Predictive Modeling with LSTM and Python

1st Joseph Trierweiler
Computer Science and Engineering Department
University of Nevada Reno
Reno, United States
https://github.com/jotreewater/Deep_Learning_Project.git

2nd Harrison Hong
Computer Science and Engineering Department
University of Nevada Reno
Reno, United States

***Abstract*—Forecasting stock data is one of the most potentially lucrative projects a data scientist can undertake. It also makes for an easy starting point to learn about machine learning, as the amount of preprocessing required is minimal. Thus Team OR4 began research into how to create an accurate predictive model for forecasting stock data. From that research, the team found the LSTM model and its extensions to be the most popular machine learning model for studying stock data. The Team's implementation takes this research a step further, and optimizes such models to find what parameters create the most accurate models. What was found was that different predictive model settings worked better for different data sets. Tailoring parameters to their stock data sets received better results than what would have received with only one model.**

I. Introduction

The project of Team OR4 was to build a model using recurrent neural networks to predict future stock market values. The team utilized tools such as Python, Tensorflow, Keras, Jupyter Notebook, and Tensorboard to develop the project. The implementation consisted of utilizing various deep learning models such as stacked long short-term memory(LSTM) and bidirectional LSTM. To optimize the model's results for lowest loss, the team adjusted parameters such as test ratios, look backs, models, layers, and epochs. Lastly, the team tested the various models on different stock data sets such as Advanced Micro Devices, Microsoft, Apple, and Disney stock.

II. Tools

A. Python

The team had to figure out which tools would be best to implement the project. The first tool that was decided upon was Python. The team found that Python is the most popular language for data scientists and machine learning developers [1]. Part of the reason for Python's popularity is how simple it is to code with. All library extensions can be found using the 'pip' package manager, and all data types are implicit, meaning they don't have to be defined. There is also several powerful deep learning libraries written for python. This allowed the team to focus almost exclusively on the manipulating the data rather than optimal coding practice.

B. Libraries

Most of the libraries used in the project had to do with data preprocessing. One of the libraries that the team utilized was Pandas which allowed the program to read in stock data from a csv file and convert it to an easy-to-use data set. This plays nicely with the next library that was utilized named Numpy, which allowed Team OR4 to transform the data into dimensional arrays. These arrays were then scaled down to float values using SKlearn. This library was useful for transforming the data to be between 0 and 1 while maintaining its information. Proof of its effect can be seen in figure 1. The reason for doing this was because LSTM reads sequence data in this format. Already having that data scaled decreased the amount of time it took to process and also improved model accuracy. The last library that was utilized was Pyplot. This library helped the team visualize the data to see what stock prices looked like relative to time. This aided in evaluating how well a model's prediction did compared to the stock's actual closing price. The team also utilized the Tensorflow and Keras libraries for performing deep learning techniques.

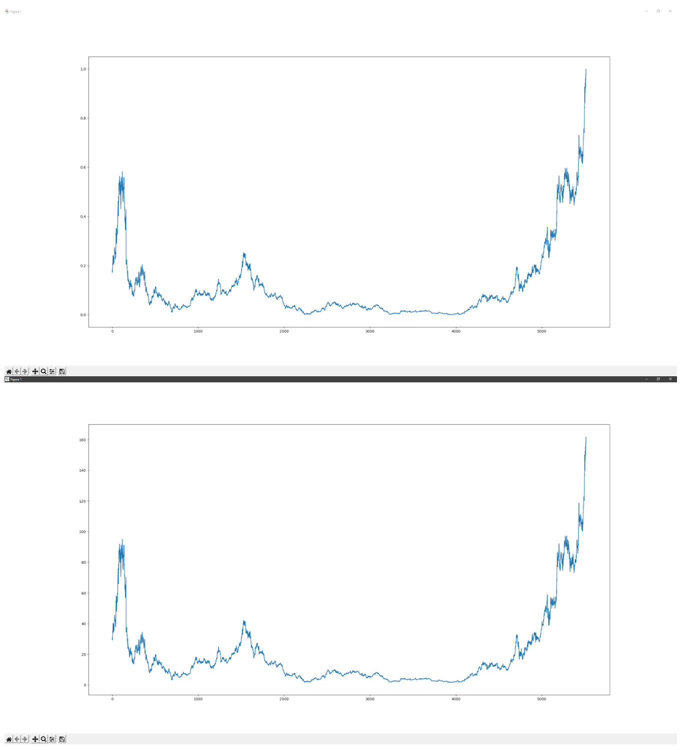


Fig. 1: Two plots showing how the stock data set can be transformed from USD to be floats between zero and one without losing any information

