Stock Market Investing Using Neural Networks

Matthew Bodenhamer MBODENHAMER@OU.EDU
University of Oklahoma

Abstract

The fundamental hypothesis of this paper is that certain trends exist in stock market data that, if properly recognized, could be used to invest well on certain stocks. This paper details an attempt to apply neural networks to stock market investing, in order to create an agent that can learn to invest profitably in certain stocks. To accomplish this, a classification scheme is developed which maps certain stock closing price patterns to future movements in the price of the stock. A neural network is trained on this data, with the hope of identifying certain patterns in the data not obvious to human analysts. Finally, the output of the neural network is coupled with high-level decision rules to direct the agent when to buy or sell for given input data. As a result of these efforts, an agent is developed which makes a profit when investing upon the stocks given it in this project.

1. Introduction

Stock market investing is a common venture undertaken by Americans and others around the world. Yet, despite its ubiquity, the stock market is hardly understood, and investments made by the average person are usually little more than proverbial "shots in the dark," resulting in either gains or losses with seemingly random regularity. The question has been posed, and still remains, as to whether the movements of the stock market can be predicted, and if so, whether or not these predictions can be used to make profitable investments in the market. Some economists theorize that even if certain trends could be identified, that the market would quickly compensate for their use, rendering little advantage to the investor. Such ideas are embodied in the Efficient Market Hypothesis (Lawrence, 1997). These notions are not discussed in this paper. Rather, the technical details of identifying and properly using trends in the closing price of stocks are the focus of this paper, with the assumption that the trends, if identified and used, could be used to full advantage in the process of investing in the market. So to this end, an agent was developed, which uses a neural network to identify trends in test data and then apply it to test data in order to make good investing decisions. The design and development of this agent are the main contents of this paper, along with the results of its performance on test data.

2. Problem and Solution Overview

As previously stated, the problem addressed by this paper is that of attempting to invest "well" in the stock market by identifying certain trends in stock price, and putting them to good use. For this particular project, the input domain was restricted to historical stock data. Therefore, the process of investing involved running a simulation over some historical data.

The process of investing is defined as follows. First, an arbitrary agent is given a set of stock market data. The data is a list of closing prices of that particular stock, where each data point corresponds to a particular day that the stock was traded. The data is organized in chronological order so that the first data point corresponds to the first day the stock was traded, and the last data point to the last day of trading. For example, the data set for Google's stock runs from 08/19/04 to 09/05/06, and also includes all of the days in-between on which the stock was traded. After receiving the data, the agent is given some initial capital, and performs a simulated investment run, starting at the first data point, and ending at the last. At each data point, the agent makes a decision to sell, buy, or do nothing, based on that particular agent's trading rules. After the run is completed, the profit is calculated, which determines the success of the run. In addition, the agent is forbidden to go into debt, meaning that it cannot buy more stock than it has capital enough to purchase. To model real-world transactions, the agent is also charged a fee of 0.9 cents per gross profit on a transaction.

Three agents were developed to operate in this domain, with the goal of making a positive profit on any given simulation. The first agent, also the main subject of this paper, is the neural network agent, which is described in detail below. The other two agents were constructed as benchmarks, for the purpose of gauging the performance of the neural network agent. The first benchmark agent is a random investor, which buys or sells at random data points. The second benchmark is a heuristic that attempts to model some human investing methods, by buying at local increases in price, and selling at local decreases. The description of the design, development, and testing of these agents will henceforth occupy the rest of the paper.

3. Implemented Agents

3.1 Random Agent

The algorithm for the random investor is agent is relatively simple. At each data point, a random number is obtained. Based on this number, the agent will buy some shares 10% of time, sell some shares 10%, and do nothing the other 80%. To be consistent with the constraints upon the problem domain, the agent will do nothing if it is given the order to sell, and currently has no shares, nor will it do anything if ordered to buy when it has no capital with which to purchase stock. This agent provides the lower bound on performance, in the sense that any somewhat intelligent agent should at least perform better than random.

3.2 Heuristic Agent

The heuristic agent attempts to mimic a short-sighted approach to investing that humans commonly use, in the author's experience. Basically, if the stock price has been increasing for the last few days, the agent assumes that it is on the up-and-up and decides to buy some shares. Conversely, if the price has been decreasing over the last several days, then the agent assumes that the stock isn't doing so well, and sells some shares to cut its losses. Many human investors mimic this behavior in some form, buying stock when they think it is going to increase in price, and selling when they think they might lose money by holding on to the stock.

