Predicting Bitcoin Prices with a Ruby Artificial Neural Network

Sara Bocabella (snb42), Ryan Kass (rck72)

December 11, 2012

1 Abstract

To investigate the Efficient Market Hypothesis and to look at the effects and properties of neural networks, we investigated the feasiblity of predicting financial information using an Artificial Neural Network. We wished to find if we could predict the bitcoin USD price by modeling the price fluctionation with a neural network and by tuning properties of the network to optimize network preformance. Our method involved creating a feedforward neural network that used back-propagation, the details of which can be found in the Method section. By evaluating Related Work, we were able to see different approaches used by other researchers and to consider different approaches such as suggestions as to which variables to experiment over and which to keep constant, as well as suggestions on the structure of the network. The System Architecture involved the interactions of Node, Network and other classes, with most of the computation occuring in Node (the propagation, error and weight adjustment) while Network simply helped to iterate over the network and the other files provided supporting methods, such as automated ways of supplying input and calculating error. We then give an Experimental Evaluation, discussing the Methodology as well as evaluation techniques, the results found and a discussion of these results in the context of the related work mentioned earlier. Future work suggests areas for improvements, such as varying the transition function, adding algorithms to find the independant components and other techniques to possilby improve the network. To conclude, we maintain that while our network did not produce the expected results, based on related work this approach is feasible and could be implemented if more advanced techniques were included in the algorithms.

2 Introduction

Neural networks have been successfully used to model commodity pricing data and even to refute the Efficient Market Hypothesis through their ability to generalize commodity data into non-linear, stochastic functions previously thought to be entirely stochastic and non-generalizeable [1]. The commodity we chose to test our network on, Bitcoin, was established in 2009 and has experienced massive volatility in its price since its inception. We chose to use Bitcoin precicely for this reason, and the fact that it is an immature market not yet flooded with banks and finance firms controlling market movements.

For three months, we have been collecting hourly data on Bitcoin, recording ask volume, bid volume, last price, and the spread between the highest bid and the lowest asking price. These four features are the input to the network, which then outputs a guess of the price twenty-four hours after the input data. The network was able to learn to predict the price with an average error of 49¢, compared to an average twenty-four hour price fluctuation of 37¢.

3 Problem Definition

Can an artificial neural network constructed in ruby be trained to accurately predict the price in USD of a bitcoin twenty four hours in the future. The network is trained on approximately 3,000 examples. Each example contains four data points, last price, difference between highest bid and lowest ask, amount of bitcoins up for sale, and amount of bitcoins asked for. The output is a floating point which represents the predicted price twenty four hours from now. By varying network architecture, such as the number of nodes per layer, the number of hidden layers in the network, and the learning rate in the network we wished to see if we could optimize preformace of the network.
As a subtask, we set out to implement a highly flexible neural network that can be initialized with any number of inputs, hidden layers, nodes per hidden layer, and output nodes. This, along with a gui would allow for easier testing and presentation of the data and results we obtained.

4 Method

The network we implement is feed-forward, and can act like a perceptron or have multiple layers, depending upon how it's initialized. We use backpropagation to adjust the weights. When the network is run, the input layer gets its input and propagates that input to the next level without feeding it through any transition function. The nodes on the higher layers then have weights associated with each of their inputs, as well as a transition function which is defined as $\frac{1}{1+e^{-x}}$. Finally, at the output layer there are weights associated with every input from the most upper hidden layer, or in the case of a perceptron, the input layer. Since we are trying to obtain a real valued output, the top layer is not fed through a transition function. At the outset, all weights are randomly initialized. At each example they are adjusted according to what the price

[1] Ramon Lawrence, Using Neural Networks to Forecast Stock Market Prices 1997

was twenty-four hours after the time the data they analyzed was recorded. This is done through standard backpropagation. At the top layer, the error is the raw difference between what the price actually was and what it was predicted to be. Each node not in the top layer calculates its error in a composite manner. From each node j to which node i outputs synapse:

$$error_i \mathrel{+}= error_j * weight_{i,\, j} \qquad .$$

This process of error calculating is propagated all the way back down to the first hidden layer (the input nodes have no associated weights). After each training iteration the errors are reset back down to zero. Next, the weights must be adjusted using

$$w = w + \Delta \text{ where } \Delta \text{ is defined as:}$$

$$\Delta = derivative * error * step * output$$

The step is a number between 0 and 1 that determines how fast the weights get adjusted. Our results keep improving with higher learning rates, suggesting that the network is not ever fully trained, and needs more data. The output is whatever this node outputted to its synapses. In the case of the top most layer this will be boundless, but in the case of the hidden layers, this will necessarily be a number between 0 and 1, because it was fed through a transition function. The derivative is the summed weights and inputs fed into the derivative of the transition fucntion, which is $\frac{e^x}{(1+e^x)^2}$. This formula is applied to every node of every layer to adjust weights, with the exception of the input layer, as it has no associated weights.