C. *Deep Learning Tools*

Tensorflow is a deep learning framework developed by Google that has libraries for Python development. There are a couple reasons the team elected to use Tensorflow over other libraries, such as Pytorch. The first reason was because of Tensorboard, which was a great aid for data visualization. Tensorboard allowed for the team to see how each variation of the model was performing in relation to its peers. Another factor the team considered was the learning curve between frameworks. Tensorflow has the reputation of having a Fsteeper learning curve than Pytorch, however this can be addressed with the framework Keras. Keras is a high-level library built on top of Tensorflow that abstracts some of the lower level features that Tensorflow is capable of. Essentially it reduces the amount of knowledge the user needs of deep learning to begin getting their feet wet. Tensorflow's abundant documentation made the team feel comfortable in their choice and allowed Team OR4 to create two primary deep learning models for the project.

III. Deep Learning Models

Both of the deep learning models implemented by Team OR4 utilized Long Short Term Memory networks (LSTM). LSTM networks are a type of recurrent neural network (RNN) used for recognizing patterns in sequential data [2]. An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of memory found in a computer. Each block contains multiplicative units that can consist of either an input, an output, or a forget gate. Through feeding sequential data through this network the outputs create a model that can be used for forecasting sequential data such as stock data. The two different LSTM models that the team implemented for experimentation were the stacked LSTM and the bidirectional LSTM models.

A. *Stacked LSTM*

Stacked LSTM is a deep learning technique in which sequential data output from an LSTM model is then put through another layer of LSTM learning. This makes makes the model 'deeper,' potentially reducing the amount memory blocks required in each layer to attain the same, if not better, results [3]. The effect of this layering is creating multiple interpretations of the data over different time scales. All of this data is then compiled back into a single model using a final dense layer after the LSTM layers as shown in Figure 2. This deep learning technique has become a stable for understanding complex sequence prediction problems such as the one being explored in this project.

B. *Bidirectional LSTM*

Bidirectional LSTM is similar to a stacked LSTM in that both use multiple layers of LSTM to accomplish deep learning. However what makes bidirectional LSTM distinct is that it has one layer that takes the sequence as is, and another that trains on a reversed copy of the first [4]. This helps give the network gain additional context and can again aid in creating a faster and fuller result than traditional LSTM learning.

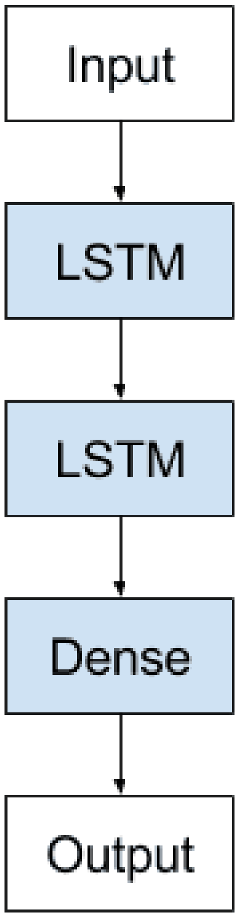


Fig. 2: A Visual Representation of a Stacked LSTM

IV. Implementation

There are a set of general development steps recommended by the Keras documentation that Team OR4 chose to follow [5]. These steps were as follows: access and collect the data, prepare the data, build and train the model, evaluate the model, and then to make predictions using the model. Each of these steps were integral for evaluating the model's results. A link to implementation can be found under Joseph's author block.