The heuristic can be expressed formally as follows. For some data point i, let v_i be the value of that particular data point, as calculated by the heuristic. Then,

$$v_i = \frac{1}{w}\sum_{k=i-w}^{i-1} D[k] - D[i-w]$$

In the above equation, w is the window size, and D is the array of data points. The index k refers to a trading day, and $D[k]$ is the closing price of the stock on day k. As one can see, v_i is nothing more than a linearly adjusted average of the w previous data points. If $v_i > l_u$ then the agent buys some shares, and if $v_i < l_l$, then the agent sells some stock. In this project, the upper limit, l_u, was set to \$5, the lower limit, l_l, was set to –\$5, and the window size, w, was set to 5. These parameters could be fine-tuned to change the behavior of the agent. For instance, to make the agent more cautious about transactions, the absolute value of the upper and lower limits could be increased. The window size could also be increased to make the agent account for a greater amount of history in the decision-making process. This approach is flexible in that it allows for buy/sell decisions to be made on data that could be noisy, as long as it is increasing or decreasing on average by enough of an amount. However, it is conceivable that some data patterns have an average that does not correspond to their direction of movement, i.e. a pattern that is moving down, but has a large "spike" that causes it to have a positive average. Such patterns are possible in theory, but for a window size of 5, are extremely rare in stock market data in the author's experience. In most all cases, data that is noisily moving up has a positive average, and vice versa.

3.3 Neural Network Agent

The Neural Network (NN) agent was designed for the purpose of extracting patterns of stock price fluctuation from a set of training data, and using that same data to invest intelligently on a set of test data. Unlike the benchmark agents, the goal of the NN agent is to use what it has learned to "guess" what the market will do next from any given data point, and then to invest appropriately given this guess. The task of designing the agent fell into four steps: designing a classification scheme, deciding upon an appropriate neural network architecture, implementing an effective training scheme, and defining high-level decision rules.

3.3.1 Classification Scheme

The intent of the classification scheme is to map a set of data points to the stock's subsequent movement. So, for some data point i, the previous w_{pre} points are considered. The average of these points is taken, and denoted as Avg_{pre}. Next, the w_{post} points following the current data point are considered. The average of these points is taken, and denoted as Avg_{post}. Then, the vector containing the points $D[i\text{-}w_{pre}]$ to $D[i\text{-}1]$ is mapped to the difference of Avg_{post} and Avg_{pre}. In this way, a set of data points is mapped to the relative future change in the data. In this project, w_{pre} and w_{post} were set to 20.

Formally, the classification can be described thus. A classified example, e_i, is the example corresponding to an arbitrary point i in the data set. Symbolically, e_i is a pair, $\langle \mathbf{x}_i, v_i \rangle$, where $\mathbf{x}_i$ is a vector, and v_i its corresponding value, according to the following equations. First, $\mathbf{x}_i$ is a vector such that $|\mathbf{x}_i| = w_{pre}$. For an integer index variable k, $\mathbf{x}_i[k] = D[i - w_{pre} + k]$, where $0 \le k \le w_{pre} - 1$ and D represents the data set. The value of v_i is as follows:

$$v_i = Avg_{post} - Avg_{pre}$$

$$v_i = \frac{1}{w_{post}}\sum_{k=i+1}^{i+w_{post}} D[k] - \frac{1}{w_{pre}}\sum_{k=i-w_{pre}}^{i-1} D[k]$$

The classification occurs, then, for all data points i such that $w_{pre} < i < |D| - w_{post}$. The classifier iterates across the data set, producing an example for each point i that falls within the given bounds. As the iteration occurs, the vector $\mathbf{x}_i$ forms a colloquial "moving window." This method of classification was developed by the author of this paper, and is novel as far as he knows. This classification approach is similar to one used in

Chenoweth & Obradovic (1996), in the sense that it is also a moving-window approach. However, the system referenced in this paper employed a somewhat different classification scheme, leveraging an ensemble of different statistical classification methods to produce an example set.

