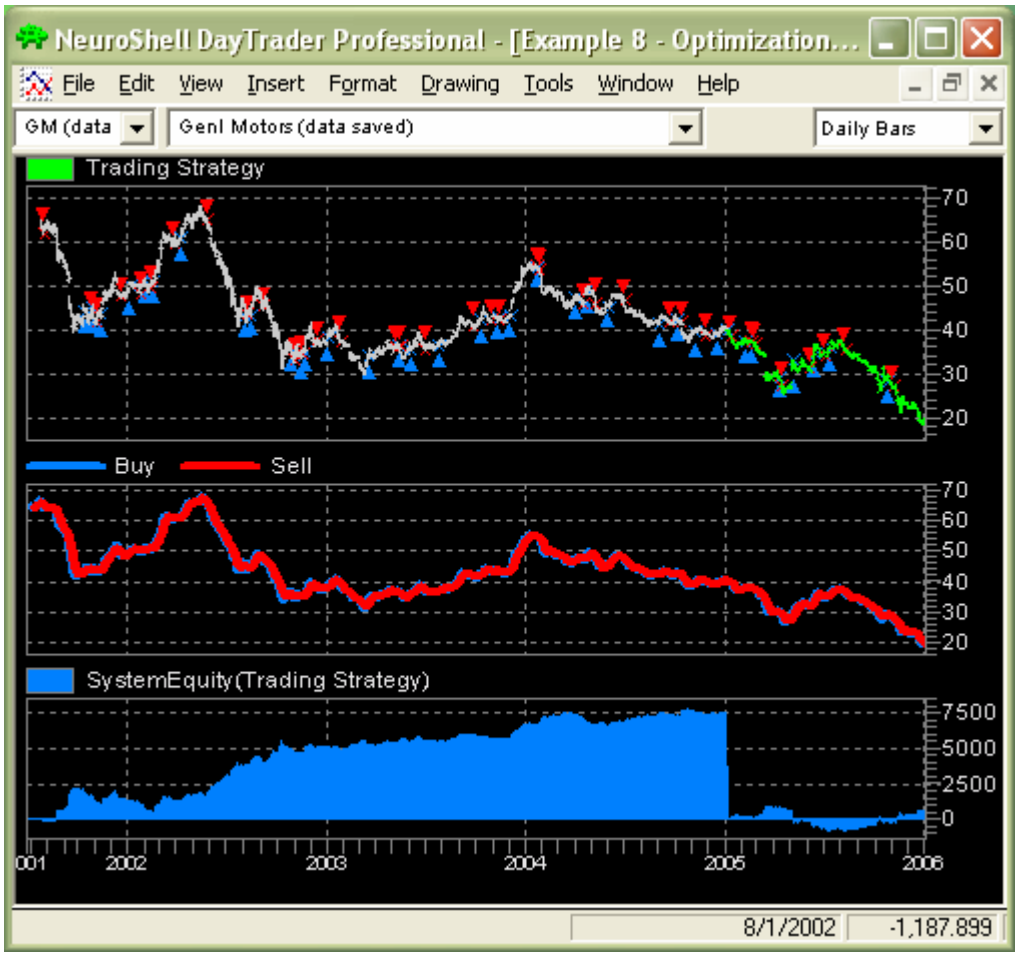


Figure 5: Preventing Overfitting. The genetic algorithm optimized the crossover periods until the end of 2005 data. The out-of-sample signals are shown by the green line in the Trading Strategy.

Optimization with Paper Trading

Let's consider the last example further, where we optimized in an earlier period and evaluated on a later period. Using the evaluation of new data (called out-of-sample data by statisticians), what you will wind up doing is repeatedly optimizing models until you find the one that works the best on the new data, i.e. the one that shows up best during the evaluation (out-of-sample period). This is sometimes called "data snooping" because your evaluation data is not really out-of-sample anymore. Nevertheless it is still the most effective method to arrive at a model which has the best likelihood of holding up as you trade with it in the future.

The NeuroShell Trader automated this process of building models, then evaluating them, keeping the one that works best during evaluation.

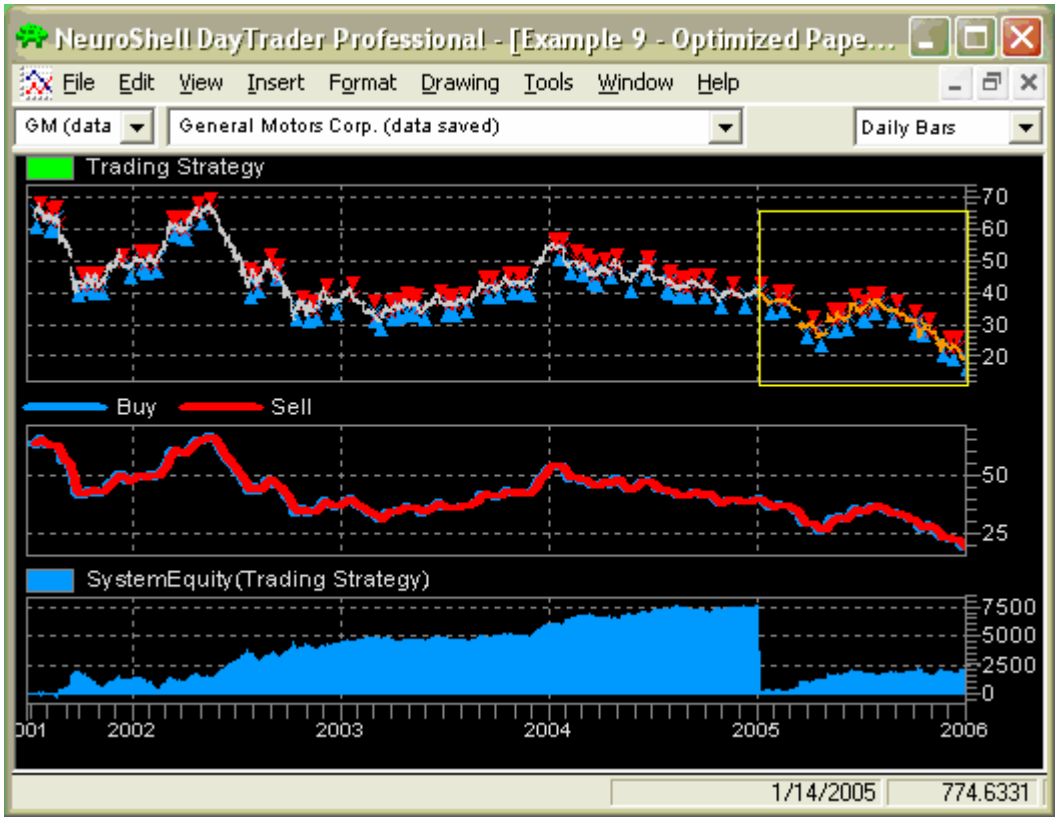


Figure 6. This chart displays the buy/sell signals for a moving average crossover system that was optimized on the period beginning in 2005 (outlined in yellow on the graph).

In the chart displayed in Figure 6 we have done just that, creating what we call in NeuroShell a paper trading period outlined in yellow on the graph.

We optimized on data before 2005, the goal being to find the best profit before 2005. However we kept the combination which worked best starting 2005, not the combination which worked best before 2005. In other words, each time the

genetic algorithm found a new combination of averaging periods, the profit was tested both for the period before 2005 and the period starting in 2005. NeuroShell remembered the best combination starting in 2005 and left us with that combination, even though it may not have been best before 2005.

In the case of the example chart, the optimizer found the same settings for paper trading (2 and 9) that it did for optimizing over the whole chart, but that is not always the case. The paper trading feature is designed to prevent optimization from over-fitting, and finding a solution which works the best in the future.

Of course if you data snoop, statisticians will say that you still haven't properly evaluated your model with real out-of-sample data. Of course statisticians usually assume normal distributions, and a number of other factors not present in market trading (see *The (Mis) Beahvior of Markets* by Mandelbrot and Hudson, references).

