A "Net" Profit Project Summary

Isaac McAuley

April 25, 2016

Abstract

This project is about predicting the price of Intel stock with a neural network. The network uses various inputs to predict the stock prices for the next day. All data in this project was collected via web scrapers that ran every week day. All of the project's programming was done in the programming language Python. With only around 3 months of training the network is able to predict stock with a mean error of 1.12% error.

1 Background

This project started last summer when I saw a video of an evolving neural network playing the original Mario. Since then I have been looking for various problems that could be solved by using neural networks, or other forms of machine learning. A few different problems were tried before trying the stock market. Predicting the price of a stock seemed to be a perfect fit for a neural network for many reasons that will be shown in this project. Programming started in mid-December and the first prediction was made in early February.

2 Purpose

The price of a future stock is a very desirable number to predict. A neural network was used because it will do more than just extrapolate. Just using classic extrapolation techniques does not work for stock prices because the past price cannot necessarily tell you the future price of that stock. Aside from the appeal of predicting the stock market, this is just a very interesting computer science problem. It is able to showcase what modern machine learning programs can achieve, even if they are fairly simple.

3 Procedure

3.1 Network Overview

The finished network has 14 input nodes, 1 hidden layer with 30 nodes, and 1 output node. Each day of data has an input and output matrix. The input matrix is of size 14×36 and is detailed more in section 3.3. The output matrix is of size 1×36 and only includes the stock data of Intel for that day. When the network is learning, the output is offset by 36 data points ahead of the inputs. This trains the network to correlate past data with future stock price. Although it is possible to change the offset that was not tested due to time constraints.

After each day the next prediction would be made after the data collection. The weights were never saved so each day the network could retrain itself with the new dataset from that day. Data collection started in January and has ran for most days up until April 8. By the end of this testing period there were 1548 data points per input making a total of 21672 data points to train off of. Each time the network trained it went through 3000 iterations of weights.

This network was designed to predict the stock of Intel, but it could be expanded to predict other stocks. Intel was picked as a proof of concept because they are a consumer facing company, focused on one market, and they are mentioned in technology websites quite a lot. Predicting a company like Apple or Google would be much more difficult because they are in so many different areas with so many different products. It would be possible to predict non-consumer facing companies or larger companies, it would just require different inputs and more data.

3.2 Data Scraping

All data used in this project was collected via Python web scrapers. This was done so the program was not connecting with various programming interfaces and overall it gave more control. The scrapers were running between the hours of 9:00 to 3:00 CST in order to only collect data through the regular hours of the stock market.

The sites that were being scraped were *yahoo.com* for stock data, *xe.com* for dollar conversion data, and *indexmundi.com* for material price data. For the news scraping the front pages of *andandtech.com, theverge.com, ycombinator.com, arstechnica.com, pcgamer.com, tomshardware.com, and techradar.com* were all scraped. Then for each company the number of times that company was mentioned was divided by the total number of words.

After the data was scraped all of it was scaled so no input would be above 1 or below 0. Each kind of input was then put into their own separate JSON files as a 1 dimensional matrix.

3.3 Inputs

The inputs of this network is what really makes it perform at such a high level. Using just the previous price of the stock is not a good way to predict the stock market, because general trends can be made, but stock prices do not go up and down solely based off of past prices. Stock values change depending on how many people want to buy the stock. The 14 inputs are as follows:

1. Date
2. Time
3. Intel Stock
4. AMD Stock
5. Nvidia Stock
6. Intel in the news
7. AMD in the news
8. Nvidia in the news

9. Price of copper
10. Price of aluminium
11. Price of crude oil
12. USD to EUR conversion
13. USD to CNY conversion
14. USD to GBP conversion

The inputs that have been picked are trying to replicate what other people would look at when trying to decide if they want to buy this stock. This is the rationalization behind all of the inputs. Every input is something a potential investor would look at/consider when deciding to buy stock in Intel. Because of the way stocks work there is nothing that directly correlates to a stock price, it is various things that have evolving affects on the price. This is why neural networks can work so well for this problem.

