S&P 500 Prediction Using Long Short-Term Memory Neural (LSTM) Networks

Kazuya Okamoto, Minji Yun, Om Chaudhary, and Sean Chen
Department of Computer Science, University of California, Davis
1 Shields Avenue, Davis, CA 95616, United States of America
kaokamoto@ucdavis.edu, myun@ucdavis.edu, ochaudhary@ucdavis.edu
Department of Statistics, University of California, Davis
1 Shields Avenue, Davis, CA 95616, United States of America
seache@ucdavis.edu

***Abstract*— In this paper, it explains what Long Short-Term Memory (LSTM) model is, how it is used in machine learning, and how it is implemented in this project using S&P 500 and NASDAQ 100 dataset. The paper closely examines the models parameters and how this project utilized and fine tuned them to achieve better result quality. The ultimate goal of this project is to predict the stock market prices in near future and to possibly predict the next crash.**

Keywords— LSTM, Tensorflow, Mean Squared Error (MSE), Neural Networks, accuracy, Keras, preprocessing.

I. Introduction/Background

Stock prediction is an immensely important yet complex task. Accurately predicting stock prices is crucial to countless industries and has extensive impact across various careers. Stock prediction has far reaching implications not only for investors and financial institutions, but for the overall stability of the market and broader economy. In specific, the S&P 500, one of the most widely followed stock market indices in the US, serves as an indicator for the wider condition of the economy and provides a benchmark for investment performance. For financial institutions, accurately predicting the S&P 500 allows for informed decision making and risk assessment. On a broader scale, accurately predicting this index enables stable financial markets which in turn leads to a healthy and stable economy. Clearly, predicting the S&P 500 accurately is an important task- though notoriously challenging due to the complex nature of financial markets.

In the field of financial prediction, various machine learning techniques have dominated the landscape in recent years. Unlike many traditional and outdated statistical models that are limited to linear relationships, machine learning models have the capability to learn from huge datasets while discovering complex patterns and adapting to changing markets.

Across all various machine learning approaches and implementations, deep learning has proved especially useful in the task of predicting financial data. Deep learning utilizes neural networks with multiple layers that can be trained on raw data. For our task of predicting the S&P 500, there were multiple different AI models in consideration. Initially, we considered three different models: Gradient Boosting Machines (GBM), Logistic Regression, and Long Short-Term Memory Networks (LSTM).

RNNs (Recurrent neural networks) are a specific type of deep learning model that are designed to process sequential data, making them ideal for time series analysis. Traditional RNNs (Recurrent Neural Network) face the vanishing gradient problem, caused when the gradients used to update the weights of a neural network become too small, rendering them ineffective. This problem is even more prevalent with longer sequences of data, such as our S&P dataset. However, one particular model, the LSTM (Long Short-Term Memory), is specifically designed to combat this issue. LSTMs are a special type of RNN that can learn long-term dependencies. These were introduced by Hochreiter & Schmidhuber (1997) and refined and disseminated by many in subsequent studies1. LSTMs mitigate the vanishing gradient problem by utilizing a unique architecture of memory cells with a system of three gates. It is explicitly designed to avoid long-term dependency problems. Remembering information for long periods of time is practically its default behavior. This means LSTMs are useful for time series data with long term dependencies, the exact quality needed for a model to accurately predict stocks.

In the specific context of stock market prediction, LSTMs have multiple strengths when compared to other traditional machine learning models.

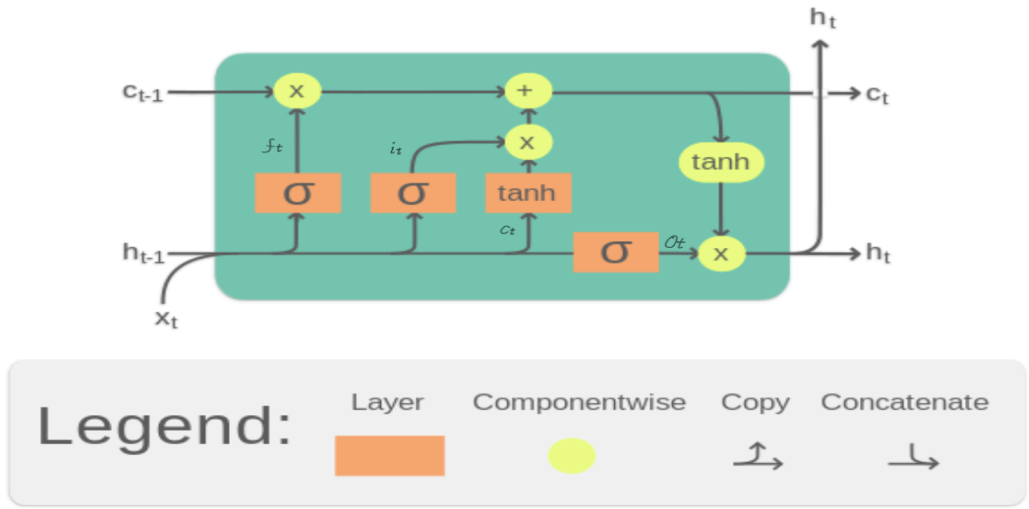

Fig. 1. LSTM Cell Process cell [4]