However, if you want to build your model with paper trading and still satisfy the statistician in you, you can select both of the options:

"Save optimization which performs best on later paper trading"

"Start trading before last chart date"

That will enable saving the model which works best on paper trading, while still giving you a real out-of-sample period after the paper trading period. The disadvantage is that you have an old model, at least as old as the out-of-sample trading period. Given that the market is frequently changing, and we suspect the number of statisticians who got rich in the market is quite small, we suggest that you consider using the paper trading feature without the added out-of-sample period.

Predictions

Unoptimized Prediction for FOREX

In this example we introduce neural network predictions, and we use a FOREX instrument since they are so popular these days. However, the techniques explained herein are just as applicable to stocks, commodities, options, etc.

Neural nets are useful when you have some an idea of which indicators might be predictive of market movements, but you don't know how to create a set of rules that will generate trading signals.

Neural nets make predictions (the output) about the future value of some data series based on some indicators that you feed into them (inputs). The neural net uses some historical data (the training set) to "learn" how to make accurate predictions of the output using the inputs. NeuroShell teaches the nets how to make those predictions, a process called training. You do not need to know how neural nets do this learning in order to use them effectively, but **Appendix B, How a Neural Network Works**, will explain the math behind neural net training if you feel you need to know. Just remember that there are many, many types of neural nets, and something you study about one type may not be applicable to other types, like ours.

After the net is trained NeuroShell applies threshold rules to the predictions to decide whether to buy or sell. The Prediction Wizard not only trains the neural net, but applies the threshold rules for you to create a simplified trading strategy.

For example, if the neural net is predicting the percent change in open over the next 3 days starting with tomorrow's open, some of the simplest rules (and the ones used in this example) might be:

If prediction > 0% then enter long

If prediction < 0% then exit long

If prediction < 0% then enter short

If prediction > 0% then exit short

In the FOREX example displayed in Figure 7 we are using daily Euro/dollar pair bars. We know that regression lines (straight lines placed through past closes) are often somewhat predictive of the future if we measure the slope of such lines. Positive slopes mean rising prices, negative ones mean falling prices. In the chart we inserted the slope measurements of straight lines each 10 bars long. One slope is of the line through the most recent 10 closes, the next through the 10 bars before that, etc. The straight lines aren't actually plotted on the chart; it is the slopes we need. The indicator we used to measure the slope of an (unplotted) regression line is displayed on the chart in purple.

Then we took 10, 20, and 30 bar lags of the indicator in purple. This measures the 10 bar slopes of the curve 10, 20 and 30 bars back.

Through hours of inspection we might be able to figure out some buy/sell rules based on these past slopes to put into a trading strategy, but why bother? Instead we put them into a neural net prediction. Here are some points to notice:

1. We are predicting the percent change in open 3 bars from the next open
2. We did not optimize the 4 inputs
3. We built the neural model using the whole chart as a training set
4. We selected our own threshold rules (as shown previously)
5. We defaulted to 10 "hidden neurons". Hidden neurons are a technical aspect of nets. The main thing you need to know about them is the more you use the tighter the curve fitting will become.

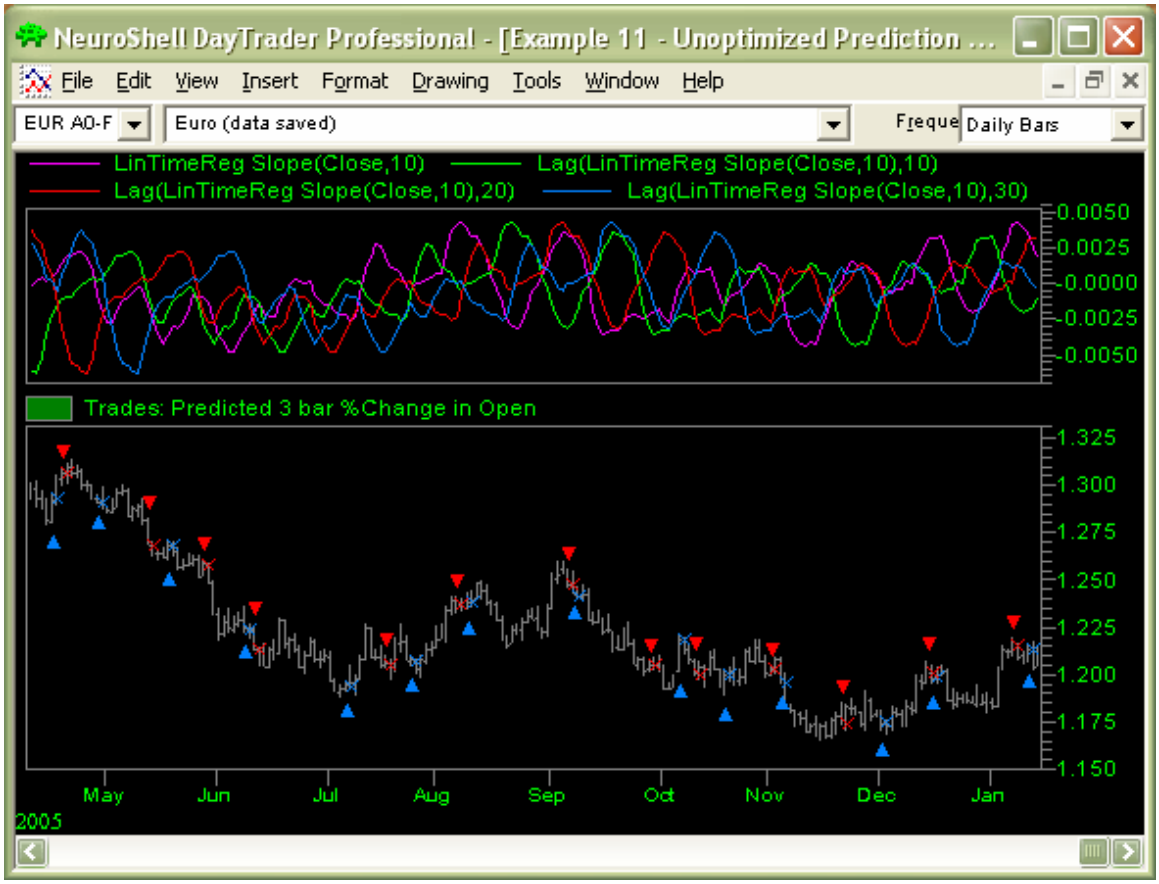


Figure 7. This graph displays the buy/sell signals generated by a neural network designed to predict the 3 bar percent change in the open. The network used a linear time regression slope of the close and several different lags of that indicator as inputs to the neural network.

Now here are some cost related aspects to note:

1. We assumed one point is worth $100,000

2. We assumed commissions of $30 per side (many brokers don't charge a commission, but the spread can be several pips)

3. We assumed 1% margin

The result was a continuous rise in the equity curve based on the signals generated by the neural net.

Unoptimized Prediction of Stocks

The example in Figure 8 is just like the last one, the FOREX example, except that we have modeled the nine Select Sector SPDRs (ETFs) instead of the Euro. The neural network treatment is the same as the FOREX example. Each stock has its own neural net. We've also changed the costs to $10 per side per trade, and we are buying $10,000 worth of each stock with each trade.

