Predicting Movements in Stock Price: Deep Learning Analysis

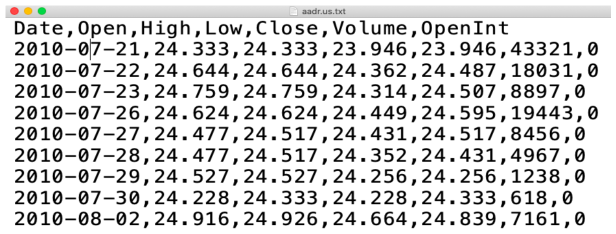
Nelson Blickman
Deep Learning; Professor Burlick
Drexel University
Philadelphia, USA
ntblickman@gmail.com

I. Abstract

Despite there having been many attempts in the past to use Deep Learning or Machine Learning techniques to successfully predict price movements in the stock market, few, if any, have succeeded. In my paper, I will explain how I create a handful of multi-layer perceptrons within Pytorch and an auto encoder to attempt to find patterns in the market myself. There are infinite ways to attack this problem, and I am not sure that the approach I picked led to results that stand out from other attempts. The results of my algorithm highlight how price movements in the market are random, and the difficulty of finding patterns within it. I think there is a way though to take the core structure of my algorithm and improve upon it in the future, with the potential to make progress with this endeavor.

II. Background/Related Work

There may be a reason that money managers in the Finance world, who can successfully predict which company's are going to grow, bring in the big bucks. Not only is it challenging, it's debated to what extent this is even achievable. That doesn't stop people from trying. There are copious amounts of data and copious amounts of features that an engineer could use in this type of Machine Learning analysis to predict prive movements in the market. There are hundreds of large companies with mountains of stock information that go back over 20 years. In addition, features to use may include, technical indicators of the company or it's industry, the state of the economy during that time, and many other possibilities. For example, in 2015, an attempt to use Convolutional Neural Networks to make predictions on future stock price used a feature in their model that was based off of the wording of an organizations announcement about a future event that may affect their company. Events, articles, and information would be great features for a model as they impact which way the stock moves. This approach used Natural Language Processing in order to take a statement of information or a news article's heading about a company, and incorporate that as an input feature. Along with the amount of data and different features to potentially use, the types of approaches can vary greatly. In 2018, a model was created that used multiple methods such as RNN, MLP, CNN, and LSTM. As we will see, my approach focuses on a Multi-Layer Perceptron Network and an Auto Encoder that focuses on past price movements from a handful of companies.



```
Date,Open,High,Low,Close,Volume,OpenInt
2010-07-21,24.333,24.333,23.946,23.946,43321,0
2010-07-22,24.644,24.644,24.362,24.487,18031,0
2010-07-23,24.759,24.759,24.314,24.507,8897,0
2010-07-26,24.624,24.624,24.449,24.595,19443,0
2010-07-27,24.477,24.517,24.431,24.517,8456,0
2010-07-28,24.477,24.517,24.352,24.431,4967,0
2010-07-29,24.527,24.527,24.256,24.256,1238,0
2010-07-30,24.228,24.333,24.228,24.333,618,0
2010-08-02,24.916,24.926,24.664,24.839,7161,0
```

Fig. 1.

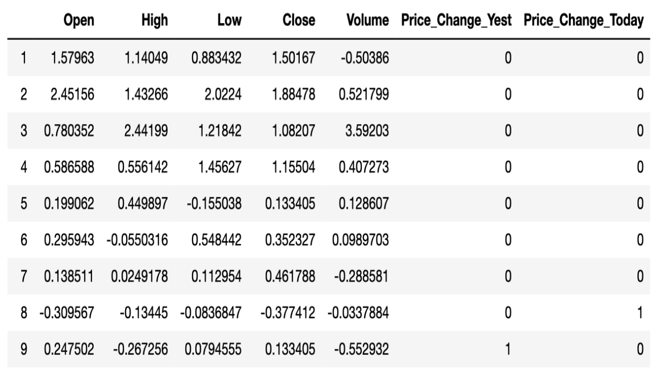
III. Approach/Math/Evaluation

A. The Data

To understand my approach, the math I used in my approach, and my overall analysis, I will begin by talking about my data and the features I used. Given the large amounts of data that the website, Kaggle, provided, I could not use it all. Instead, I picked a handful of companies, and each company had data on some thousands of rows (samples). Each row consists of approximately one day of stock information (see Fig 1). Keep in mind, if 15 companies were used, the approach was to concatenate all of their information together. The approach is looking at what happens to a stocks price movement on a particular day (type of company, etc, is not a feature).