For example, both GBM and Logistic Regression struggle with time dependencies. GBM models need considerable feature engineering and need to use 'lagged variables' when predicting patterns in time series. Logistic Regression, a simpler model used for binary classification, can not adequately handle the complexities of time series data. On the other hand, LSTMs inherent architecture makes them good at recognizing patterns both short term, and long term. LSTMs are built to learn from raw data and capture the complexities of long term dependencies, all while being robust to noise and requiring less feature engineering than other traditional models[4].

This study seeks to provide a comprehensive analysis of LSTM-based stock prediction models, evaluating their performance, and identifying key factors that influence their accuracy. All recurrent neural networks are chains of repeating neural network modules. In a standard RNN, this iterative module has a single tanh layer for each cell.

LSTM also has this chain-like structure, but the repeating module has a different structure. Instead of a single neural network layer, it has four layers that interact in a very special way.

Equations for the layers are as follows:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$g_t = \phi(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$
$$o_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \phi(c_t)$$

(where c_{t-1} is state of last cell, h_t is last hidden state, x_t is the input at time t.)

A. Forget Gate (f_t)

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

The forget gate is calculated using the current input (x_t) and the previous hidden state (h_{t-1}).

It is transformed into a value between 0 and 1 using the sigmoid function (σ). This value determines which parts of the cell state (c_{t-1}) should be forgotten.

B. Input Gate (i_t)

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

The input gate decides which new information should be stored in the cell state.

It is calculated using the sigmoid function (σ).

C. Cell Candidate (g_t)

$$g_t = \phi(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$

The new candidate cell state is calculated using the hyperbolic tangent function (ϕ).

This results in values in the range of -1 to 1.

D. Output Gate (o_t)

$$o_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

The output gate determines the hidden state (h_t) from the current cell state (c_t). It is calculated using the sigmoid function (σ).

E. Updating the Cell State (c_t)

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

F. Updating the Hidden State (h_t)

$$h_t = o_t \odot \phi(c_t)$$

The hidden state is obtained by applying the hyperbolic tangent function (ϕ) to the updated cell state (c_t), and then taking the element-wise product with the output gate (o_t).

Following this detailed explanation of LSTM models' structure and functionality, it is essential to also properly understand the various parameters that influence their performance. Our LSTM model utilizes several key parameters that all convey crucial information in relation to its effectiveness and performance. These parameters include: input variables, time steps, data normalization, accuracy, mean squared error (MSE). This following provides a brief overview of these basic parameters and their respective utility within our LSTM model.

G. Input Variables

Input variables are what we call features or predictors. These features are directly used by the model to make predictions. When training and testing AI models, selecting the correct input variables is crucial as they directly impact the model's ability to capture patterns/trends in the data. In the context of a CSV file, each column represents a feature, and is assigned to the variable X.

H. Time Step

A time step in general analysis means "a way of examining and analyzing your data through specified time intervals"[2]. In the context of our model, it represents the number of data points a LSTM network considers at a time when making future predictions. It constitutes "how many values exist in a sequence"[3]. Selecting an appropriate time step length is crucial to capture relevant patterns/trends and ensure model accuracy.

I. Data Normalization

Data normalization is an important part of preprocessing. It allows data to be structured and organized into a common scale, ensuring that all features proportionally impact the model. Normalizing data correctly guarantees that different ranges within the data do not distort the model's performance.

J. Accuracy

Accuracy in terms of AI model training is the measure of correct prediction the AI model is outputting. It is calculated by dividing the number of correctly predicted outputs by the total number of outputs.

K. Mean Squared Error (MSE)

MSE stands for mean squared error, which is utilized for checking how close the predictions are to true values. In other words, it shows how good the AI model is at modeling the given dataset. The closer it is to zero, the better it is - we want to minimize MSE to get more accurate models.

L. Reasons For Choosing LSTM

Our dataset consists of more than 10,000 data points, which means we can confidently declare it to be a large dataset. As mentioned earlier, we initially had three models up for selection: Gradient Boosting Machines (GBM), Logistic Regression, and Long Short-Term Memory Networks (LSTM). All three of these models are fit for large datasets.