A. *Data Collection and Preparation*

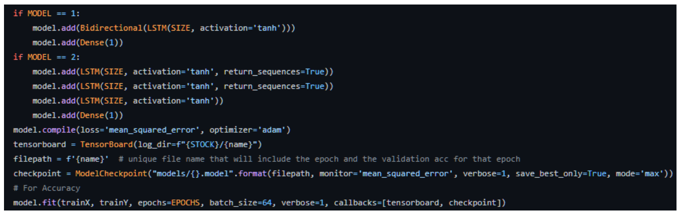
The first step was to find a way to load data, which was accomplished via the API 'Tiingo.' Tiingo can be used to retrieve stock closing prices from the internet as CSV files. The second step is to preprocess the data for the model. As mentioned in section II-B, Team OR4 used a series of libraries to do this. The data was read from the CSV files with Pandas and transformed into data sets. These data sets contained NumPy dimensional arrays with dimensions for closing prices with dimensions depending on the number of ”lookbacks” defined by the user. This is one of the parameters that was adjusted during development that will be further discussed in section IV-C. The data set was then split into a training set, and a testing set. The percentage split between these sets was another parameter adjusted during optimization. The purpose of this methodology is to give the model the training data set for learning, and then to evaluate its performance on the testing data set. [5].

B. *Building the Model*

As mentioned in section III there were two deep learning techniques implemented for building models. Model 1 was a Bidirectional LSTM method, the code for which can be seen in Figure 3. The number of memory cells allocated for each layer was defined by a size parameter that was adjusted during optimization. The effect it has on model prediction can be seen in section V. Layers were activated with a tanh function. This was required for the model to be compatible with Nvidia's GPU CudaNN drivers. These drivers drastically sped up the time spent training via leveraging the cudacores in the team's GTX 1080Ti. Model 2 was a Stacked LSTM deep learning method. A point of note is that each LSTM function has a parameter to return sequence data. It allows layers to build off one another's memory cells, which is what makes stacked LSTM so powerful [3]. The final LSTM layer does not return sequence data because it's passing its results to a Dense layer. The Dense layer in both models is used to convert the data back into its original dimensions. The "model.compile" function in Keras allows users to define what optimizer to use when fitting the data. Team OR4 chose to use 'adam' as its optimizer and did not mess with Keras's default parameters for it. Loss, or inaccuracy, was measured with mean squared error. Mean squared error is the euclidean distance between two points and can be defined by the equation below.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2 \tag{1}$$

The two points of interest to the compiler was the model's prediction value compared to the training data set's value. The Keras function "model.fit" starts the learning process, passing the training data set through the model for a predefined number of epochs. The team used 16 epochs for the examples found in section V. The best model from each epoch is saved with a callback function called checkpoint. These saved models are then ready to be visualized using Tensorboard, which will be further discussed in section IV-C.

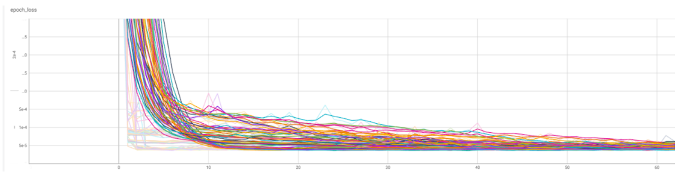


```
if MODEL == 1:
    model.add(Bidirectional(LSTM(SIZE, activation='tanh')))
    model.add(Dense(1))
if MODEL == 2:
    model.add(LSTM(SIZE, activation='tanh', return_sequences=True))
    model.add(LSTM(SIZE, activation='tanh', return_sequences=True))
    model.add(LSTM(SIZE, activation='tanh'))
    model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
tensorboard = TensorBoard(log_dir=f"{STOCK}/{name}")
filepath = f'{name}'  # unique file name that will include the epoch and the validation acc for that epoch
checkpoint = ModelCheckpoint("models/{}.model".format(filepath, monitor='mean_squared_error', verbose=1, save_best_only=True, mode='max'))
# For Accuracy
model.fit(trainX, trainY, epochs=EPOCHS, batch_size=64, verbose=1, callbacks=[tensorboard, checkpoint])
```

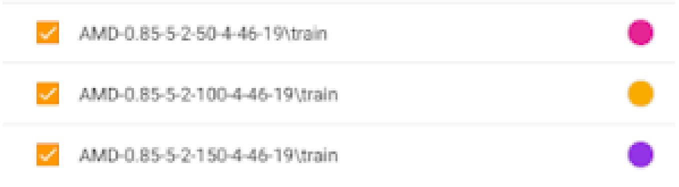
Fig. 3: A snippet of the code used for building the model

