Implementing Long Short-Term Memory Recurrent Neural Networks for Stock Market Prediction

Ayman Haque[a]

[a]*Department of Computer Science, Missouri State University, 901 S National Ave, Springfield, 65897, Missouri*

Abstract

The Stock Market is a fickle and volatile field which makes it a challenging endeavor to predict its trend. The interaction of a gigantic number of individual agents makes it harder to derive a pattern or motivation behind the change in stock values. Environmental and Political factors as well as social sentiment significantly affects the stock market and, to what extent they may influence the stock market is not always quantifiable, or even conceivable at times. Stock returns forecasting is essential for business entities to build strategies and make informed business decisions. It also plays a major role in sustaining economic stability in the society. With the advent of several machine learning algorithms, it has been possible to tackle the abnormality and non-linearity of the character of the stock market in forecasting future trends. In this article we discuss how we have implemented stock prices prediction using Recurrent Neural Networks (RNN), specifically a brand of RNN called Long Short-Term Memory RNN to achieve satiable accuracy. In this article, we state our motivation behind using this model, discuss several aspects of the model and the results derived from this endeavor.

1. Introduction

Understanding the trends in financial markets has been a popular research area for many years. The financial market is quite dynamic and unpredictable, which makes forecasting the future values of financial assets quite a daunting and challenging task. The plethora of factors and volatility of the different aspects of the market render it quite impossible to be predicted with spot on accuracy by simple mathematical and statistical methods or linear computational algorithms. With the advent of sophisticated machine learning techniques, we can understand the complex patterns and multi-collinearity of the market [1]. In this paper we are using Recurrent neural network with long short-term memory (LSTM) to implement such a predictor. We implement this by using historical data (Time-Series data) of a particular company (TESLA). The prime reason for using this model is because the LSTM cells can retain information from the time series for prolonged periods.

Email address: aymanhaque001@gmail.com (Ayman Haque)

2. Vanilla Recurrent Network and Vanishing Gradient Problem

In a simple neural network, the inputs are independent at each state and the prediction is determined by the corresponding input. In a vanilla recurrent neural network, at each state, the prediction is determined by the features at previous states alongside the input at that corresponding state. In other words, the information is persistent throughout the network and the cells make informed decisions while generating the output and passing it along to a successor. This also indirectly implies that the model can take a vector of multiple inputs and provide with a vector of multiple outputs. This makes RNN incredibly useful in time series analysis, speech recognition, image captioning, machine translation etc. The vanilla RNN has the following structure:

$h_t = f_w\ (h_{t-1}, x_t)$,

where h_t denotes the current state, h_{t-1} denotes the previous state and x_t is the current input. An issue with a vanilla RNN is that in practice, it does not simply handle long term dependencies well, although in theory it should. The prime reason for this is the Vanishing Gradient problem. In a RNN, the error is calculated from the output vector and the gradient is calculated by backpropagating through time. The gradient is used to update the weights. When the gradient becomes smaller and smaller as it backpropagates it becomes minute and the weights are not significantly updated; hence the system ends up learning nothing. This issue is solved by using an LSTM- RNN model instead.

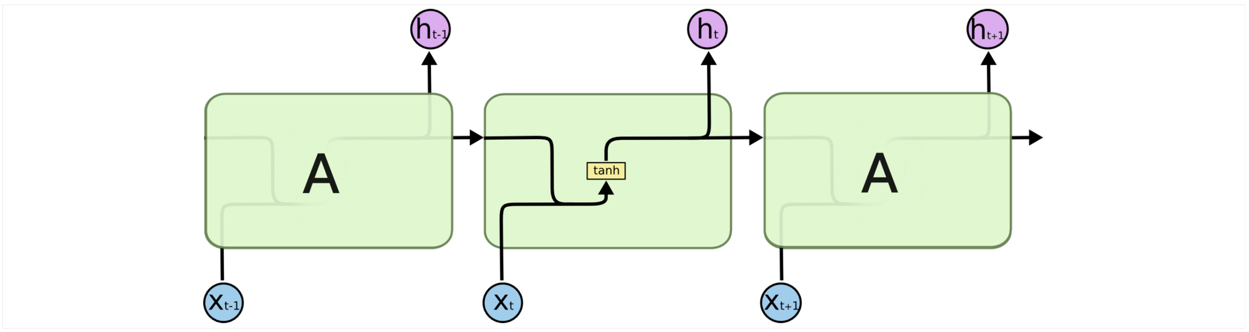


Figure 1: Figure 1: Vanilla Neural Network

3. Methodology

An LSTM cell is very similar to a vanilla RNN cell except it has a cell state and a few gates that regulate the flow of information through the cells over time. The gates are composed of a layer of neural net with a sigmoid activation. There are four such gates. The outputs from the gates are as follows:

$f_t = \text{sigmoid}(\ W_f\ [h_{t-1}\ , x_t] + b_f$ [f1]

$i_t = \text{sigmoid}(\ W_t\ [h_{t-1}\ , x_t] + b_i$ [f2]

$\Delta C_t = \tanh(\ W_c\ [h_{t-1}\ , x_t] + b_c$ [f3]

$o_t = \text{sigmoid}(W_o\ [h_{t-1}\ , x_t] + b_0$ [f4]

May 14, 2020

$C_t = f_t * C_{t-1} + i_t * \Delta C_t$ [f5]
$H_t = o_t * \tanh(C_t)$[f6]

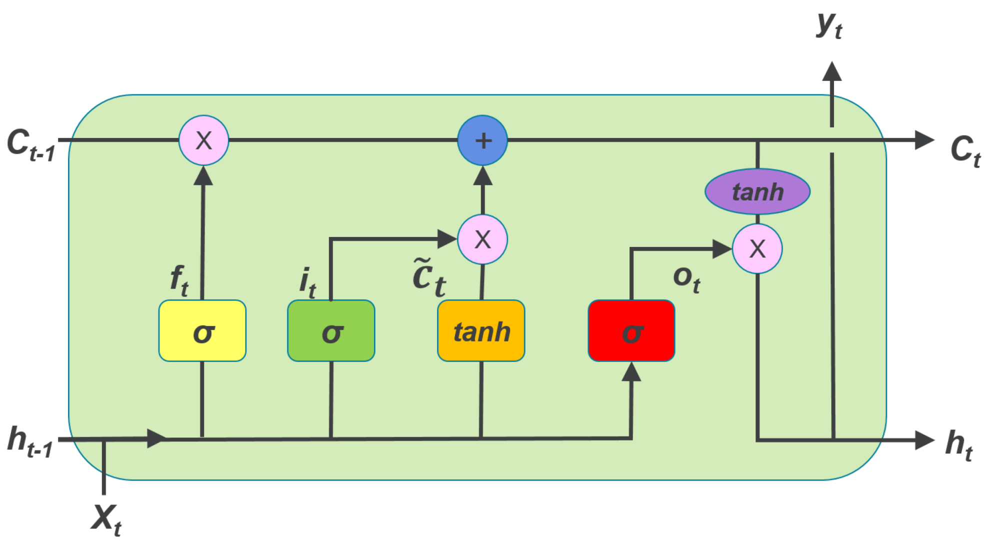


Figure 2: An LSTM cell demonstrated with the gates

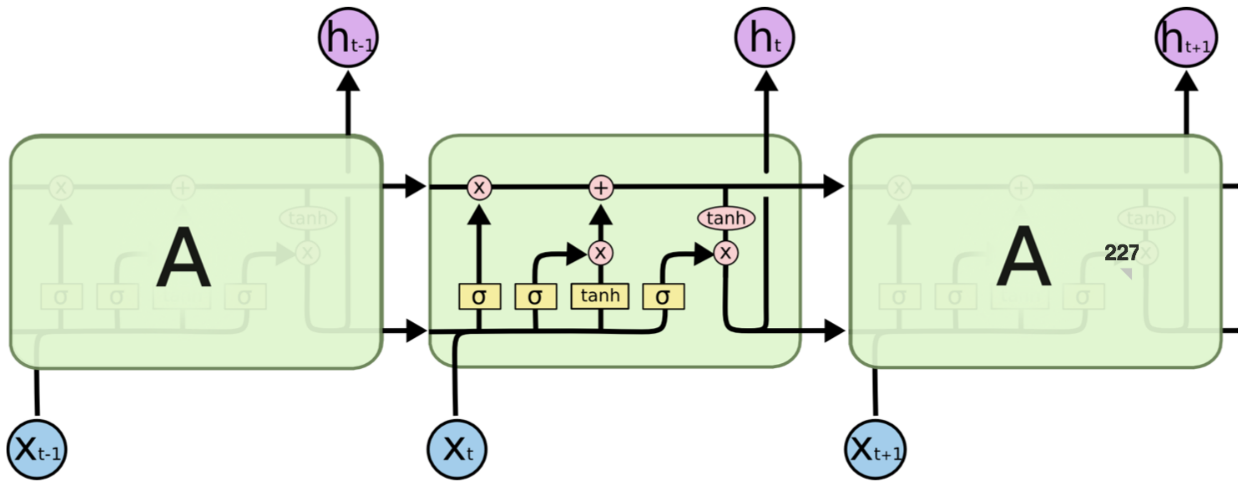


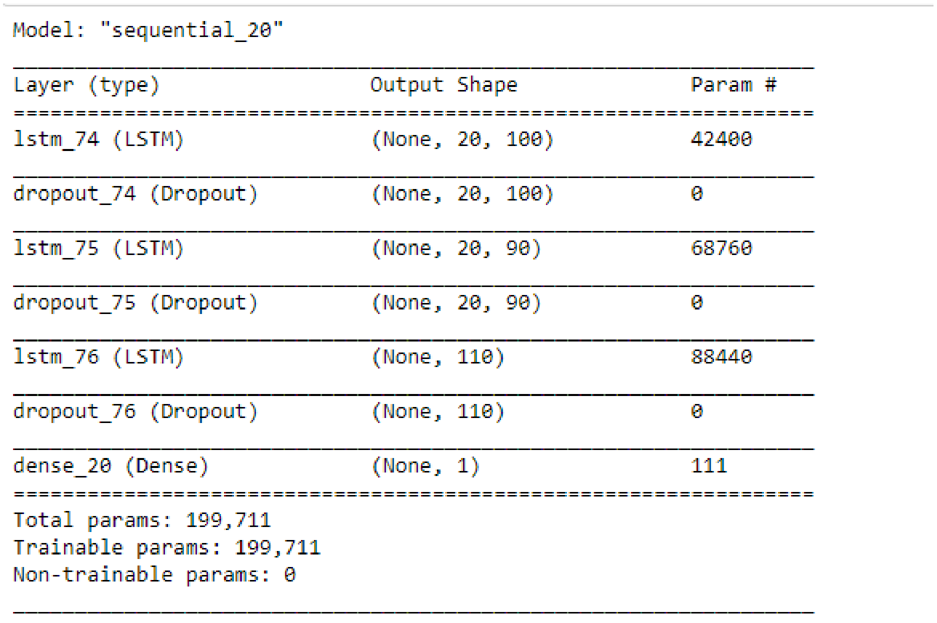
Figure 3: LSTM network

The forget gate [f1] determines whether the neural network should keep the information from previous state or drop it. [f2] and [f3] determines the sections of data in need of update. In [f5] the cell state is being updated. [f4] is the output gate that decides the value of Ht and though a tanh function, spews a value between 0 and 1. This is the method followed in the implementation of my system. The implementation of this model in this project is discussed in the next few sections.

4. Data Collection and Pre-processing

We used the Historical Time Series data of TESLA stock prices from the 6/29/2010 to 5/4/2020. We obtained this data set from the Yahoo finance website. There are seven columns on the data set namely Date, Open, High, Low, Closed, adj close and volume. Open indicates opening prices for a specific day, closed price indicates the last price for the day and so on. We drop the date and adj closed column from the data set. The remaining are the features that we shall use to train our model. We split the model into training and test set. For the training set we use all the prices before the year of 2020. We are now going to rearrange the data set in to chunks, and there will be 20 days inside each chunk of data. The idea is that, for each chunk our Training input is going to be the first 19 days of the sub-data and the 20th day is Training Label. This means that the first 19 days are our input vector, and we will predict the stock price of the 20th day for that iteration. Similarly, the next chunk is going to start from the 2nd data point on the data set to the 20th, and we will be predicting for the 21st day. In other words, at each chunk of data, the starting data point will be the second data point of the previous chunk and so on. This will go on continually in this recursive manner until the end of the data set. The new data set that we acquire is a three-dimensional array. Similarly, we can arrange the test data set and in addition, include the last 19 data points of the training data as we want to predict the 1st 20 data points in the test set. We normalize both the training and data set to be used with the model.

