A Beginner Approach to Predicting Stock Closing Price using Long Short-Term Memory Models

Group 11-Cost Function

Zixia Luan
(zixial2)
zixial2@illinois.edu

Alexander Sheehan
(aps2)
aps2@illinois.edu

Ishan Nagpal
(inagpal2)
inagpal2@illinois.edu

Zhuo Lei
(zlei5)
zlei5@illinois.edu

Hannah Gong
(hangong2)
hangong2@illinois.edu

I. Introduction

Since the establishment of stock and equity markets around the world, shareholders and traders have attempted to predict the growth and shrinkage of their shares. While far more attempts ended up being a futile endeavor, the age of deep learning has shed a new light on the methods of predicting an overall trend in the market. Artificial Neural Networks have shown promise in accurately predicting trends in future data, by "learning" from past data.

The ability to predict a trend in market data, and subsequently predict the price of an individual stock, could prove highly lucrative. In an era of low commission fees through various financial companies such as Robinhood and TD Ameritrade, as well as an increased interest in trading without paying into an expensive brokerage firm, personal stock trading continues to grow in popularity. Simultaneously, resources for deep learning have become readily available to the general public in recent years. Many companies now offer subscriptions to cloud computing resources made specifically for deep learning, such as Amazon Web Services and Google Colab.

For our final project in STAT 430: FDL, we will construct an Artificial Neural Network that can accurately recognize an increasing or decreasing trend in market data, and predict the closing price for the following day. This will be achieved through the use of the plethora of financial data available for free to the general public. Many websites provide daily quote data for individual stocks that date back a varying number of years. We have narrowed down the stocks we will be using to Apple Inc. (AAPL), Google LLC (GOOGL), Amazon Inc. (AMZN), and Microsoft Corporation (MSFT). The majority of this data will be used to train our model and the remainder will be used to test it. Each observation used to train and test this model will be the daily closing price of the respective stock. The closing price is the last price the stock was at before the market closed for the day. The input neurons in this model will contain the aforementioned closing prices, and will be the only data used to train this network. Our expected output is the predicted closing price for the next day.

II. Related Work

In order to gain a solid understanding of how to predict stock market trends and prices using deep learning methods, we consulted a number of sources. Below are sources that pertain to basic LSTM architecture, as well as attempts at predicting using LSTM and series data:

[1] This article gives a good basic explanation of an LSTM network. It also includes ways to test the accuracy of our network and how to refine it that we may end up using.

[2] LSTM networks and ARIMA time series models are compared and contrasted in this article. The LSTM section gives a good explanation on the differences between LSTM and a regular neural network, and what each "gate" is for.

[3] The emotional analysis in this article adds an interesting tweak to just a regular LSTM network, although it seems a bit more advanced for what we're learning in class so we most likely will not use that part. However, the article does have a good example of an LSTM network with time-series so we may end up using that.

[4] This article has great examples with actual code that use time-series forecasting. We'll be using the standardization technique used in this article of "differencing" the data to remove the time trend. The model used in this article is close to the one we are building, however instead of using stock prices it uses sales data of shampoo. It also is only a somewhat basic model with only one layer, while we hope to have at least two.

[5] We were able to gain a good baseline understanding of how an LSTM model works compared to a regular neural network. We most likely will not be using the GRU explanation and examples mentioned later in this article.

[6] This paper compared different RNNs architectures including basic RNNs, LSTM and GRU performances on forecasting Google stock price movements with data from 2012-2016. In the conclusion, the study confirmed that LSTMs approach, compared to other random architectures, is able to provide a high enough accuracy, up to 72% for 5 days prediction horizon.

[7] This online tutorial article explained the process of establishing a LSTM model from data exploration to model implementation, giving us more insights on data cleaning, normalizing and visualization.

III. Data

A. Data Collection

For our project we are using discrete, numerical data and it was collected from the Nasdaq Composite's website [8]. In our proposal we mentioned that initially we were using Yahoo Finance, but we found that the Nasdaq website offers larger datasets. Nasdaq generates a comma separated value file for each stock listed on their website. The csv file contains 6 variables: "Date", "Close", "Volume", "Open", " High", and "Low". We have approximately 5 years' worth of data, excluding weekends and Federal holidays, so that leaves us with a little over 1260 observations per stock. That being said, we are using the 80/20 test/train method. With this model we will have 1008 train observations, and 252 test observations

Below is an example of our initial data:

Date	Close/Last	Volume	Open	High	Low
5/7/20	$303.74	28803760	$303.22	$305.17	$301.97
5/6/20	$300.63	35583440	$300.46	$303.24	$298.87
5/5/20	$297.56	36937800	$295.06	$301	$294.46
5/4/20	$293.16	33391990	$289.17	$293.69	$286.32
5/1/20	$289.07	60154180	$286.25	$299	$285.85
4/30/20	$293.80	45765970	$289.96	$294.53	$288.35
4/29/20	$287.73	34320200	$284.73	$289.67	$283.89
4/28/20	$278.58	28001190	$285.08	$285.83	$278.20
4/27/20	$283.17	29271890	$281.80	$284.54	$279.95
4/24/20	$282.97	31627180	$277.20	$283.01	$277

B. Data Normalization and Augmentation

Our dataset initially required some cleaning and overall preparation. The first observation in our data was the most recent closing price, while the last was the oldest. Using a simple indexing method from the Pandas library, we were able to flip the entire dataset. Since we will only be working with the closing price of each stock, there is little use in passing in "Volume", "Open", "High", and "Low". These columns were promptly dropped. Each entry for closing price has a '$' sign at the beginning, and as a result is read in as a string. This issue was solved by using a simple "replace" method for each observation. A strptime method was also used to parse in each observation in the "Date" column and change its data type from a String to a Time Structure.

Because we will use the Keras API form TensorFlow to build, train, and test our model, we have to be sure our data is structured in an input/output format. The "Output" in this sense will be the time (t), and the input is the prior time (t-1). We will use the shift() function to move our initial closing price data downward by one observation, so that there is an empty row at the top of the input columns. This will be replaced by a 0.

Since each observation in our data is a specific day within the last ten years, we will need to remove the time trend. To do this, we will "difference" the data, by subtracting each observation by the previous so t - (t-1). This will remove the increasing trend that is caused by our time variable. Since our activation functions will be the hyperbolic tangent function, the default activation for LSTMs, we will scale our data to [-1,1]. To do this we will use the "MinMaxScaler" method from the sklearn preprocessing library.

Below is an example of our data after cleaning and normalization, including both the aforementioned input and output columns:

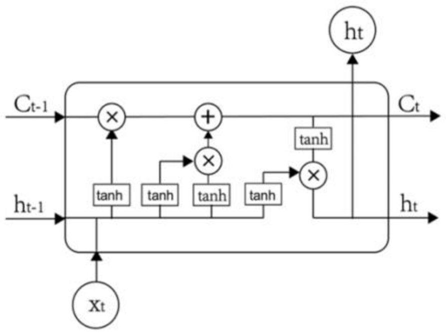
Date	Close/Last	Close/Last
5/10/10	0	36.28
5/11/10	36.28	36.65
5/12/10	36.65	37.44
5/13/10	37.44	36.91
5/14/10	36.91	36.26
5/17/10	36.26	36.32
5/18/10	36.32	36.05
5/19/10	36.05	35.48
5/20/10	35.48	33.97
5/21/10	33.97	34.62

We decided against augmenting any data for this model. Firstly, since each observation in our data is heavily dependent not only on the time set directly before it (t-1), but also nearly *all* of the time steps before it (t-2, t-3, t-n….), augmenting individual observations would make little sense in this real-world problem. More simply, we would not expect the closing price of a stock to increase or decrease by such a large amount for a single step (t), if it did not increase or decrease at all in the previous time step (t-1). Secondly, if more training data was needed, it would make more sense to incorporate new data of different stocks, than it would be to create artificial data from the stocks you are currently working with. This is because the closing price data we are using is from the Nasdaq Composite. The Composite is made up of various different stocks, and is used as index to gauge the performance of the economy. If more data is pulled and used to train this model as we did, we would assume each stock to follow relatively the same trend as the one initially being predicted.

IV. Methods

Our final model will utilize a special type of Recurrent Neural Network, called Long Short-Term Memory. First off, a Recurrent Neural Network (RNN) is based on the basic feedforward neural network architecture. However, RNN differentiates itself from regular Neural Networks and Convolutional Neural Networks because it is able to process data in a specific sequence. The aforementioned Convolutional Neural Network, as well as traditional neural networks, assume that each input is independent of one another. A Recurrent Neural Network is perfect for our application, since the closing price of a stock is very much dependent on the closing price the day before.

