Predictions of Stock Market Trends After Earnings Release (April 2020)

Carter Porter (cporte05) - 0990287, Jason Quon (jquon) - 0969675

***Abstract*— This project revolves around the idea of using an artificial neural network to predict the value of a stock. The model aims to predict a company's stock price based on their quarterly performance. While the value of a stock is dependent on many factors, this model aims to simplify the concept of stock valuation to achieve the relatively simple goal of gaining further understanding of a company's performance.**

***Index Terms*—artificial neural network, backpropagation, epochs, feedforward, finance, Jupyter Notebook, Keras, multilayer perceptron, neuron, perceptron, Python, stock, TensorFlow**

I. Introduction

This model aims to provide investors and stakeholders an additional way of assessing risk to enable better informed investing and business decisions. Additionally, this model will allow for easier navigation of the highly nonlinear and often seemingly random stock market by taking advantage of a neural network's inherent strength in finding underlying patterns and using it to classify new data. The model in question will use a feedforward artificial neural network with backpropagation. More specifically, a multilayer perceptron model utilizing a supervised training technique was developed for this application. This model was used to analyze the performance of stocks based on a company's quarterly earnings report. In understanding how a company's stock price is affected by a standardized set of metrics, we aim to enable business insights that can be used to advance business processes and investing decisions.

II. Background

To best understand how our model works, we must first understand the metrics that are used to evaluate a company's stock performance. Our neural network model was designed to analyze stock performance based on a company's quarterly finance report. In this report required by stock exchanges, companies provide 3 months of financial information to investors regarding the state of the company. In our analysis, only private sector companies were evaluated for simplification. The metrics in these financial reports that we focused on a company's revenue, net income, and earnings per share (EPS). Revenue, the amount of income generated, and net income, the overall profit for that quarter were chosen to gain an understanding of the company's business activities and expenses. EPS was an important factor to evaluate in company performance as it is calculated by dividing profit by outstanding shares. The actual EPS compared to the market estimate of the EPS was also used as a metric to identify if the company is meeting expectations, referred to as the EPS surprise. This has a strong correlation to the company's short-term movements. Along with these standard financial data points, 3 additional financial ratios were used to determine the overall health of the company; Enterprise Value (EV) to free cash flow, EV to operating cash flow, EV over Earnings Before Income Tax Depreciation and Amortization (EBITDA). All these metrics were chosen due to its general indication of the company's short-term profitability. Short-term profitability was chosen as the key metric as it is the most basic indicator of the performance of a business. These metrics were used to predict the overnight percentage change of a stock's value after a company's quarterly finance report was released, that is how the price of the stock changed between closing on the day that the report is released and opening on the next business day. The prediction of this metric was chosen to better understand how the market immediately reacts to a company's financial performance. This information can be useful to investors and stakeholders to better understand the direction of the stock value in relation to the company's future performance.

III. Literature Review

This project builds on the foundation provided by previous work on this topic. In the past 10 years, many works have been done and proved powerful in the context of stock price prediction. The general process of developing a data science project of this nature is derived from the process outlined by Vivek Palaniappan and Sebastian Heinz in the articles *Neural Networks to Predict the Market* [1] and *A simple deep learning model for stock price prediction using TensorFlow* [2] respectively. The models in each of these articles presents

interesting and unique ideas for creating a relatively simple model that predicts stock price based on its previous value. In the work put forward by Palaniappan, a multilayer perceptron model (MLP) and a long short-term model (LSTM) are created. Where the MLP is developed for its simplicity, the LSTM is developed to address the MLP's lack of memory compared to a recurrent neural network. For the application of this model, only an MLP model will be developed with the development of different models to be explored in future work. The basic concepts and software libraries used in this MLP model was used as a starting point for the development of our model. Where our model deviates from the concepts in this work is that it does not predict the value of the stock itself, but the percentage change in stock price based on the company's financial performance metrics. This required extensive modification to the model's training as well as its inputs and target values. Further influence in the design of the model for our application came from the article *A Bayesian regularized artificial neural network for stock market forecasting* by Jonathan L Ticknor [3]. In this article, an algorithm derived off the Naïve Bayesian (NB) algorithm achieved a 98% accuracy in stock market forecasting. These past studies use various algorithms such as State Vector Machine (SVM) and Locally Weighted Linear Regression (LWLR), some more viable than others in the application of stock market price prediction.