5 Related Work

Several studies and analysies have been done to examine the usefulness of Neural Networks in predicting financial data, especially in the prediction of stock. Upon begining the project, we based a great deal of our planning on a piece written by Dr Clarence N W Tan, PhD titled “Applications Examples in Financial Distress Predictions and Foreign Exchange Hybrid Trading System.” [2] This paper gave a brief explanation of neural nets and discussed their ability to predict financial information. Specifically, it discussed the history and fundamentals of neural networks, explaning the concepts of feedforward networks and backpropagation along with their formulaes (for a recap on these concepts, see Method above). After describing these principles, it continued by discussing the parameters along which you could change a neural network and perhaps

[2]Clarence Tan “Applications Examples in Financial Distress Predictions and Foreign Exchange Hybrid Trading System” 1997

optimize preformace. It listed the dynamic variables (refered to as the "neurodynamics" of the system) as: "combination of inputs, production of outputs, types of activation functions and weighting schemes" and the architecture variables (the variables affecting the structure of the network) as "types of interconnections, number of neurons and number of [hidden] layers." . After discussion, we decided to vary the network architecture, experimenting on the number of nerons and the number of hidden layers. The inputs, and outputs would remain constant, as would the weighting scheme (initiliaze with random weights, update based on Back Propogation). Finally, while we hoped to evaluate the effects of changing the transition function, we decided that this manipulation might be too much and instead simply varied the nodes and layers, sticking with a traditional sigmoidal logistic function as an activation funciton for each node. After discussion with the professor, it was decided that the interconnections, too would remain static, simply attaching all nodes in the previous layer, to all nodes in the current layer, in order to allow the network to prune itself. We were also intrested in the effect of the learning rate on the network and decided to vary along this dimension as well.

Other related works include "Using Neural Networks to Forecast Stock Prices" [3] which also touches on the Efficient Market Hypothesis, attemting to realistically evaluate how much Neural Networks can contribute to the prediction of stocks. In the paper, Lawrence argues that NNs have outpreformed all other forms of financial prediction, but cautions readers about the limitations of NNs. Comparing NNs to expert systems, he states that "neural networks are faster" and that "the major problem with applying expert systems to the stock market is the difficulty in formulating knowledge of the markets" [4]but that NNs can extract the rules without need of humans to understand or expliclty formalize the rules. He does recognize the limitations however, citing a study by McCluskey where "neural networks all beat the buy and hold strategy and the S and P 500 index, but often failed to beat a hand-coded optimization." [5] Finally, we reviewed many other papers which discussed the several other extensions of Neural Networks. One entertained the possibility of doing component analysis so that the inputs were independent and which reduced the noise put into the system and analyzed two such techniques an Independent Component Analysis that assumed a non-Gaussian distribution of inputs and a Principle Componnet Analysis which extracted independent components while assuming a Gaussian distribution. In both cases the researchers found that removing noise and only inputing independent factors increased the accuracy over a simple neural network [6]. While our experiment didn't expliclty run algorithms to remove dependent factors, while choosing the input vector, we did take independance of factors into ac-

[3]Ramon Lawrence, Using Neural Networks to Forecast Stock Market Prices 1997

[4]Lawrence p 8

[5]Lawrence p 18

[6]Liu and Wang "Integrating Independent Component Analysis and Principal Component Analysis with Neural Network to Predict Chinese Stock Market" 2011

count. These relevant works aided in planning our network and experiments, and as you will see later, can be used to help us evaluate the results we receive and account for succeses and failures in our approach.
For background, we also consulted the class's textbook "Artificial Intelligence: A Modern Approach" and an online article by the AGH University of Technology and Science in Poland. Both provided us with general background knowledge of Artificial Neural Networks and the ability to understand and code concepts such as Back-Propagations.

