

# **Stock Market Predictor using Backpropagation Neural Networks**

Anish Karki

Prabhav Panta

Pranav Sharma

Date: 02/11/18

## Contents

Introduction .....	2
Financial Factors .....	2
Scaling Time Series Data .....	2
BPN Architecture.....	3
Demystifying BPN Connections.....	4
Training methodology.....	4
Activation Function .....	4
Output.....	4
Testing methodology .....	5
Flowchart .....	5
Results.....	5
Improvements.....	7
Conclusion.....	7
<b>References</b> .....	8
Appendix .....	8

## Introduction

One of the biggest fields in the financial sector is machine learning. More specifically, understanding how to properly build algorithms which can self-learn and have a certain degree of predictive power over the trends of the stock market is quite a commodity. Several investors have seen varying degrees of success by building out their own neural network models and using them to create predictive models.

While the predictive power of any model declines rapidly the further into the future you go, it is certainly possible to get within a respective confidence interval. However, obtaining such a confidence interval may end up costing the investor thousands of dollars so we must try to minimize the errors between the predicted values and actual values. Throughout the course of this report, we have implemented use of prior experience with machine learning to reinforce implementation choices.

## Financial Factors

Before being able to build a back propagation network, we must determine what our inputs will be. There are several inputs which may be used to build a predictive model for future stock prices. However we have settled for only three key indicators as opposed to the expected amount of about five to ten. These indicators that have been chosen are all based on the closing values of four weeks' worth of data on the GOOG stock. Building up the training data in the form of [ [average, minimum, maximum], [normalized price] ] has seen the most success in the application, where average is the average closing price, minimum is the lowest closing price and maximum is the maximum closing price of the stock. Proper scaling and preparation of this data must be done before they may be used in the predictive algorithm.

## Scaling Time Series Data

In order to derive meaning from our closing stock values, implementation of a basic sliding window algorithm has been done which computes a rolling average, rolling minimum and rolling maximum of 7-day windows from our first day values to the end of the month. By doing this, we've effectively built a meaningful training set - however we must still figure out how to scale our closing price. To do so, a simple function was built which normalizes the closing stock price between -1.0 and 1.0:

```
def normalize ( price , minimum , maximum ) :  
  
    return (( 2 * price * (maximum - minimum ) ) / (maximum + minimum ) )
```

Along with a roughly inverse method for de-normalizing the data:

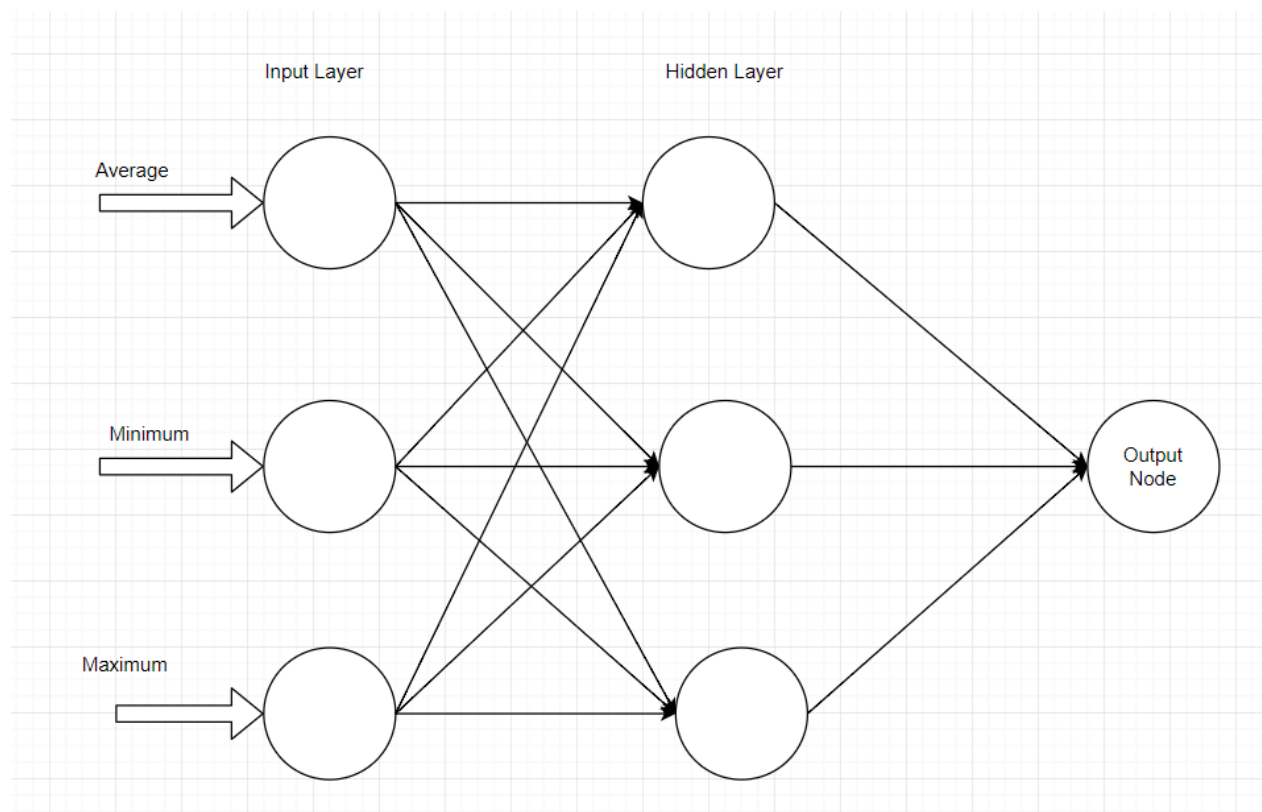
```
def denormalize ( normalized , minimum , maximum ) :
```

```
    return ((( normalized *(maximum - minimum )) / 2) + (maximum +  
    minimum))/2
```

With this, we can properly represent our training data as an array of tuples in the form of [[rolling average, rolling minimum, rolling maximum], [normalized price]]. Next, we sketch our BPN architecture.

## BPN Architecture

With our three inputs average, minimum and maximum, we arrive at the following architecture:



It is important to note that this is a layered network in which all hidden units are deeply connected with both the output unit and the input units.

## Demystifying BPN Connections

In this architecture, we have a deeply connected network because there is a correlation between average, minimum and maximum prices - whereas there might not be a correlation between volume, volatility and earnings. In the case where these factors are to be used, we must be careful not to inner-connect hidden units and input units which have no direct correlation and as such we have chosen to implement a more simple model. For more in-depth networks, the number of hidden units would be tweaked as well as the inputs while varying the connections in order to determine the optimal layout. Moreover, it is more important to build a base model which can easily be tested first and then improve upon it later with new inputs and different architectures then to throw all existing data at a deeply connected network - which will also help to reduce false positives.

## Training methodology

As we have discussed in section 1.1, we now have a proper representation of our training set through transforming the moving window time series data generated by GOOG stock closing prices. We train the BPN as the algorithm normally progresses until it can properly approximate all of the expected training outputs. The final source code presents the patterns in order, as experimenting with randomly selecting training vectors did not have much of an improvement over sequential training. The training data itself is taken from the Kibot API.

## Activation Function

As we have learned in class, the sigmoid function won't suffice in cases where we need to scale data within certain ranges. As such, we have used the hyperbolic tan function as the activation function for the BPN, as it approximates the ramp function for different values of  $x$ .

## Output

The final output of the neural network will be a normalized predicted value which we then de-normalize afterwards. A table will be shown as well as a plot of the actual closing stock price versus predicted closing stock price in the Results section.

