Commodity Futures Price Prediction, an Artificial Intelligence Approach

by

Ernest A. Foster

(Under the direction of Walter D. Potter)

Abstract

This thesis describes an attempt to predict the next value in a financial time series using various artificial techniques. The time series in question consists of daily values for commodities futures. First, an artificial neural network is used as a predictor. Then the neural network is augmented with a genetic algorithm. The genetic algorithm first is used to select the parameters for the neural network. Then in a seperate experiment the genetic algorithm is used to evolve the weights of the network. The various approaches had similar results.

Index words: artificial neural networks, genetic algorithms, commodity futures, time series analysis

Commodity Futures Price Prediction, an Artificial Intelligence Approach

by

Ernest A. Foster

B.S., The University of Alabama, 1993

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2002

Commodity Futures Price Prediction, an Artificial Intelligence Approach

by

Ernest A. Foster

Approved:

Major Professor: Walter D. Potter

Committee: Donald Nute
Khaled Rasheed

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2002

Table of Contents

Chapter 1

Introduction

1.1 Commodities, Markets and Profits

A commodity is a good, especially an agricultural or mining product, that can be processed and resold. Some examples of such goods are: soybeans, wheat, crude oil and gold. Large volumes of commodities are bought and sold around the world each day. Farmers and miners need to sell their products and manufacturers need to purchase raw materials. This can be done on the spot market, where commodities are bought and sold "on the spot". The spot market is actually a term that denotes many decentralized locations at which the good may be bought or sold.

However, commodities trading is actually much more complex than simply buying and selling goods on the spot market. Most commodities trading is actually done on the futures market as futures contracts. That is, at time t_0 an individual promises to sell x bushels of soybeans at price y at a time t_1 in the future [Hul97]. Someone else would enter into the other side of the contract, promising to buy that quantity of the commodity at the agreed upon price at the proposed time. The development of futures can be traced to the middle ages when they were developed to meet the needs of farmers and merchants [Hul98]. Futures make a great deal of sense for both parties because they reduce risk. The farmer knows that he will sell his crop for a certain price and the manufacturer can rest assured that his business will be able to continue production with an ample supply of affordable raw materials.

Commodity futures contracts were first traded on an organized exchange, the Chicago Board of Trade (CBOT), in 1850. The first contract called for the delivery of 3,000 bushels of corn. The corn would be exchanged for a cash payment at a pre-specified future time and location. In 2002, over 260 million contracts for 47 different products were traded on the CBOT. Comparable trading volume exists on the Chicago Mercantile Exchange and the New York Mercantile Exchange. The enormous size of the futures market has sparked researchers to examine the motivations of traders.

Commodity futures contracts originated as a way for farmers to reduce their price risk. Prior to these contracts, a farmer would have to bear the cost of planting and cultivating a crop with no guarantee of the price for which that crop could be sold. By contracting to sell his crop at a certain price before planting it, the farmer did not have to be concerned with future price changes. His only risk was the actual production of his crop. Therefore, a farmer's compensation would depend on his ability to produce a crop rather than his ability to produce a crop and future changes in the price of the commodity being produced. The farmer, as is anyone else who seeks to reduce risk, is called a hedger.

The farmer's need to hedge is fulfilled by speculators. Speculators are willing to bear the price risk in hopes of favorable price changes. Since farmer's hedge their position by contracting to sell in the future (aka shorting the contract) speculators take long positions (agree to buy). Speculators will only take a long position if they expect the price of the commodity to be greater at time of delivery than the price specified in the futures contract [Key78]. Then they can buy from the farmer at a set price and immediately sell the commodity for a greater price. The difference between the price stated in the futures contract and the speculators expected price at that future time is the risk premium. The hedger is willing to allow speculator to have this premium in exchange for the reduced risk of a guaranteed price. The speculator's

compensation is this risk premium. Since it is the speculator's belief about the probability distribution of future spot prices that determines the size of the risk premium that he is willing to accept, the speculator's success is dependent on the validity of his beliefs. A comparative advantage can be gained by acquiring superior information about the underlying dynamics of the commodity's price behavior.

The use of futures contracts to hedge is not limited to farmers. A manufacturing firm may require large amounts of a commodity to produce its final product. Airlines must buy large amounts of jet fuel. Any entity which is exposed to the price risk of a commodity that underlies a traded futures contract may reduce their risk by taking the appropriate position futures contracts. Those seeking to reduce risk are hedgers [CWS95].

The futures market has grown well beyond the previous examples but the logic still applies. It should be noted that most futures contracts do not result in physical delivery. A trader can realize the financial gain or loss of his position by taking a position in the same futures contract that is opposite to his current position.

The effects of various factors on a particular commodity's price are unique to that particular commodity. As with any economic good the ultimate arbiter of price is supply and demand. Efficient markets imply that all information is fully reflected in the commodity's price and no one can make a economic profit by trading futures contracts.

If a pattern can be found in past price data that accurately predicts future price movement, then markets are not efficient and an economic profit can be made by those who recognize the pattern. NN and GAs are used to find such a pattern. In an attempt to predict commodity futures prices, people attempt to use historical price data for a commodity future to predict the value of that commodity future. Mathematically, values which run in a series and vary over time are known as a time series.

1.2 Time series and financial analysis

Technical analysis focuses on the past data of an asset, such as a commodity futures contract, in order to predict its future behavior [BKM98]. Thus, in essence, technical analysis is devoted to studying time series of data related to financial instruments, including commodities, futures, stocks, etc. Technical analysts examine this historical data and attempt to formulate rules which explain the behavior that they observe. They look for patterns in the data and attempt to generalize from those patterns in order to predict the future behavior of the variable in question such as the price of the asset. Some technical analysts specialize in stocks, others in futures or commodities, yet others in foreign exchange rates.

Technical analysts use a variety of methods in their attempts to extract valuable information from time series. Some of the methods, such as Box-Jenkins methods [BJ76] or other linear regression approaches rely on their assumption of a linear relationship among the variables. However, most experts have found that the relationships among time series values for various financial instruments are not linear [BLB92, DGE93]. Other, non-linear methods have also been used to perform financial time series analysis [Sav89, Sch90].