For GBM, it had high predictive accuracy and was able to catch high prediction accuracy but was going to be computationally expensive.

For logistic regression, it is able to predict based on known conditions, it is simple and really good for when there are two or more responsive variables. Our variables were change percent, binary crash indicator (equals 1 when a substantial stock market downturn occurs during month , and 0 otherwise), binary crash signal (equals 1 if the model, incorporating all information available at the end of month t , expects a stock market crash to occur during month t+1, and 0 otherwise.), maximum drawdown(max drop from peak to bottom price in set amount of days). But crucially, we had to choose logistic regression's type such as linear, polynomial, etc. and our dataset had a trend that did not strictly match any of the types, and also assumed a linear relationship between features, which made us feel hesitant on choosing it.

On the other hand, LSTM was known for being flexible, which means it can add time series prediction. This was a very important feature to us since our dataset heavily relied on. It was also known for improving accuracy over larger data. It was also adaptable, meaning that we could add other neural networks to it, potentially opening a way of ensemble learning, which is a way of achieving higher accuracy for training the models.

Some disadvantages LSTM had were computational intensity and complexity, issues with overfitting (which could be fixed using regularization techniques such as dropout, reducing the number of units, L1/L2 regularization, etc.), and the complicatedness of the training process. Despite these disadvantages, we still chose LSTM over other methods since overfitting is a common issue on other models too, and so is computational intensity. The only unique problem it had was the complicated training process, but we believed that we were capable of building and training a working LSTM model. Our main focus was to train it over the large S&P 500 dataset, and let it predict the next stock market crash, or highly accurate future stock prices. We theorized that it would be accurate to some level if we train the LSTM model using at least two regularization techniques, and with the large size we have for our dataset, it will be a perfect model for predicting future prices.

II. Literature review

A. Introduction

The application of machine learning in economics and in financial markets in particular has gained significant attention over recent years. Among various other models and techniques, LSTM networks have proven to be especially promising in the task of predicting stock prices due to their capability to capture long-term dependencies in time series data. This literature review aims to explore the effectiveness of

LSTM models in stock market prediction while highlighting their advantages over traditional models and other deep learning techniques.

B. Body

The foundational work of Hochreiter and Schmidhuber (1997) introduced LSTM networks as a solution to the vanishing gradient problem faced by other traditional Recurrent Neural Networks. Their research paved the way and provided a solid basis for subsequent studies on LSTMs and their utility for time series prediction in financial markets. LSTMs are inherently designed to remember information over long sequences which makes them very suitable for handling the complexities of financial data.

In the specific context of stock market prediction, Shi demonstrated how much more effective LSTM networks are in comparison to traditional models [1]. This study's overarching conclusion showed that LSTM models' performance is significantly greater than other models such as ARIMA and Gradient Boosting Machines. The study highlighted how LSTMs' uniquely excel at capturing temporal dependencies in financial data- a crucial advantage over other methods that struggle with this aspect. Overall, Shi's research emphasized the importance of LSTMs in improving prediction accuracy and adapting to market changes. One strength of this source is that it uses robust methodology, including a detailed comparison of multiple models. However, one primary limitation is the lack of real-time application and performance analysis. This leaves the paper mostly theoretical, while a performance analysis would provide more practical insights.

However, utilizing the findings of another more specific study can provide these additional insights. Nelson, Pereira, and de Oliveira (2017) analyzed the use of LSTM models in the specific case of high-frequency trading. Their research found that these models considerably outperformed other deep learning techniques in terms of both prediction accuracy and computational efficiency. Their study revealed that LSTMs could effectively handle the volatility and rapid fluctuations that are characteristic of high-frequency trading data. They concluded that LSTMS are a valuable tool for financial analysts and traders. One drawback of this study is that the scope is somewhat narrow and only focuses on high-frequency data, which may not generalize to other types of financial time series. Nevertheless, it is apparent that LSTMs have the exact set of traits that makes them ideal for working with stock markets and financial data.

C. Conclusion

The literature and various studies reviewed above demonstrate that LSTM models are highly effective in predicting stock prices primarily due to their ability in capturing long-term dependencies and complex patterns in time series data. They offer significant advantages over traditional machine learning models and other deep learning techniques, especially within the scope of economic data. Future research would benefit by focusing on improving the computational efficiency of LSTM models and further exploring their application in different financial contexts. The strengths of LSTMs, including their robustness to noise and reduced need for feature engineering, make them a valuable tool for financial analysts and traders. However, LSTMs are far from perfect. Their computational demands and sensitivity to hyperparameters highlight the need for further research to continue refining and optimizing these models.