After loading in this data, I pre-processed and cleaned it in order to create the features I desired for my analysis. I wanted to have six independent variables. So, I changed the above structure so that one sample data point included the following: The price the stock opened at today, how high the price rose yesterday, how low the price dropped yesterday, the closing price of the stock yesterday, the volume of shares traded yesterday, and whether the price went up or down yesterday. My dependent variable would be the price change in the stock today. The idea behind this approach was that an individual, on any given day, could simply look at what the stock did yesterday and the opening price today, plug that into the model, and then based on the output of the model (which will say whether it thinks the the stock will go up today or down today), make stock trades accordingly. Please see the code to understand how this preprocessing was performed. Finally, I standardized my data and binarized one of the independent variables "Price Change Yest" and my dependent variable "Price Change today" (0 for price went down, 1 for price went up; see Figure 2 below.)

:



	Open	High	Low	Close	Volume	Price_Change_Yest	Price_Change_Today
1	1.57963	1.14049	0.883432	1.50167	-0.50386	0	0
2	2.45156	1.43266	2.0224	1.88478	0.521799	0	0
3	0.780352	2.44199	1.21842	1.08207	3.59203	0	0
4	0.586588	0.556142	1.45627	1.15504	0.407273	0	0
5	0.199062	0.449897	-0.155038	0.133405	0.128607	0	0
6	0.295943	-0.0550316	0.548442	0.352327	0.0989703	0	0
7	0.138511	0.0249178	0.112954	0.461788	-0.288581	0	0
8	-0.309567	-0.13445	-0.0836847	-0.377412	-0.0337884	0	1
9	0.247502	-0.267256	0.0794555	0.133405	-0.552932	1	0

Fig. 2.

B. *The Model and Architecture*

- Part 1 Analysis:

  From here, I used Pytorch and put the data into "Tensors" as the application requires. At a high level, the next steps involved taking the tensors, splitting them, putting my independent features through my MLP Pytorch model, putting the outcome of that model into my loglikihood objective function to see how often I correctly predicted if a stock price should go up or down, backpropagating and updating my Pytorch optimizer, and then applying those trained weights to my testing data to calculate that model's accuracy.

  Now, lets take a closer look. This paper will eventually explain how I used a number of different inputs into my models architecture. Therefore, for simplicity, I will use specific numbers to explain an example. (Just remember that most of the numbers are subject to change as I perform different tests). First, the model lines would take in an input matrix of 20,000 by 6, each sample row and it's six independent features. The input will then go through three Fully Connected Layers back to back. FC 1 will be (20,000 x 6) @ (6 x 50) . There would be 50 output nodes in our first hidden layer. Therefore, FC2 will be (20,000 x 50) @ (50 x 100). There would be an output of 100 nodes in our second hidden layer. Therefore, FC3 will be (20,000 x 100) @ (100 x 1). We are predicting one output, and then putting that through a sigmoid activation function which will not change our final shape of (20,000 x 1), which is an array of probabilities that show how likely the model believes it is that the price will go up or down today, based on the independent features. These probabilities will be rounded to 1 or 0 when we calculate the model's accuracy on our testing data.

  The Pytorch optimizer that I use handles the back propagation and parameter updates. It uses an "Adam" approach and a learning rate that I specify, as it moves towards the optimal point. The optimizer does not show this, but in our architecture, it is updating three sets of weights. In the example described above, the three sets of weights are (50x6), (50x100), and (1x100). Please see Figure 3 for more details on the architecture and how we backpropagate to update our weights.