3.3.2 Neural Network Architecture

The NN agent uses a multi-layer perceptron network. For general information on artificial neural networks, the reader is directed to Mitchell (1997). The number of inputs to the network is equal to w_{pre}, the size of the classifier's moving window. Thus, the input to the neural network, for a given data point, is the previous w_{pre} data points. This approach is not universal, however. In Yao and Poh (1995), an input configuration is used which utilizes several statistical calculations performed over recent data that attempt to convey certain characteristics of the data, as opposed to just using the data itself. Several examples include the moving average over the previous 10 nodes, the moving average over the previous 50 nodes, and the difference in closing prices over the previous 5 days. The goal of the network used in this project is to detect trends in data not necessarily known to human observers, rather than attempting to isolate certain trends beforehand, on which to train.

The output of the network is a single node, which produces a real-valued number indicating the direction the network thinks the market will go in the near future; positive values indicate an expected increase in price, and negative values indicate an expected decrease. In addition, the absolute value of the output also indicates the amount of expected change, so an output of +15 would indicate a greater expected increase than +5. To accomplish this kind of operation, linear input and output units are used, along with hidden nodes using *tanh* as an activation function.

In the course of performing many experiments, it was determined that the network performed best, in terms of smallest error, for the classified data when the number of hidden nodes, H, approximately satisfied the following relation, with $k = 10$, and E_{given} the set of examples given to the neural network, on which to train:

$$H = k \cdot \left| E_{given} \right|$$

This is most likely due to the fact that the data is not linearly separable and correct correlation to certain output values requires that many factors be considered. In any case, the network did not perform well for values of $k < 10$, in many cases not even learning to differentiate between widely separate inputs. Through experimentation, it was determined that training times for networks in which $|E_{given}| > 20$, while following the above relation, were intractable. So, $|E_{given}|$ was set to 20 in this project, the number of hidden nodes was thus set to 200. Several different topologies were considered over the course of the project, including using several layers of hidden units. In the end, however, the network of 200 hidden nodes experimentally performed the best. In an interesting side note, one paper found that networks with only one hidden layer are generally more accurate than those with two or more when training on stock market data, although this finding did not really influence the architecture used in this project (Egeli, Ozturan, & Badur, 2003).

3.3.3 Training Scheme

The neural network of the NN agent was trained using backpropagation. An interesting alternative method of error minimization is discussed in Hochreiter and Schmidhuber (1997). Although this method, which attempts to identify large flat regions in the error space, shows some promise, implementing it proved to be beyond the scope of this project.

In general, the neural network in the NN agent is trained over an example set E, where $E = \{e_1, e_2, \ldots, e_n\}$, where each e_i is of the form $e_i = \langle \mathbf{x}_i, v_i \rangle$, where $\mathbf{x}_i$ is a vector of length w_{pre} and v_i is a real-valued number. The network performs some number of training iterations over E, where in each iteration, backpropagation is performed for each $e_i \in E$. Once again, the reader is referred to Mitchell (1997) for the details of backpropagation.

This general approach, however, did not work. In the course of many experiments, it was determined that the network could not learn properly if two different training examples e_i and e_j contained vectors such that $\mathbf{x}_i[k] \approx \mathbf{x}_j[k]$ and $v_i \neq v_j$, where k is some index variable denoting a specific element in the vector. More specifically, it was found that the network had trouble learning if any $|\mathbf{x}_i[k] - \mathbf{x}_j[k]| < 0.5$ and $|v_i - v_j| > \varepsilon$, $\forall i,j$ such that $1 \leq i,j \leq n$, $i \neq j$, where $n = |E|$, and some small ε. To account for this issue, the training set E was modified in the following manner prior to training. First, each $\mathbf{x}_i$ is modified, such that each element $\mathbf{x}_i[k]$ is rounded to the nearest integer. Let these modified vectors be denoted $\mathbf{x}_i'$. Next, each $\mathbf{x}_i'$ is stored, with its corresponding v_i, in a table, T. A scan is performed on T, and identical vectors $\mathbf{x}_i'$ are noted. Next, for each class of identical vectors $\mathbf{x}_i'$, a corresponding v_i' is found, which is the average of all of the v_i's of the vectors $\mathbf{x}_i'$. Formally, if a certain class contains m identical vectors, $\mathbf{x}_{i1}', \mathbf{x}_{i2}' \ldots \mathbf{x}_{im}'$ then v_i' would be:

$$v_i' = \frac{1}{m} \sum_{l=1}^{m} v_{il}$$

Finally, the modified $\langle \mathbf{x}', v' \rangle$ pairs are collected into a new set of the form $\{e_1', e_2', \ldots, e_r'\} = E'$, which is called the *reduced example set*, where $r \leq n$. The reduced example set can be successfully trained on the neural network, since it satisfies the property that all corresponding elements of different vectors $\mathbf{x}_i$ are at least 0.5 apart. To produce E_{given}, some $|E_{given}|$ elements are taken from E' and collected to form a set. In the implementation of the NN agent, elements are drawn

from T in a semi-random fashion, such that the order of E' does not resemble E. From this, the first $|E_{given}|$ elements of E' can be selected to form E_{given}. Due to the semi-random order of E', E_{given} is likely contain a diverse set of examples, and since w_{pre} is relatively large, the examples in E_{given} should also mostly "cover" the data set, in the sense that a good portion of the data points in D are present in the $\mathbf{x}_i$'s of the examples in E_{given}. Thus, even though E_{given} may be relatively small, its size should not adversely affect the ability of the neural net to find the desired patterns in the data by a great amount. This conclusion is supported by the results in section 4.3.

This approach to reducing the example set was developed by the author, and is novel as far as he is aware. Chenoweth & Obradovic (1996) also note the importance of reducing the example set, eliminating non-relevant and noisy data, grouping the rest into well-separated classes. They suggest the usage of several sophisticated statistical models to perform this reduction. Although the approach developed here is more naïve, it attempts to accomplish something similar, by separating the examples out into classes so that the neural network can learn effectively.

Through experimentation, the leaning rate, α, was set to $5x10^{-8}$ and the momentum, μ, was set to 0.9. It was also found that at least 1000 training iterations need to be run for the neural net to perform to any degree of accuracy on test inputs provided after training.

3.3.4 High-Level Decision Rules

When performing a simulation, the agent needs to have a method by which to decide whether or not to buy, sell, or do nothing at a given time. For a given data point i in a simulation, let $\mathbf{x}_i$ be a vector containing the previous w_{pre} data points. Also let $F(\mathbf{x})$ be the output of the neural network. Then, at each data point i encountered in the simulation, $F(\mathbf{x}_i)$ is computed. At this point, a decision is made. If $F(\mathbf{x}_i) > L_u$, an upper limit, then the decision is made to sell. Conversely, if $F(\mathbf{x}_i) < L_l$, a lower limit, then the decision is made to buy. Nothing is done for all other values of $F(\mathbf{x}_i)$. This particular set of decision rules attempts to impart buy low, sell high behavior, by buying stock when the price is expected to drop, and selling when it is expected to rise. In the implementation of the agent L_u was set to \$10, and L_l to -\$10.

4. Experiments and Results

Two sets of experiments were run to test both the NN and benchmark agents: investing on Google stock (GOOG) and investing on eBay (EBAY). GOOG and EBAY have different features, so investing on both should more accurately reveal the character of the agents being tested. In this regard, GOOG is a "nicer" stock, as it starts out at a relatively low price, increases over time, and then levels off after a slight decrease. This is almost perfect for investing, as it gives an agent a great opportunity to buy low, sell high, and make a profit. EBAY, on the other hand starts out by rapidly climbing to a high price, then decreasing and approximately leveling out. This stock is not so friendly for investing agents, as there is not much room to buy low and then sell at a higher price. One can still make a profit, but it is not as easy as on GOOG. All three agents were tested by investing on GOOG, then investing on EBAY. The results and methodologies of these experiments for each of the agents are detailed in the following sections.

4.1 Random Agent

The random agent was simulated over GOOG for 100 runs. In each run the agent was given \$1000 initial capital. Also, due to the stock's high average closing price of \$297, the agent was allowed only to buy or sell one share per data point. The profit of each of these runs is shown below, in Figure 1.