## Testing methodology

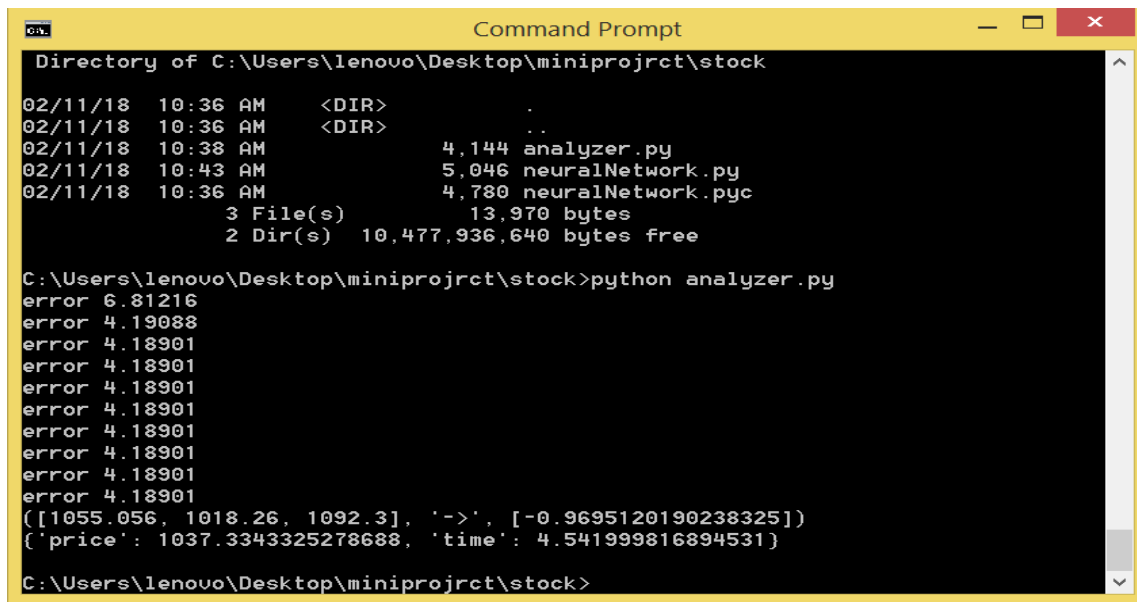
As mentioned earlier, it is better to build a network which can be trained and tested easily. One way to test our network is to take the last two months of closing prices for our stock in question and partition it with 6 weeks being used for training, and the last 2 weeks for testing. During the testing phase, we build sliding window averages, minimums and maximums of the closing prices during the last 2 weeks, feed it into the network and then check the prediction. By doing this, we can appropriately scale our network by observing the performance - predicting values days in advance will actually be difficult to test in the real world due to having to wait for these values to be released and then correlated.

## Flowchart

The flowchart is in appendix.

## Results

With all of the technical jargon out of the way, let's take a look at the results. The prediction of the new stock is done and the error propagated along the 100 iteration is printed. The images of results in the appendix below:



```
Command Prompt
Directory of C:\Users\lenovo\Desktop\miniprojct\stock
02/11/18  10:36 AM  <DIR>          .
02/11/18  10:36 AM  <DIR>          ..
02/11/18  10:38 AM                4,144 analyzer.py
02/11/18  10:43 AM                5,046 neuralNetwork.py
02/11/18  10:36 AM                4,780 neuralNetwork.pyc
                3 File(s)            13,970 bytes
                2 Dir(s)  10,477,936,640 bytes free

C:\Users\lenovo\Desktop\miniprojct\stock>python analyzer.py
error 6.81216
error 4.19088
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
error 4.18901
([1055.056, 1018.26, 1092.3], '->', [-0.9695120190238325])
{'price': 1037.3343325278688, 'time': 4.541999816894531}

C:\Users\lenovo\Desktop\miniprojct\stock>
```