3.4 Math

Neural networks have been theorized for over 70 years now. The original neural network paper was *A Logical Calculus of the Ideas Immanent in Nervous Activity by Warren McCulloch and Walter Pitts*. But these kinds of programs have evolved much since then. This program uses the back propagation method found in the paper *Efficient BackProp by Yann LeChun, Leon Bottou, Genevieve Orr, and Klaus-Robert Muller*. The network starts with the feed forward equation, which just puts data through the network. X is the input matrix, W_1 and W_2 are the matrices of weights, and $\sigma(x)$ is the activation function.

$$\hat{y} = \sigma(\Sigma\sigma(\Sigma(X * W_1)) * W_2)$$

The activation function for the nodes is a sigmoid function, this was chosen because sigmoid functions are non-linear and are easy to find the derivative for.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \frac{e^x}{(1+e^x)^2}$$

When the network starts learning the weights are initialized with random values and then data is put through the network. After the data has been put through there is a $\hat{y}$, but it is very incorrect. The network can then find how incorrect it is with this error function:

$$J = \frac{1}{2}\Sigma(y-\hat{y})^2$$

Once the error has been found the network can start to change the weights. The goal of a neural network is to find the global minimum of the error function by finding the right combination of weights. Each weight matrix has a separate derivation and because this network has two weights matrices, the network has two derivative functions.

$$\frac{dJ}{dW_2} = (-(y-\hat{y})\sigma'(W_2 * \sigma(X * W_1)) * \sigma(X * W_1)$$

$$\frac{dJ}{dW_1} = ((-(y-\hat{y})\sigma'(W_2 * \sigma(X * W_1)) * W_2) * \sigma'(X * W_1)) * X$$

Because this problem is of dimension 450, just solving for the derivative would be too computationally intensive. To approximate the derivative functions gradient descent was used. For this project the BFGS optimization algorithm was used to implement gradient descent because of its speed and accuracy when applied to neural networks.

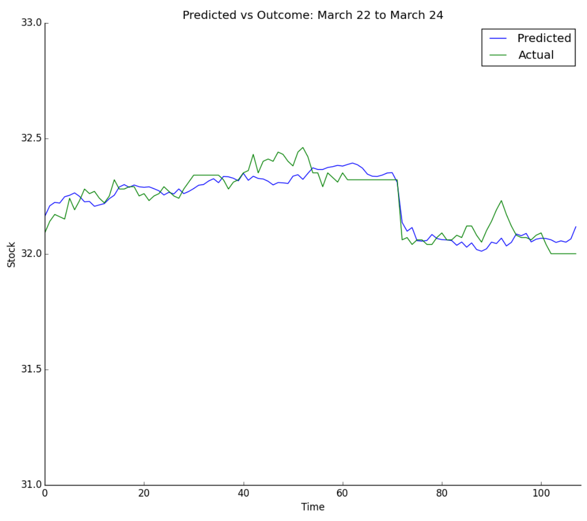
3.5 Programming

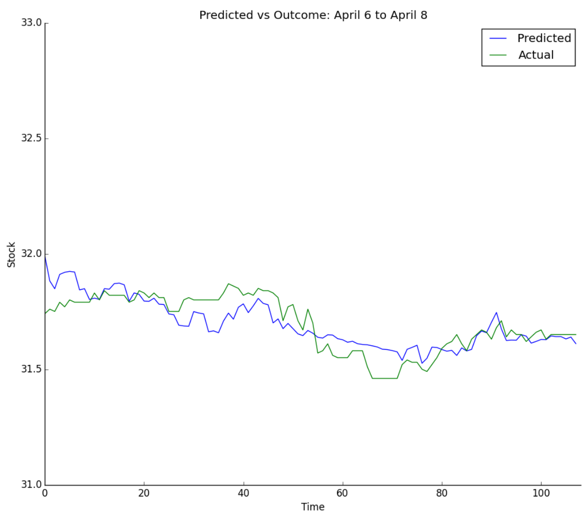
All of the project was coded in the programming language Python. Python was used because right now it is the most supported language for simple machine learning algorithms. For the web scraping BS4 was used as the parsing library. To crate matrices and to extend mathematical functions SciPy was used. And to graph the results MatPlotLib was used. Only a basic network was needed, so the base of the network was forked from GitHub user stephencwelch. Then it just needed to be modified to the particular topology and to be able to accept the format of the inputs. The code is split between 3 directories: scraping, network, and editing. The entire project ended up being around 15 files and over 1000 lines of code.