6 System Architecture and Implementation

The data was obtained with a ruby script that accessed Mtgox.com every hour in order to pull down the information we used. Mtgox.com distributes API keys to users who want to datamine their site or who want to be able to execute trades without a browser. There is a ruby module that we used that handles all interactions with the Mtgox.com API. This information was then recorded into a Postgresql database every hour. We used Heroku to host the script and the database, as well as some add-ons provided free by Heroku to monitor the functionality of our app. We then pruned our data so that the only examples we were using had appropriate feedback–that is, we cut out the last 24 examples since we did not have information on their future price. The ruby script that accesses the Mtgox.com API was executed as a cron task every hour.

We designed the neural network in ruby. It can be initialized with any reasonable parameters for the number of input nodes, hidden layers, nodes per hidden layer, output nodes, and learning rate. We did enforce, however, that it is fully connected, so all input nodes feed to all nodes at the next layer, and so on. $Layer_j$ strictly outputs synapse to $Layer_{j+1}$. Each layer in our network is represented by an array of nodes, so a network with five layers will have five elements in the array, with each element being an array of nodes in that particular layer. Each node then has a an array of nodes from which it receives input, and a corresponding array which determines the weight that is associated with each input node. All calculations remain internal to the node class, and the network class strictly is in charge of the control flow of the program.

We have a separate script devoted to parsing the data and getting it ready for input to the neural network. This file converts the data in the database into segregated input arrays and output arrays. Note that the output is an array, but in our case it only holds one element: the price tweny-four hours in the future. Assigning the output to an array gives the network the flexibility to output more than one value. There exists another script then, that trains the network. It does this by loading in the data, and repeatedly calling asking the initialized network to train itself on successive data. This script takes as input the ratio of training set to validation set given the fixed size of our data. When the network is finsihed training, the network's predictions are then computed using the weights it attained from training. At this point the weights are fixed and the network is no longer learning. These results are then compared against the actual observed results. We then square these differences and divide by n to get the average squared difference. As a reference point, we then calculate the squared difference attained by strictly guessing the current price.

Since there is some variability between identically intialized neural networks given that the weights are intialized randomly, we have also made a script that repeatedly trains, validates, and gets the squared differences for networks of identical parameters. This takes as input the parameters with which to build the network and the number of networks that it should train and validate with these parameters. It then outputs the best network, the squared difference attained by this network, as well as the squared difference attained by making a naive guess (guessing the current price).

When the network is initialized, random weights are assigned to every input of every node with values ranging from -1 to 1. At each iteration of training, the network takes in two arrays as input: the input data and the expected data. When calculating the derivative in the process of adjusting the weights we found that when the argument fed to the derivative function was greater than 709, it would output Nan. Since as x goes to infinity, the derivative function, approaches zero, we manually check is the derivative returns Nan, and if it does we output zero.
Supporting methods were created to generate the input from data tables, to calculate mean squared error and to find the best network after generating mulitple networks with the same parameters.
The GUI was created by using an extension called Ruby Shoes which allows simple calls to Shoes in order to create simple graphical interfaces. We used a simple layout to take in all the experimenatal variables and compute x neural networks with the given parameters (where x is defined as the number of loops – the last input in the GUI). We then select the best neural network result and display it alongside the naive results.