1.3 Methodology

1.3.1 Experiments

A neural network is an artificial intelligence technique that is especially useful for recognizing patterns in complex data sets and generalizing from those patterns in order to work with new data. Thus, a neural net would seem to be an ideal technique to use in time series analysis. The neural net should be able to take n values of the time series as inputs and estimate the next value as an output. The network should be able to find any existent relationship between these variables. The neural net

should be able to recognize any pattern in the data and predict future values. The theoretical limit to the accuracy of a neural network is the amount of noise present in the data.

Neural networks are quite useful for recognizing patterns and generalizing from them. However, optimal network performance is quite dependent upon making the proper choices when constructing the network. Furthermore, the construction of a neural network is very much an art form and the ideal network architecture can be quite elusive. It is possible to combine the neural net with another artificial intelligence technique such as a genetic algorithm. Genetic algorithms excel at minimization / maximization problems. Thus, a genetic algorithm can be used to determine the proper architecture to maximize the accuracy of the neural network.

In addition to the architecture problem, the methods used to train a neural net can fall prey to local minima. Genetic algorithms can be used once more to find optimal values for the weights of the neural networks as well. Thus, local minima can be avoided and optimal neural network results can be obtained.

The goal of the current research is to find a preferred artificial intelligence approach to predict future values for commodities futures. Project objectives are:

1. Develop a neural network model to predict the next value in the commodity future time series.

2. Use a genetic algorithm to select the proper architecture for a neural network model to predict the next value in the commodity future time series.

3. Use a genetic algorithm to search for weight assignments for a neural network model to predict the next value in the commodity future time series.

Chapter 2 examines objective 1, while chapters 3 and 4 address objectives 2 and 3, respectively. Chapter 5 compares results from all of the objectives and discusses overall conclusions.

1.3.2 Data

The data consist of daily commodity data for soybeans at the Chicago Board of Trade starting with January 1, 1980 and ending with August 30, 2002. Different portions of this data are used in the various experiments. The data are obtained from Datastream. In order to more easily optimize neural network performance, the data are normalized before being used as input for the neural networks. Each data set is divided by its largest member to provide values between zero and one. In turn, the predicted, output values are multiplied back to their original values and compared to actual observed values in their original form. Error statistics are generated using these non-normalized values.

Chapter 2

Pure Neural Network Approach

2.1 Background on Neural Networks

The brain is made up of neurons. These neurons are connected to each other by synapses which vary widely in strength. Each neuron receives input from a number of other neurons. This input may be excitatory, i.e. positive, or inhibitory, i.e. negative. If the input activation level is sufficient, then the neuron will "fire". The firing neuron will then send activation to those neurons which are connected to its output. In this manner activation spreads through a network of neurons. The human brain contains between 10^{10} and 10^{11} neurons. Each of these neurons is connected to hundreds or thousands of other neurons[And95]. Thus, the human brain understands speech, recognizes a loved one's face, and accomplishes abstract thought, etc. by spreading patterns of activation throughout this complex neural network. Though each neuron is making a simple calculation, complex learning is accomplished by altering the connection strengths between neurons. The calculation performed by each neuron remains the same, however the weight given to the various inputs changes. This enables the marvelous complexity of function shown by the human brain.

Neural networks are based on this system. There are nodes which have an activation function, most commonly the sigmoid function, $1/(1 + e^{-x})$. This function determines if the input activation is sufficient to cause the node to fire. Biological synapses are modeled by weighted links between the nodes. These weights can be

positive or negative, modeling the excitatory and inhibitory function found in biological neural networks. In general, there are one or more nodes which receive the data as input. These are called input nodes. In most neural network implementations, the input nodes do not use an activation function. These nodes simply function as storage bins for the input being fed into them. There are also usually one or more nodes which provide the output, or answer from the network. These are called output nodes. In many neural network architectures, including all of those used in this research, there are additional nodes which reside between the input and output nodes. These nodes are called hidden nodes and they provide much of the computing power for the network.

A wide variety of neural network architectures have been used. However, this research focuses on the most commonly used architecture, the multilayer perceptron, also known as a three layer, feed-forward neural network. Hereafter "network" or "neural network" will refer to a network of this type. In this sort of network, nodes in one layer may only influence those residing in a layer closer to the output layer. Thus, activation may only feed forward through the network from input nodes to hidden nodes and then on to output nodes. No feedback loops are allowed.

A properly constructed network can recognize patterns in the input data. Different patterns of input produce different output. The network can also generalize by responding to similar patterns in a similar manner. However, these desirable behaviors are dependent upon the weights in the network being properly set. Calculating the proper weights for a small network quickly becomes a daunting task. Directly calculating the weights for a large network is practically impossible. However a system of using supervised learning called back propagation of errors[RHW86] provides an answer to this problem. This technique uses a gradient decent method to establish values for the network weights which minimize the output errors. Thus, both the input and output must be known for the network to be trained.

2.2 Neural Networks and Financial Time Series Analysis

In addition to the more traditional approaches, a artificial intelligence techniques have been applied to a number of financial instruments. Garcia and Gençay[GG00] use a neural network to estimate a generalized option pricing formula. Walczak[Wal01] uses neural networks to forecast foreign exchange rates for a variety of currencies. Trippi and DeSieno[TD92] examine a day trading system for Standard & Poors 500 index futures contracts which is based on a neural network combined with rule-based expert system techniques. Hutchinson, et al.[HLP94] construct neural network models for predicting Standard & Poors 500 futures options prices. Castiglione[Cas00] uses neural networks to predict a variety of financial time series. Yao, et al.[YTP99] use neural networks to forecast the Kuala Lumpur Composite Index.

