Commodity Trading Using Neural Networks: Models for the Gold Market

by E. Brauner, J. Dayhoff, X. Sun

T.R. 97-41

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry
Commodity Trading Using Neural Networks: Models for the Gold Market

Erik Brauner and Judith Dayhoff
Institute for Systems Research
University of Maryland, College Park
College Park, MD. 20742

Xiaoyun Sun
BehavHeuristics, Inc.
7240 Parkway Dr., Suite 170
Hanover, MD. 21076

Abstract
Essential to building a good financial forecasting model is having a realistic trading model to evaluate forecasting performance. Using gold trading as a platform for testing we present a profit based model which we use to evaluate a number of different approaches to forecasting. Using novel training techniques we show that neural network forecasting systems are capable of generating returns far above those of classical regression models.

1. Introduction
We present a trading model that gives a profit measurement based on forecasts of gold prices one day in advance. Using this model to test performance, a number of different time series forecasting techniques were evaluated. In the last few years much interest has been generated by non-classical techniques for forecasting financial markets[1,2]. Due to their unique ability to approximate arbitrary non-linear functions[3] and their ability to mimic classical regression models[4], neural network techniques were chosen as the primary focus of study. We present modifications to the standard back-propagation learning methods as well as a novel method for training time series data. These methods were used to identify the best error functions to use in training as well as to decide how much past data should be used during training a network. When these methods were compared to classical statistical methods, a significant improvement over a similar regression model was achieved. The best network tested by our trading model was found to have average annual returns of over 20% above market profit for a seven year period.

2. Data Segmentation and Correlation
2.1 Fundamental Choices
Gold was selected as our commodity to test our forecasting and trading systems due to its heavy trading and its key role in many financial markets. To have a large enough data set to draw statistically sound conclusions, it was decided that the seven years of data from 1988 through 1994 would be used as performance data to test any models. Gold was not a good investment for a buy-and-hold strategy during this period of time losing 17% of its value, but this does not lessen the utility of an information and prediction tool. The value of such a tool is its ability to "beat the market" through accuracy of forecasting and capability of identifying market mode shifts.

One day in advance prediction was chosen as a means to explore prediction methods. This allows sufficient data for building and testing models but is still reflective of real world trading practices where traders may change investment positions on a daily basis. One difficulty with predicting tomorrow's price given past data is that daily changes in price tend to be only a fraction of a percentage of the gold's value. Thus, when considering the total price of a commodity such as gold, the daily fluctuations become "lost" when considering the total price. In contrast, if the percentage change on a day-to-day basis is tracked, the data becomes more meaningful and a neural network or statistical tool will likely do a better job at predicting these changes than at predicting the total price. For these reasons it was percentage change in price that was used as a forecasting target to evaluate our models. For a trader this percentage change is the important information to predict since the buying price is only a baseline and it is these changes which represent profit or loss.

Copyright (C) 1997 E. Brauner, J. Dayhoff, and X. Sun. All Rights Reserved.
Figure 1 shows the daily percentage change in price of gold over the period 1988-1994. Once transformed it can be seen that the data is heteroskedastic, that is, the distribution of price changes is variable over time. Inspecting Figure 1 it can be seen that the vertical range of the data passes through various phases or modes indicating different market cycles.

![Figure 1. Daily % change of the price of gold from 1988-1994.](image)

### 2.2 Time-Windowing

For purposes of building and evaluating network models, the data was always segmented into three sets: a training set, a validation set, and a forecast set. The training set was used to build the network model, the validation set was used to monitor forecasting performance to know when to stop training, and the forecast set was used to test the resulting model on completely blind data. To ensure that training of networks was really blind to the forecast data the training and validation sets contained data strictly from before the forecasting period for that model and the training data always preceded the validation data.

![Figure 2. Windowing of data into training, validation, and forecast sets. The window is then shifted forward to cover a new forecast period.](image)

As an example, if a one year period of forecasting, and 3 month period of validation were chosen to build a model to forecast the period Jan. 1990 - March 1990, then the validation data would be the period Oct. 1989 - Dec. 1989, and
the training period would be the year directly preceding this, i.e. Sept. 1988 - Sept. 1989. Once a network has been trained and evaluated the sets are slid forward in time until the new forecast set is directly after the previous one. Figure 2 illustrates this windowing procedure. This process is then repeated until the entire performance data set has been covered by the forecasting windows. In practice we frequently used forecast set sizes of 1 month. Thus, over the seven year performance set period 1988-1994 this windowing procedure was repeated 96 separate times resulting in a different network model for each month of data. In practice it might be beneficial to extend this to rebuilding the network model for each day (i.e. forecast set length of one day), but as there are over 2000 days of trading data in the performance set this would have required a prohibitive amount of time for experimentation.