D. Sources

Title of the paper is "Stock Transition Prediction Modeling and Analysis Based on LSTM". It was published in 2018 at the 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), under the doi of 10.1109/ICIEA.2018.8398183. Authors are Siyuan Liu, Guangzhong Liao, Yifan Ding. It was published to the IEEE Xplore site, https://ieeexplore.ieee.org/. It has 55 citations in papers and 3333 full text views. Two of the three authors have 2 publications to this site and 58 citations. All three authors are associated with the School of Computer Science and Technology at Wuhan University of Science and Technology, Wuhan, China.

III. Exploratory data analysis on S&P 500

1. Introduction

a. Data collection (Kaggle)

We used an open source data collection site called Kaggle Datasets. We aimed to search for a dataset with a mix of categorical and numerical data consisting of at least 5 features (columns) and 100 rows, since for training the AI models, we need fairly large datasets to prevent oversimplification and to enhance accuracy.

We searched for datasets with the keywords such as 'stock dataset', 'stock trends', 'stock market', 'stock exchange', etc. Some of the datasets we considered consisted of Inflation Interest Rate with 13 features and 265 rows, IMF FOREX with 31 features and 9,855 rows, Bitcoin Historical Data with 8 features and 3,650 rows, etc.

We finalized our decision for using S&P 500 and NASDAQ 100: Daily Data that has 7 features with 14,235 rows. The reason for choosing this dataset was that it had more than 5 features, 1 categorical and 6 numeric features, plenty of data points to use for fine tuning the model, and it being a non-generic dataset compared to Apple's stock price change dataset or Bitcoin dataset.

b. Trend date

Our dataset has a date range that spans over 100 years. The NASDAQ dataset ranges from 09-24, 1985 to 04-14, 2024. The S&P 500 dataset ranges from 12-29, 1927 to 04-14, 2024.

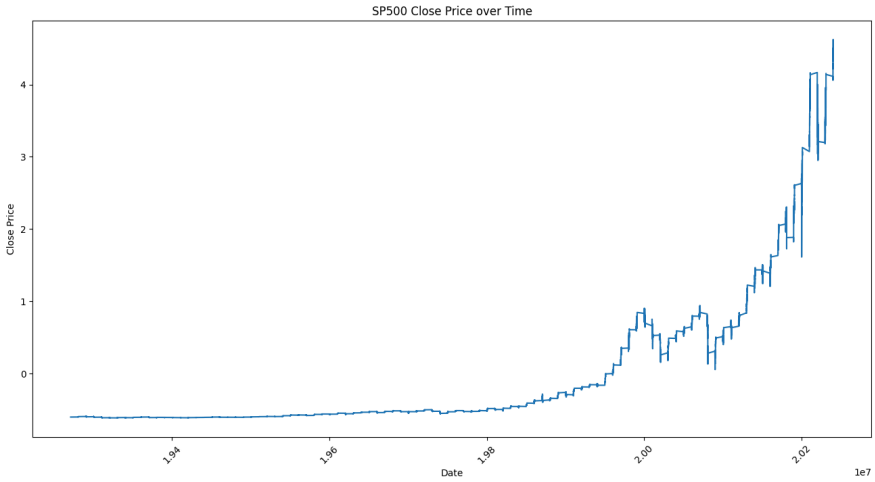

Fig. 2: Normalized Close Price for S&P 500.

c. Significance of data

The dataset's summary suggests that it provides a "granular look at the daily Open, High, Low, Close, and Volume (OHLCV) data for some of the U.S.'s most tracked financial indicators: S&P 500 and NASDAQ 100 indexes" and is "updated every trading day [to] serve as a resource for researchers, traders, and finance enthusiasts".

From the description above, we know that this data has detailed, up-to-date data points. And as it aims to provide help for researchers, it was neatly organized into CSV files with relatively clean and acceptable values.

d. Data Inspection

There are 7 features and a column for the date of trading day. Open indicates the opening price for the specified stock on the given trading day. High indicates the highest recorded price during that trading day. Low indicates the lowest recorded price during that trading day. Close indicates the closing price of the specified stock. Volume indicates the volume traded for the specified stock. Change percent indicates the day-to-day percentage change in the closing price. 20 day average volume is used as a metric to measure the trading interest for the past 20 trading days. Adjusted close is in some of the csv files, but not all, which reflects the close value after dividend distribution and applicable splits. Since our data does not include adjusted close, it will not be used in consideration.

e. Data cleaning

The dataset used required some cleaning, as there were columns that will not be used in our final model as well as some null values. To do so, data.describe() is first used to determine the total number of values within our dataset, which is 24187. Then, to find the total amount of null values for each column, data.isnull().sum() is used. The resulting output displayed a total of 1 null value in the change_percent column and 19 null values in the avg_vol_20d column.