2.3 Software used / Methodology

For this research, a variant form of backpropogation called resilient propagation (RPROP)[RB93] was used. RPROP utilizes a local adaptation of the weight updates according to the behavior of the error function in order to overcome some of the disadvantages of a pure gradient decent approach. The RPROP algorithm is not effected by the influence of the size of the error derivative, since it relies only on the sign of the error derivative. Instead, the size of the weight change is determined by a weight-specific update value $\triangle_{ij}^{(t)}$:

$$\triangle w_{ij}^{(t)} = \begin{cases} -\triangle_{ij}^{(t)} & , \quad \text{if } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\triangle_{ij}^{(t)} & , \quad \text{if } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & , \quad \text{else} \end{cases} \tag{2.1}$$

The new update-values $\triangle_{ij}^{(t)}$ must also be determined. This is based on a sign-dependent adaptation process.

$$\triangle_{ij}^{(t)} = \begin{cases} \eta^+ * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \triangle_{ij}^{(t-1)} & , \text{ else} \end{cases} \tag{2.2}$$

$$\text{where } 0 < \eta^- < 1 < \eta^+$$

Thus, when the last update is too large and the algorithm has jumped over a local minima, the partial derivative of the corresponding weight w_{ij} changes its sign and the update value $\triangle_{ij}^{(t)}$ is decreased by the factor η^-. When the derivative retains its sign, the update value is increased by the factor η^+. Additionally, in the case of a change in sign, there is no adaptation in the succeeding learning step. This is achieved by setting $\frac{\partial E}{\partial w_{ij}}^{(t)} = 0$ in the above adaptation rule.

In order to reduce the number of freely adjustable parameters, the increase and decrease factors are usually set to fixed values of $\eta^- = 0.5, \eta^+ = 1.2$.

This research also utilized weight decay. Weight decay is a modification that may be used with neural network's backpropagation learning technique in order to improve generalization. Using a weight decay term (α) can assist in preventing over training and preserving generalization by performing a local reduction of the weights, discouraging large weights. This research used the weight decay variant of the RPROP algorithm. Thus, the composite error function is:

$$E = \sum (t_i - o_i)^2 + 10^{-\alpha} \sum w_{ij}^2$$

For implementation of the neural nets, this research used the Stuttgart Neural Network Simulator, version 4.2. This software was run on a Pentium 4 PC with the Linux operating system. A number of different network architectures were used along with a number of variations of the network parameters. The networks were evaluated by comparing predicted versus actual values on the data sets. For each

network, the sum of square errors (SSE) was collected and used for evaluation. Each combination of variables was set up in SNNS and trained until either:

- the test data error began increasing
- there was no further improvement in the training data error.

This data should provide insight into the problem as well as establishing a baseline for comparison with other techniques.

2.4 Results / Conclusions

One set of experiments uses one hundred days worth of input into the neural net. The other set uses only ten days worth of input. The longer data set should allow longer term patterns to be detected, while the shorter data set should focus the network on recent values for the variable in question.

Additional, longer time periods could be tested in an attempt to find still longer term patterns. A time period of approximately a year would be a fairly logical extension, especially for agricultural commodities such as the soybeans used in this research. However, longer time periods are outside the scope of this research.

The number of hidden nodes used were five, ten, and one hundred. The weight decay parameter was allowed to assume the values five, ten, and twenty. The initial weight range values were kept semetrical and had values of plus and minus: 1.0, 0.5, 0.25, 0.125, 0.0625.

The results of the experiment can be seen in Table 2.1 through Table 2.6. There is considerable variety in the quality of solutions found by the different networks. These tables contain the error calculations on the out of sample validation set. The mean squared error (MSE) was calculated by dividing the SSE by the number of observations in the data set. The mean error is simply the square root of the MSE.

These values are in cents per bushel. So, a mean error of 15.20 means that the network missed its prediction by an average of 15.20 cents per bushel. The average price per bushel was 651.37 cents. Even relatively small differences in error could mean a considerable difference in the profit made by using the network since the volume of the commodity traded can be rather large.

It appears that the most recent data is the most important for calculating the next value in the time series since there is no improvement gained by increasing the number of inputs from ten to one hundred. Indeed, there is evidence of decreased performance on the experiments which used one hundred data points as input. There seems to be a negative effect of including additional hidden nodes as well. The error values for the five and ten hidden node subsets are comparable, however the error generated by the one hundred hidden node sets is noticeably greater. This difference seems to only exist in the experiments which used one hundred input values. This seems to indicate that the greater number of hidden nodes combined with the larger number of input nodes allowed the network begin to over train despite the weight decay parameter and the stopping condition. Indeed, the results for a weight decay parameter of five are slightly better than the others. Since a lower weight decay parameter causes more severe weight decay and thus a more severe limit on over training, this discrepancy supports the hypothesis. These findings are in agreement with those of Gerson and Fuller[LF95]. They found that including additional input or hidden nodes reduced network efficacy for time series evaluation.

Table 2.1: Results with 100 input values, average price 651.37 cents per bushel

Hidden Nodes	Weight Decay	Initial weight range	SSE	MSE	Mean Error
100	5	-1.0–1.0	253563	329.30	18.14
100	5	-0.5–0.5	236477	307.11	17.52
100	5	-0.25–0.25	304617	395.60	19.88
100	5	-0.125–0.125	224618	291.71	17.07
100	5	-0.0625–0.0625	233060	302.67	17.39
100	10	-1.0–1.0	321903	418.05	20.44
100	10	-0.5–0.5	242407	314.81	17.74
100	10	-0.25–0.25	202006	262.34	16.19
100	10	-0.125–0.125	171655	222.92	14.93
100	10	-0.0625–0.0625	242307	314.68	17.73
100	20	-1.0–1.0	478583	621.53	24.93
100	20	-0.5–0.5	207835	269.91	16.42
100	20	-0.25–0.25	326325	423.79	20.58
100	20	-0.125–0.125	375068	487.10	22.07
100	20	-0.0625–0.0625	139092	180.64	13.44