3. Trading and Decision Support Performance

3.1 A Trading Model

To evaluate the performance of our neural models, a trading model was chosen rather than a standard statistical measure such as mean squared error. This model is based on the one proposed by Blake LeBaron[5]. The idea is to simulate the daily trading practices of a large trading firm which is changing its position on a daily basis. The presumption of the model is that at the close of each trading day you have a forecast for what the market will do tomorrow. Using this information alone you must decide to put all of your capital into either a long or short position. Thus if you have forecast that the market will go up tomorrow you choose to take a long position which is equivalent to buying the actual commodity. If you forecast that the price will go down tomorrow, you take a short position, which is equivalent to purchasing put options. Using this sort of a trading model it is really only the direction of the forecast that matters. If we guess the direction correctly then our profit is the absolute value of the percentage change the commodity undergoes; if we guess incorrectly we lose this amount. For ease we assume that our total capital on the first day is simply 1. Thus if our capital at the end of the forecast period were 1.5 we would have made a profit of .5 or 50%. Mathematically the change in capital from day t to t+1 can be expressed:

if our forecast for t+1 is correct in direction:

\[ \text{Capital}_{t+1} = \text{Capital}_t \times (1 + \% \text{ change in gold}) \]

if our forecast for t+1 is wrong:

\[ \text{Capital}_{t+1} = \text{Capital}_t \times (1 - \% \text{ change in gold}) \]

As a basis for comparison we note that guessing that the price will increase every day is equivalent to buying gold and holding it through the entire period. If one were to do this during the forecast period of 1988-1994 your final capital would actually be .83 indicating that gold lost 17% of its value over that period– not a very good investment. For our purposes it is this profit, defined as the capital at the end of the period minus the starting capital, which is used as our primary measure of model fitness.

4. Neural Network Training

![Diagram of a typical feed-forward network configuration used for these experiments.](image)
4.1 The Neural Network Model

The neural networks used for our experimentation were 3-layered, fully interconnected, feed-forward networks[6], with an input layer, one hidden layer composed of three nodes, and an output node which supplied the prediction for percentage change in the price of gold. Figure 3 illustrates a typical network configuration. The input layer nodes had no transform function and were simply the values of the corresponding inputs, while the hidden and output layer nodes used the standard sigmoidal transform function. Since a sigmoidal node can only output values between 0 and 1 all inputs were scaled at the input layer to be between .1 and .9, then the output from the network was scaled back to a real target value. This allowed the output from the network to be an actual value representing a predicted percentage change in the price of gold and made error measurement and analysis much easier.

4.2 Weight Adjustment

In order to propagate the error backward in the network for adjustment of weights, it is necessary to define an error function which we will be attempting to minimize. The general gradient descent learning algorithm gives the update rule for weight \( i \) on node \( j \) as:

\[
\Delta w_{ij} = -\eta \frac{\partial E(n)}{\partial w_{ij}}
\]

where \( E(n) \) is some function measuring the error of pattern \( n \), and \( \eta \) is the learning rate parameter. The most common error measure used in gradient descent learning is that of squared error. The difficulty with such an error measure is that it can be overly influenced by outliers—data points with extreme values which are a result of unpredictable factors. Outliers will be weighted too heavily. An alternative to using a squared error function which is less susceptible to outliers is to use an absolute error measure. The function for such an error measure is given by

\[
E(n) = |\text{networkOutput}_n - \text{desiredOutput}_n|
\]

which yields

\[
\frac{\partial E(n)}{\partial w_{ij}} = \pm \frac{\partial \text{networkOutput}_n}{\partial w_{ij}}
\]

The problem with such an error measure is that all errors are given equal weight and are only dependent on direction. Thus, no matter how accurate a given network prediction is, it will force a correction of equal magnitude to that of the worst prediction. One solution to this problem is to only train on a pattern if the error of the prediction is above a certain threshold. This means that once the prediction for certain patterns comes close enough to the desired value we stop training it. All experiments described in this paper which use absolute error for training used a cutoff which was set at .01, which was determined after a few preliminary tests.