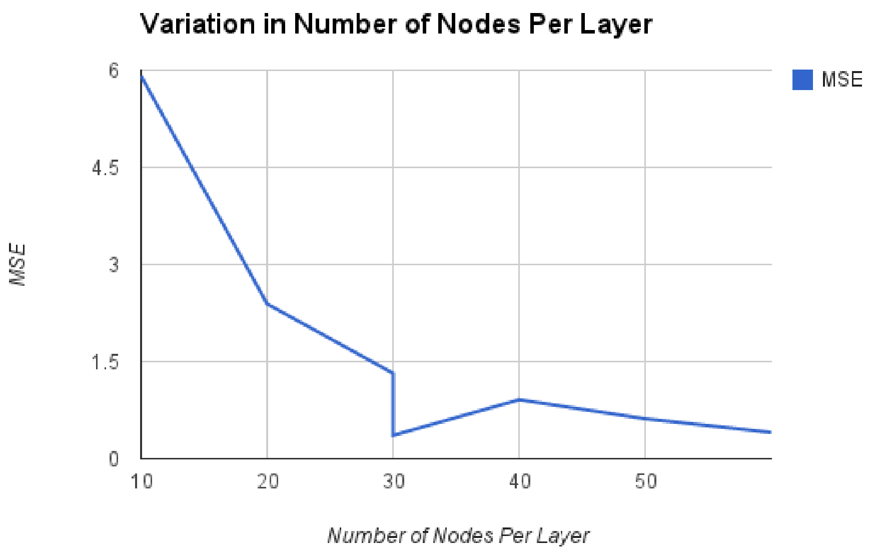
7 Experimental Evaluation

7.1 Methodology

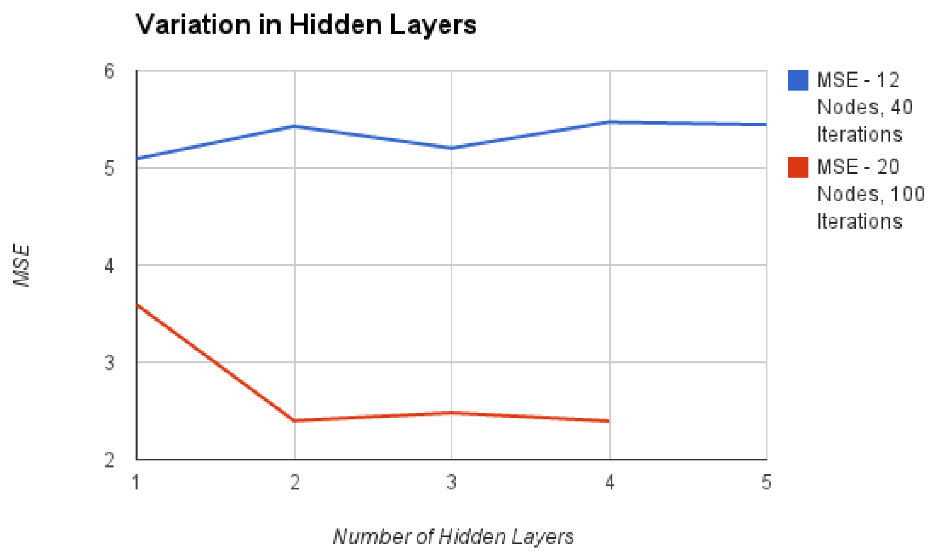
We evaluate our neural network by taking the mean squared error of the output and the expected result. We then find the mean squared error (MSE) between the naive prediction and the expected result. For the purposes of this experiment, we defined the "naive" result as the result that would be obtained had we simply outputted the price in the market 24 hours prior to the predicted time. Our neural network was trained by the data collected from the bitcoin site mtgox.com. We input as a parameters to the network the ratio of training to validation data out of our 3000 samples and would take the first x(3000) sets as training data and take the last (1-x)3000 sets as validation data. This allowed us to simulate how the network would be used in real systems, training on recent data and predicting the unknown values for events in the next few days/weeks.

7.2 Results

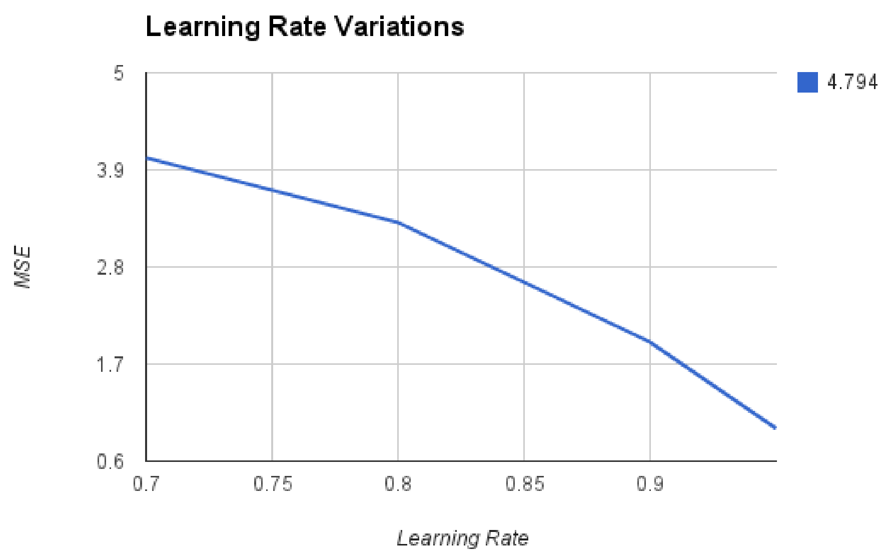
To provide a simple summary of the factors we analyzed, we created graphics showing the basic trends discovered (for the data that created these specific graphis, please see the Appendix). Each graph shows the variation of the network results while varying one of the architecture parameters. The MSE is the average Mean Squared Error found over the number of times we iterated (detailed in the tables). Note that while we did several runs, we only show a small sample of the data and that, due to the random initial weights, there is some variation due to that randomness and small sample size that may cause small anamolies in the graphs.
We found that by increasing the number of nodes we could increase the accuracy of the network asymptotically. We had expected to find some level where the ability to increase terminated, at a place where overfitting occured, but a value was never found. A graph of the results on networks of 10-60 nodes is shown below.