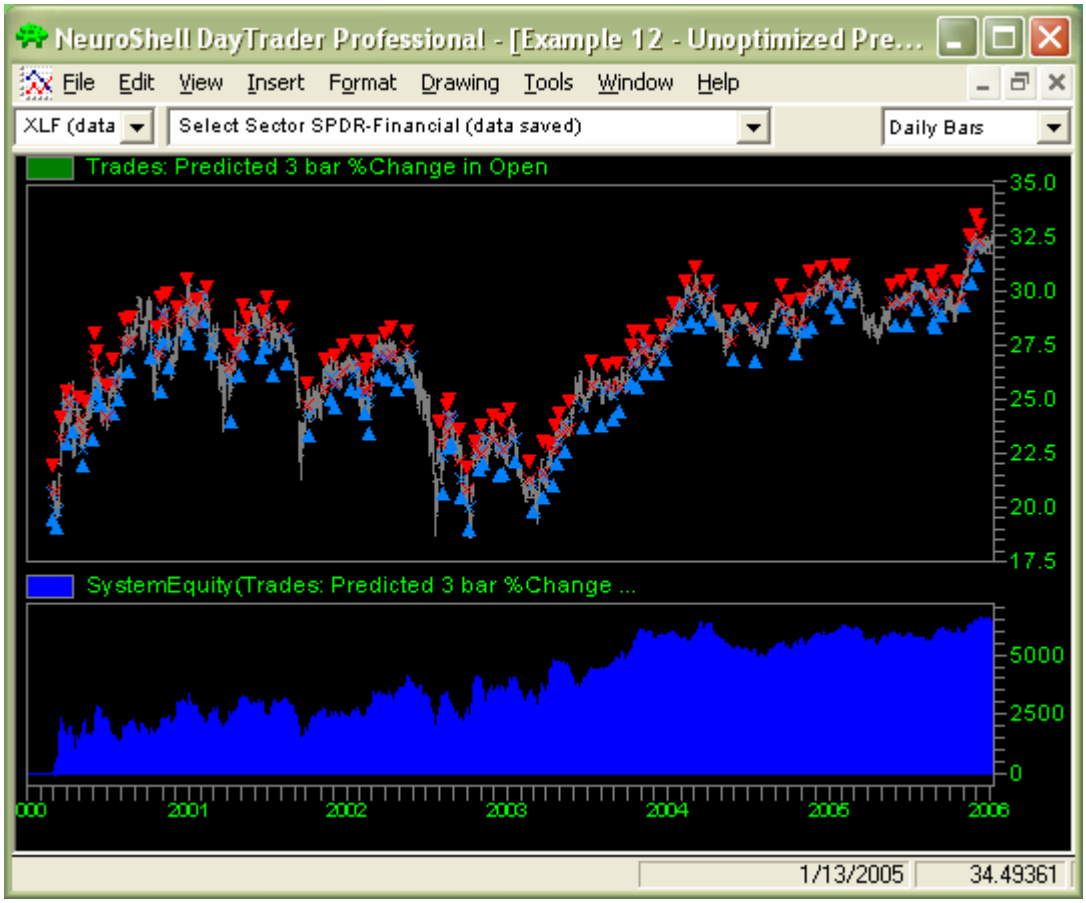


Figure 8. This graph displays the buy/sell signals generated by a neural network designed to predict the 3 bar percent change in open for nine different ETFs. This model for the SPDR financial sector generated an equity curve that increased overall despite a few dips along the way.

Optimum Prediction Thresholds

Remember the Unoptimized Prediction FOREX example where the buy/sell signals were based on the following:

If prediction > 0% then enter long

If prediction < 0% then exit long

If prediction < 0% then enter short

If prediction > 0% then exit short

The 0% change included in the rules above was simply using the default value. But what if different percentages yielded higher profits? This is once again where optimization can improve model performance.

Figure 9. This neural network prediction optimizes the thresholds for the buy/sell decisions. The bottom green subgraph displays the sum of the equity curves for all of the sectors modeled in the chart. This is useful in monitoring an entire portfolio.

The neural network prediction displayed in Figure 9 used a linear time regression slope of the close and several different lags of that indicator as inputs to the neural network. The optimizer found different threshold values for each of the different ETFs (Exchange Traded Funds) sector models (see Figure 10).

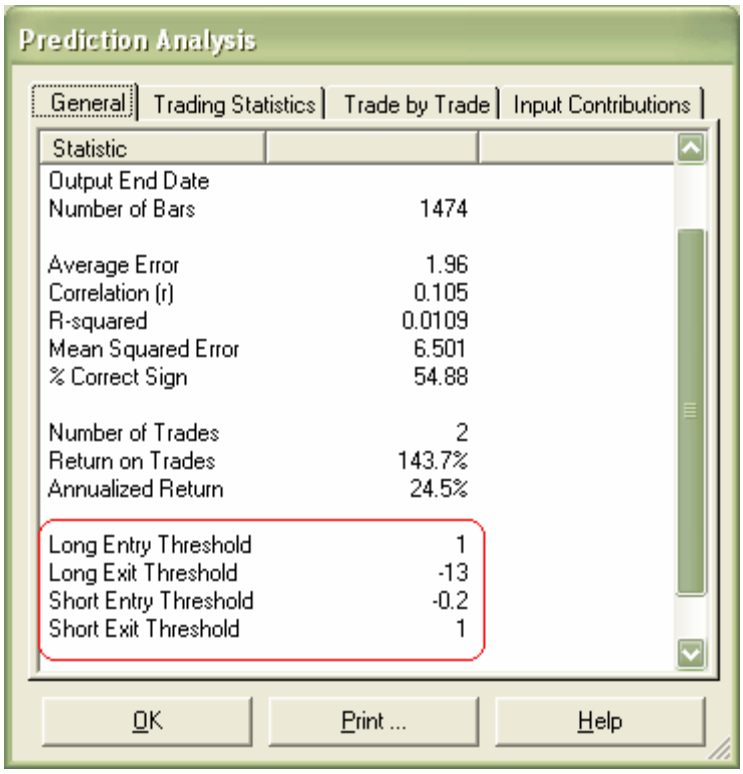


Figure 10. The optimizer found the thresholds outlined in orange above for the SPDRs financial sector. The optimizer generates individual trading thresholds for the entry and exit rules for each sector in the chart.

Optimum Prediction Based on Paper Trading

The previous neural network example based its buy/sell decisions on thresholds that were optimized across all of the data in the chart. This example introduces the concept of saving a model that shows the best results on a separate set of data called the “Paper Trading” set.

The example displayed in Figure 11 is like the previous one, except for two things:

We set the optimizer to optimize for a fixed time (2 minutes) on each stock (see Figure 12). Remember we still are not optimizing inputs, only thresholds

We held out 6 months for a paper trading period, and instructed NeuroShell to keep the model that worked best on the paper trading period (see Figure 13). Using this paper trading feature, we feel more confident trading this model into the future.