As stated before, we will be using a Long Short-Term Memory network. A Long Short-Term Memory (LSTM) network improves upon an RNN by eliminating the "long-term dependency problem", that is to say that they are able to remember important information from an earlier point in the network very easily. LSTM networks do this by "forgetting" information that it deems unimportant, and remembering information that it deems worthwhile to keep. LSTM networks do this through a series of "gates" that are contained within each activation layer of a network. Below is a diagram detailing how LSTM layers process data:



Since LSTM networks are a new concept to us, we decided to take independent approaches and create different models. We ended up creating two different LSTM models, as well as a basic Auto Regressive Integrated Moving Average (ARIMA) model to compare the fit of the LSTM models to. For our activation functions, we were unsure if an activation layer was supposed to follow each LSTM layer. This is because the output is already passed through the hyperbolic tangent function when it leaves the LSTM layer. Our first attempt at an LSTM model does not contain individual activation layers, while our second LSTM model does. Regardless, every layer that contains an activation function uses the hyperbolic tangent function.

In terms of hyperparameters, we decided we would use Dropout regularization as a means of preventing overfitting. We also decided to use Root Mean Square Error (MSE) as our loss function for both of our LSTM models. Our intuition being that since we were not classifying our data into multiple classes, it would be inappropriate to use a cross-entropy-loss function here.

A. *LSTM Model 1*

Our initial LSTM model has 2 layers in total, 2 LSTM layers with 60 neurons, each followed with Dropout regularization. Since our model only has two hidden layers and is relatively "small" we kept our dropout values at 50% for the first layer and 30% for the second. As mentioned before, our first model did not contain extra activation layers.

B. *LSTM Model 2*

Our revised LSTM model has a total of 6 layers, three of which are LSTM layers with 100, 30, and 3 layers respectively. Each LSTM layer has an extra activation layer using the hyperbolic tangent function. Three Dropout regularizations also follow each LSTM layer, with values of 50%, 30%, and 20% respectively.

C. *ARIMA Model*

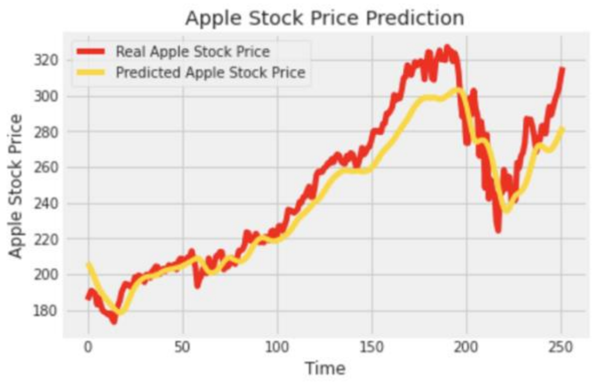
Finally, we constructed an ARIMA model to compare our LSTM models to. We decided on ARIMA model since it is not a deep learning model, but it is commonly used to predict future points in a time series.

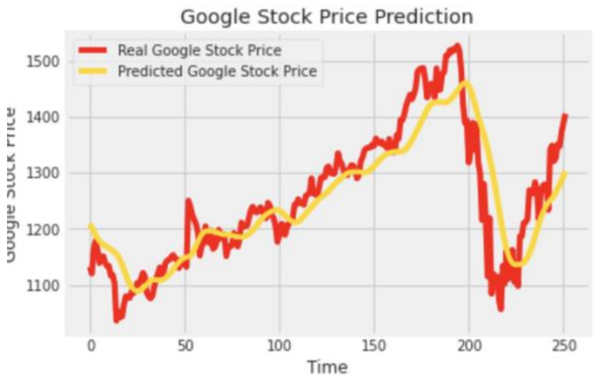
While a standard Recurrent Neural Network could predict stock price like our network to a somewhat accurate degree, we feel that a Long Short-Term Memory model better suits this problem. Using an LTSM model gave us the opportunity to not only gain experience using a different type of neural network, but also to better fine tune our network to remember import points from earlier in our data.

V. Discussion of Results

A. *Classification*

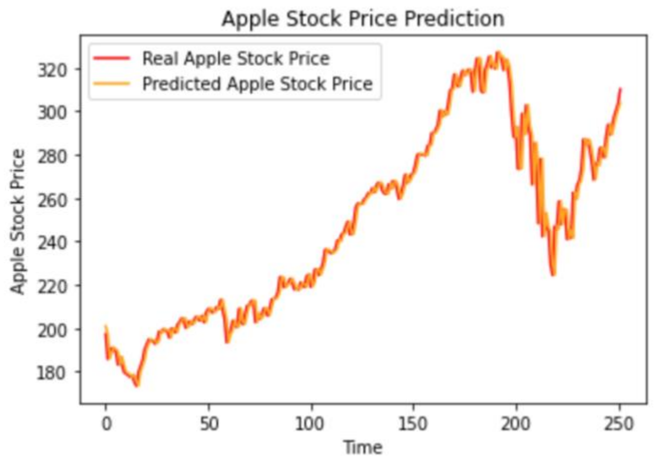
After applying our first LSTM model to our subsetted test data, we graphed our predicted values with the ones actually in our dataset. Below are graphs of two of the four stocks we predicted, Apple and Google:

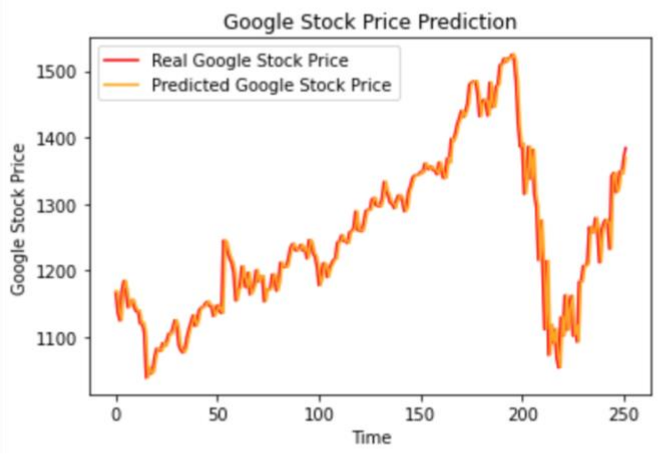




These graphs show that overall, the model can identify and predict with the long upward trend and sudden downward spike in the data. There definitely seems to be some underfitting in this initial model, because the prediction line is offset to the right somewhat from the actual data. We believe the issue is either too many Dropout layers, or the lack of the extra activation layers in this model.

Next, we ran our test data through our larger model. Below are the graphs of Apple and Google, trained and tested with our second LSTM model.



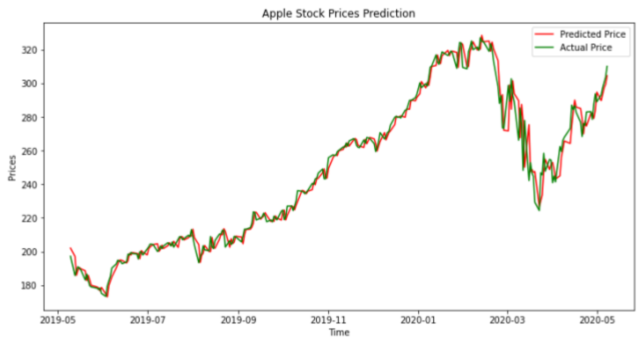


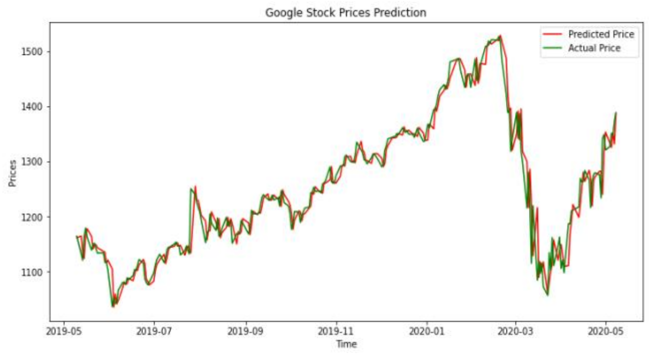
This model seems to not only be predicting with the trend, but actually staying very close to it. There is still *some* evidence of an overfit, but it is negligible when compared to our first LSTM model. A closer look at the predicted vs. expected data gives us a better look at how accurate our model actually is Below is our predicted vs. expected data for Google, the data is recent as of 5/11/20:

```
Predicted=1271.662883, Expected=1232.590000
Predicted=1233.392982, Expected=1342.180000
Predicted=1342.982907, Expected=1346.700000
Predicted=1347.502977, Expected=1317.320000
Predicted=1318.123078, Expected=1322.900000
Predicted=1323.703096, Expected=1349.020000
Predicted=1349.823059, Expected=1345.430000
Predicted=1346.233054, Expected=1369.280000
Predicted=1370.083030, Expected=1384.340000
```

While these predictions are obviously not 100% accurate in most cases, these results do tell us that this model can recognize the changes in a trend very well. In the first few rows of the above image a sudden decrease followed by a major increase can be seen. The model correctly accounts for this sudden change by changing the direction in which it is predicting.

Now we will compare our LSTM models to a model derived from Time Series Analysis, the ARIMA model. Below are the ARIMA graphs using the same Apple and Google data we used for our LSTM models.



