CS230

# Get Rich or Die Trying: Predicting QQQ ETF Price Movement with Numerical and Sentiment Data

**Aparna Tumkur**
Department of Electrical Engineering
Stanford University
atumkur@stanford.edu

**Namit Mishra**
Department of Electrical Engineering
Stanford University
namishra@stanford.edu

**Nikhil Hira**
Department of Electrical Engineering
Stanford University
nhira7@stanford.edu

## Abstract

In this paper, we discuss an attempt to predict the price of an exchange traded fund based on price movements and news for its ten largest constituents by weight. After considering various approaches, we settled on using an LSTM, along with various normalization and Dense layers, to classify an output into categories reflecting the index price's percentage change on a daily basis. Although we found our model able to correctly predict percentage change with 45% accuracy, it shows improvement over simply including prior stock prices.

## 1 Introduction

The overall task involves predicting day-to-day percentage price changes for every day of 2018 of the QQQ exchange traded fund (ETF) by incorporating technical indicators and sentiment analysis from company mentions in the financial news media. Past work has mainly focused on predicting prices of a single company's stock (or a single ETF price), primarily using numerical data from the stock market. Our approach using an LSTM includes sentiment analysis along with multiple constituent stock data (numerical indicators), to see if that has any noticeable impact on the prediction accuracy. We referred to past projects using deep learning to predict stock prices, to understand which market indicators may or may not be useful for developing an accurate model.

## 2 Related work

We referred to past projects using deep learning to predict stock prices, to understand which market indicators may or may not be useful for developing an accurate model. Much like our project, the author of [1] looked into predicting the price of an ETF, and the model used in that instance relied heavily on data derived from the prices. For instance, the model extensively used price momentum metrics, moving averages, and tangent slopes. The model also considered the weighting of each of the stocks, which is not a characteristic we considered. The end result, much like in our case, was a softmax classification of percentage price movements of the ETF. However, we did not rely so heavily on metrics, instead banking on sentiment analysis to provide accurate information regarding price movements.

The approach discussed in [2] considered news sentiment for stock prediction. In this approach, stock price data was taken on every change (known as a tick) in the price. To align news data with these ticks, sentiments are assigned to every hour of ticks; multiple sentiments being released during a particular period are averaged to assign a sort of index to each time period. However, since we chose to use stock price data with the granularity of a single day (e.g. the highest price on a single day or the closing price on a single day), we could instead just assign per-day sentiments. Nonetheless, we share in our approaches to assign sentiments based on the frequency of data used.

Our initial model was based on the Tensorflow Softmax assignment [3]; we chose to use it simply because it allowed us to get started somewhere. We used the following classification layers: LINEAR > RELU > LINEAR > RELU > LINEAR >

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

SOFTMAX. The model, unfortunately, was plagued by stubbornly low performance (with accuracy no better than 20% on the test set), likely because it could not take advantage of the "memory" afforded by an LSTM. Thus, it probably is not an ideal model for use in stock price prediction.

The work in [4] provided us with great insight into using an LSTM for price prediction. This model serves as the primary basis for our project. In this project, the author attempts to predict the price of a particular stock solely based on the closing price, volume, and intraday price extrema. Corrective actions to improve the accuracy used in this work include normalization of data, a simple moving average, and various dropout layers. All of these are aspects we eventually included in our model as well, though we eventually experimented with the exact values used in these actions (i.e. the exact dropout layer).

For understanding the web scraping and sentiment analysis, we relied heavily on a DataCamp course on the topic [7]. Quickly perusing the analyzed sentiment analysis, we found that the approach from this course was largely successful, so we did not see a need to explore the topic further beyond this. There were other models [5][6] we had initially wanted to look into for more understanding, but we did not have time to do so.