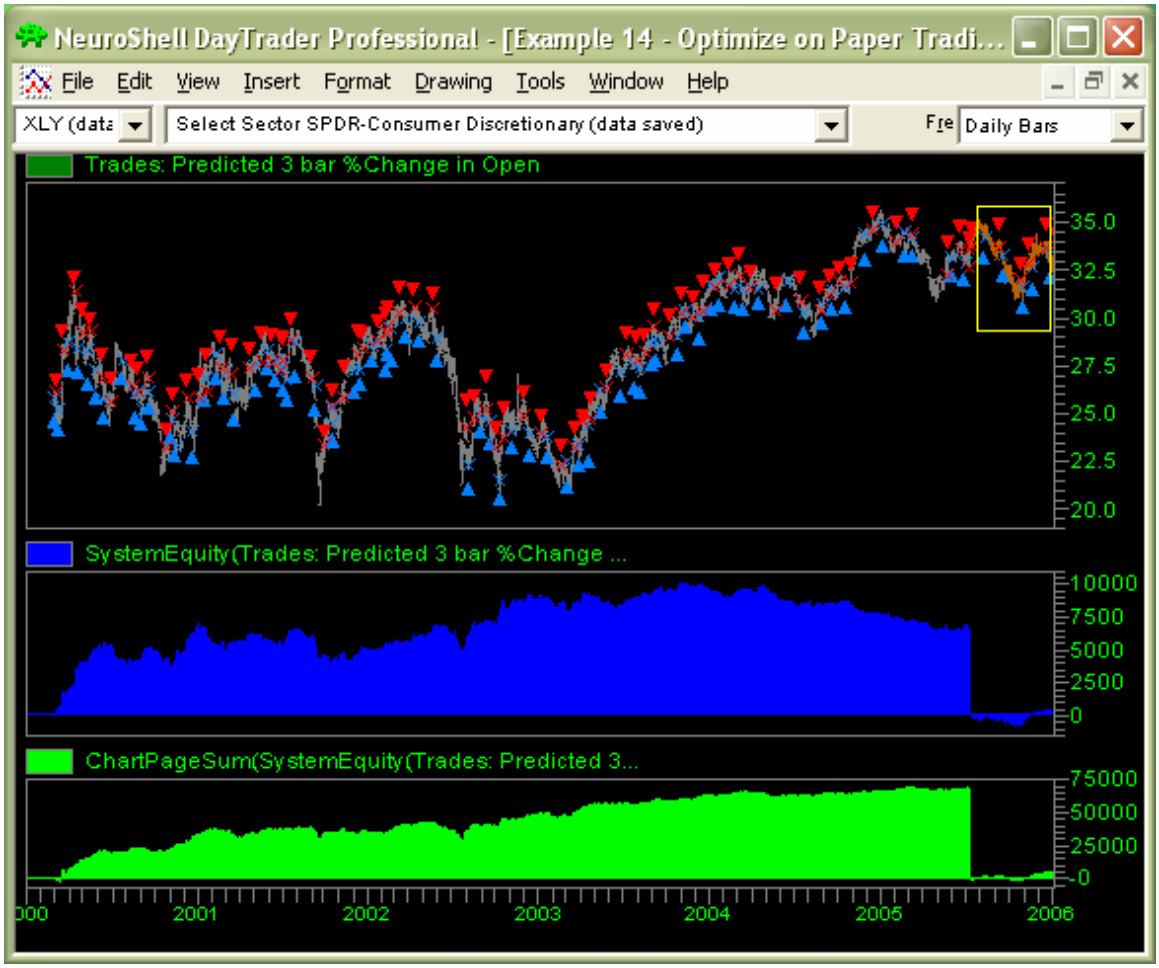


Figure 11. The final version of this neural network model was saved based on how well the model performed on a separate "paper trading" set of data (outlined in the yellow box). Note that the equity indicators displayed on the chart are reset for the paper trading period.

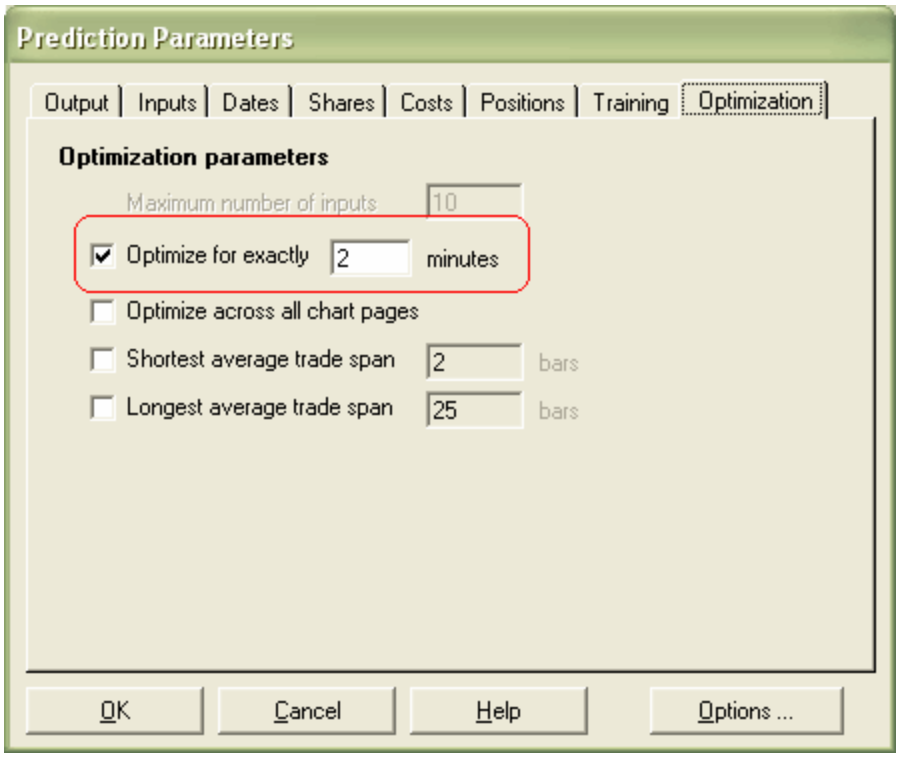


Figure 12. The optimizer is set to only spend two minutes on each issue in the chart in order to find the best threshold for each trading rule.

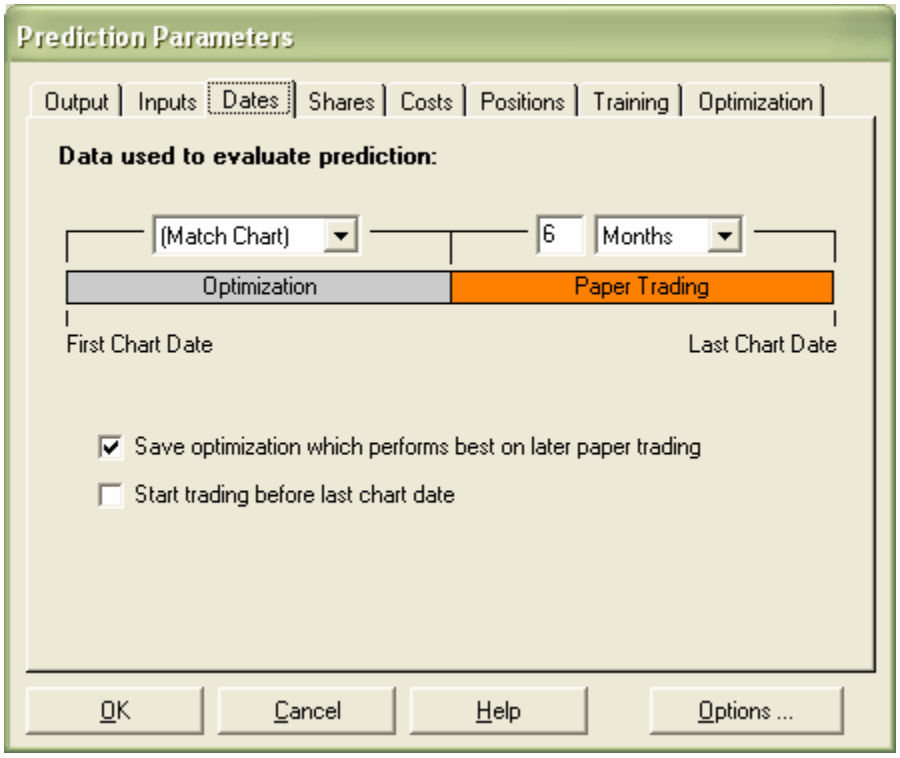


Figure 13. The dates tab allows you to specify the amount of data used for the paper trading period.

Optimum Prediction Based on Multiple Stocks

One technique you can use to keep the optimizer from doing its job too well (otherwise known as overfitting) is to let the optimizer pick the thresholds that work best on a basket of stocks rather than a single stock.

You can accomplish this in the NeuroShell Trader simply by selecting the option to optimize across all chart pages. For example, you could build a prediction in a chart for Coca Cola Co., and then add additional chart pages for American Express Co., Wal-Mart Stores Inc., Wells Fargo & Co., and Anheuser-Busch. The result should be a robust model that predicts well into the future.

Other ways to apply this technique are to build sector models such as a chart that includes the stocks that make up the Dow Jones Transportation Average Index or the 30 stocks that make up the Dow Jones Industrial Average.

Input Selection

One advantage of using neural networks to build successful trading models is that the nets themselves rank the independent variables to the models.