5. Construction of the LSTM model



```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_74 (LSTM)               (None, 20, 100)           42400
_________________________________________________________________
dropout_74 (Dropout)         (None, 20, 100)           0
_________________________________________________________________
lstm_75 (LSTM)               (None, 20, 90)            68760
_________________________________________________________________
dropout_75 (Dropout)         (None, 20, 90)            0
_________________________________________________________________
lstm_76 (LSTM)               (None, 110)               88440
_________________________________________________________________
dropout_76 (Dropout)         (None, 110)               0
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 111
=================================================================
Total params: 199,711
Trainable params: 199,711
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Summary for the Initial LSTM model of this project

6. Results

In the initial model (Figure 4) as described in the previous section, we first run it for 10 epochs, 20 and 40 respectively to see the changes in loss and accuracy. It can be seen that Mean absolute error for both training and test set is increasing. Train loss is decreasing with increase in epoch which should indicate the model is working better with increase in epoch(refer to figure 5, 6 and 7), but increase in the test loss indicates that the model might be over-fitting in the training phase. Please refer to the table 1 below:

Table 1: Changes in loss and frequency for increasing number of Epochs

Epochs	Loss at Training	Final Mean Absolute Error at training	Loss at Test Phase	Mean Absolute Error-Test Phase
10	0.004	15.9733	2.79099e-05	0.16
20	0.0013	20.09	4.14118e-05	0.17
40	8.3535e-04	26.997	0.001	0.18

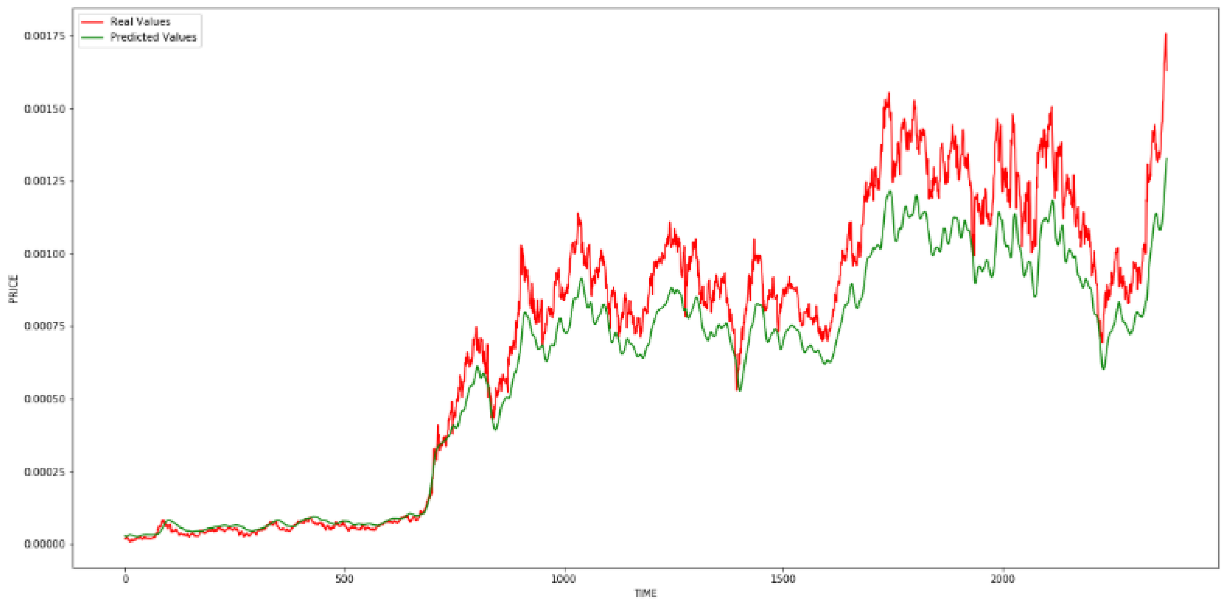


Figure 5: Price vs Time Curve when Epoch = 10

When we keep the epoch constant at 10 and increase two more layers (figure 8), the projections are much more accurate. We also observe that loss in training set significantly decreases along with the loss in the test phase, both in reference to the initial model(Please refer to figure 10 and Table 1). In figure 9, we draw a comparison between the real values, the prediction curve of the initial model (in green) and the prediction curve with the layers increased (blue). It is evident that increasing LSTM layers increase the precision of the model

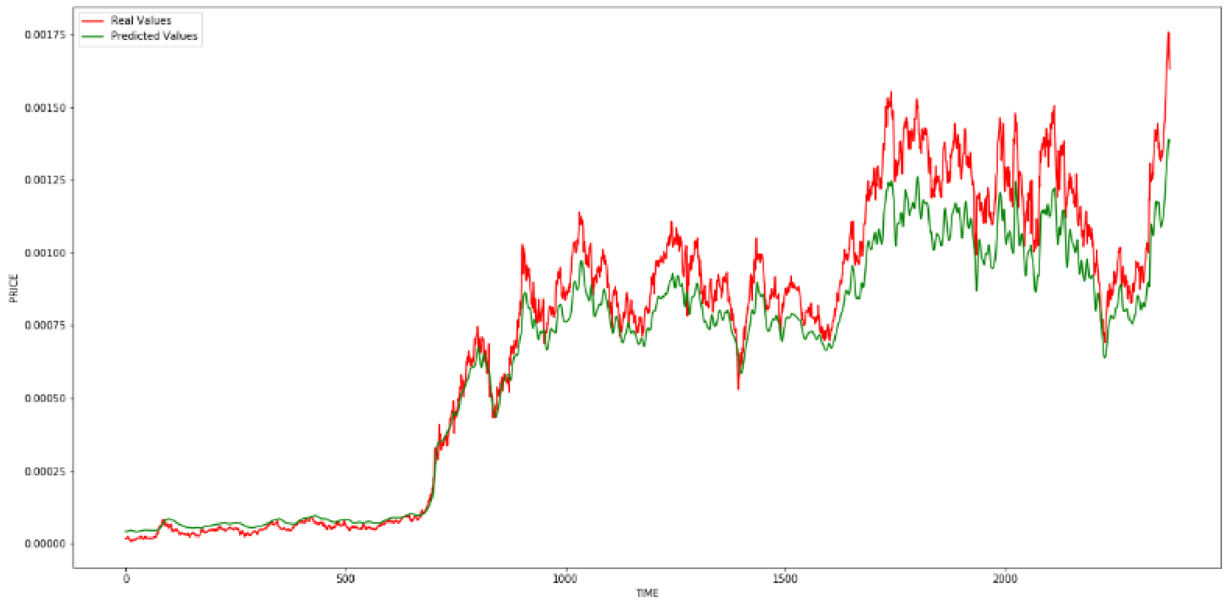


Figure 6: Price vs Time Curve when Epoch = 20

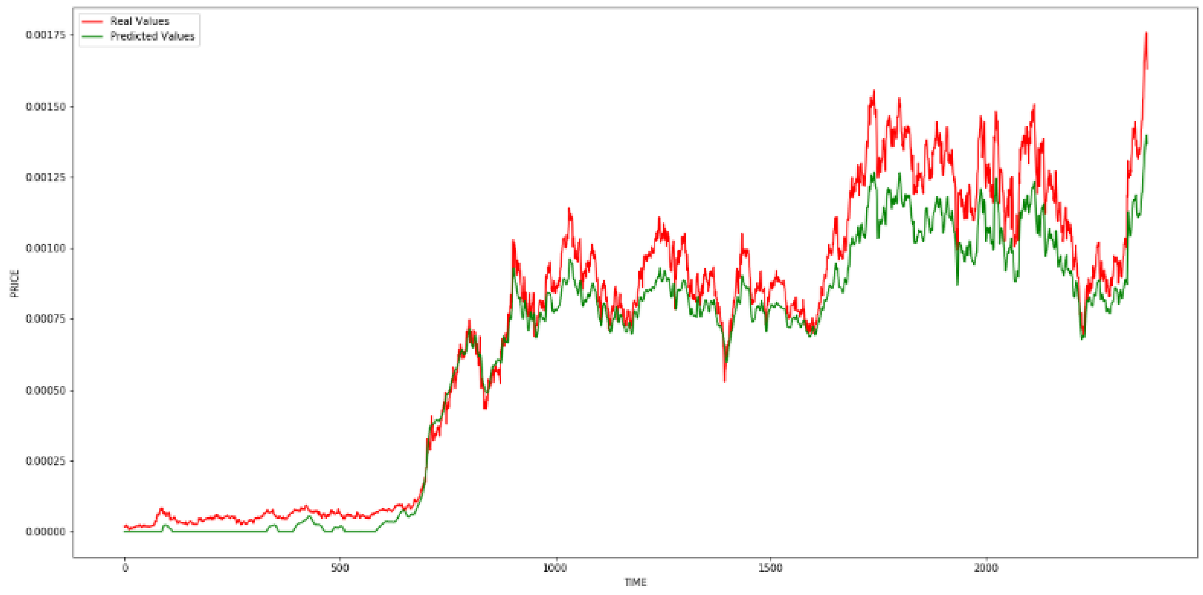
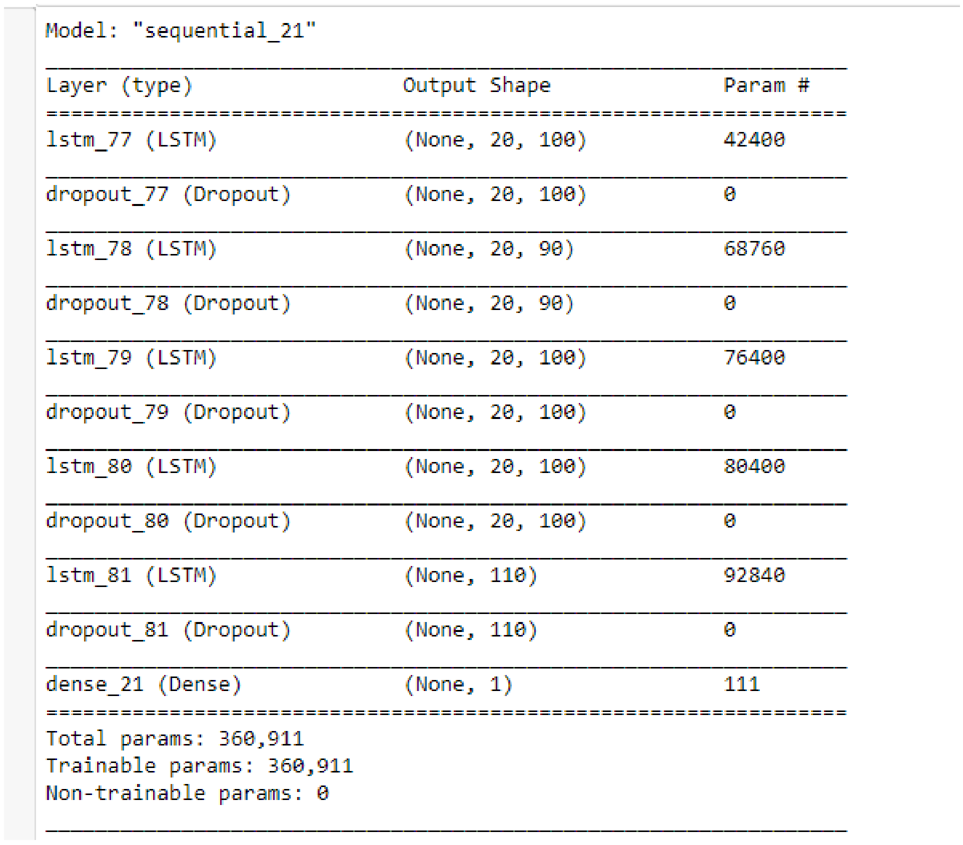


Figure 7: Price vs Time Curve when Epoch = 40