# 3 Dataset and Features

## 3.1 Financial Data

We downloaded June 2017 - December 2019 stock price data from Yahoo Finance for the ten largest companies in the QQQ index by weight, as given in [8]. These companies were, in descending order of weight, Microsoft, Apple, Amazon, Facebook, Google (classes C and A), Intel, Netflix, PepsiCo, Nvidia, and Cisco. After downloading the data, we calculated the 5 day exponentially weighted moving average for each stock, the values of which will be used as a feature for price calculation. We chose to calculate this value to see if using pricing data with less noise (as opposed to the actual adjusted closing price) would improve our model prediction.

We also found the Bollinger Bands for each stock. These are trendlines calculated m standard deviations above and below the adjusted closing price, and can help indicate if a stock is oversold or overbought. We had to experiment slightly with the value of m to get the stock price to better fit between the bounds; normally, a value of 2 is selected, but experimentation led us to choose 2.5. If a stock is trading close to the lower trendline, it is considered to be "oversold" and trading below its intrinsic value. This could indicate that its value might rise later. The opposite is true if the stock is at the upper trendline.

## 3.2 News Sentiment Data

We gathered from both Reuters and Barrons, to provide diversity to the news and increase data points. Each company had the following number of relevant articles:

| Count of Headline | Date | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ticker | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec | Grand Total |
| AAPL | 450 | 380 | 344 | 287 | 213 | 149 | 164 | 227 | 250 | 273 | 313 | 225 | 3275 |
| AMZN | 337 | 386 | 372 | 465 | 301 | 246 | 221 | 191 | 184 | 276 | 276 | 308 | 3563 |
| CSCO | 27 | 77 | 43 | 28 | 79 | 35 | 34 | 53 | 27 | 29 | 71 | 28 | 531 |
| FB | 246 | 185 | 351 | 336 | 200 | 204 | 217 | 127 | 252 | 326 | 177 | 192 | 2813 |
| GOOG | 87 | 88 | 75 | 91 | 77 | 126 | 133 | 68 | 62 | 109 | 59 | 89 | 1064 |
| INTC | 176 | 80 | 90 | 130 | 103 | 78 | 126 | 89 | 90 | 122 | 88 | 65 | 1237 |
| MSFT | 155 | 143 | 148 | 154 | 88 | 89 | 104 | 52 | 78 | 168 | 103 | 103 | 1385 |
| NFLX | 168 | 98 | 115 | 191 | 88 | 76 | 120 | 92 | 105 | 216 | 95 | 103 | 1467 |
| NVDA | 68 | 83 | 68 | 55 | 79 | 58 | 45 | 71 | 49 | 51 | 84 | 38 | 749 |
| PEP | 39 | 53 | 42 | 52 | 39 | 26 | 53 | 31 | 20 | 38 | 15 | 19 | 427 |
| **Grand Total** | **1753** | **1573** | **1648** | **1789** | **1267** | **1087** | **1217** | **1001** | **1117** | **1608** | **1281** | **1170** | **16511** |

A concern we share is that an uneven distribution of news articles between the companies could potentially skew our prediction of the overall QQQ index price. If the distribution between the different companies is uneven, it could unfairly skew the QQQ price prediction towards the movements of some companies with more data over others. On the other hand, a lack of headlines involving a company could be a sign of neutral sentiment.

The sentiment analysis was conducted using the NLTK sentiment analysis toolkit, which is commonly used in natural language processing in Python. As the dictionary this kit uses is somewhat generic, its categorization of financial terms, in financial contexts, may be somewhat limited.

# 4 Methods

## 4.1 Main Code Walk-Through

The first part of the code loads the datasets. Here we define the number of "history points" to create windows of data that are used to predict the following day's price movement. Both the x and y data are .csv files imported to numpy arrays. The x data consists of close price, trading volume, 5-day Exponentially Weighted Moving Average (EWMA) price, Bollinger band max and min (at 2.5$\sigma$ using 10-day simple moving average), and sentiment score for each of the ten stocks chosen. The close price of QQQ was also included, bringing the total input features to a count of 61. The input data was normalized to the maximum value of the corresponding input feature.