Table 2.2: Results with 100 input values, average price 651.37 cents per bushel

Hidden Nodes	Weight Decay	Initial weight range	SSE	MSE	Mean Error
10	5	-1.0–1.0	152358	197.86	14.06
10	5	-0.5–0.5	127937	166.15	12.89
10	5	-0.25–0.25	150851	195.91	13.99
10	5	-0.125–0.125	328134	426.14	20.64
10	5	-0.0625–0.0625	194167	252.16	15.87
10	10	-1.0–1.0	170047	220.84	14.86
10	10	-0.5–0.5	134972	175.28	13.23
10	10	-0.25–0.25	108339	140.70	11.86
10	10	-0.125–0.125	114269	148.40	12.18
10	10	-0.0625–0.0625	160197	208.04	14.42
10	20	-1.0–1.0	159494	207.13	14.39
10	20	-0.5–0.5	164318	213.40	14.60
10	20	-0.25–0.25	125927	163.54	12.78
10	20	-0.125–0.125	120299	156.23	12.49
10	20	-0.0625–0.0625	112058	145.53	12.063

Table 2.3: Results with 100 input values, average price 651.37 cents per bushel

Hidden Nodes	Weight Decay	Initial weight range	SSE	MSE	Mean Error
5	5	-1.0–1.0	170248	221.10	14.86
5	5	-0.5–0.5	116781	151.66	12.31
5	5	-0.25–0.25	123515	160.40	12.66
5	5	-0.125–0.125	120098	155.97	12.48
5	5	-0.0625–0.0625	234568	304.63	17.45
5	10	-1.0–1.0	135776	176.33	13.27
5	10	-0.5–0.5	220699	286.62	16.92
5	10	-0.25–0.25	115173	149.57	12.23
5	10	-0.125–0.125	114570	148.79	12.19
5	10	-0.0625–0.0625	118490	153.88	12.40
5	20	-1.0–1.0	117686	152.83	12.36
5	20	-0.5–0.5	112862	146.57	12.10
5	20	-0.25–0.25	129846	168.63	12.98
5	20	-0.125–0.125	371349	482.27	21.96
5	20	-0.0625–0.0625	111053	144.22	12.00

Table 2.4: Results with 10 input values, average price 651.37 cents per bushel

Hidden Nodes	Weight Decay	Initial Weight Range	SSE	MSE	Mean Error
100	5	-1.0–1.0	131555	152.97	12.36
100	5	-0.5–0.5	507729	590.38	24.29
100	5	-0.25–0.25	112058	130.30	11.41
100	5	-0.125–0.125	99294	115.45	10.74
100	5	-0.0625–0.0625	129746	150.86	12.28
100	10	-1.0–1.0	120902	140.58	11.85
100	10	-0.5–0.5	98591	114.64	10.70
100	10	-0.25–0.25	104118	121.06	11.00
100	10	-0.125–0.125	100701	117.09	10.82
100	10	-0.0625–0.0625	129947	151.10	12.29
100	20	-1.0–1.0	576371	670.19	25.88
100	20	-0.5–0.5	214066	248.91	15.77
100	20	-0.25–0.25	118490	137.77	11.73
100	20	-0.125–0.125	100802	117.21	10.82
100	20	-0.0625–0.0625	130751	152.03	12.33

Table 2.5: Results with 10 input values, average price 651.37 cents per bushel

Hidden Nodes	Weight Decay	Initial Weight Range	SSE	MSE	Mean Error
10	5	-1.0–1.0	410846	477.72	21.85
10	5	-0.5–0.5	107133	124.57	11.16
10	5	-0.25–0.25	134168	156.00	12.49
10	5	-0.125–0.125	103113	119.89	10.94
10	5	-0.0625–0.0625	108641	126.32	11.23
10	10	-1.0–1.0	98189	114.17	10.68
10	10	-0.5–0.5	328034	381.43	19.53
10	10	-0.25–0.25	461699	536.86	23.17
10	10	-0.125–0.125	137082	159.39	12.62
10	10	-0.0625–0.0625	247231	287.47	16.95
10	20	-1.0–1.0	102812	119.54	10.93
10	20	-0.5–0.5	101405	117.91	10.85
10	20	-0.25–0.25	96882	112.65	10.61
10	20	-0.125–0.125	104018	120.95	10.99
10	20	-0.0625–0.0625	117686	136.84	11.69

Table 2.6: Results with 10 input values, average price 651.37 cents per bushel

Hidden Nodes	Weight Decay	Initial Weight Range	SSE	MSE	Mean Error
5	5	-1.0–1.0	97787	113.70	10.66
5	5	-0.5–0.5	100400	116.74	10.80
5	5	-0.25–0.25	97485	113.35	10.64
5	5	-0.125–0.125	105827	123.05	11.09
5	5	-0.0625–0.0625	124017	144.20	12.00
5	10	-1.0–1.0	104319	121.30	11.013
5	10	-0.5–0.5	99294	115.45	10.74
5	10	-0.25–0.25	97083	112.88	10.62
5	10	-0.125–0.125	97586	113.47	10.65
5	10	-0.0625–0.0625	107837	125.39	11.19
5	20	-1.0–1.0	110550	128.54	11.33
5	20	-0.5–0.5	150750	175.29	13.23
5	20	-0.25–0.25	112962	131.35	11.46
5	20	-0.125–0.125	108641	126.32	11.23
5	20	-0.0625–0.0625	98892	114.99	10.72

Chapter 3

Use a Genetic Algorithm to evolve architecture for a neural network

3.1 Introduction

Commodity trading is a complex and difficult business. Those that do it successfully make a great deal of money. One of the keys to succeeding in the commodities markets is information. If one could predict future prices within a known error tolerance, it should be possible to gain a distinct advantage in the marketplace.

Technical analysts believe that the price of a financial asset, including commodities futures, contains all of the information needed to predict future values of the asset. These analysts are, to some degree, involved in time series analysis. Many different approaches have been used in time series analysis. Some time series are quite amenable to linear regression. However, most financial asset time series have been found to show a high degree of non-linearity. Neural networks provide a valuable alternative to complex parametric non-linear regression techniques. The only limit to the complexity of the function that a neural network can fit is the number of hidden nodes present in the network architecture. Additionally, neural networks have the ability to model noisy data with respectable accuracy.