```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_77 (LSTM)               (None, 20, 100)           42400     
_________________________________________________________________
dropout_77 (Dropout)         (None, 20, 100)           0         
_________________________________________________________________
lstm_78 (LSTM)               (None, 20, 90)            68760     
_________________________________________________________________
dropout_78 (Dropout)         (None, 20, 90)            0         
_________________________________________________________________
lstm_79 (LSTM)               (None, 20, 100)           76400     
_________________________________________________________________
dropout_79 (Dropout)         (None, 20, 100)           0         
_________________________________________________________________
lstm_80 (LSTM)               (None, 20, 100)           80400     
_________________________________________________________________
dropout_80 (Dropout)         (None, 20, 100)           0         
_________________________________________________________________
lstm_81 (LSTM)               (None, 110)               92840     
_________________________________________________________________
dropout_81 (Dropout)         (None, 110)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 111       
=================================================================
Total params: 360,911
Trainable params: 360,911
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Summary of the LSTM model with increased layers

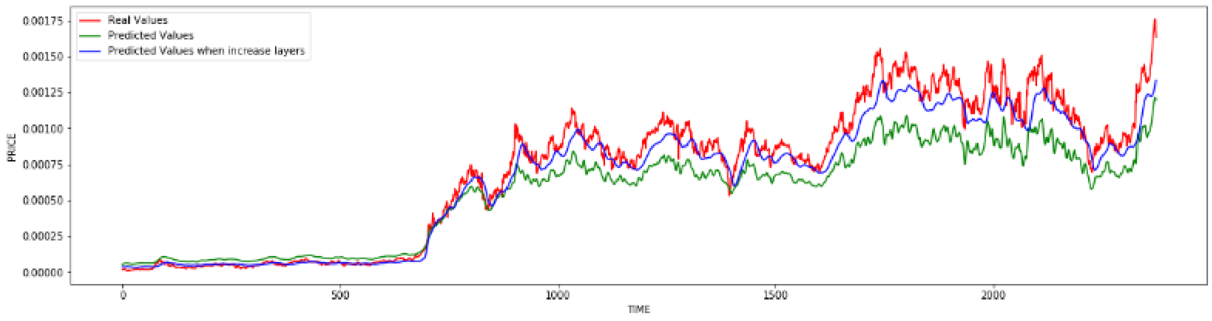
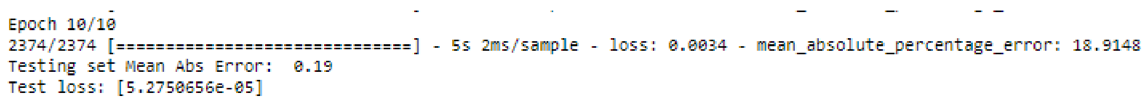