After loading the dataset, we split the data into train and test sets (80%:20%). The next block of code is where we define our metric. It checks for equality between the index of the maximum of the softmax output for both predicted and true values and then takes the mean over all predictions. This function is given to the model.fit metrics input. Next, we define our model graph as seen below:

```
# model architecture
lstm_input = Input(shape=(history_points, 61), name='lstm_input')
x = LSTM(100, name='lstm_0', activity_regularizer=keras.regularizers.l2(0.00012))(lstm_input)
x = BatchNormalization(name='batch_norm_0')(x)
x = Dropout(0.1, name='lstm_dropout_0')(x)
x = Dense(128, name='dense_0', activity_regularizer=keras.regularizers.l2(0.00012))(x)
x = BatchNormalization(name='batch_norm_1')(x)
x = Activation('relu', name='relu_0')(x)
x = Dense(8, name='dense_1', activity_regularizer=keras.regularizers.l2(0.00012))(x)
output = Activation('softmax', name='softmax_output')(x)
```

Before compiling the model, we also define a callback class to be used to observe the total accuracy of the model after each epoch. We did this because we noticed that the reported metric at the final epoch was not matching our final accuracy metric. Using this callback allowed us to monitor the total accuracy of the model throughout the learning process.

```
model = Model(inputs=lstm_input, outputs=output)
adam = optimizers.Adam(lr=0.0016)
model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=[our_metric])
history = model.fit(x=x_train, y=y_train, batch_size=32, epochs=200, shuffle=True, callbacks=[total_accuracy])
```

Details of the learning model can be seen in the image above. Values for various parameters were determined from a hyperparameter sweep that is covered further down. An Adam optimizer was used with a categorical cross-entropy loss function and our previously defined custom metric.The remainder of the code simply uses predicted values to verify final accuracy and plots the loss and accuracy over epochs.

A second metric was also used at the end of training to quantify how far off the predictions are from the actual true values. This metric counted not only the correct predictions, but also included the predictions that were only one bucket away in a sum before taking the mean. The code for that can be seen below.

```
train_prediction_distance = keras.backend.argmax(y_train_predicted, axis=1) - keras.backend.argmax(y_train, axis=1)
test_prediction_distance = keras.backend.argmax(y_test_predicted, axis=1) - keras.backend.argmax(y_test, axis=1)

train_pred_dist_1 = keras.backend.less_equal(train_prediction_distance, 1)
test_pred_dist_1 = keras.backend.less_equal(test_prediction_distance, 1)

train_accuracy_dist_1 = keras.backend.mean(keras.backend.cast(train_pred_dist_1, "float"))
test_accuracy_dist_1 = keras.backend.mean(keras.backend.cast(test_pred_dist_1, "float"))
```

The remaining lines of code are for viewing the model plot, summary, and other details for troubleshooting and informative purposes.

### 4.2 Hyperparameter Tuning

In order to find optimal hyperparameter settings, we chose to create a combination grid on which to train our model iteratively. The code snippet below shows how we generate the "sweep" grid. Some parameters had specific chosen values while others were assigned randomly within a range. This was by all means not a comprehensive sweep, but we were curious to see the results of this type of approach. Due to the long amount of time it takes to train a model, we limited our total combinations to 600. This resulted in a total run time of about 14 hours, hosted on AWS.

```
def get_h_params():
    history_points_list = random.sample(range(7, 31), 5)
    # dropout_rate_list = [0.5 * random.random() for i in range(4)]
    dropout_rate_list = [0.1, 0.5]
    batch_size_list = [32, 64, 128]
    r1 = -4 * np.random.rand(5)
    learning_rate_list = (10**r1).tolist()
    r2 = -4 * np.random.rand(4)
    l2_reg_list = (10**r2).tolist()
    # epochs_list = (np.random.randint(200, 500, 5)).tolist()
    epochs_list = [200]

    sweep_grid = itertools.product(history_points_list,
                                   dropout_rate_list,
                                   batch_size_list,
                                   learning_rate_list,
                                   l2_reg_list,
                                   epochs_list)

    return sweep_grid
```

For future improvements and testing, we would take a more individualized approach to trying to optimize one parameter at a time. Results from the hyperparameter sweeping are covered in the next section.