For example, we hypothesized that indicators based on some chip makers might be good inputs to a neural net to predict the change in the stock price for several computer manufacturers.

For the Dell model, the RSI and MACD Signal for Dell's close were included in the network as well as the RSI and MACD Signal for Intel and Motorola. In the NeuroShell Trader we selected the "Input selection" optimization. We wanted the optimizer to find which of our inputs were most predictive. In order to help prevent over-fitting, we specified that we wanted no more than 4 inputs maximum. The contribution factor for each of the inputs is listed in Figure 14. The inputs without a contribution factor were eliminated by the genetic algorithm optimizer.

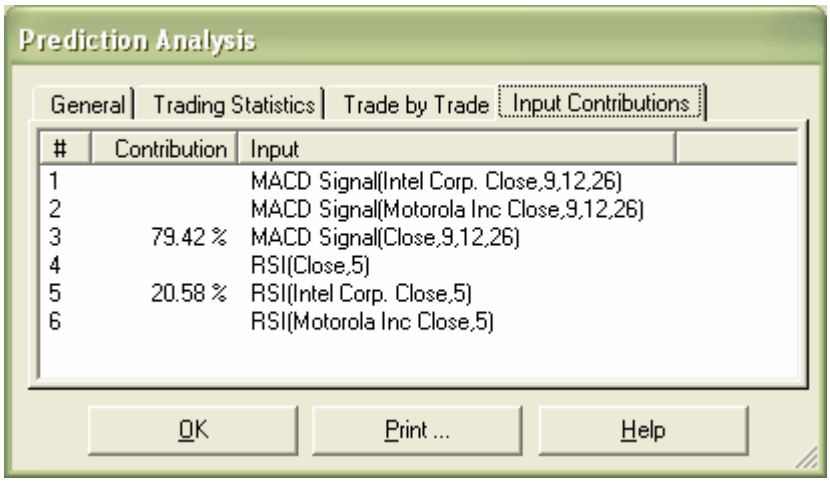


Figure 14. Neural network predictions can rank the importance of each input to the model's effectiveness.

Portfolio Management

Portfolio Management Overview

Many traders have a prescreened basket of stocks which they trade. The question is which stocks should be traded at any given time to maximize the amount of return. NeuroShell has some powerful portfolio management capabilities when you make use of the indicators in the Chart Page Indicators category. We will explore just one of the many possibilities in this example.

In our portfolio management example we will build a portfolio trading strategy based upon the RSI indicator, but obviously any indicator or neural net prediction signal could be used instead. The idea is that the trading strategy buys stocks in the chart that have the lowest RSI values, and sells those same stocks when the RSI is no longer among the lowest in the chart.

For such a determination we use the indicator Chart Page Lower Rank. If you inspect the portfolio details at the bottom of the chart you will see that a stock has a rank of 1 whenever the RSI is the lowest of all stocks in the portfolio. Therefore if we buy whenever the Chart Page Lower Rank is < 2, which we did, we will be buying the stock when its RSI is the lowest of all in the chart. Note that if we had used Rank < 3 we would be buying two stocks at once.

Our exit condition is simple, too. We exit when the Rank is > 3, in other words when at least three stocks have higher rank (lower RSIs).

If you examine the portfolio view at the bottom of Figure 15, you will see that on average there are one to three stocks in a long position at any given time.

To top everything off, we used the chart page sum indicator to sum the equity curves of all stocks for both the trading strategy and the buy and hold strategy. It should be noted that there are several other Chart Page indicators that can be used in place of the Lower rank, such as Percentile for instance.

Figure 15. Chart page calculations allow you to pick and choose which stocks, futures, etc. to include in your portfolio based upon ranking standard indicators.

Portfolio Management Long and Short

This example is like the previous Portfolio Management example, except that we used the Chart Page Upper rank to make short entries and exits, as well as the Chart Page Lower rank to make long entries and exits (as we did in the previous model). Frankly, we could have used just one of the rank indicators since they are symmetric and opposite.

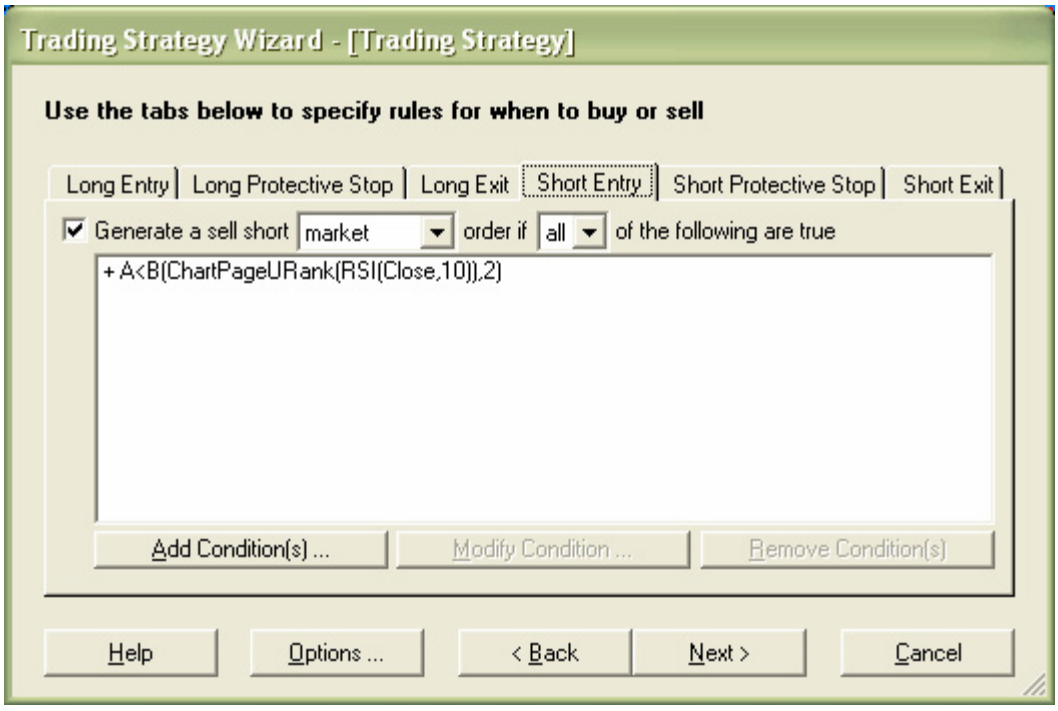


Figure 16. In this example, the short entry tab above displays the rule that sells short when the upper rank of the RSI is less than 2, in other words sell short when a stock has the highest RSI value compared to the RSI values of the other stocks in the portfolio.

Hedged Portfolio

It only took a slight adjustment to turn the previous example into a nicely hedged system. At all times the system is long one stock and short one stock. Since the strategy always invests $10,000 in each stock, the hedge is maintained on a dollar basis too, which would not have happened if we were always buying and selling a fixed number of shares. All we did was to adjust the rules so that an exit occurs as soon as the rank drops below one. We called that strategy Hedge1 because there is always one long and one short.

The Trading Strategy enters the long position for one of the issues in the portfolio when it reaches the lowest rank for the RSI indicator compared to all of the other issues in the portfolio. The long position is exited as soon as the issue is no longer the lowest in the portfolio according to RSI value.

The Trading Strategy enters a short position for one of the issues in the portfolio when it reaches the upper rank compared to all of the other issues. The short position is exited as soon as the issue is no longer the top one in the portfolio according to RSI value.

To prove that we could trade more than one long and one short stock at a time, we built Hedge 2, which always has two stocks long and two short. Selections are based on the top two upper and lower RSI values. The number of stocks

traded at any one time could be adjusted up to the number of stocks in the portfolio.

Figure 17. The Hedge 1 Trading Strategy is always trades one long and one short position based on the lowest and highest RSI values.

Obviously the best use of such a hedging strategy would be to pick a portfolio of stocks in the same sector, say, so that they are likely to move together when news hits.