Conveniently, the columns that will not be used in the final model are the volume and avg_vol_20d columns, therefore the 19 null values will not matter. The 1 null value in the change_percent column can be replaced with 0.57 to maintain consistency with the rest of the dataset.

After cleaning the data, there will be some reformatting required, more specifically for the date. In our data, the date is formatted using YYYY-MM-DD, which is easy to read, but will be difficult to work with for our model. Therefore, reformatting the data to YYYYMMDD is optimal for usage. To do so, data["date"]=pd.to_datetime(data["date"]).dt.strftime("%Y%m%d") is used.

2. Data Visualization (graph)

a. Closing prices over time (normalized)

Refer to figure 2. Normalized means scaled and fixed to the state where the data distribution follows normal distribution, with a mean of zero and variance of 1 (N~(1,0))

b. trends/patterns

Stock prices have been consistently rising and are showing an exponential rise.

Characteristic crashes include the invasion of Kuwait in 1990, the bursting of the dot-com bubble in the early 2000s, the financial crisis in 2008, and the sharp decline in 2020 due to Covid-19.

3. Summary of process

With all the data cleaned and visualized, we decided to plot it in a candlestick chart to see the OHLC (Open-High-Low-Close) of our dataset all at once, just like what the 'usual' stock graphs would look like.

IV. Time Series Forecasting and LSTM Model Implementation

1. Strengths, Limitations, and Parameters

LSTM is used in various fields that have time series concepts. For instance, Natural Language Processing(NLP) and sound processing, time series prediction for financial markets and weather forecasting

all use some form of LSTM models [7]. LSTM is quite effective at preserving long-term dependencies since the vanishing gradient problem that existed in RNNs is overcome in LSTM. Additionally, it is quite effective at handling sequential data as well, which reflects the data that our project uses [8].

However, as LSTM performs calculations using multiple gates in case cells, it has many parameters, resulting in high memory usage and high computational cost. Calculating recursively means calculating sequentially, and it is difficult to speed up calculations because it cannot be parallelized and the characteristics of GPUs etc. cannot be utilized [8]. This was reflected within our run time, where most of the time it ranged between 40-70 seconds per epoch, meaning that since we had 10 epochs it took more or less 6-11 minutes to run. It is also important to note that LSTM tends to be prone to overfitting.

The parameters used in LSTM are number of hidden Units, learning rate, batch size, dropout rate, sequence length, input size and bias for each gate.

2. Methodology

Our machine learning model was built using Python. After cleaning and processing the data, we began the model by selecting important columns that were going to be used for training, and settled on the columns “open”, “high”, “low”, and “close”. These values within the column were then converted into float values to ensure precision in the predictions.

Then, empty lists for X_train and y_train are initialized so that we can store training values for features X and target y. Since the values of our data are quite large, there will need to be some normalization. Therefore, by using StandardScaler() which removes mean and scales the data to unit variance, it leaves us with rescaled values ranging from 0 to 1.

After processing, cleaning, and normalizing the data, we can begin to implement time series forecasting. The purpose of time series forecasting is to analyze historical data patterns and create a prediction based on those patterns [5]. To begin time series forecasting, we initialize two integer variables; futuredays = 1 and pastdays = 24. These variables symbolize the amount of days into the future we want to predict and the amount of days in the past we want to use for pattern analysis. The integer for past days was determined as the most optimal after doing multiple trial and errors. The reason why the integer for future days is set to 1 is due to this section being still considered the “training” phase of our times series forecasting. By setting future days to 1, we train the model to learn the relationship between pastdays and futuredays as: *given 24 days of historical data, predict the data on day 25.*

To prepare and fit the training data for LSTM, there will need to be a loop created so that the training data for features and targets can take into account the past and future day variables we set. In this specific loop, the range is set to begin from pastdays (24) to the length of the dataset so that the loop will not go out of bounds during the creation of sequences. The lists are then converted to arrays so that the values are compatible with Keras.

Next, to apply LSTM into our scenario, we imported Keras, an API with a library of high-level neural networks, from TensorFlow [6]. For our model specifically, we used two layers of LSTM to achieve hierarchical learning; the first layer uses 64 neurons, processing the input sequences and retaining information and the second layer uses 32 neurons, taking the processed sequences from the first layer and extracts temporal features that are higher level [7]. To prevent overfitting we used dropout, a regulation technique that randomly drops neurons in each epoch during the training process. We tested several regularization techniques, including ridge (L1) and lasso (L2) regularization, and batch normalization, however dropout was the technique that yielded the best results.