IV. Methodology

A. Data Acquisition

To create a stock market model, we would need years of past relevant financial data. With this in mind, we decided to gather data on 20 different companies from varying sectors with two things in common, the companies market cap and their earnings financial report date. Each of the companies chosen, at the time of this report, had a market cap of at least $100 billion and a quarterly earnings report date within two weeks of each other company. The market cap boundary was chosen as to negate large daily fluctuations as higher market cap companies daily stock price tend to vary less. A similar financial earnings release date for all companies was used to negate the impact of sporadic market fluctuations. The idea being that if the market fluctuated sporadically during the earnings release period, all the companies surveyed would fluctuate similarly. A total of 174 quarterly earnings reports were gathered from between 2017-2019, with each company having about 9 reports each.

We looked to several online APIs each with a different subset of data with the intent of creating one complete unified set as some sources had only a portion of the data we were looking to utilize. The Python module *yfinance* was used to interface with Yahoo! Finance's online earnings report calendar which acquired the EPS estimate, and the reported EPS for that quarter. This allowed us to calculate the EPS surprise value as such in equation (1).

$$Surprise = \frac{(Actual - Estimate)}{Estimate} \quad (1)$$

The next source was an online web API provided by *www.thelast10k.com* which allowed us to acquire the companies updated balance sheet. Finally, the last source used to acquire the financial data required was another web API *www.financialmodellingprep.com*. This allowed us to gather all the income statement financial features for each of the report dates, being the EV over EBITDA, EV to operating cash flow, and EV to free cash flow. All the data collected was consolidated into a JSON file with each object pertaining to a single quarterly report creating a large training set.

B. Normalization

As the absolute values of the chosen inputs for our model vary in magnitude and are not directly related to each other, it would be meaningless to compare them as such. For instance, a company with higher market cap will likely have more revenue, and without normalizing the data, this would appear to be a better company, which is not necessarily true. To solve this problem, we must find a meaningful way to normalize the data. To do so, we calculated the quarterly growth of each financial feature, except for the EV ratios, taking the previous quarters financial report and comparing it to the present quarters financial report as such in equation (2).

$$Growth = \frac{(Present - Past)}{Past} \quad (2)$$

By looking at the growth of each financial feature the data becomes relativized in terms of percentage and the effects would be consistent across all companies. This has some conflicts based on time of year however it was assumed that all companies experienced the same change in growth relative to the time of year. I.e. Revenue growth for July is less than the end of year due to Christmas.

C. Data Preprocessing

The data recorded in the JSON file was uploaded to Google Drive. From there the *PyDrive* library, a wrapper library that simplifies many Google Drive API tasks, and the Google Cloud SDK were used to access the file within a Jupyter Notebook. Once the JSON file was imported into the Jupyter Notebook, the Python JSON library was used to read the contents of the file into a Python dictionary object. The necessary data was then

transferred into a 2D array with each element being the data for one stock's quarterly data. The array was then randomly shuffled and split into 2 further arrays with 80% of the data being allocated for training and the remainder for testing. The training and testing arrays were then further separated into arrays of inputs and target values.

```
with open('consolidatedData.json', 'r') as outfile:
    m = sklearn.model_selection()
    json_data = json.load(outfile)
    data = getRequiredData(json_data)
    data = pd.DataFrame.from_records(data)
    X = np.array(data.drop(columns=4))
    Y = np.array(data[4])
    test_size = 0.2
    train_x, test_x, train_y, test_y = m.train_test_split(X, Y, test_size)

return train_x, test_x, train_y, test_y
```

Figure 1: Code to parse and separate data into training and test sets

As seen above in Figure 1, the data set were read in from the JSON file as objects and parsed into their respective arrays before being split into train and test datasets. The *test_size* variable allows for different train and test allocations to be made, the 0.2 being 20% into the test dataset.

D. Model

Development of this model began with choosing an appropriate neural network model and training technique. The multilayer perceptron model of the ANN was chosen due to its inherent strength in stochastically computing approximate solutions to complex problems. Moreover, the application of multilayer perceptrons as predictive classifiers lends itself well to the application regarding stock performance through its proven ability as a universal approximation algorithm. We found that the multilayer perceptron's ability to learn mappings between input data and predicted output variables was best used to predict the overnight movement of a stock price after earnings. More specifically, the model at hand was developed to take normalized inputs to predict the percentage change of a stock price between closing on the day of earnings being released and market opening the next day. The model was developed using the Keras neural network library running on top of the TensorFlow data flow and differentiable programming library. These libraries were used for their ease of use and implementation due to their widespread adoption amongst neural network applications. Furthermore, these libraries are both free and open source, meaning they are both easily portable and open to future improvements.