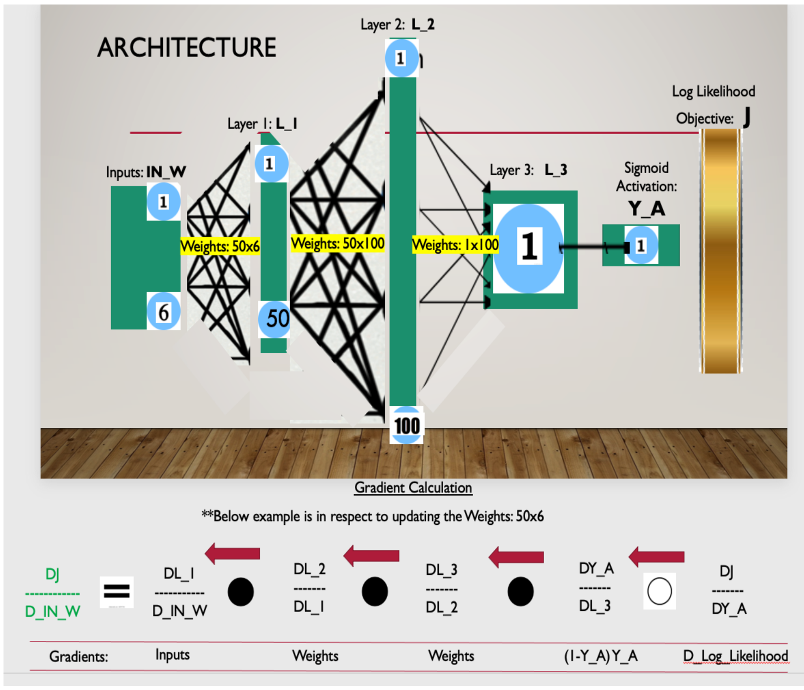


Fig. 3.

- Part 2 Analysis:

  In my next analysis, I created two other networks for comparison purposes, and to see how it was different from the network in analysis Part 1. First, I conducted the same preprocessing of the data as I did in the first analysis (except for the use of variable inputs I used in Part 1 which we will see in the results section). Next, I used Pytorch to create two additional neural networks, and then applied a couple of rounds of tests to gauge their accuracy. The first neural network included was the same one used in analysis 1. The second network though had a different architecture. It's architecture ordering was a Fully Connected Layer, a Relu, FC2, Relu2, FC3, Sigmoid (see Figure 4).

  This architecture is relatively similar to Analysis 1, but we will see how the results vary slightly. Finally, the third architecture that I used was a deeper neural network with the ordering of an FC, Relu, FC2, Relu2, FC3, Relu3, FC4, Relu4, FC5, Relu5, FC6, Sigmoid.

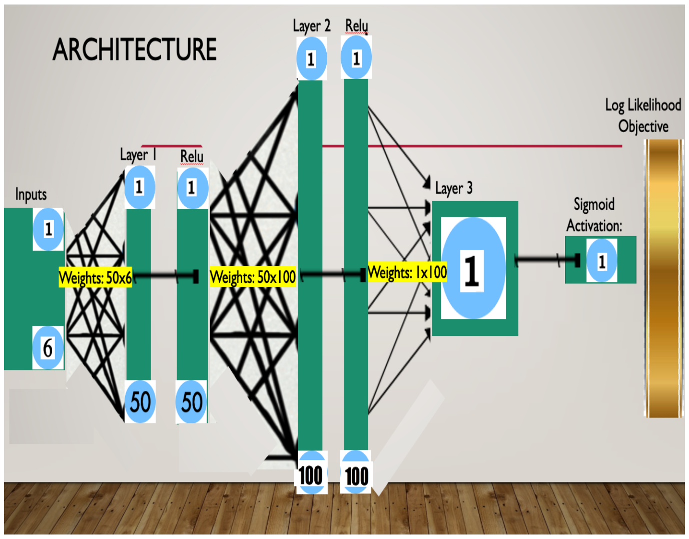


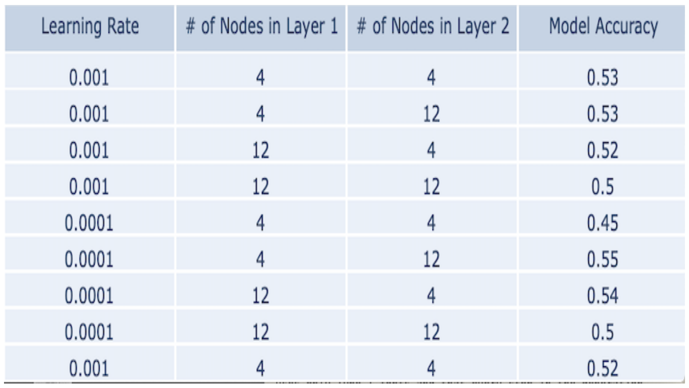
Fig. 4.