Additionally, we deemed it to be the most convenient as well since the dropout rates could be modified if needed. We set it to around 20% initially, but after repeating results and getting various answers, we set it to 5%. In order to shape the LSTM output to our desired shape, we added a dense layer, which performs a linear regression onto the inputs. After adding all the required layers onto our model, we then compile it using the adaptive moment estimation (adam) optimizer and mean squared error loss function. To train our model, we simply use *model.fit(X_train, y_train, epochs = 10, batch_size = 16 , validation_split = 0.25, verbose = 1)* where we split the dataset into 25% validation, and set our verbose argument to progress bar mode. The model will reiterate 10 epochs with 16 training examples in each epoch.

Next, using the trained model the forecast can be made. Since we are implementing our model onto the website, futuredays is set as int(input()), which allows for users to input the days they want to predict as an integer. Using this information, a forecast_period variable is defined to create a date range for the forecast

period, then using the trained model the future values are predicted using model.predict. Since the results that we want to display will need to be the original format (for convenience), we had to inverse transform the normalized values back into the original scale. In this process, we also had to convert forecast dates to a list so that it can be used in the plot. Lastly, a data frame for the forecasted data is created using pandas, which is used to plot our prediction model output.

Our website was built using Python, JavaScript, HTML, and CSS. The server was hosted through Flask, and the backend was done through Python and Javascript; to make the website capable of saving the uploaded csv file and processing it through the model, and to compile the output plots to a base64-encoded image.

V. Results & Evaluation

1. Parameter Optimization

In our first run, the prediction forecast started well below the latest price at between 1700 and 1800 dollars.

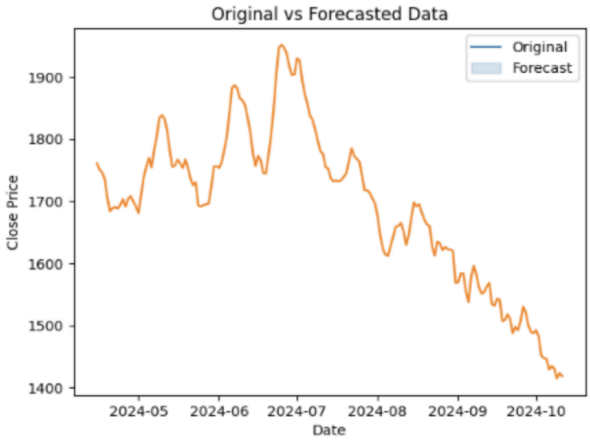

Fig. 3. Initial forecast run

This is where we figured out that our model configuration was off and had to look into optimizing our parameters. At first we tried various different techniques such as adding more LSTM layers, switching activation functions from RELU (Rectified Linear Unit) to ELU (Exponential Linear Unit), and increasing the number of epochs. None of these techniques worked, and that is when we tried out using regularization techniques. We began by trying L1/L2 regularization, but it gave an output that was far out of range, with a close price range being 0 to 400. Batch normalization yielded similar results, with the price range being around the low 1000s. We then terminated this technique and decided to try dropout with a rate of 0.2. This brought the close price range up to 1000 to 3000. We saw some improvements, and experimented to see if adding one more dropout layer would improve these results, but that ended up giving us a similar price range. We decided to keep just one layer and tried many different dropout rate values, including 0.5, 0.2, 0.1, 0.05, 0.01. The most optimal dropout value was 0.05, which was the one we also used in the final model.

2. Model Loss Optimization

We visualized how the loss function decreased as each epoch ran for training and validation data. Generally speaking, the loss value was constantly low for the training data while the validation data had more significant changes. Initially, our model saw constant fluctuations in validation loss, indicating that there were potential issues of overfitting and how validation data was being split. This can be seen in figure three below.