While neural networks do provide a useful model of the underlying nonlinearities, they are not without limitations. There are parameters which must be chosen when constructing a neural network. First, the designer must select an architecture. Even if the standard three layer feedforward architecture is chosen, the number

of hidden and input nodes must still be selected. This is especially true of time series data, since dividing the data into input sets is a completely arbitrary decision. Furthermore, after the architecture is selected, learning algorithm parameters must be selected. The relationship between network performance and architecture and learning parameters is poorly understood and often searching for the best one is a heuristic task. This search can be conducted in a trial and error fashion and an acceptable network can generally be found. However, this is a laborious task consuming a great deal of time and requiring a substantial amount of expertise.

Alternatively, an automated technique for finding parameters which would maximize the network's accuracy would be quite desirable. Genetic algorithms provide an excellent way to maximize a function. They allow for searching broad, poorly understood solution spaces and finding maximizing values for function parameters. By combining the search capabilites with the modeling power of the neural networks, a powerful, usable predictive tool can be generated. This research constructs such a system and examines its performance.

3.2 Background on Genetic Algorithms

Just as neural networks are biologically based, genetic algorithms are based on elements of the natural world. Genetic algorithms are search algorithms based on the mechanics of natural selection[Gol89]. They begin with random individuals, which represent possible solutions and mirror chromosomes in biological evolution. These individuals are compared against one another using an objective function. Those which are judged better by this objective function are then selected as progenitors for the next generation of solutions. These selected individuals exchange some of their information, reflecting sexual reproduction in biology. The canditate offspring are then exposed to mutation. This allows for a small chance that a portion of the

childs information will be altered. This preserves diversity in the population and provides a bit more power to the search. This cycle of evaluation, selection, crossover, and mutation continues until the stopping criterion is met.

There are a number of techniques used for each of the steps of a genetic algorithm. The objective function is obviously different for each problem. The selection scheme can also vary. There are probabilistic methods in which the more fit individuals are more likely to be chosen. There are also methods where a strict percentage of the most fit individuals is chosen. The crossover step can also be implemented in a variety of ways. There can be fixed single point crossover where each individual is seperated into two parts of a predetermined size and then these subportions are recombined. The crossover can also be random single point crossover. In this case, for each set of individuals the program determines a random point for the split and recombination to take place. Each of these can also occur in more than one place so that there can be two, three, or an arbitrary number of points of crossover. Mutation most often means randomly changing a bit of the information content of the child. However, mutation is most dependent upon the representation chosen for the genetic algorithm.

By beginning with a random population of individuals and including random mutation, the genetic algorithm can globally sample a large solution space. This makes it ideal for poorly understood or high order solution spaces. By then recombining portions of good solutions, it is hoped that one of the ensuing offspring will be an even better solution than the parent.

3.3 Using a Genetic Algorithm to Evolve Parameters for Neural Networks

Many people have used genetic algorithms to evolve parameters for neural networks. Representation is perhaps the most important issue confronting the designer of such a system and the representations chosen may be roughly divided into direct and indirect encodings.

Miller et al.[MTH89] provide a good example of direct encoding. They evolved the internode connectivity for a feedforward network with a fixed number of nodes. Each connection was represented by a binary bit. A 1 at a particular position in the chromosome signified that the connection was present, while a 0 indicated that the nodes in question were not connected. This representation illustrates direct encoding. There is a discrete element, in this case a bit, in the chromosome which directly represents each element, in this case a connection, that is represented in the network. Miller et al. used a standard fitness-proportionate selection scheme and standard mutation (bits in the string were flipped randomly). However, they modified the crossover operator so that certain blocks of bits were always copied together. Their scheme is essentially multi-point fixed crossover. They chose this implementation of the crossover operator because each such block represented all of the incoming links into a node. They felt that these bits represented a functional building block of the network and that preserving them would speed convergence. The objective function decoded each chromosome into a neural network, trained that network using backpropagation for a fixed number of epochs and then returned the sum of the squares of the errors on the training data at the last epoch. Miller et al. tried their network on three problems: the XOR problem, a real valued four quadrant problem, and a pattern copier that contained fewer hidden nodes than inputs. While their approach worked, the problems to which they applied it were quite simple and did not

supply a rigorous test of this method [Mit96]. Direct encoding causes problems that are related to the directness of the direct encoding. Because the chromosome maps directly to the architecture, the size of the required chromosome must increase at the same rate that the network architecture increases. This rapid increase in size may make direct encodings computationally unfeasable for large problems. Additionally, direct encodings cannot efficiently represent repeated or nested structures, though these sorts of structures are common for some problems.

Kitano [Kit90] provides an example of indirect encoding. He uses what he terms grammatical encoding. The grammars are a set of rules that can be used to build a set of structures. He encodes the grammars into the chromosomes of the genetic algorithm. Thus, the genetic algorithm evolves the grammars. To evaluate each individual, first the chromosome must be decoded to produce its grammar. The grammar is then used to construct a neural network. The network is then trained as usual and the ones with a lower error receive a higher fitness. Kitano's genetic algorithm used normal fitness-proportionate selection, standard multi-point crossover, and a modified mutation operator. Instead of randomly flipping a bit, the mutation operator replaced one symbol of the encoded grammar with another. Furthermore, the mutation rate was variable. The variation was dependent upon the similarity between the individual's parents. If the parents were quite similar, then the mutation rate was increased. Two dissimilar parents result in a child with a lower mutation rate. In this way, the genetic algorithm responds to a loss of population diversity by increasing the mutation rate. This technique was found to be effective on a number of relatively simple problems [Mit96].