Using these libraries, the model was built as a sequential model which consists of a linear stack of layers. The first layer had an input dimension of 7 in which the ratio of estimated EPS to actual EPS, EPS growth percentage, revenue growth percentage, net income growth percentage, and the 3 EV ratios were passed in as inputs. It was important to ensure that all inputs were normalized as a ratio or percentage to ensure consistency across the vast diversity of values for different company's stocks. The second layer, or first hidden layer, is densely connected and consists of 15 neurons using a rectified linear unit activation (ReLU). The ReLU activation was chosen because it does not bound the output between 0 and 1 like a sigmoid function, or between -1 and 1 for a hyperbolic tangent function. The rectified linear function is a piecewise function in which values of 0 or less are rounded to 0 but all other values remain the same. This activation was found to be more desirable for its use in training through backpropagation. The third layer, or second hidden layer, is densely connected with 15 neurons using a linear activation. While less suitable for complex mappings, the use of a linear activation simplified training the model. The output layer of the model consisted of a single neuron which would then represent the predicted overnight percentage change of a stock's value. A visualization of the model and each of its layers can be observed in Figure 2.

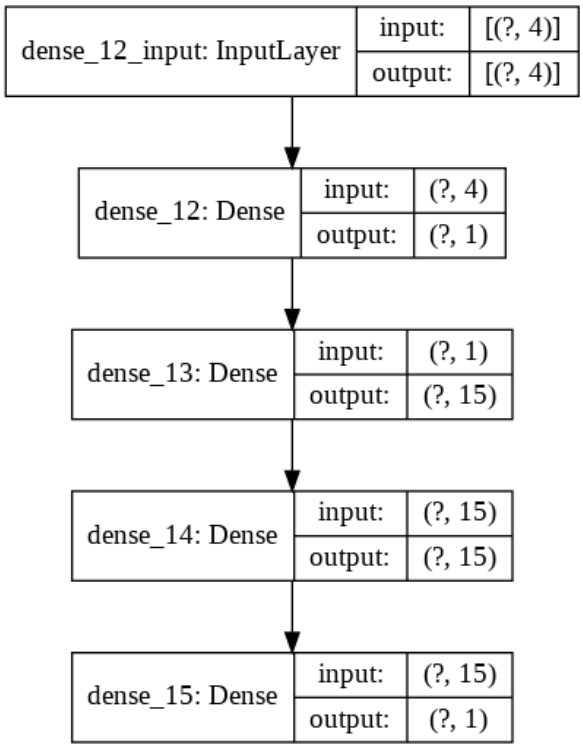

Figure 2: Neural Network Layer Diagram

The model was then compiled using the adam optimization algorithm with loss calculated through mean square error. The Adam optimizer was chosen due to its strength over stochastic gradient descent particularly in its computational efficiency. More specifically, the adam optimizer was used because of its inherent combination of the advantages of adaptive gradient descent and root mean square propagation by calculating an

exponential moving average of the gradient and the squared gradient.

V. RESULTS AND DISCUSSION

The overall loss and accuracy were used to evaluate our model. When the model was simulated to run 150 times to get a fitted model on the training data, and then evaluated with test data, the best loss was 5.25, and the highest accuracy achieved was 10.46%. It can be noted that this loss and accuracy value was recorded using the adam optimization algorithm and rectified linear activation. When experimenting with optimizers such as adaptive gradient descent and root mean square propagation, the resulting loss and accuracy were significantly worse. Specifically, the model tended to output mostly 0 when using different optimization algorithms. This is further validation of the decision to use the adam optimizer as discussed earlier. Furthermore, different activation functions were experimented with to validate the decision to use the rectified linear activation function. It was observed that when using the hyperbolic tangent or sigmoid functions that the model's performance was noticeably worse when compared to that of the rectified linear function, likely since the functions have strict upper and lower bounds. Additionally, the decision to use a linear activation in the second hidden layer allowed the model to produce negative outputs suitable for this application. The code used to modify the model parameters can be seen in Figure 3 on lines 18 and 21. These parameters allow for the alteration of optimizers, loss function and the desired metrics.

```
1 # Generate MLP model, 2 hidden layers with 15 neurons each
2 model = tf.keras.models.Sequential()
3
4 #Input layer with input dimension of 7
5 model.add(tf.keras.layers.Dense(1, input_dim=7))
6
7 #Hidden layer with rectified linear activation
8 model.add(tf.keras.layers.Dense(15, activation=tf.nn.relu))
9
10 #Hidden layer with linear activation
11 model.add(tf.keras.layers.Dense(15))
12
13 #Output layer
14 model.add(tf.keras.layers.Dense(1))
15
16 #Using adam optimizer, could also use Adaptive Gradient Descent
17 #(AdaGrad) or Root Mean Square Propagation (RMSProp)
18 model.compile(optimizer="adam", loss="mse", metrics="accuracy")
19
20 #Train model
21 history = model.fit(train_input, target_output, epochs=50)
22
23 #Test model
24 print("\nLoss value: ")
25 model.evaluate(test_input, test_target)
26
27 #Compare output to target values
28 test_results = model.predict(test_input)
29 for i in range (35):
30     print("\nTest input: ", test_input[i])
31     print("Target output: ", test_target[i])
32     print("Model output: ", test_results[i])
```