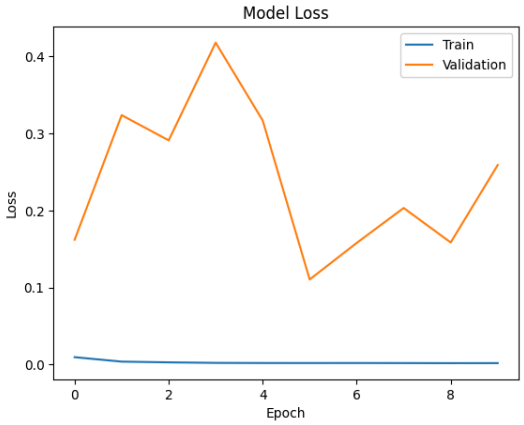

Fig. 4. Model Loss Plot Before Regularization

To reduce overfitting issues, we decided to implement the dropout regularization technique as described in section IV and yielded results that were more closely aligned to the graph we expected. The new model loss graph is displayed in figure 4 below. It can be seen that there is now less fluctuations in the validation loss trend, and shows a downward trend leading to a consistent loss value at around the same as the training loss. This means that the overfitting issue was fixed through regularization, and that the original

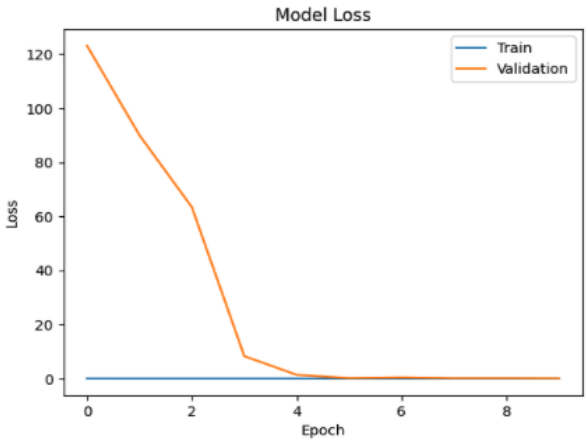

Fig. 5. Model Loss Plot After Regularization.

fluctuations were not due to the splitting of validation data.

3. Output Analysis & Performance

The output of our model is a line graph displaying the predictions of the S&P 500 closing values beginning from 2024-04-15 and ending in the 2024-04-15 + date input. Displayed in figure 5 is the output graph that ran with our most optimal model on the website. Our input for the amount of days into the future we want the model to predict was 180 days, which is approximately 6 months.

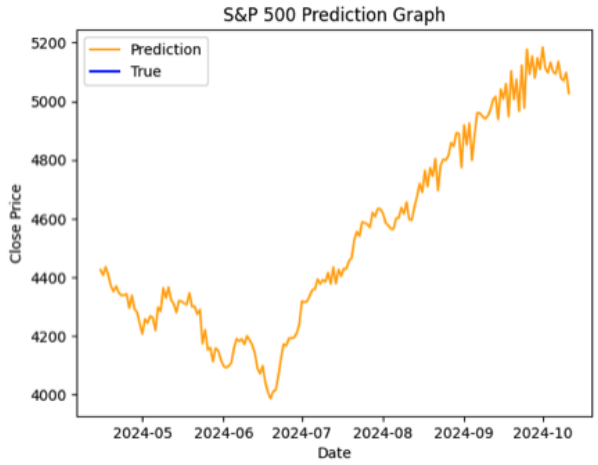

FIG. 6. S&P 500 Predicted Closing Prices

The accuracy of our forecast is difficult to specifically obtain, since it is trying to predict future values. The data that we used for our forecast was from the date range of 1927-12-30 to 2024-04-15, and the current data goes to 2024-06-13. This meansthat we can compare the minimum and maximum of the predicted and true data and that we have to define somewhat of how accurate our model is at a range of between 2024-04-15 and 2024-06-13.

Fig. 7. S&P 500 True Closing Prices (courtesy of Apple Stocks app)

Depicted in figure 6 is the true closing prices for the S&P 500 from the dates 2024-04-15 to 2024-06-13. The minimum closing price from figure 6 on 2024-04-19, with a closing price of 4967 dollars. The maximum closing price was on 2024-06-13 at 5434 dollars. Compared to our figure, at a similar range, we can see that the minimum closing price was between 4000 and 4200 at around 2024-06 and the maximum closing price was between 4400 and 4500 at around 2024-04. To calculate the accuracy we simply found the percent difference between the maximum and minimum values, which, for the minimum was between 18.26 and 24.16% and for the maximum between 20.76 and 23.5%. Since the values for the percent difference was greater than 5%, we can conclude that there is a significant difference between the values and more tuning of the hyperparameters can be utilized for improvement.

4. Evaluation

As it is, the model is in a state of slight overfitting, which is reflected from the validation data. The accuracy of the trend of our prediction model is acceptable, yet some improvements can still be made. To improve the model, we think that tuning the hyperparameters using methods such as grid search may be appropriate to find a more accurate and realistic model. However, the output prediction we have gotten as a result after finishing up adding regularization layers shows an acceptable trend and prediction.

While building our website using HTML, CSS, JavaScript, and Python, we also encountered some difficulty in implementing the machine learning model. However, we overcame this issue by utilizing print statements at every processing step for debugging, and analyzing the error statements flask was displaying on the web.