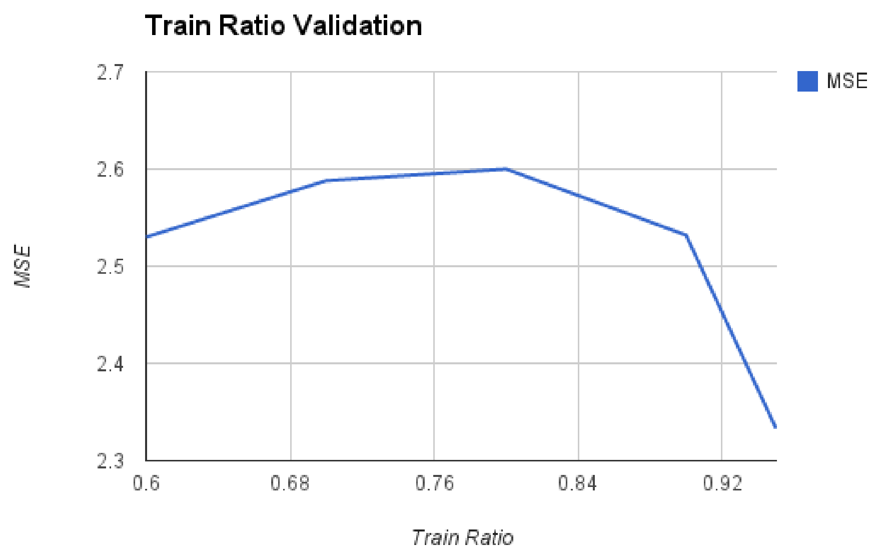
Hidden layers were slightly harder to investigate, several of our first attemts to discover the optimal number of layers failed because we found no variation in the MSE for trials (see blue line that tracks a network with 12 nodes and 40 iterations). Upon further experimenation we found that by increasing the number of iterations and nodes per layer, we could find a distinction between having 1 layer (simply modeling a perceptron) and having mulitple layers, but after 2 layers it seemed highly arbitrary to the results how many layers occured. Again, this was unexpected. We expected to find some optimized MSE and then a drop-off where overfitting had occured.



The third architecture parameter to optimize was the learning rate. This value, most definitely will peak and give a bound if given enough data, because as a learning rate approaches 1 the function will simply learn the training set and not the true function. Unfortunately, we could not even find a reasonable bound for this value, it too caused an asymptotic increase in accuracy (and corresponding decrease in average MSE). When doing research we saw typical learning rate or step values between 0.3 and 0.65, with one or two a few orders of magnitude below 0.3, but never did we see an optimal network around 0.95. We believe this is a strong signal that we do not have enough data to train our network and that we did not collect enough data to properly analyze the effectiveness of some of these techniques. See below for the graph of learning rates vs MSE and the appendix for the tabular values.



Finally, we varied the ratio of training values to validation values. On this final parameter, we also found an asymptotic increase in accuracy. While this could occur for many reasons, we believe that in this context it again indicates a lack of large enough sample size for our network. Another potential cause might be that as we increase the training size, the network is better able to generalize the inputs and does less overfitting than occured in lower training sizes. This we discard becuase we found no evidence of overfitting in any of the other results, and because the need for a larger sample size is consistent with the rest of the data.