Stanley and Miikkulainen [SM02c, SM02b, SM02a] provide an interesting alternative paradigm for combining genetic algorithms with neural networks. They propose a system which they call NeuroEvolution of Augmenting Topologies (NEAT). Their system evolves both the weights and the architectures of the neural network. The

NEAT chromosome contains node genes and connection genes. The connection genes refer to the node genes. There are a variety of mutation operators. The node weights are modified by a standard mutation operator. However, the connections and nodes themselves are modified by a pair of unusual mutation operators. The add connection mutation adds a connection gene which refers to a pair of nodes which were not previously connected. The add node mutation inserts a node in between two previously connected nodes. The old connection between the two original nodes is removed and a connection from each of the nodes to the new node is added. These mutations schemes allow the network to slowly grow and become more fully connected. The crossover scheme is even more unusual. In order for crossover to occur, Stanley and Miikulainen create an intriguing specifier for their genes called an innovation number. Whenever a new connection or node is created, they assign it a unique number. This number is inherited by all offspring that inherit those genes. When crossing over two chromosomes, genes that do not match up are inherited from the most fit parent, or randomly if the parents are equally fit. The goal of this method is to allow crossover without the need for any topological analysis. The NEAT method also uses speciation and explicit fitness sharing to preserve genetic diversity, most importantly topological diversity. Perhaps the most unusal thing about the NEAT algorithm is that the network populations begin as uniform networks with no hidden nodes. This encourages networks with a minimal number of nodes and weighted connections. In turn this makes the search for ideal weights much easier. The NEAT algorithm was found to be quite successful at solving the pole balancing problem.

3.4 GA evolved NN Parameters in Finance

El Shazley and El Shazley[SS99] use a genetic algorithm to evolve a neural network which is used to forecast currency prices for the British pound, the German mark,

the Japanese yen and the Swiss franc. They use a genetic algorithm to evolve a number of network parameters. The best individual is then fully trained on the data. An innovation with their approach was that this individual is then fed back into a genetic algorithm which is used to evolve the network weights using the weights of the trained network as a starting point. Their results showed marginal improvement over common methods of predicting currency prices.

3.5 Software used / Methodology

The genetic algorithm portion of this research relies heavily on a set of publicly available classes. These classes belong to the GAlib, version 4.5 which is available from the Massachusetts Institute of Technology. GAlib is available at http://lancet.mit.edu/ga/.

The neural network portion of this research consists of custom written code which implements the RPROP algorithm with a weight decay parameter. This code was written in C++.

This experiment was developed and run on a Pentium 4 PC under the Linux operating system.

Each of the paramters was represented by a real number. Thus, the chromosome representation was an array of real numbers, not a simple binary string. However, each of the parameters had a different bound on its range of allowable values. The genetic algorithm was used to encode the following parameters (with their bounds in parenthesis):

1. number of input nodes. (1–100)
2. number of hidden nodes. (1–100)
3. initial weight range. (0.0625–2.0)

4. initial step size. (0.0626–1.0)

5. maximum step size. (10–75)

6. weight decay parameter. (0–20)

These ranges were chosen to include the best values found in Chapter 2 as well as a sizable area on either side in order to find better possible values.

This choice of representaion leads to some interesting results. While standard roulette wheel selection and single point crossover were used the mutation operator was rather unusual. Since the representation was not a bit string, simply flipping a bit on the real number would not be very helpful. However, the GAlib offers a mutation operator which makes Gausian random modifications to the real number. Thus, additional possibly useful genetic diversity can be realized without the negative effect of almost certain uselessness.

The population size was limited to a very small number. This limitation was due to practical considerations. For each individual in the population in each generation the objective function trains a neural network for 1000 epochs. This is a computationally expensive process and thus the population size was kept small in order to limit the demand for computational resources.

Additionally, the mutation rate was set unusually high, 0.1. This choice was made because of the relatively small population size of twenty and the choice of real number representation. With a small number of initial individuals, the variety of any one of the parameters would be limited. Also, since the real number representation precludes the crossover operator from generating a new value for a parameter, a higher mutation rate was selected to restore the genetic diversity that had been lost because of the chosen representation. Alternatively, more sophisticated crossover and selections schemes may have provided other, possibly more effective methods for restoring this lost genetic diversity.

The neural network served as the objective function for the genetic algorithm. The process of evaluation of a chromosome was as follows:

1. Decode the parameters and instantiate a network using those parameters.
2. Train the neural network using the RPROP + weight decay algorithm and the evolved learning parameters for 1000 epochs.
3. Report the lowest total sum of squared error for both the training and testing data sets.
4. Fitness equals the inverse of the total error reported.

Note that the lowest total sum of squared error is returned, not that of the last epoch. Furthermore note that the returned error is the sum of the error on the training and testing data sets. When the sum is calculated, it is checked against the current low. If the new error is less that the previous low, the new low is saved along with a copy of the network at that point. Saving the sum of the error of both data sets should allow the genetic algorithm to find the network with the best performance without being concerned with over training.

After the genetic algorithm produced its fittest individual, a network was produced with the paramters coded therein and this network was trained until convergence. As stated above, during the training a snapshot of the network that produced the lowest error was retained. The algorithm then retrieved this low error network and used it to process the validation set.

3.6 Results

The results of the experiment can be seen in Table 3.1. The entry specified by PureNN is the best network result found in Chapter 2. The entry specified by GANN shows the results found with the genetic algorithm / neural network hybrid. It is

slightly disappointing that the genetic algorithm did not out perform the best network found by trial and error searching. However, it should be noted that the two networks are very close indeed and that the GANN outperformed the vast majority of networks whose error values can be found in Table 2.1 through Table 2.6. One possible explanation involves the difference between the genotype and the phenotype. The genetic algorithm encodes a genotype. This genotype decodes into a phenotype, an individual network. However, it does not decode directly into an individual. The initial weight range element of the genotype is used as the boundry points for a random number generation for each of the weights of the network. Thus, one genotype decodes to a vast number of phenotypes. This will, in general, produce only small differences in the fully trained network. However, it could produce a difference on the order of magnitude of what is observed in Table 3.1. It is possible that when the network was evaluated inside the genetic algorithm's objective function that a much more fortunate set of randomly generated initial weights led to a slightly more desirable outcome.