You might be asking the question of what role does artificial intelligence play in creating the hedged portfolio trading strategy described above. The answer is a small but potentially profitable one. You would probably not want the genetic algorithm to adjust the number of stocks you trade at any given time (unless you currently have a lot of money to trade). But perhaps you would like to adjust the number of time periods involved in the RSI calculation (or another indicator of your choice) or use neural nets instead of rules. That's the competitive edge you can gain from using artificial intelligence tools.

Panel of Experts

E-minis Combined Nets Hybrid

It is possible to include predictions (neural nets) in a trading strategy. You could to this to enhance the simplified trading strategy that is created when building the neural net (prediction > threshold). Some reasons for doing this include:

1. Making a strategy with a neural net that has added trailing stops.
2. Making a trading strategy that uses several neural nets to make a decision (called ensemble nets in the literature).
3. Incorporating rules with the neural net for a hybrid strategy.

In this example we examine both number 2 and number 3 above by making a strategy with three neural networks and a traditional rule. Our strategy makes entries and exits when two of the four conditions (3 nets and a rule) are true.

Figure 18. The Trading Strategy displayed in the bottom subgraph combines signals generated by combining three neural networks and a traditional trading rule.

The inputs to the three predictions where not optimized, but we did optimize the threshold (trading) rules. Of course, we could have optimized the predictions if we had wanted to.

Next we'll examine the trading strategy to see how we inserted the completed nets. The NeuroShell Trader's Prediction Wizard makes these long and short entry and exit rules just so that we can insert them into a trading strategy.

In our trading strategy we used the venerable RSI rule. Then we optimized the trading strategy. In this case the combine strategy out-performed the individual nets, which is what we hoped for in combining them.

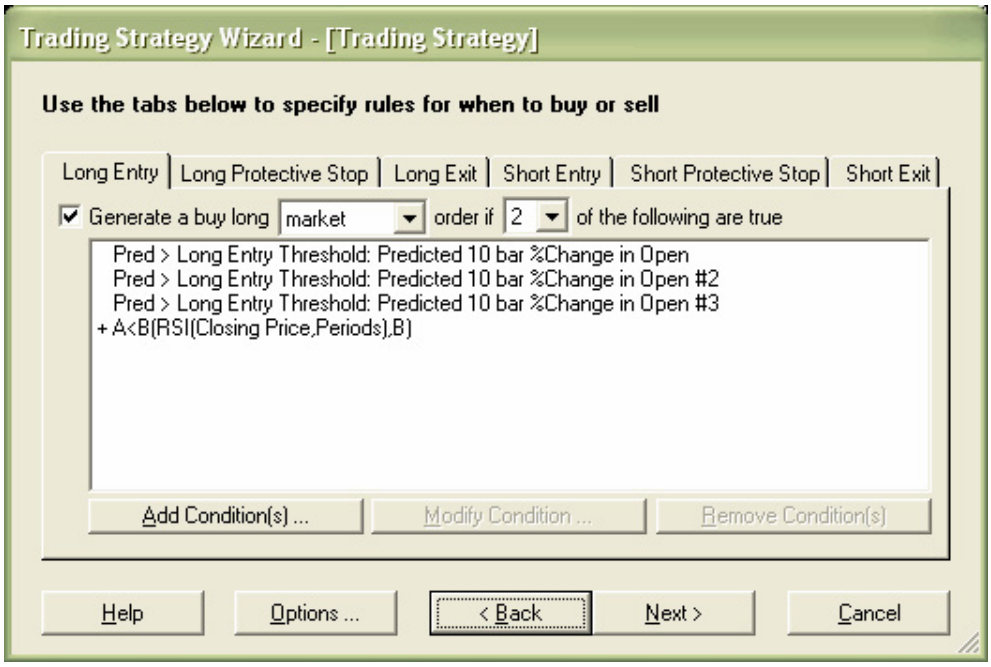


Figure 19. A long entry signal is generated if two of the listed conditions are true.

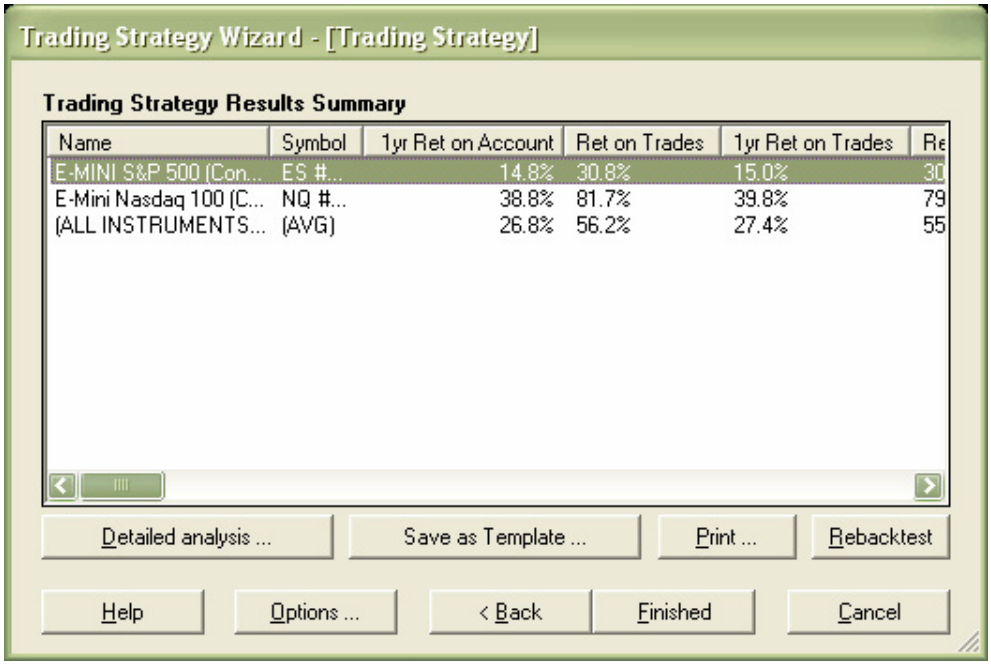


Figure 20. The hybrid trading strategy of three neural nets and a trading rule outperformed a single neural net.

In this example we used continuous futures contracts where the historical contracts are "glued" together to make a long running contract that we can use when we want plenty of historical data. We used eSignal® data to load the NASDAQ and S&P 500 E-minis, but other data suppliers have similar contracts with different symbol nomenclature. We did not bother to set the point values, because we were more interested in finding correct peaks and valleys than in correct profit calculations.

DayTrading – Models for Specific Times

Many people believe, as do we, that institutions fire their program trading operations at specific times of the day. Either that or the professional traders finish coffee and lunch breaks and decide to trade about the same time every day. Therefore, we like to build models that trade only during specific high volume time intervals. If you can predict which way the high volume trading is going to go, you can move to Palm Beach and trade.

In this chart we built a trading strategy that decides whether to go long or short at 9:55 am eastern time each day. The exit is always at 10:30. We used the familiar regression slopes from earlier examples. When you optimize such a strategy, the optimizer evolves a model which is only concerned about the rules as they affect the 10-10:30 timeframe as long as you don't optimize the time too. That is exactly what we did. Note that you can have many such models for different time periods, including the overnight gap.