4.3 Training Schedule

For each data window or group of training, validation, and forecast sets, a new network model was trained starting from small randomized weights. The network weights were adjusted by successive loops through the training data set. After each loop through the training set, the network was used to forecast the data in the validation set without training on it. A measure of the network's average error over the validation set was then used as a stopping criterion, to avoid overtraining. In all experiments described in this paper the networks were allowed 10,000 training passes on the training set. The model from the pass which had the lowest average forecast error over the validation set was then chosen to use on the forecasting set. One key idea behind using a validation set was the intuition that since this data is the closest in time to the forecast set it should best represent the behavior of the forecast set. This was used to identify phase transitions in the data in the following manner: if the data in the training set had very different behavior than the data in the validation set, then what frequently occurred was that refinement of the model on the training data only worsened with repeated training. If the behavior of the two data sets was sufficiently different, the minimum validation error often occurred in the first few training passes, indicating a model which was likely to perform poorly on the forecast set. In our experiments, if the minimum validation error was achieved before pass 20, then the model was not used and we assumed that we could not accurately predict the behavior of the forecast set. In these cases the profit over the period of the disregarded model was assumed to be zero, as though we were to withdraw all investment from the market.
5. Experimental Results

5.1 Choosing the Best Error Function for Training

The two choices of error functions tested for training the network weights were the traditional squared error measure, and absolute error. To test which method of training was more appropriate for our trading model a set of experiments was run utilizing both methods of error correction. The experiments all used a training period length of one year and a validation period length of 3 months to forecast successive 3 month periods over the span of 1988-1994. The networks used all had three hidden layer nodes and one output layer node and the weights were randomized for each of the 28 models built. 11 inputs were chosen from the list of possible inputs and included some time delayed values for the past changes in gold. A momentum of 0.25 was used and the learning rate for the two error correctors were set at $\eta=0.01$ and allowed to linearly decrease to zero over the 10,000 training passes. Each experiment was run 5 times with different random starting weights to ensure that no results were anomalous. Both average absolute error and mean squared error over the validation set were tried as stopping criteria. Results from this experiment indicated that using absolute error for training was far superior to squared error regardless of the stopping criteria used.

5.2 Comparison to a Regression Model

As a basis for comparison, the experiment described in the above section was reproduced using classical linear regression to build the model. The same 11 inputs were used as the independent variables as those used for the network inputs. Again, 28 separate regression models were built, each one to forecast one of the successive 3 month periods in 1988-1994. The one difference was that no validation data was needed for the regression model. Because of this only the 1 year of data immediately preceding each forecast period was used to train each model. When this method was used in the trading model it yielded a profit of 0.462. This was not a bad profit considering the market profit was -0.17 during the same period but is not nearly as high as the 0.6098 profit from the neural network trained by absolute error. Other variations lead to even higher yields than 0.6098.

5.3 Choosing a Stopping Criteria

Another question explored was which error measure over the validation set should be used to decide when to stop training. Three obvious measures came to mind: average absolute error over the validation set, average squared error over the validation set, and profit over the validation set. To explore this question more fully, 5 different choices of inputs were tried ranging in number from 8 to 15 and each was trained using all three stopping criteria. The training validation period's lengths were left at 1 year and 3 months respectively, however the forecast period was changed to one month necessitating the building of 96 models rather than the 28 described above. Again the experiments were run five times each with different randomized starting weights to remove dependence on this factor. The results showed that profit over the validation set was a poor choice for stopping criteria. The differences between using mean absolute error and mean squared error were less dramatic, but mean absolute error appeared to outperform other choices of stopping criteria for a range of models.

5.4 Comparison of Validation Period Lengths

How long a validation period should be used for training a network was an issue essential to constructing a good network model. If the forecast data were equally well represented by all historical data, then the correct amount of data would be all available data. The changing behavior of the market, however, indicates that the smaller the validation period, the better it would represent the coming period. Too little validation data would make it impossible to get a good measure on the model's fitness, so we were left to search for the minimum period which would give a good measure of network fitness. Differing validation periods were tested on 3 network models having 8, 11, and 15 inputs respectively. Figure 5 illustrates the results of these tests. Each data point given is the mean of 5 separate trials run with differing random starting weights for the networks.
Figure 5. Comparison of differing validation periods on three different network models. Profit given is the mean over 5 separate trials with different random seeds.

It can be seen from the data that the best choice of validation length varies by the number of inputs. More generally, if we consider the number of adjustable weights in each network rather than the number of inputs an interesting relationship emerges. The network with 8 inputs has 27 adjustable weights and a best validation period of 60 days. The network with 11 inputs has 36 weights and a best validation length of 70 days. The network with 15 inputs has 48 adjustable weights and a best validation length of 100 days. From this it appears that the best validation length is approximately $2^*(\text{number of adjustable weights})$.

6. References


7. Acknowledgments

Special thanks to: Sharon Hornby of BehavHeuristics, Inc.
This work was supported by: The Maryland Industrial Partnership Program (MIPS contract 1716.17), the National Science Foundation (NSF grant# CDF88-03012), the Institute for Systems Research at the University of Maryland, College Park, and BehavHeuristics, Inc.