As we move forward, we plan to research more machine learning models and their parameters, to find the best fitting model for the prediction. Additionally, we hope to separate the data in chunks that align with the historical events and let the model compare those to the current trends for better accuracy. We also hope to find a better dataset or to make our own dataset with more details and features.

VI. CONCLUSION & DISCUSSION

Our thesis was that if we used at least two regularization techniques and took advantage of the large dataset we have prepared, the model will predict future stock prices fairly close to the actual values. While our output had predicted values and the model was successfully implemented, there were quite a bit of differences as we discovered in our results. The differences between our true and predicted models ranged from 18 to 24% in terms of the minimums and maximums. Despite this, there were valuable insights gained from this project, improving our knowledge on machine learning model implementations and learning effective parameter optimizations within a model. Additionally, we were able to fully understand the usage and limitations of LSTM within time series forecasting.

More specifically, during the debugging of our model, we optimized a variety of parameters through trial and error. Along the way, we learned how different dropout rates affect the generalization of LSTM models, reducing overfitting and therefore improving the model performance. We also learned the effects that different activation functions as well as different amounts of layers within a neural network had on the trend and range in our model output. Overall, the

project went quite smoothly, with our team being able to create a fully-working model and implement it onto an interactive website.

VII. GITHUB, ROADMAP

1. Github link: https://github.com/minjiyun02/stock-prediction
2. Demo link: https://youtu.be/l3Hm7fiV--E
3. Task Distribution/Roadmap

Shared tasks for everyone in the team were sharing personal schedules, cooperating in scheduling meetings and showing up to meetings, working on individually assigned tasks, writing the portions of the report that correspond with the individual tasks, and providing feedback and help for other team members. We were also all responsible for deciding on what machine learning model we were going to use, and to do research on it. Here are the individual's names and the tasks they completed:

a. Sean Chen (Team Leader)

Sean wrote the first and second progress report papers along with team members Kazuya and Minji. He also built the working LSTM model with the help of sources in *references* and worked on debugging and fine tuning the model's parameters along with Kazuya and Minji. Additionally, he wrote part of the exploratory data analysis, and evaluations with the help of Minji, and fully wrote the methodology, results, and conclusions portion of the final report. As a team leader, he coordinated meetings, regularly reminded everyone of the tasks and deadlines, encouraged equal participation, and managed work distribution.

b. Kazuya Okamoto

Kazuya completed the mathematical explanation of LSTM, and gave a full overview of how the model works. He assisted Sean and Minji with fine tuning the model, and provided some insights for preventing overfitting and underfitting. He also wrote parts of the introduction of the paper as well as the first and second progress report papers.

c. Minji Yun

Minji built the website to deploy and test the finished model with, assisting Sean and Kazuya with debugging and fine tuning along the way. She also built the server connection and data management in python for flask a flask server and recorded the demo of the website. Additionally, she wrote the first and second progress report papers and wrote half of the introduction and data analysis and collection part in the final report.

d. Om Chaudhary

Om wrote part of the introduction with the help of Kazuya and Minji and the literature review.

References

[1] S. Liu, G. Liao and Y. Ding, "Stock transaction prediction modeling and analysis based on LSTM," 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), Wuhan, China, 2018, pp. 2787-2790, doi: 10.1109/ICIEA.2018.8398183.

[2] "How time stepping works—ArcGIS Pro | Documentation," *pro.arcgis.com*. https://pro.arcgis.com/en/pro-app/latest/tool-reference/big-data-analytics/how-time-slicing-works-with-big-data.htm#:~:text=Time%20steps%20are%20a%20way

[3] "LSTM: Understanding the Number of Parameters," *kaggle.com*. https://www.kaggle.com/code/kmkarakaya/lstm-understanding-the-number-of-parameters

[4] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: https://doi.org/10.1162/neco.1997.9.8.1735.

[5] "A Guide to Time Series Forecasting in Python | Built In," *builtin.com*. https://builtin.com/data-science/time-series-forecasting-python

[6] J. Terra, "Keras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework," *Simplilearn.com*, Jul. 26, 2020. https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#:~:text=TensorFlow%20is%20an%20open%2Dsourced

[7] DigitalSreeni, "181 - Multivariate time series forecasting using LSTM," *YouTube*. Dec. 08, 2020. Available: https://www.youtube.com/watch?v=tepxdcepTbY

[8] Nelson, David & Pereira, Adriano & de Oliveira, Renato. (2017). Stock market's price movement prediction with LSTM neural networks. 1419-1426. 10.1109/IJCNN.2017.7966019.