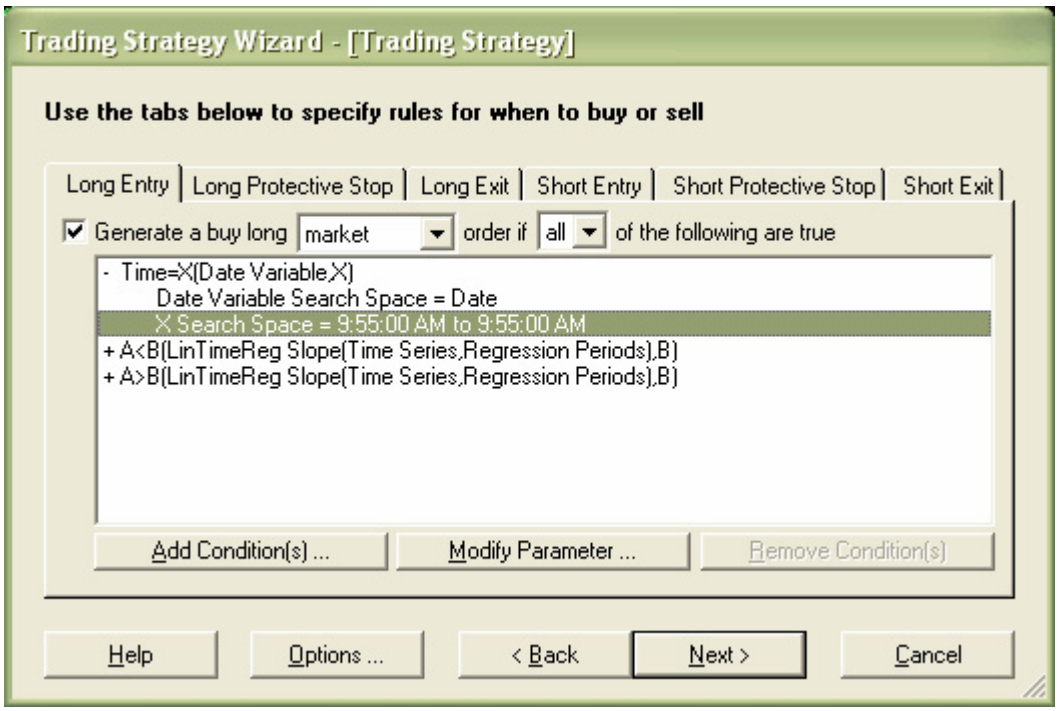


Figure 21. The long entry rule lists the time range from 9:55 am to 9:55 am so the value is not changed by the optimizer.

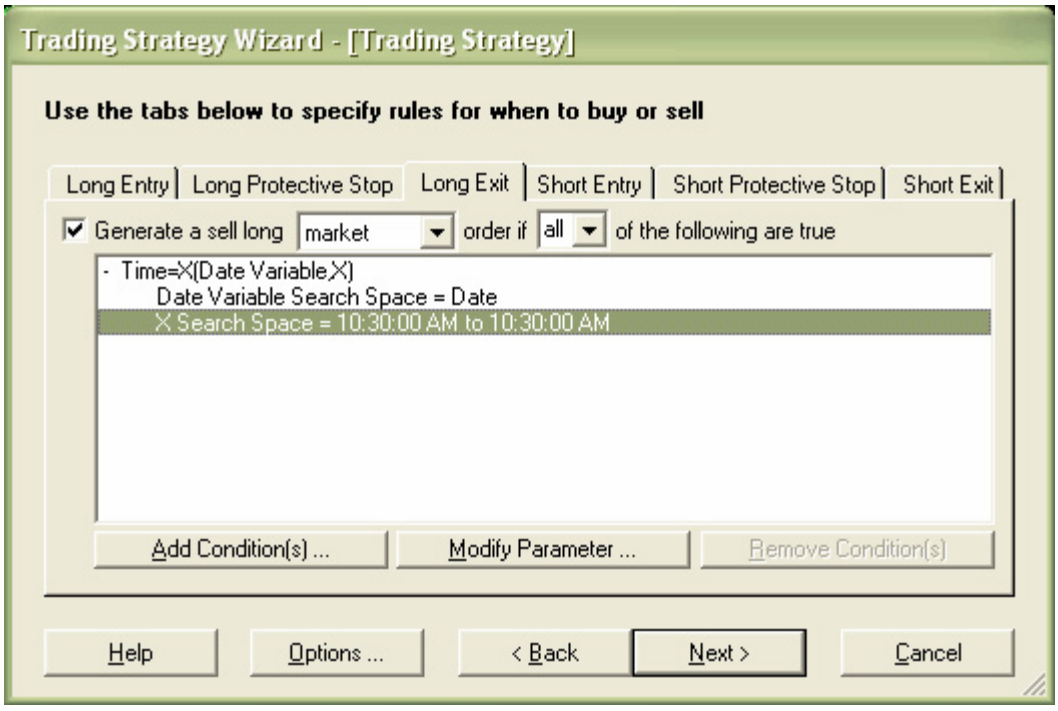


Figure 22. The single long exit rule gets out of the trade at 10:30 am.

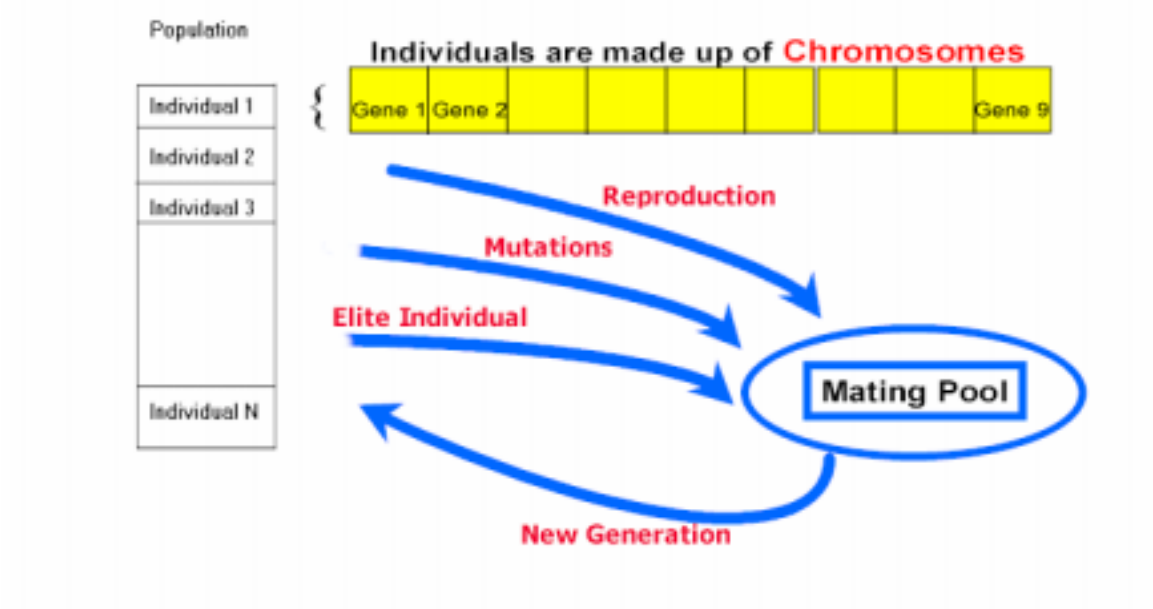
Often we use QQQQ as a proxy for continuous NQ futures data, and SPY as a proxy for continuous ES data. Frequently, NQ moves as QQQQ moves etc so it becomes profitable to daytrade these time intervals because you can use highly leveraged Eminis instead of stocks.

Appendix A

How a Genetic Algorithm Works

A genetic algorithm solves optimization problems by creating a population or group of possible solutions to the problem. The individuals in this population will carry chromosomes that are the values of variables of the problem, such as the number of periods in a moving average.

The genetic algorithm actually solves your problem by allowing the less fit individuals in the population to die (peacefully) and selectively breeding the most fit individuals (the ones that solve the problem best by making the most profit, for example). This process is called selection as in selection of the fittest. The genetic algorithm will take two fit individuals and mate them (a process called crossover that takes some genes from the mother and others from the father). The offspring of the mated pair will receive some of the characteristics of the mother, and some of the father.



In nature, offspring often have some slight abnormalities, called mutations. Usually these mutations are disabling and inhibit the ability of the children to survive, but once in a while they improve the fitness of the individual (like toes stuck together in a web-like fashion). The genetic algorithm similarly occasionally causes mutations in its populations by randomly changing the value of a variable.