7.3 Discussion

As mentioned above, our network most likely suffered from a lack of training data and never fully converged to a result, meaning that the results could be improved by simply running the network over more data (which we unfortunately do not have access to). Comparing to research in the area, we found studies that collected as many as 500 data points a day [7], and iterated over significantly more instances to train.
Along with our need for more data points to train over, research into different financial and stock prediction systems suggested other improvements that could be made to the system. Gryc discussed that training over different stock types could produce different results. They suggested that certain stocks or prices are more 'predictable' and therefore better suited to Neural Network analysis [8]. Perhpas running our network over different stocks or prices could improve (or reduce) or success in finding accurate prediction values. Another idea would be to compare Bitcoin's relationship to a different currency (perhaps the Euro instead of USD). Another issue found was that in the Spanish market, the stocks were affected by the month of the year, and other cyclic "calendar effects" [9]. Perhaps because we only inspected 3 months of data, we were unable to separate the noise of month effects from our data. Having a year or two worth of data and potentially inputting relevant date information might also yeild more valuable results.

8 Future Work

In the future, it might be interesting and educational to try many of the more advanced suggestions from the Related Works section. By examining different types of stock (or simply the Bitcoin price compared to currencies other than USD), or examining the market over a longer period of time we could mitigate some of the challenges discussed by Gryc and by Forradellas & Sarrió. We could also experiment with multiple tranisition functions as suggested by Tan or perhaps see the affects of using more inputs to approach the larger numbers evaluated by Vaisla & Bhatt in the network that took 500 inputs daily. Finally we could prune our input, removing all dependent or noisy data through methods such as ICA as suggested by Liu & Wang. Other vectors of experimenation might include changing the prediction period to a week or month, or by changing the time between the inputed sets of information. Extending the prediction period will show the network's ability to discern much more distant effects which creates many more degrees of uncertainty, and which might introduce more noise into the system. To change the time between input sets, we could input more recent information, and choose to exclude data from the past, or add more data points from the past and less

[7]Vaisla et al. "An Analysis of the Performance of Artificial Neural Network Technique for Stock Market Forecasting"

[8]Gryc 'Neural Network Predictions of Stock Price Fluctuations'

[9]Forradellas, M. Teresa S., and Dídac R. Sarrió. "AN ANALYSIS OF MONTHLY EFFECTS IN THE SPANISH STOCK MARKET USING ARTIFICIAL NEURAL NETWORKS."

from more recent times to evaluate which information is more useful to the network.

9 Conclusions

Our neural network can learn to output a guess that is expected to be 49¢away from the actual price. Only 12¢away from the target value of 37¢, which is the naive guess. This value was attained through much testing and tweaking of the parameters of the neural network. Eventually we found the following to be the values which attained the best results:

Nodes Per Hidden Layer: 20

Hidden Layers: 2

Learning Rate: 0.9

As nodes per hidden layer increased, we found that average squared distance was asymptotically increasing to a value of .2 (given that hidden layers>1). At 20 nodes per hidden layer we can frequently obtain a value within .05 of the asymptote. While at times, on certain iterations this number is not achieved, running this network until we achieve such a value is quicker and just as reliable as adding nodes per hidden payer. When the amount of nodes per hidden layer increases past 20, the reliability in coming closer to this asymptote is not as noteable as the speed cost incurred by adding another node, especially when the amount of hidden layers >1.

The same can be observed regarding amount of hidden layers: any value less than 2 performs significantly worse, and any value greater than two performs negligibly better, but at a huge performance cost. The results we have presented here are juat the result of working with fixed features and number of inputs. A major accomplishment has been creating the framework that allows for very easy tweaking of they system, so that this network can be tested with totally different features, as well as different numbers of features. It can also be used for a totally different problem space, however given how close we came to breaking through the barrier of the naive prediction with no tweaking of the input features, we think that with more data and more intelligent feature selection this network can be used to model bitcoin prices more effectively than it does here, and possibly break through the barrier of naive prediction.

In commodities trading it's considered vital to have data on the relationship between your commodity of interest and many other commodities. Since we only use the USD as a comparison point for our commodity of interest, it's very possible that adding other currencies and examining the relationships that they all undergo would significantly improve the performance of the neural network. However, our pre-collected data only paid attention to the dollar, and so our model only pays attention to the dollar as well. Notwithstanding its obvious shortcomings, we think we have created a valuable tool in the very mutable neural network and were pleasantly surprised with the performance of the network in this problem domain, and think it's a great jumping off point for more accurately modeling bitcoin price behavior.