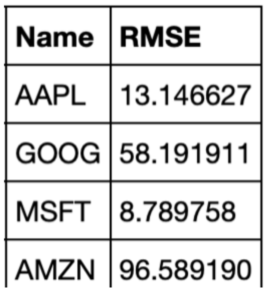


When compared to our prior models, it seems our second model predicts better than what a simple time series analysis. Which given what we've learned in STAT 430, this should be the case.

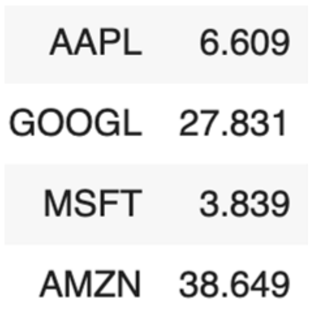
B. *Regression*

As mentioned before, we used Root Mean Square Error (RMSE) as our loss function. Since RMSE is the standard deviation of the error in our model, the lower our RMSE, the better. Below is a comparison of RMSE values for each of the

4 stocks we predicted, between our first and second LSTM models, respectively.

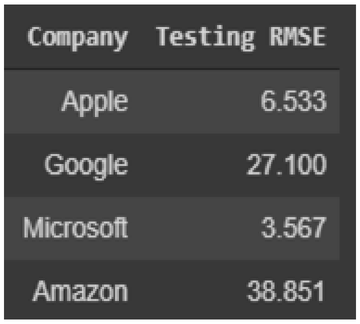


Name	RMSE
AAPL	13.146627
GOOG	58.191911
MSFT	8.789758
AMZN	96.589190



AAPL	6.609
GOOGL	27.831
MSFT	3.839
AMZN	38.649

It's immediately obvious that our second model predicts far better than the first. It seems that each RMSE value from the first model is more than halved in the second model. With this we can confidently say that our second model is a major improvement over our initial attempt. Finally, we will compare the RMSE of our second model with the RMSE of our ARIMA model. The RMSE's of the ARIMA model can be found below:



Company	Testing RMSE
Apple	6.533
Google	27.100
Microsoft	3.567
Amazon	38.851

VI. Conclusion and Future Work

Given the results mentioned we can firstly conclude that a Long Short-Term Memory model is a fantastic tool used to predict time series data, and is a great improvement off of a regular Recurrent Neural Network. Secondly, we can confidently say that adding an independent activation layer after each LSTM layer adds tremendous predicting power.

In future work, we would be curious to see how changing activation functions affect predictions. We decided on the hyperbolic tangent function since it was described as the "default" activation for an LSTM network. Computing hyperbolic tangent also resulted in long computation times. Perhaps we could implement some sort of parallel programming to minimize our runtime. We also feel that in hindsight, using only the closing prices of stocks is too singular. We would be interested to see the effects of using stock-related news or quarterly earnings reports in addition to historical closing prices. A natural language processing model could be used to scrape various Bloomberg articles and perform a sentiment analysis to gauge overall market sentiment on a particular stock. Using these two models, we could construct a training bot that makes trades based on the models' predictions.

VII. References

[1] Karim, Fazle, et al. "LSTM fully convolutional networks for time series classification." *IEEE access* 6 (2017): 1662-1669.

[2] Siami-Namini, Sima, and Akbar Siami Namin. "Forecasting economics and financial time series: ARIMA vs. LSTM." *arXiv preprint arXiv:1803.06386* (2018).

[3] Zhuge, Qun, Lingyu Xu, and Gaowei Zhang. "LSTM Neural Network with Emotional Analysis for Prediction of Stock Price." *Engineering letters* 25.2 (2017).

[4] Brownlee, Jason. "Time Series Forecasting with the Long Short-Term Memory Network in Python." *Machine Learning Mastery*, 5 Aug. 2019, machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/.

[5] Nguyen, Michael. "Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation." *Medium*, Towards Data Science, 10 July 2019, towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[6] Persio, L. D., & Honchar, O. (2017). Recurrent neural networks approach to the financial forecast of Google assets. *INTERNATIONAL JOURNAL OF MATHEMATICS AND COMPUTERS IN SIMULATION*, *11*, 7–13.

[7] "(Tutorial) LSTM in Python: Stock Market Predictions." *DataCamp Community*, www.datacamp.com/community/tutorials/lstm-python-stock-market.

[8] "Exercise and stress: Get moving to manage stress," 08-Mar-2018. [Online]. Available: https://www.mayoclinic.org/healthy-lifestyle/stress-management/in-depth/exercise-and-stress/art-20044469.

Here is the distribution of work amongst the final report for our project:
Introduction – Alex 20%
Data – Zixia 20%
Technical Details – Ishan 20% and Zhuo 10%
Results – Zhuo 10%
Citations – Hannah 20%