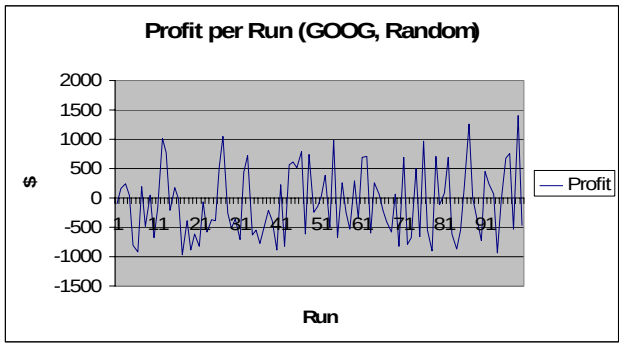

Figure 1. Profit per run over GOOG for the random agent.

The average profit over these runs is -\$84.07. For a random agent, this seems to be pretty good performance, and is most likely due to the investor-friendly features of GOOG mentioned above.

For the second experiment, the random agent was simulated over EBAY for 100 runs. As with the previous experiment, the agent was given \$1000 initial capital for each run, and was also restricted to only buying or selling one share per data point. The results of these runs are shown below in Figure 2.

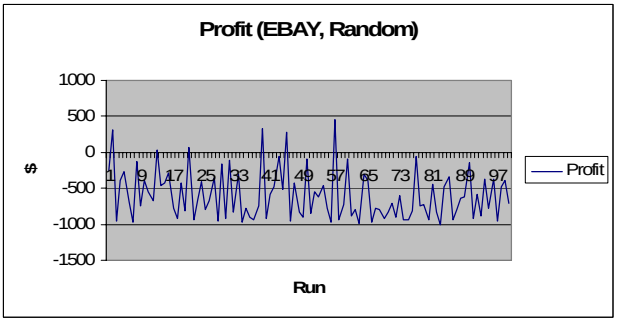

Figure 2. Profit per run over EBAY for the random agent.

The average profit over these runs is -\$595.51. This isn't nearly as good as the results over GOOG, and is

consistent with the assertion that EBAY is not as "nice" as GOOG for investing. These results are less than ideal for a human investor; one would at least like to make a profit! Therefore, it seems safe to say that these results form a lower bound on performance for the agents, and that the other more intelligent agents should perform better.

4.2 Heuristic Agent

The heuristic agent was then run over GOOG. Like the experiments involving the random agent, the agent was given $1000 initial capital, and allowed to only buy or sell one share per data point. Since the heuristic is deterministic, it is only necessary to perform one run to gauge the agent's performance. The results of the heuristic run are shown below in Figure 3.

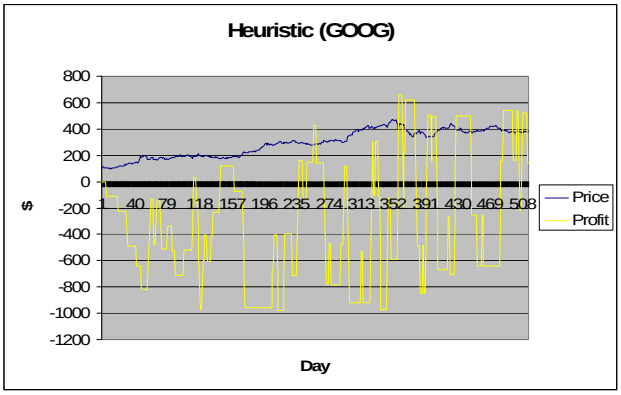

Figure 3. Simulation run of heuristic agent over GOOG.

The profit for this run was $141.12. One can observe that the agent buys stock, decreasing the profit, when the stock price increases, and sells, increasing the profit, when the price decreases. The "good" features of GOOG are visible in this graph, and the investment pattern of the heuristic follows, resulting in a respectable profit.

For the next experiment, the heuristic agent was simulated over EBAY. The same rules applied, with $1000 initial capital and the limit of one share bought/sold per transaction.

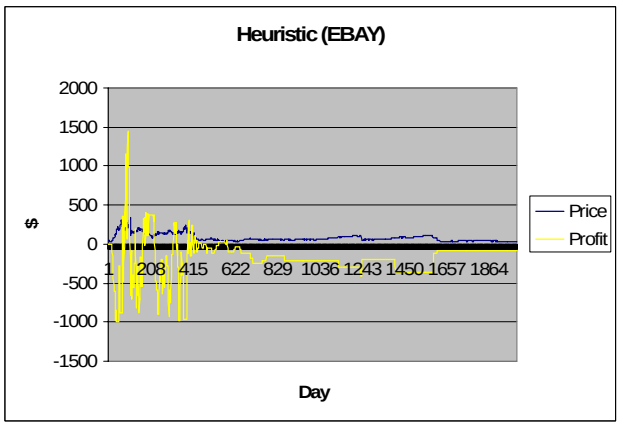

Figure 4. Simulation run of heuristic agent over EBAY.