Fig: The actual Prediction

```

C:\Users\lenovo\Desktop\experiments>python stocks-experiments.py
error 1.09757
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
error 0.00000
([1145.0120000000002, 1129.79, 1169.97], '->', [-0.9170984455956367])
([1151.464, 1129.79, 1169.97], '->', [-0.9170984455956367])
([1159.58, 1137.51, 1170.37], '->', [-0.9170984455956367])
([1167.2459999999999, 1155.81, 1175.84], '->', [-0.9170984455956367])
([1171.2, 1164.24, 1175.84], '->', [-0.9170984455956367])
([1169.944, 1163.69, 1175.84], '->', [-0.9170984455956367])
([1171.084, 1163.69, 1175.84], '->', [-0.9170984455956367])
([1170.55, 1163.69, 1175.84], '->', [-0.9170984455956367])
([1157.762, 1111.9, 1175.58], '->', [-0.9170984455956367])
([1133.806, 1055.8, 1169.94], '->', [-0.9170984455956367])
([1117.188, 1055.8, 1169.94], '->', [-0.9170984455956367])
([1092.916, 1048.58, 1167.7], '->', [-0.9170984455956367])
([1059.6799999999998, 1001.52, 1111.9], '->', [-0.9170984455956367])
([1044.856, 1001.52, 1080.6], '->', [-0.9170984455956367])
+-----+-----+-----+-----+
|   Date   | Predicted | Actual | Error Percentage |
+-----+-----+-----+-----+
| April 12th | 1140.67 | 1169.97 | -1.27 |
| April 13th | 1140.67 | 1164.24 | -1.02 |
| April 14th | 1146.41 | 1170.37 | -1.03 |
| April 15th | 1161.23 | 1175.84 | -0.63 |
| April 18th | 1167.38 | 1175.58 | -0.35 |
| April 19th | 1166.98 | 1163.69 | 0.14 |
| April 20th | 1166.98 | 1169.94 | -0.13 |
| April 21th | 1166.98 | 1167.7 | -0.03 |
| April 22th | 1129.14 | 1111.9 | 0.77 |
| April 25th | 1086.70 | 1055.8 | 1.44 |
| April 26th | 1086.70 | 1080.6 | 0.28 |
| April 27th | 1080.83 | 1048.58 | 1.51 |
| April 28th | 1031.40 | 1001.52 | 1.47 |
| April 29th | 1022.93 | 1037.78 | -0.72 |
+-----+-----+-----+-----+
C:\Users\lenovo\Desktop\experiments>

```

Fig: Result of the output of the Algorithm

Not surprisingly, the BPN does not accurately predict the prices. It often underestimates the values, but it does come within a respectable margin of error given the minimal inputs and architecture trickery.

## Improvements

As mentioned in section 2.1, introducing new inputs such as stock volatility, volume sold and company earnings could be beneficial inputs if fed in and interconnected correctly. Another interesting input would be a sentiment analysis with regards to the company (Google, in this case) - where various articles published about the company can be analyzed using a tool such as IBM Watson (they have a publicly available API through IBM Bluemix) in order to determine the overall trend towards the company. Naturally, a company with more positive sentiment analysis would correlate to an improvement in stock. These features would be great for a thesis project or in the real-world, along with looking into deep learning for stock market predictions.

## Conclusion

Overall, this project gave us an opportunity to learn a great deal about various aspects of neural network implementations as well as machine learning overall. In the process, we've learned how to build matrices of the proper shape, how to update the appropriate weights through back propagation. While our understanding of neural networks is still novice at best, this project was a great eye-opener to all of the tiny details that go into designing these applications - especially when neural networks appear as black boxes that compute functions and can only be improved by getting under the hood.



## References

- [1] KiBot API. [http://www.kibot.com/api/historical\\_data\\_api\\_sdk.aspx](http://www.kibot.com/api/historical_data_api_sdk.aspx)
- [2] Jaydip Sen, Tamal Datta Chaudhuri. Decomposition of Time Series Data of Stock Markets and its Implications for Prediction An Application for the Indian Auto.
- [3] S. C. Nayak, B. B. Misra, H. S. Behera Impact of Data Normalization on Stock Index Forecasting. International Journal of Computer Information Systems and Industrial Management Applications, 2014.
- [4] Abhishek Kar. Stock Prediction using Artificial Neural Networks (Y8021).
- [5] Steven Miller. Mind: How to Build a Neural Network (Part One)  
<http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/>
- [6] Steven Miller. Mind: How to Build a Neural Network (Part Two)  
<http://stevenmiller888.github.io/mind-how-to-build-a-neural-network-part-2/>

## Appendix