- Part 3 Analysis:

  My third analysis did not include pytorch. Instead, I manually created an auto-encoder for the purposes of dimensionality reduction. My data preprocessing did not change, so I was still working with 6 features for each data sample. To begin I created three sets of weights. The first 2 had the shape of (6x3) and I initialized the last 1 to be a shape of (6x1). My auto-encoder started by encoding my data down from 6 features to 3 features. Next, I decoded the data back up to it's original size. Finally, I put this output through a Fully Connected Layer in order to get to a shape of (num rows x 1), since I was still predicting the same binary output of 0 or 1 (weather the stock will go up or down). I slapped on a Logistic Activation function at the end. Finally I calculated my 3 gradients in order to update my 3 sets of weights. The gradient calculation included multiplying the derivative of my Log Likelihood function, the derivative of the Sigmoid Function, the FC layer, Decoder, and Encoder. Please see my code "Part 3 Analysis Code" for the exact calculations.

IV. Results of Part 1 Analysis

I tested a variety of different numbers as inputs into the model from Part 1: the number of companies, the total number of samples (individual days), learning rate, and the number of nodes in the first and second hidden layers of my network. See Figure 5 below to see how these inputs affected the overall accuracy of the model. The accuracy lingers around 50 percent, which is a growing theme we will see in our other results



Learning Rate	# of Nodes in Layer 1	# of Nodes in Layer 2	Model Accuracy
0.001	4	4	0.53
0.001	4	12	0.53
0.001	12	4	0.52
0.001	12	12	0.5
0.0001	4	4	0.45
0.0001	4	12	0.55
0.0001	12	4	0.54
0.0001	12	12	0.5
0.001	4	4	0.52

Fig. 5.

V. Results of Part 2 Analysis

Figure 6 conveys the results from the different architectures described in Part 2 of the analysis. There are 9 graphs listed in the image. Each graph represents one of three networks described and was created with a variable amount of input data. Each graph shows the epoch versus the movement of the objective function during the training process, along with it's final testing accuracy. It is interesting to point out that with enough data, all of these architectures led to having the same exact accuracy. I think this may show that these models may be overfit to this particular data. When you have about 10 nodes as output at your hidden layer, and you use a number of different weights at each layer, that is a lot of complexity and training for an analysis. I believe this is a reason as to why these three models produced such similar accuracy's. A future extension of this would definitely be to find testing data that is from a completely separate company, or at least is more unrelated to the training data. Another solution may be to try and simplify the models.

It was also interesting to see how when there was very little input data, the accuracy of the three models did vary. The architecture from analysis 1 with 3 FCs and a sigmoid did much worse than our second architecture which is where we slapped two Relu's on, right after the FC's. Of course the data input was small, however, it is still curious that this had this affect. The Relu's did not change the shape of our structure, rather, they simply stripped away the negative numbers. Theoretically, this makes sense, and it does seem like a good thing to do for this analysis, because we are dealing with stock prices which can't be negative numbers. All of my inputs were positive. However, after we standardized this data, we were of course left with negative inputs. Given my final outputs are supposed to be binary categorical variables, it does not seem like the Relu should have actually been needed, or even helpful. So why the accuracy jumped up initially is an interesting question.

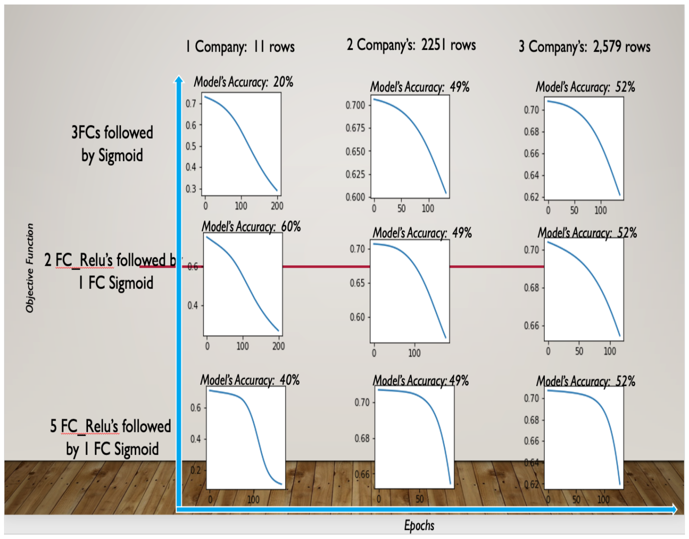


Fig. 6.