The profit for this run was -$82. If the agent were to stop early in the simulation, it might make a nice profit, but the price of EBAY quickly drops and the agent loses money. The price never recovers, and neither does the agent. Despite, the "bad" features of EBAY, the agent still has a respectable performance, ending the simulation with a minor loss.

As expected, the heuristic agent performs better than the random agent. Nevertheless, one would hope for the NN agent to perform better than the heuristic, and make a profit over both stocks. Thus, the heuristic agent provides another bound for performance, one which the NN agent should hopefully surpass.

4.3 Neural Network Agent

Two sets of experiments were performed on the NN agent, the difference lying in the training method used. First, since GOOG was determined through visual inspection and from the results of the benchmark heuristics to have "good" features, it was determined that some $|E_{given}|$ examples of GOOG should be used to train the NN agent before being tested on GOOG and EBAY. Since GOOG's features are "good," one would expect the agent to perform well testing on GOOG, and possibly other stocks. In order to prevent overfitting, the use of two common methods was devised, which in turn led to two different sets of experiments. The first method involves stopping training after a set number of iterations. For this project, training was stopped after 5000 iterations. The second method involves using a validation set, disjoint from the training set, to determine when to stop training. When the error from computing values in the validation set starts increasing beyond some threshold, then one can conclude that the network is starting to learn specific idiosyncrasies in the training set, and stop the training. So, in both sets of experiments, the NN agent was trained on a small set of examples drawn from GOOG's reduced example set, and tested on both GOOG and EBAY. After, this, the same experiments were performed by training on EBAY, then testing on both GOOG and EBAY. Since EBAY's features, are not as "good" as GOOG's, one would expect performance to be worse overall, especially when testing on stocks other than EBAY. As with the other experiments, the agent was given $1000 initial capital, and limited to buying or selling on share per data point. The size of the training set was placed at 20 for both sets of experiments, and the size of the validation set was placed at 20 for the second set. Also, since the neural network weights are randomly initialized, there is some room for variation in the network output after training has completed. To account for this, five runs of each class of experiment were performed.

4.3.1 Set Number of Iterations

After training on the given example set the agent averaged a profit of $1269.48 on GOOG and a profit of $887.47 on EBAY.

Table 1. Profit on GOOG and EBAY using a neural net trained on GOOG with a fixed number of iterations.

Run	GOOG Profit ($)	EBAY Profit ($)
1	1258.38	895.46
2	1268.62	989.46
3	1261.76	900.28
4	1251.52	794.94
5	1307.10	857.22

From this data, the NN agent appears to have done extremely well using this method. Even more indicative of success is the fact that the agent used trends learned from GOOG to invest successfully in EBAY, a data set unrelated to the training set in any way. This seems to support the hypothesis certain trends can be identified and used to invest profitably in certain stocks.

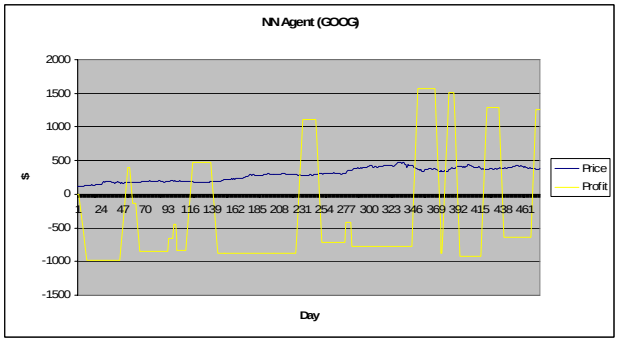

Figure 5. Individual simulation run of NN agent over GOOG

The above figure shows one of the runs performed by the agent over GOOG. Many of the spikes in profit occur after a peak, indicating that the agent has learned some buy low, sell high behavior. This once again supports the initial hypothesis of the paper.