4 Results

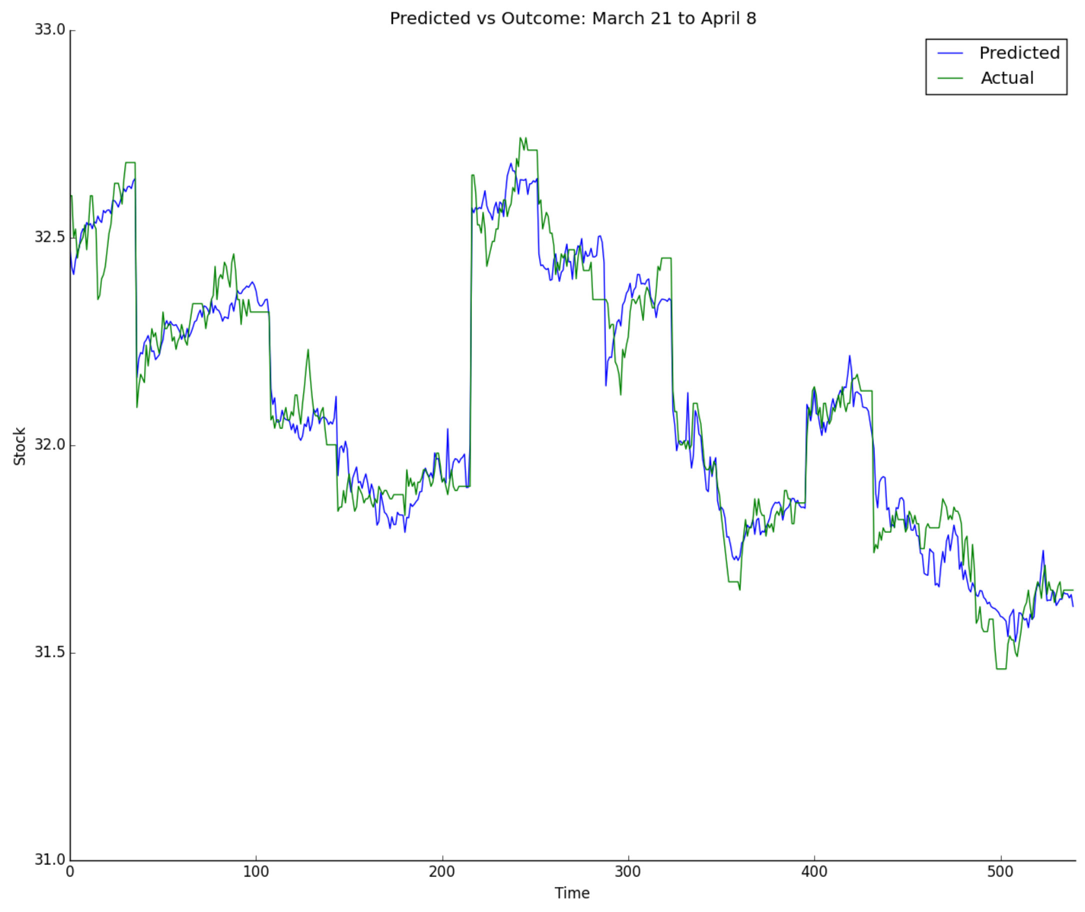
These results are from data that was collected from March 21st to April 8th. This gave the network 15 full days and 540 data points to predict. This data set has an error of 1.12%. To calculate the error I used this error function:

$$E = \frac{\Sigma(|(y - \hat{y})/\hat{y}| * 100)}{n}$$





Above are fragmented parts of the final run of predictions. The graph of the complete final results can be found on the next page. On the Y-axes is the stock price in USD, and on the X-axes is time where 1 data point is 10 minutes, and every 36 data points is a new day.



Just by looking at these 3 figures it is easy to see what the network is good at and where it fails. This network is extremely good at predicting macro changes. It will know if the price will drop or raise a massive amount. But the predicted points are never quite on the line of the actual outcome. But a prediction of ±$0.01 is still a very useful prediction.

5 Conclusion

After seeing the data it is clear that neural networks can work for predicting the price of a stock. Obviously there are some limitations. The network only works for Intel right now, it has not been tested for long term prediction, etc. But I was surprised a network with only 1 hidden layer could perform at this level. With a few improvements this project could actually be a commercially viable product. It would just need to be able to work for more companies and to be able to train for larger time offsets.

The network ended up being much better at recognizing trends rather than absolute values. I was hoping to that it would eventually train enough to go down to a $< 1\%$ error, but an error of 1.12% is still functional.

References

[1] Alpaydin, E. *Introduction to machine learning.* The MIT Press, Cambridge, 2014.

[2] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[3] McCulloch, W. S., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics 5*, 4 (1943), 115–133.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR abs/1312.5602* (2013).

[5] Shanno, D. F. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation 24*, 111 (1970), 647–656.

[6] Welch, S. Neural networks demystified. `https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU`, 2014. [Online; accessed 24 April 2016].

[7] Winston, P. Mit 6.034 artificial intelligence, fall 2010. `https://www.youtube.com/playlist?list=PLUl4u3cNGP63gFHB6xb-kVBiQHYe_4hSi`, 2010. [Online; accessed 24 April 2016].