Figure 9: Price vs Time Curve for real, predicted and predicted when increased LSTM layers



```
Epoch 10/10
2374/2374 [==============================] - 5s 2ms/sample - loss: 0.0034 - mean_absolute_percentage_error: 18.9148
Testing set Mean Abs Error:  0.19
Test loss: [5.2750656e-05]
```

Figure 10: The MSE and MAPE value of the training and test set when LSTM layers increased

7. Conclusion

The model does a satisfactory job at stock market trend forecast. I believe that accuracy of the model can be improved if the dropout percentage in the layers are increased to prevent overfitting. Another way to make the model more realistic would be to integrate sentiment analysis into the system. The social sentiment towards the stock market can be a reflection of political, environmental and social atmosphere affecting the market. A practical way would be to analyse the sentiment of news headlines around a specific Stock each day and generate a cumulative score, which could be an element in the unit vector of our system.

8. References

[1] A. Moghar, M. Hamiche, Stock Market Prediction Using LSTM Recurrent Neural Network, Procedia Comput. Sci. 170 (2020) 1168–1173.

[2] C. Peng, Z. Yin, X. Wei, A. Zhu, Stock Price Prediction based on Recurrent Neural Network with Long Short-Term Memory Units, 2019 Int. Conf. Eng. Sci. Ind. Appl. ICESI 2019. (2019) 1–5.

[3] S. Sable, A. Porwal, U. Singh, Stock price prediction using genetic algorithms and evolution strategies, Proc. Int. Conf. Electron. Commun. Aerosp. Technol. ICECA 2017. 2017-January (2017) 549–553.

[4] A. Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network, Phys. D Nonlinear Phenom. 404 (2020) 132306.