## 5 Experiments/Results/Discussion

The output of the model is a classification into one of eight 1% change buckets. The classifications are as follows: $\Delta < -3\%$, $-3\% < \Delta \leq -2\%$, $-2\% < \Delta \leq -1\%$, $-1\% < \Delta \leq 0\%$, $0\% < \Delta \leq 1\%$, $1\% < \Delta \leq 2\%$, $2\% < \Delta \leq 3\%$, $3\% \leq \Delta$. $\Delta$ is the percentage in price for a particular day compared to the previous day.

We used two accuracy metrics in our model, the pre-defined categorical cross-entropy accuracy and a custom defined metric called "our_metric." This metric compares the target one-hot vector with the softmax output of the model, and checks if the index of the '1' in the former matches with the index of the highest probability in the latter. The reported accuracy is the fraction of predictions which lie in the correct bucket. While "our_metric" imposes a stricter criterion for reporting the accuracy, the categorical cross entropy compares the probability distributions of the target and actual output. The formula for the two are as follows:

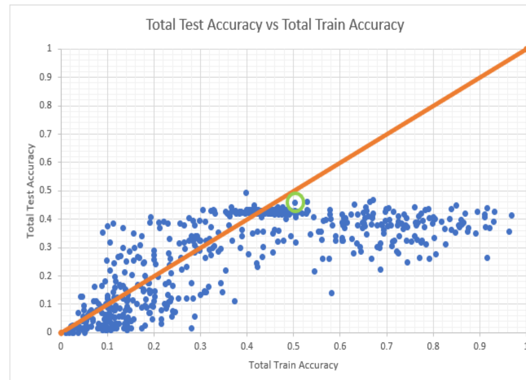
```
def our_metric(y_true, y_pred):
    correct_prediction = keras.backend.equal(keras.backend.argmax(y_pred, axis=1),
                                             keras.backend.argmax(y_true, axis=1))
    return keras.backend.mean(keras.backend.cast(correct_prediction, "float"))
```

$$CategoricalCrossEntropyAccuracy = \frac{1}{N} \sum_{s \epsilon S} \sum_{c \epsilon C} 1_{s \epsilon c} log(p(s \epsilon c)) \quad (1)$$

where S is collection of samples, C is collection of classes, and $s\epsilon c$ means sample s belongs to class c

The model on which our project is based uses price prediction [4], whereas our model predicts or classifies the output into a bucket corresponding to percentage change in the stock price compared to the previous day. We used this idea to make learning easier for the model, while also getting a good idea of the direction and amount of change in the stock price. The model we referred to does not report an accuracy metric, but uses the "mse" (mean square error) loss. An appropriate metric, if we were to report it for the referred model, would be the mean square error regression metric.

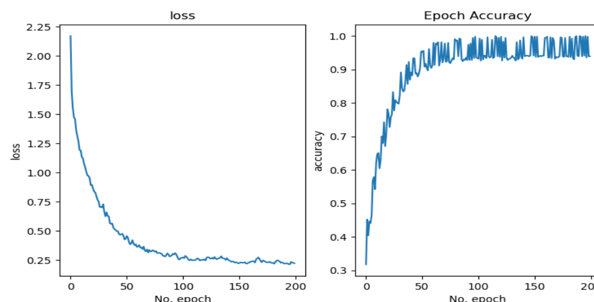
Sweeping the hyperparameters as described in the previous section gave us the train and test accuracies for different combinations of them. To make the final hyperparameter selection, we chose the points farthest from the origin, but also closest to the total train accuracy = total test accuracy line.