Table 3.1: Comparing NN with GANN

Description	Hidden Nodes	Weight Decay	Initial weight range	SSE	MSE	Mean Error
GANN	8	8.1472	-0.6211–0.6211	100601	117.11	10.82
NN	10	20	-0.25–0.25	96882	112.65	10.61

Chapter 4

Use a Genetic Algorithm to evolve the weights for a neural network.

4.1 Introduction

Commodity traders are interested in maximizing their profits. The most reliable way for one to do so is for the trader to have information that no one else has. Of course, insider trading is against the law. However, there is nothing illegal about analysing existing data to produce information that few if any others have obtained.

Technical analysts assert that the price of a financial asset, such as commodities futures, contains all of the information needed to predict the future value of the asset. They believe that any other information about the company or product in question is incorporated into the price. Assuming that this assertion is valid, one should be able to analyse the historical price data for a commodity and obtain information allowing one to better predict its future price.

Neural networks allow the analysis of complex, non-linear data. More complex and non-linear data simply requires a neural network with more complexity. This additional complexity is achieved by adding hidden nodes together with their connections to the other network nodes. Neural networks have a notable ability to obtain acceptable results despite noisy data. This makes them an especially interesting alternative for modeling financial time series data.

Though neural networks can prove to be a very useful and powerful modeling technique to obtain information about the non-linear data, they must be trained

in order to perform acceptably. Generally this training involves a gradient decent method for minimizing network error. However, gradient decent can easily become stuck at a local minima. Additionally, backpropagation and its variants require some number of parameters to be set. The learning algorithms are quite sensitive to changes in these parameters and the relationship between the parameters and network performance is poorly understood. Often they must be chosen by trial and error. This is time consuming and requires some level of sophistication of the user.

Genetic algorithms provide a method for searching broad, poorly understood solution spaces and maximizing (or minimizing) values for funciton parameters. By using a genetic algorithm to find the values for the weights that minimizes the error of the network, the modeling power of the neural network can be utilized without exposing the user to the laborious task of finding adequate learning parameters. This research constructs such a system and exmamines its performance.

4.2 Background on using a GA to evolve weights for NN

Many people have used genetic algorithms to evolve the weights for neural networks. Representation is perhaps the most important issue confronting the designer of such a system and the representations chosen may be roughly coupled into real valued and binary bit string encodings.

Montana and Davis[MD89] used neural networks to classify underwater sonar data. The network used was a fully connected feedforward neural network. The genetic algorithm chromosome consisted of vectors of real values that corresponded to the network weights. The genetic algorithm objective function was computed by reading the weight values off of the vector and assigning them to weights in the neural network. The network was then run on the training set and the sum of the squares of the errors for the training cycle was returned. The mutation operator

selected a node and for every incoming link added a random value between positive and negative one to the weight. The crossover operator took two parent weight vectors and randomly selected from which parent to copy the real valued weight. Montana and Davis compared the performance of their genetic algorithm learning algorithm with that of backpropagation and found that the genetic algorithm found better weight assignments in less time than did backpropagation.

There is a problem associated with the use of GAs for evolving ANNs. It is known as the permutation problem or the problem of competing conventions. The problem is that many permutations of a neural network are equivalent. That is, one hidden node can easily be swapped with another, including all of the connection weights, and the networks are exactly equivalent. Thus, the genetic algorithm is searching through a much larger solution space than is necessary. It is searching through a space that includes all of the permutations of the valid neural network solutions. This expanded search space results in degraded GA performance. There are sophisticated, specialized methods for preventing this problem. However, these techniques were not used in this research. Instead, the GA was allowed to run for a longer period of time in order to reduce the negative effects of this problem.

4.3 Software used / Methodology

The genetic algorithm portion of this research relies heavily on a set of publicly available classes. These classes belong to GAlib, version 4.5 which is available from the Massachusetts Institute of Technology. GAlib is available at http://lancet.mit.edu/ga/.

The neural network portion of this research consists of custom written software which implements simple feedforward neural network functionality.

The code for this experiment was written in C++ and developed and run on a Pentium 4 pc under the Linux operating system.

Representation is the main issue that must be decided when designing a genetic algorithm to evolve the connection weights for a neural network. There is some contraversy over whether real valued chromosomes are preferable or not. There seem to be a number of factors which encourage real number representation.

1. Many researchers have used real valued representation with good result.

2. Real valued representations are more efficient since there is no further decoding necessary.

3. Floating point processors make manipulation of real valued chromosomes quite efficient.

4. Real valued representations are easier to understand and program, reducing development time.

For these reasons, this research uses a real number representation. A real number representation is not without repercussions. The primary effect that a real number representation has is its degradation of the efficacy of the crossover operator to produce new values for the representation. Indeed, with a real number representation, crossover can only occur at the real number boundries. Therefore, the initial population of values at each location in the chromosome will not be altered with crossover. A further result is that mutation must play a larger factor in retaining and generating genetic diversity. To that end, a higher than normal mutation probablity, 0.1 was used.

The mutation operator must be modified as well. Using a binary representations bit flipping mutation operator would result in extremely drastic changes in weight values and thus in the individual's fitness. A more feasible solution was presented by

the GAlib software. It allowed the mutation operator to induce a gausian random alteration to the weight. This generally results in small weight change while allowing for the rare more substantial change.

To evaluate a chromosome, the objective function took the weight vector and instantiated a fixed architecture neural network with those values as the network's connection weights. One epoch of the training data was then fed through the network and the inverse of the sum of the squares of the network error was returned.

Since the fitness is evaluated only against the training set, over fitting the training data is a serious possibility. Thus, the best solution to the problem might not be the individual that the genetic algorithm finds as the fittest. However, given the high rate of mutation, it is likely that the best solution will be evaluated at some point as the algorithm evolves. Thus, each time the objective function evaluates an epoch of the training set, it also evaluates an epoch of the testing set. Each individual is compared to the current lowest error and if it produces a lower error than the current minimum, it will be saved as the best solution. When the genetic algorithm completes its evolution, the best solution is retained, not necessarily the individual with the highest fitness score. In a sense, this is a kind of meta-fitness. In any case, the best solution network was then used to process the validation data and generate the sum of squared error for that data set.