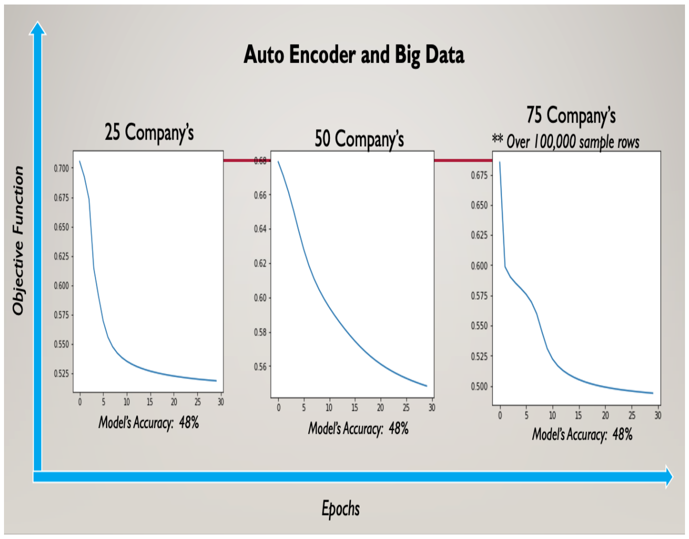


Fig. 7.

VI. Results of Part 3 Analysis

The Auto Encoder seemed to get a model accuracy around the 50 percent mark as well, and did not find any real correlations in the data set. The graphs in figure 7 show the results of the Encoder. Each graph used a large amount of data. For example, the third graph shows the decrease in the objective function over 30 iterations. 75 company's, which in total had over 100,000 individual days (or sample points) of stock information, were input into this third graph. The accuracy was the same for each of the three graphs, and I am assuming the model was a bit over trained, due to the massive amount of data. The movement along the gradient towards the optimal point varied slightly between the graphs. This is another good example of needing different testing data. In the future, using testing data from separate company's that were not in the training data would be something to attempt.

VII. Conclusions/Analysis/Future Work

Overall, the idea that the market is unpredictable prevailed. In my Part 1, 2, and 3 of my Analysis, the accuracy of my testing data set on my trained parameters stayed at around 50 percent, which is the same as if someone took a random guess between two options. There are only two possible outcomes which are whether the stock price will go up or down on a given day. Therefore, it is difficult to gauge which inputs or architecture was optimal. For example in Part 1, whether our first hidden layer had 4 nodes and our second hidden layer had 12 nodes, or vice versa, it was not easy to notice a big difference. However, it is interesting to note how quickly the Part 1 model was subject to overfitting once I started to increase the amount of nodes in the hidden layers above 12 or so. This made sense given there were already 6 weights, and although not an incredibly deep structure, three fully connected layers is definitely more than one. The increase in data, learning rate, and train and test split ratio did not seem to affect the accuracy significantly.

Based on these results in Part 1, I think that trying to vary my inputs into my model was not the best method. I think is more difficult to see subtle patterns when you have so many inputs that are changing on each round of testing. Parts 2 and 3 showed though that this was not the only issue. It seems clear that there is so much randomness in the market that the goal should be to find patterns in the data that could lead to slight profit margins, as opposed to trying to hit a home run. I think it is pretty clear that the features I picked for my analysis did not show any signs of correlation on a given day. Therefore, I think this analysis needs to use a different set of features in the future. There are thousands to pick from, and my focus on the stock price information like closing and opening price was not optimal. At the same time, another approach may be to keep the same features, but focus on a time series, long-term, approach, using RNN's.

References

[1] "Find Open Datasets and Machine Learning Projects." Kaggle, www.kaggle.com/datasets.

[2] GS. Atsalakis, KP. Valavanis, et al. "Short-Term Stock Market Price Trend Prediction Using a Comprehensive Deep Learning System." Journal of Big Data, SpringerOpen, 1 Jan. 1970, journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00333-6.

[3] M, Hiransha, et al. "NSE Stock Market Prediction Using Deep-Learning Models." Procedia Computer Science, Elsevier, 8 June 2018, www.sciencedirect.com/science/article/pii/S1877050918307828.

[4] "Papers." Https://Www.ijcai.org/Proceedings/15/Papers/329.Pdf, 1 Jan. 2015.