C. *Evaluation and Optimization*

After the model is built, can be tested using the Keras function "keras.evaluate". This function uses the model generated from the training data set and tries to predict values from the test data set. The result of the evaluation is the models total MSE as found in equation 1. For optimization, the team defined its goal was to minimize the MSE in the model. There is a plethora of parameters that can be manipulated to optimize the model. So many in fact, that it was difficult to diagnose what parameters were important for reducing MSE, so an automated approach was taken for finding out. There were 4 automated parameters in total: various test ratios, the number of look backs, which model to use, and how many memory blocks per LSTM layer to use. These parameters were then plotted using Tensorboard, as shown in figure 4a, for comparison. Each model was given a unique name representing its parameters as shown in figure 4b. The specific naming convention was the name of the stock, followed by the percentage of the dataset used for training, how many lookbacks were allowed, which model was chosen, the amount of memory cells permitted for each layer, and the time that the model was generated. The model with the lowest MSE would then be selected as the best model for that stock. The results of this evaluation is what will be discussed in section V



(a) Tensorboard visualization of each model's performance



✓ AMD-0.85-5-2-50-4-46-19\train ●

✓ AMD-0.85-5-2-100-4-46-19\train ●

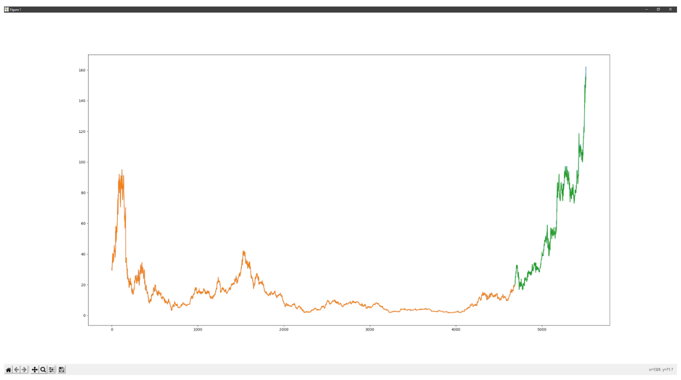
✓ AMD-0.85-5-2-150-4-46-19\train ●

(b) Model names with model parameters

Fig. 4: How optimization was performed using Tensorboard

V. Results

After the best model for a stock is found, an inverse transformation is applied on the training and testing data set for the model for visualization purposes. Such visualization can be found in figure 5a. From the AMD stock data set example, the MSE during evaluation was 0.010029%. The parameters for this model was a training percentage of 75%, the amount of look backs was only one, it used the stacked LSTM model, and it used 150 memory cells. The team found this MSE range to be acceptable, however it was found that the parameters as well as the MSE would change between stock data sets. For example, when applying the same program with Microsoft stock, it preferred to use the second learning model over the first one, and had a slightly higher MSE at 0.011902%. Apple stock preferred using model 2 but with less memory layers and had a much higher MSE of 0.024533%. Finally Disney stock was similar to apple stock in its parameters, but had around half the MSE at 0.01411%. The only parameters that were consistent between our results was keeping the number of look backs at 1, and to use a 75% training ratio when fitting and evaluating the model. The justifications for these results are speculation due to the black box nature of deep learning, however, our hypothesis is that the model was satisfied with the amount of data it had. Giving it more data to work with might actually hurt its performance through over training. LSTM models tend to be resistant to over training due to their forget gates, but it might not be a coincidence that both constant parameters had to do with reducing the amount of data for fitting the model [3]. Another point of concern is the higher than average MSE found in the Apple data set. It was the only data set to have a MSE above 0.01%, which could provide a clue as to why the model was struggling. The hypothesis the team came up with, is that the model struggles with data sets that have stock splits. The reason is that Apple stock was the only stock data set that had stock splits, however further testing is needed to confirm this notion.


(a) Training and testing data overlaid on top of AMD dataset

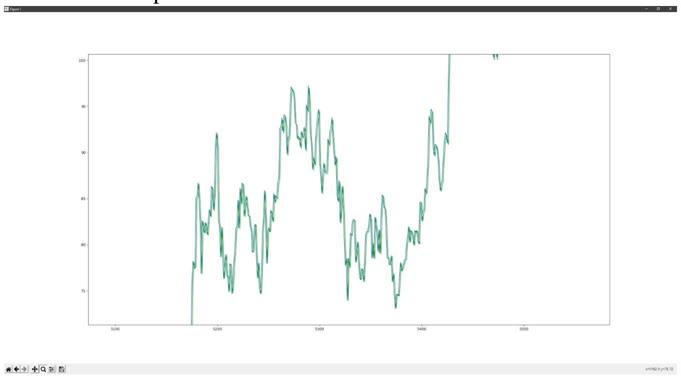
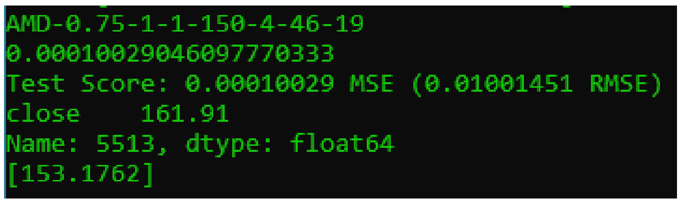