Figure 3: Creating, Training, and Testing the Model

As seen below in Figure 4, the model's loss value decreases with each epoch. An epoch being a full cycle through the entire training dataset. So, after 50 iterations through the entire training dataset, the resulting loss was 5.25.

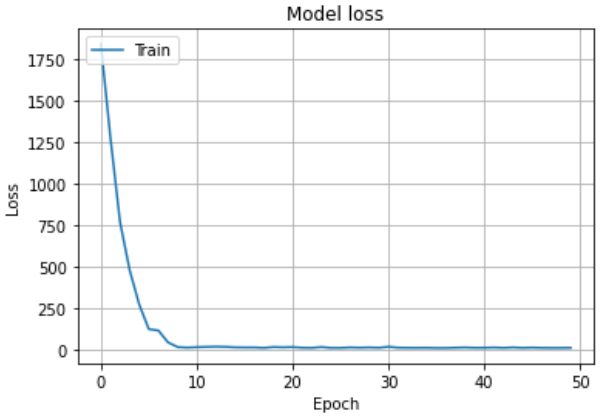

Figure 4: The Model loss after 50 epochs

Increasing the epochs in this yielded a higher loss value, indicating that the model was overfit and over-trained, showing that more training is not always an optimal design solution. With our model, that when faced with the test data, it did not work as efficiently as the slightly less trained model. It was imperative to reduce the loss function as much as possible to optimize the functions parameters to mitigate output variance. With a larger dataset, this could have been reduced even further to create a more optimized model. Our dataset is relatively small compared to more traditional works and methods and in the scope of machine learning, small training sets without strong correlation to the output often leads to high variance. With a small dataset, we also run the risk of high-bias and overfitting our model to a small subset of the overall set. Overall with more data, we could reduce the number of required epochs to try and get a fitted model and have a more comprehensive set of data to train with, ultimately increasing accuracy and minimizing loss.

VI. CONCLUSION

The overall performance of the model could be improved with a larger and more comprehensive dataset. By simply having more data to train and test the model with, the performance of the model is likely to improve as it is more likely to detect underlying patterns in a larger dataset. The inaccuracy of the model can also be attributed to the fact that the model only considers a company's profitability. While this is the main component in evaluating stock price, it is far from the only factor that affects it. More specifically, the model does not account for simple supply and demand for the stock, it can

only approximate this based on information from the quarterly finance report which can be misleading out of context. For example, Tesla Inc. ($TSLA), the producer of electric vehicles, has seen a general increase in its stock price despite reporting negative earnings per share in 4 out of the 12 quarters between January 1, 2017 and December 31, 2019. Furthermore, the company has only exceeded the estimated EPS twice in that period. Despite only generating profit in 33% of these fiscal periods, the value of the stock has continued to increase because its value reflects demand based not only on the current performance of the company, but also its perceived future performance in which electric vehicles are predicted to be more popular. Future improvements to this model could be made to increase the amount of contextual data used when training the model such as company sentiment in the investors eye. Having a better understanding of the company's long-term performance will allow the model to better predict the immediate changes put in motion by the release of the company's quarterly performance report.

References

[1] Vivek Palaniappan. *Neural Networks to Predict the Market, 2018.*

[2] Sebastien Heinz. *A Simple Deep Learning Model for Stock Price Prediction Using TensorFlow,* 2018.

[3] Jonathan L Ticknor. *A bayesian regularized artificial neural network for stock market forecasting. Expert Systems with Applications,* 40(14):5501–5506, 2013.

[4] Jason Brownlee, *Introduction to the Adam Optimization Algorithm for Deep Learning,* 2017

[5] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit,* 2019

[6] Vitaly Bushaev. *Latest Trends in Deep Learning Optimization,* 2018

[7] Porter Carter, StockMarketProject, 2020, GitHub Repository, https://github.com/c-porter15/StockMarketProject