The children, whether they were created from two parents or a mutation, are then tested and an overall score for the population is calculated. In financial

applications, this test can determine the net profit or number of winning trades. Different types of tests are often called objective functions.

After testing, the population undergoes a generation change. The population will then consist of offspring plus a few of the older individuals, which the genetic algorithm allows to survive to the next generation because they are the most fit in the population, and we will want to keep them breeding. These most fit individuals are called elite individuals.

After dozens or even hundreds of "generations", a population eventually emerges wherein the individuals will solve the problem very well. In fact, the most fit (elite) individual will be an optimum or close to optimum solution.

The processes of selection, crossover, and mutation are called genetic operators.

Appendix B

How a Neural Network Works

Neural network technology mimics the brain's own problem solving process. Just as humans apply knowledge gained from past experience to new problems or situations, a neural network takes previously solved examples to build a system of "neurons" that makes new decisions, classifications, and forecasts.

Neural networks look for patterns in training sets of data, learn these patterns, and develop the ability to correctly classify new patterns or to make forecasts and predictions. Neural networks excel at problem diagnosis, decision making, prediction, classification, and other problems where pattern recognition is important and precise computational answers are not required.

The training data includes many sets of input variables (usually indicators) and a corresponding output variable (such as the percent change in open). If you're familiar with statistics, the inputs are often called independent variables and the output (prediction) is called the dependent variable. Each set of corresponding independent variables and dependent variable is called an observation, example, or case. In more general terms, when the neural network trains, it is using historical examples to "learn" the patterns of the input variables and how they correlate to the output variable (prediction).

The range of historical examples ("training set") used to train the network should include a representative set of problems likely to be encountered in the real world. For example, if you want to predict the selling price of a stock, you need to make sure your training set includes historical examples of when the price went up, when it went down, and when it stayed the same.

You will want to provide historical examples that are relevant to predicting the current market and avoid historical examples that do not represent current market behavior. The best way to do this is to limit the amount of past history upon which the neural network trains ("learns"). This means limiting the number of rows to between 300 and 2000 so that the neural network will learn data "relevant" to today's market. On the other hand, you don't want to use too few bars or overfitting may occur.

Additionally, it is important to understand that for each observation all the inputs must exist for the neural network to use that observation to "learn". If you are trying to predict a U.S. security and using an input that comes from a foreign market that has a holiday on a day that you are expecting an observation, then the observation will be missing data. Thus the neural network will be unable to use this observation for "learning" or predicting its value.

Finally, because of inflation and an overall rise in most markets, normalizing inputs and outputs over time (detrending) is extremely important. Most technical indicators do this automatically.

The results that you achieve will only be as good as the training data (inputs and outputs) that you select.

How Does a Neural Network Learn?

The network begins by finding linear relationships between the inputs and the output. Weight values are assigned to the links between the input and output neurons. After those relationships are found, neurons are added to the hidden layer so that nonlinear relationships can be found. Input values in the first layer are multiplied by the weights and passed to the second (hidden) layer. Neurons in the hidden layer "fire" or produce outputs that are based upon the sum of the weighted values passed to them. The hidden layer passes values to the output layer in the same fashion, and the output layer produces the desired results (predictions).

The network "learns" by adjusting the interconnection weights between layers. The answers the network is producing are repeatedly compared with the correct answers, and each time the connecting weights are adjusted slightly in the direction of the correct answers. Additional hidden neurons are added as necessary to capture features in the data set.

Eventually, if the problem can be learned, a stable set of weights evolves and will produce good answers for all of the sample decisions or predictions. The real power of neural networks is evident when the trained network is able to produce good results for data that the network has never "seen" before.

Network Structure

The basic building block of neural network technology is the simulated neuron (depicted in Figure 1 as a circle). Independent neurons are of little use, however, unless they are interconnected in a network of neurons. The network processes a number of inputs from the outside world to produce an output, the network's predictions. The neurons are connected by weights, (depicted as lines).

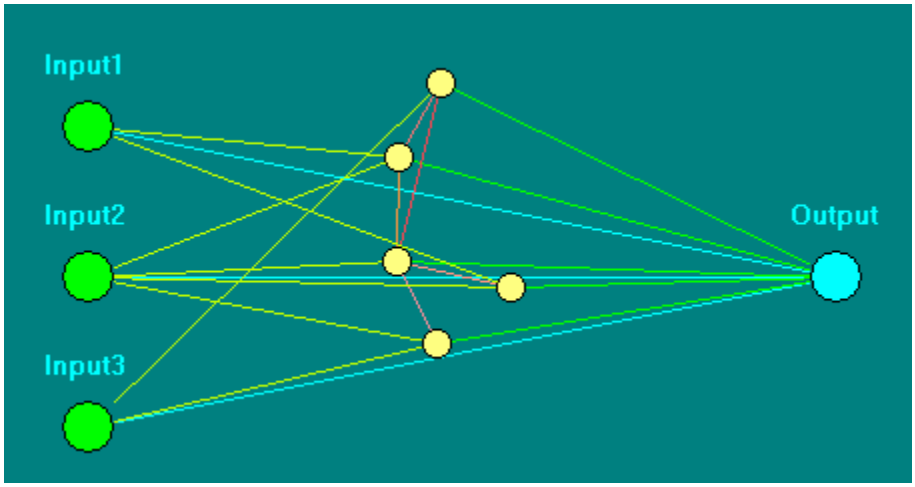


Figure 1. Network Structure

Neurons are grouped into layers by their connection to the outside world. For example, if a neuron receives data from outside of the network, it is considered to be in the input layer. If a neuron contains the network's predictions, it is in the output layer. Neurons in between the input and output layers are in the hidden layer, which serves as a feature detector.

About the Authors

Steve Ward is CEO and Chief Technical Officer of Ward Systems Group, Inc. (WSG). His 40 years of computer experience began as a programmer in college, when he was on the team that programmed the first heart EKG pattern recognition computer. Before becoming President of Ward Systems Group, Inc., his last positions were Vice President of Systems at Computer Network Corp., and Director of the US Army Computer Center at Ft. Detrick, Maryland. Steve's bachelor's and master's majors were both abstract mathematics, and he leads the technical implementation of the neural networks in WSG's product line, as well as all of it's research activities.

Marge Sherald is COO of Ward Systems Group, and has been with the company for over fifteen years. She has authored numerous articles on expert systems, neural networks, fuzzy logic and financial forecasting. Marge participates in the design and documentation of Ward Systems Group products. She has taught neural network seminars and has assisted many customers in developing their applications. Marge has a BS in communications and computer science coupled with over 25 years of experience in these fields.

References

For GeneHunter, the genetic algorithm optimizer discussed in this book:
Goldberg, D.E., “Genetic Algorithms in Search, Optimization, and Machine Learning”, Reading, Mass: Addison-Wesley, 1989.

For TurboProp 2, the neural network algorithm used in NeuroShell Trader:
Turboprop 2 is a variant of the Cascade Correlation algorithm of Scott Fahlman.

Fahlman, S.E., and Lebiere, C., "The Cascade-Correlation Learning Architecture", published in "Advances in Neural Information Processing Systems", Vol 2 1990, pp 524-532, Morgan Kaufmann Publisher, San Mateo

General
Gately, E., “Neural Networks for Financial Forecasting,” Wiley Trader’s Exchange, John Wiley & Sons, New York, 1994.

Mandelbrot, B, and Hudson, R.L., “The (Mis) Behavior of Markets, a Fractal View of Risk, Ruin & Reward,” Basic Books, New York, 2004.

Rumelhart, D.E., and McClelland, J.L. and the PDP Research Group, “Parallel Distributed Processing Explorations in the Microstructure of Cognition, Volumes 1 and 2,”Parallel Distributed Processing Explorations in the Microstructure of Cognition, Volumes 1 and 2, MIT Press, 1986.

Trippi, R. R., and Turban, E., editors, “Neural Networks in Finance and Investing”, Probus Publishing Company, 1993.

Index