We ran our model using the hyperparameters we found to be best suited for our application, and the final train and test accuracy are 50.3% and 45.7%, respectively. Yet another metric we used considered whether the classifications were either correct or one classification off from the actual one. This idea is predicated on the fact that while our accuracy in softmax prediction plateaued at around 45%, altering the accuracy metric to consider predictions which were one classification off from the correct category (such as if the model predicted a change of 2%, even when the actual change was 1.7%) improved the accuracy to around 90%. This is significant because it shows that the rigid nature of the classification is unfairly hurting our accuracy metric by punishing small discrepancies. As can be seen in the results below, using this more inclusive metric pushes the accuracy much higher.

```
Train Accuracy: 0.50292397
Test Accuracy: 0.45736435
Train Accuracy New: 0.8947368
Test Accuracy New: 0.90697676
```

We also measured the train accuracy after each epoch, and the plot of the same is as follows:




As can be seen from the plots, most learning occurs in the initial epochs, which is an interesting observation. A future work would be to probe into the reason for such a behavior of our model, and potentially address the underlying issue to improve the learning of the model. When we trained the reference model [4] using our three-year QQQ price data, we obtained an adjusted MSE of 37. The author of the reference model had trained it using 15 years of stock price data and obtained an adjusted MSE of 7. This indicates that in both cases the models are suffering from a lack of data. Future work would also rely on us gathering more data. In turn, this would mean that we need to find additional sources for historical news data, or find other creative ways to assess stock sentiment from other sources (e.g social media, blogs, etc.).

## 6 Conclusion/Future Work

In our work, we were able to show that including sentiment data provides a marked improvement over simply inputting stock prices. Further, although our test accuracy hovered around 45%, we now know that including additional years worth of sentiment data would improve accuracy even higher, providing strong motivation to add more data in a future iteration. Additional data would also likely allow for a transition to a price prediction model, as the "buckets" idea was used just to provide leeway in prediction. Another potential idea would be to take into consideration the fortunes of competitors to the constituents of QQQ. For instance, in the case of Pepsi, a drop in Coca-Cola's fortunes could be better for Pepsi from a competitive perspective, or it could signal sector instability which negatively affects both of them. Integrating some sector indices which measure the health of any particular sector might be helpful, and it is certainly something we will look into for the future. Nonetheless, the work we have done shows that stock prediction is a promising idea which can likely be used advantageously as the field develops.

# 7 Contributions

All three team members worked on and contributed to all sections of this project.

# References


[1] https://cs230.stanford.edu/projects_spring_2018/reports/8290411.pdf

[2] https://www.aclweb.org/anthology/W19-6403.pdf

[3] A. Ng, K. Katanforoosh, and Y. Mourri. "Convolutional Neural Networks," 2020, https://www.coursera.org/learn/deep-neural-network/notebook/AH2rK/tensorflow

[4] https://towardsdatascience.com/getting-rich-quick-with-machine-learning-and-stock-market-predictions-696802da94fe