4.4 Results / Conclusions

The results of the experiment can be seen in Table 4.1. The entry specified by PureNN is the best network found in Chapter 2. The entry specified by GANN shows the results found by using a genetic algorithm to evolve the network parameters and then using RPROP learning technique to train the network in Chapter 3. The entry specified by GANN-Weight is the result found by this experiment. It is extremely

close to the best performing network found with a fraction of the man hours invested in finding it.

Table 4.1: Comparing NN with GANN and GANN-Weight

Description	SSE	MSE	Mean Error
GANN	100601	117.11	10.82
NN	96882	112.65	10.61
GANN-Weight	97910	113.85	10.67

Chapter 5

Summary

Three artificial intelligence approaches were taken to develoop a reliable method for predicting the futures price for a commodity, soybeans. All approaches used the same data for training, testing, and validation.

The first experiment proceeded along traditional neural network system lines. Parameters were selected by hand and varied to do a rough search to find acceptable parameters which would lead to a network with minimal error.

The second experiment introduced genetic algorithms and used their powerful search mechanisms to find optimal network parameters. This approach produced a network that performed on a very similar, if slightly inferior level to the previous network.

The third experiment alterred the use of the genetic algorithm. In this experiment, the genetic algorithm was used instead of a traditional learning technique to find optimal weights for the network. The result of this experiment was a network that performed even closer to the standard neural network's performance.

It is disappointing that neither of the genetic algorithm hybrids produced better results than the pure neural network approach. However, the performace of all three approaches is so close as to render any differences marginal.

The goal of this research is to find a preferred artificial intelligence approach to predict future values for commodities futures. To that end, other factors must be considered since the performance of all three approches is insufficently different to

render a clear winner. Two additional factors are obvious and seem to be important. The first is the number of man hours required for each approach. The pure neural network approach from Chapter 2 was very time intensive. It required a great deal of expert time to achieve this level of performance. The hybrid approches required little in terms of man hours other than writing the custom code used for each. Thus, the two hybrid systems score much higher on the ease of use test. The other additional factor which should be considered is the amount of computer resources required by the approach. The first approach used clock cycles while running, but no more than any math intensive program would do. The GANN-Weight approach took an order of magnitude fewer clock cycles than did the GANN approach that used RPROP. Thus, by combining all three criteria, the clear winner and preferred artificial intelligence approach to predicting future commodity futures prices is the GA-Weight approach.

Bibliography

[And95] James A. Anderson. *An Introduction to Neural Networks*. MIT Press, 1995.

[BJ76] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: forecasting and control*. Holden-Day, Oakland, CA, 1976.

[BKM98] Zvi Bodie, Alex Kane, and Alan J. Marcus. *Essentials of Investments*. Irwin Mcgraw-Hill, third edition, 1998.

[BLB92] W. A. Brock, J. Lakonishok, and B. Le Baron. Simple technical trading rules and the scholastic properties of stock return. *Journal of Finance*, 27(5):1731–1764, 1992.

[Cas00] Filippo Castiglione. Forecasting price increments using an artificial neural network. *Advanced Complex Systems*, 1:1–12, 2000.

[CWS95] Jr. Clifford W. Smith. Corporate management: Theory and practice. *The Journal of Derivatives*, 1995.

[DGE93] Z. Ding, C. W. J. Granger, and R. F. Engle. A long memory property of stock market returns and a new model. *Journal of Empirical Finance*, 1(1):83–106, 1993.

[GG00] René Garcia and Ramazan Gençay. Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics*, 94:93–115, 2000.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search Optimization & Machine Learning*. Addison-Wesley, 1989.

[HLP94] James M. Hutchinson, Andrew Lo, and Tomaso Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. Technical Report AIM-1471, Massachusetts Institute of Technology Artificial Intelligence Laboratory and Center for Biological and Computational Learning, 1994.

[Hul97] John C. Hull. *Options, futures and other derivatives*. Prentice-Hall, Inc., Upper Saddle River, NJ 07458, third edition, 1997.

[Hul98] John C. Hull. *Futures and Options Markets*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, third edition, 1998.

[Key78] John Maynard Keynes. *Treatise on Money*. Cambridge University Press, April 1978.

[Kit90] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[LF95] Gerson Lachtermacher and J. David Fuller. Backpropagation in time-series forecasting. *Journal of Forecasting*, 14(4):381–393, 1995.

[MD89] David Montana and Lawrence Davis. Training feedforward networks using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1989.

[Mit96] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.

[MTH89] Geoffrey Miller, Peter Todd, and Shailesh Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Converence on Genetic Algorithms*, pages 379–384, 1989.

[RB93] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, 1993.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[Sav89] R. Savit. Nonlinearities and chaotic effects in option prices. *Journal of Futures Markets*, 9:507–518, 1989.

[Sch90] J. A. Scheinkman. Nonlinearities in economic dynamics. *Economic Journal*, 100(400):334, 1990.

[SM02a] Kenneth 0. Stanley and Risto Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02)*, 2002.

[SM02b] Kenneth 0. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*. Morgan Kaufmann, 2002.

[SM02c] Kenneth 0. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[SS99] Mona R. El Shazly and Hassan E. El Shazly. Forecasting currency prices using a genetically evolved neural network architecture. *International Review of Financial Ananlysis*, 8(1):67–82, 1999.

[TD92] Robert R. Trippi and Duane DeSieno. Trading equity index futures with a neural network. *The Journal of Portfolio Management*, 19(1):27–33, 1992.

[Wal01] Steven Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems*, 17(4):203–222, 2001.

[YTP99] Jingtao Yao, Chew Lim Tan, and Hean-Lee Poh. Neural networks for technical analysis: A study on klci. *International Journal of Theoretical and Applied Finance*, 2(2):221–241, 1999.