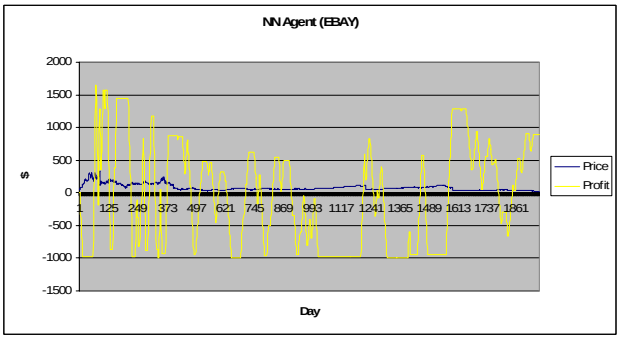

Figure 6. Individual simulation run of NN agent over EBAY.

The above figure shows one the runs performed over EBAY. Although the profit is a bit noisier than the GOOG run, the agent still appears to exhibit buy low, sell high behavior, as profit spikes appear after local maxima. It appears that this training method produces an agent that learns to invest profitably, at least on these stocks.

Next, the NN agent was trained on EBAY and tested on both GOOG and EBAY. In simulation, the agent averaged a profit of $51.38 on GOOG and a profit of $579.23 on EBAY.

Table 2. Profit on GOOG and EBAY using a neural net trained on EBAY with a fixed number of iterations.

Run	GOOG Profit ($)	EBAY Profit ($)
1	240.20	566.53
2	131.70	573.10
3	-29.45	670.09
4	-73.16	555.62
5	-12.38	530.80

From this data, the NN agent appears to have done reasonably well given EBAY data on which to train. As can be expected, the agent performed better or EBAY than on GOOG, since it trained on examples from the former. Nevertheless, despite the "bad" features of EBAY, the agent still makes a profit, on average, investing on GOOG. Thus, it still seems to have identified some trends in EBAY that are also useful on GOOG. For the sake of brevity, graphs of individual runs of this sort will not be presented here.

4.3.2 Using Validation Set

Similar results were achieved using a validation set to limit the number of training iterations. In the five runs comprising this experiment, the agent netted an average profit of $814.32 on GOOG, and made $821.61 on EBAY.

Table 3. Profit on GOOG and EBAY using a neural net trained on GOOG using a validation set.

Run	GOOG Profit ($)	EBAY Profit ($)
1	947.91	872.58
2	662.74	854.85
3	962.25	810.97
4	581.18	798.73
5	917.52	771.16

Although these numbers are slightly lower than those in the previous experiment, the agent still performs well, even doing better on EBAY than GOOG on a couple of

occasions. Once again, this data seems to suggest that the agent has learned some trends that it can put to good use in investing.

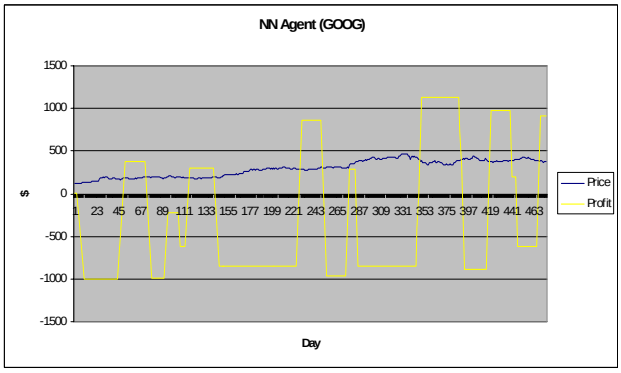

Figure 7. Individual simulation run of NN agent over GOOG.

As in the previous experiment, this agent appears once again to exhibit buy low, sell high behavior, which is the intended result. Although this run is not as profitable as its counterpart from the other experiment, its general pattern is similar, and indicates that the same kind of learning has taken place.

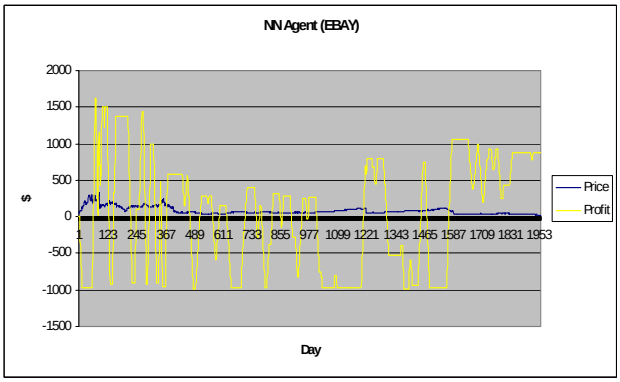

Figure 8. Individual simulation run of NN agent over EBAY.