10 Acknowledgements

We worked with the professor and asked advice at times to help in the understanding of Neural Networks as well as provide advice on how to structure the network (ex. the interconnections between neurons).

11 The Team

Sara Boccabella (snb42) CS Class 2014 Ryan Kass (rck72) CS Class 2014

Sara created the GUI, implemented the transition funciton and the back-propogation code as well as writing the weights update. Ryan worked on gathering the data, the basic code outline, and implementing several of the supporting functions including the MSE calculation and the automated creation of the input from the data. Both partners worked on research of neural networks at the begining to learn the properties and abilities of a neural network. The code structure, the decision of expermintal variables and network strucutre, the write-up and all remaining tasks were split evenly and frequently worked on together.

12 References/Bibliography

"Backpropagation." Virtual Laboratory for Artificial Intelligence. AGH University of Technology and Science, n.d. Web. 10 Dec. 2012.

Forradellas, M. Teresa S., and Dídac R. Sarrió. "AN ANALYSIS OF MONTHLY EFFECTS IN THE SPANISH STOCK MARKET USING ARTIFICIAL NEURAL NETWORKS." FUZZY ECONOMIC REVIEW 14.1 (2009): n. pag. http://search.proquest.com/docview/229038044. Web. 10 Dec. 2012.

Gryc, Wojciech. Neural Network Predictions of Stock Price Fluctuations. Initiative for Interdisciplinary Research. N.p., n.d. Web. 10 Dec. 2012.

Lawrence, Ramon. "Using Neural Networks to Forecast Stock Market Prices." Diss. University of Manitoba, 1997. THE UNIVERSITY OF BRITISH COLUMBIA. Web. 10 Dec. 2012. https://people.ok.ubc.ca/rlawrenc/research/Papers/nn.pdf.

Liu, Haifan. "Integrating Independent Component Analysis and Principal Component Analysis with Neural Network to Predict Chinese Stock Market." Mathematical Problems in Engineering Volume 2011 (2011): n. pag. Hindawi Publishing Corporation. Web. 10 Dec. 2012. http://www.hindawi.com/journals/mpe/2011/382659/cta/.

Norvig, Peter. "18.7 Artificial Neural Networks." Artificial Intelligence (A Modern Approach). By Stuart Russell. Third ed. Upper Saddle River, New Jersey: Pearson Education, 2010. 727-37. Print. Prentice Hall.

Nygren, Karl. "Stock Prediction – A Neural Network Approach." Diss. Royal Institute of Technology, KTH, 2004. Web. 10 Dec. 2012. http://www.f.kth.se/ f98-kny/thesis.pdf.

Tan, Dr Clarence N W. "'An Artificial Neural Networks Primer with Financial Applications Examples in Financial Distress Predictions and Foreign Exchange Hybrid Trading System '." (1997): n. pag. http://www.smartquant.com/references/NeuralNetworks/neural28.pdf. Web. 10 Dec. 2012.

Vaisla, Kunwar S., and Dr. Ashutosh K. Bhatt. "An Analysis of the Performance of Artificial Neural Network Technique for Stock Market Forecasting." International Journal on Computer Science and Engineering 2.6 (2010): n. pag. Web. 10 Dec. 2012.http://http://www.enggjournals.com/ijcse/doc/IJCSE10-02-06-111.pdf.

13 Appendix

Variation in Number of Nodes per Hidden Layer	
Nodes	MSE
10	5.91
20	2.392
30	1.319
30	0.36
40	0.911
40	0.912
50	0.617
60	0.406

Table 1: Hidden Layers=2, Step=0.9, Iterations=10, Train Ratio=0.9

Variation in Number of Hidden Layers		
Nodes	MSE - 12 Nodes, 40 Iterations	MSE - 20 Nodes, 100 Iterations
1	5.095	3.595
2	5.43	2.4
3	5.205	2.481
4	5.473	2.396
5	5.445	–

Table 2: Step=0.9, Train Ratio=0.9

Variation in Step (LearningRate)	
Step	MSE
0.6	4.794
0.7	4.037
0.8	3.303
0.9	1.95
0.95	0.9738

Table 3: Nodes=20, Layers=2, Iterations=40, Train Ratio=0.9

Variation in Train Ratio	
Train Ratio	MSE
0.6	2.53
0.7	2.588
0.8	2.6
0.9	2.532
0.95	2.333

Table 4: Nodes=20, Layers=2, Iterations=40, Step=0.9