(b) Zoomed in view of testing data overlaid on top of AMD dataset

Fig. 5: Visualization of Model Results

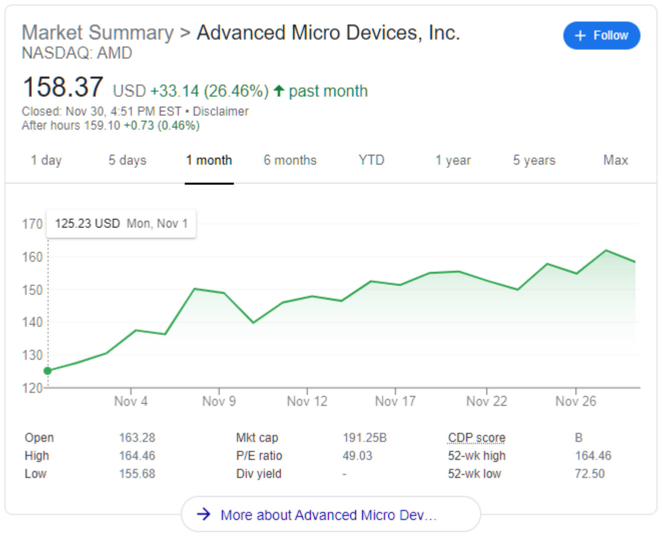
VI. Conclusion

Team OR4 accomplished its goal of using recurrent neural networks to accurately forecast stock prices. However, if asked what parameters produces the best results, the answer is more nuanced than a simple answer of Model 1 or Model 2. It was found that different data sets perform better with different model parameters. Further development of this project would be to implement some form of trading logic to utilize the predictions made by the deep learning model. Currently the model predicts one day into the future for what tomorrows stock price will be as shown in figure 6, but it could also be extended further for long term investments. Before being deployed for trading however, the hypotheses in section V should be proven to be accurate through further testing. Overall the team was satisfied with the models accuracy and will continue development for deployment at a later date.



```
AMD-0.75-1-1-150-4-46-19
0.00010029046097770333
Test Score: 0.00010029 MSE (0.01001451 RMSE)
close    161.91
Name: 5513, dtype: float64
[153.1762]
```

(a) Model results for AMD and the next predicted closing price



Market Summary > Advanced Micro Devices, Inc.
NASDAQ: AMD

158.37 USD +33.14 (26.46%) ↑ past month

Closed: Nov 30, 4:51 PM EST • Disclaimer
After hours 159.10 +0.73 (0.46%)

Open	163.28	Mkt cap	191.25B	CDP score	B
High	164.46	P/E ratio	49.03	52-wk high	164.46
Low	155.68	Div yield	-	52-wk low	72.50

→ More about Advanced Micro Dev...

(b) The next days actual closing price

Fig. 6: Example of the model forecasting a future stock price

References

[1] C, Voskoglou, What is the best programming language for Machine Learning?, Developer Nation, May. 2017. Accessed on: Dec. 10, 2021. [Online]. Available: https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7

[2] J, Brownlee, A Gentle Introduction to Long Short-Term Memory Networks by the Experts, Machine Learning Mastery, May. 2017. Accessed on: Dec. 10, 2021. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

[3] J, Brownlee,Stacked Long Short-Term Memory Networks, Machine Learning Mastery, April. 2019. Accessed on: Dec. 10, 2021. [Online]. Available: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

[4] J, Brownlee,How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras, Machine Learning Mastery, June. 2017. Accessed on: Dec. 10, 2021. [Online]. Available: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[5] fchollet, The Sequential model, Keras Developer Guides, April. 2020. Accessed on: Dec. 10, 2021. [Online]. Available: https://keras.io/guides/sequential_model/