This graph exhibits noise, much like its counterpart in the previous experiment, but it also seems to exhibit the desired buy low, sell high behavior, as major increases in profit occur after decreases in price. Once again, it appears that the agent is learning properly and validating the initial hypothesis that an agent can learn to invest profitably in stocks.

As in the previous set of experiments, the NN agent was then trained on examples from EBAY, and tested on both GOOG and EBAY, averaging a profit of -$44 on the former, and of $577 on the latter.

Table 4. Profit on GOOG and EBAY using a neural net trained on EBAY using a validation set.

Run	GOOG Profit ($)	EBAY Profit ($)
1	-28.17	624.22
2	-54.18	573.75
3	-45.18	585.24
4	-45.18	553.33
5	-47.56	548.59

In this experiment, the agent performed about the same as its counterpart in Table 2 for investing on EBAY. However, the agent performed worse, by about $95 on average, investing on GOOG. Despite the decrease in success, this is still better than the average result of the random heuristic. Thus, in this case, the NN agent seems to be affected more by the "bad" features of EBAY on which it trained, and thus performs worse on other stocks. One could conclude that the agent still found enough good patterns to prevent a large loss while investing on GOOG.

5. Future Work

Although the NN agent performed relatively well, there is still work that can be done to possibly improve the agent. One area that could be improved is in the area of the classification scheme and neural network architecture. If a way could be found to bring down the number of hidden nodes per training example and still retain good performance, then training times could be reduced and more examples incorporated into the training set, possibly improving performance in the simulation.

Another way to improve the agent would be to incorporate more information into the inputs of the neural nets. The behavior of the stock market is influenced by a number of external factors, and one would think that including as many of these as possible would improve performance as well. One interesting approach along these lines was taken in Wuthrich, et al. (1998), in which textual data from news websites was considered along with numerical data to make predictions. Although this project might not benefit from exactly reproducing this paper, implementing something along these lines might help improve performance.

6. Conclusion

This paper details an attempt to build a neural network-based agent to invest profitably on certain stocks. Using a certain combination of classification scheme, neural network architecture, and high-level decision rules seems

to have produced an agent capable of investing well on stocks, at least on GOOG and EBAY. The initial hypothesis of the paper, that certain salient trends can be identified and used to invest well, is supported by the results obtained by these experiments. Whether or not this hypothesis holds for all stocks, or whether a system like this could be implemented for use in the real world, is a question beyond the scope of this paper, and an unanswered one at that. What this paper has shown, however, is that neural networks can be used to invest profitably on certain stocks, and perhaps, with the right combination of design, experimentation, and luck, one might be able to produce an agent capable of performing extremely well on all stocks in general, reaping huge profits for whoever was fortunate enough to have access to its predictions. At this point, however, the future remains to be seen.

References

Chenoweth, T., & Obradovic, Z. (1996). A multi-component non-linear prediction system for the Sp 500 Index. *Neocumputing J. 10*(*3.3*), 275-290.

Egeli, B., Ozturan, M., & Badur, B. (2003). *Stock Market Prediction Using Neural Networks.* Retrieved September 11, 2006 from the Hawaii International Conference on Business website: http://www.hicbusiness.org/biz2003proceedings/Birgul%20Egeli.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, *9*(*1*), 1-42.

Lawrence, R. (1997). *Using Neural Networks to Forecast Stock Market Prices*. Retrieved September 11, 2006 from: http://www.cs.uiowa.edu/~rlawrenc/research/Papers/nn.pdf

Mitchell, T.M. (1997). *Machine Learning*. New York: McGraw-Hill.

Wuthrich, B., et al. (1998). Daily stock market forecast from textual web data. *IEEE International Conference on Systems, Man, and Cybernetics. 3*, 2720-2725

Yao, J., & Poh, H.L. (1995). Forecasting the KLSE Index Using Neural Networks. *IEEE International Conference on Neural Networks 2*. 1012-1017