[5] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0223593

[6] https://arxiv.org/pdf/1909.12227.pdf

[7] https://learn.datacamp.com/projects/611

[8] https://www.etf.com/QQQ#tradability

## Appendix

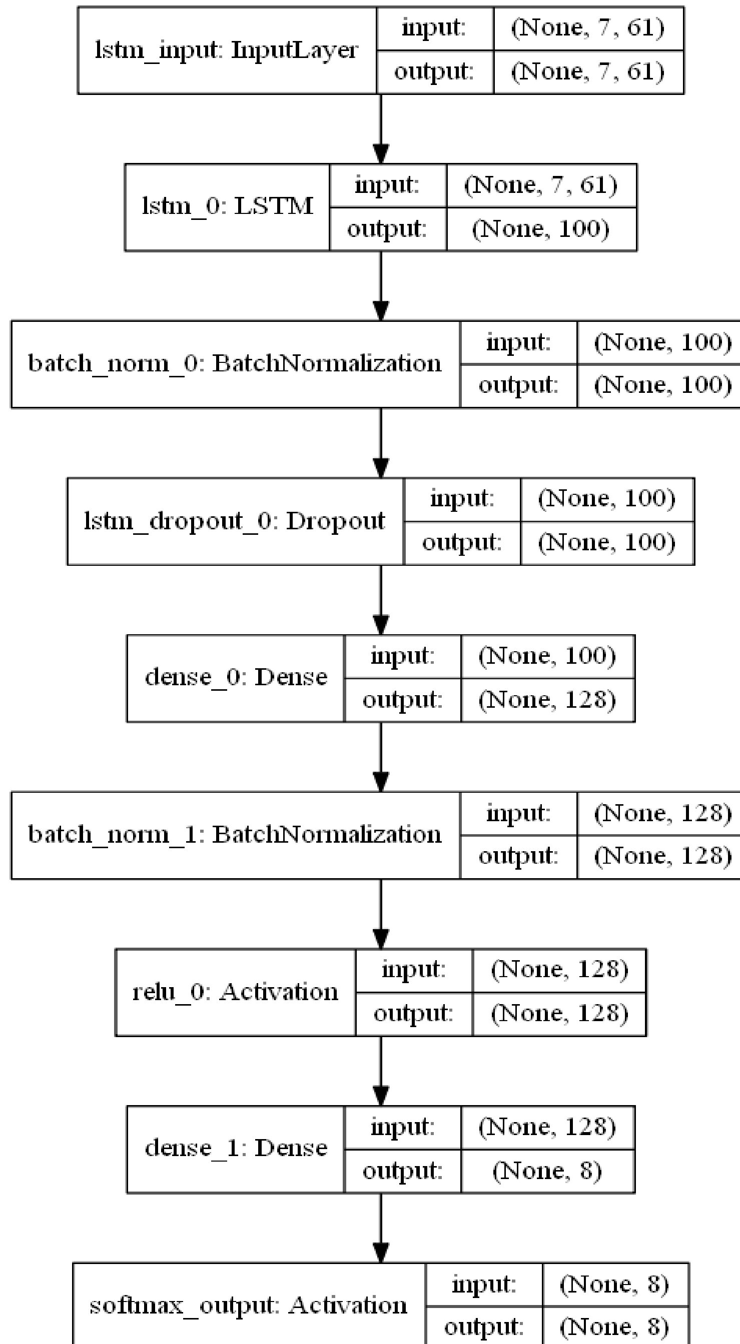



Figure 1: Final Model Architecture

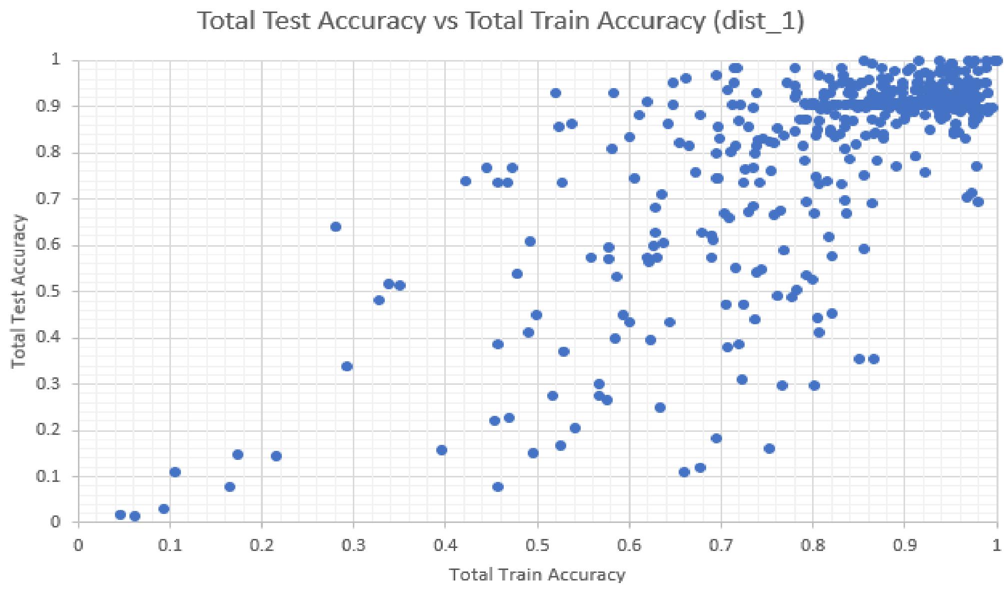



Figure 2: Train vs Test Accuracy with second metric including 1 bucket away from truth