Machine Intelligence Term Project: Machine Learning in the stock market for stock price prediction

Lucas C. Magalhaes

***Abstract*— I was watching a stock trading conference video, the head of one of Canada's top trading banks was talking about High Frequency Trading. It surprised me when he said that almost 60% of all trades in the stock market are decided by computers. It showed me the disadvantage any individual trader has when competing with big banks and trading firms in the stock market. Although it seems hopeless to compete with the professionals, maybe a machine learning model could help close this gap between these giants and the individual traders. This paper investigates the feasibility of using Machine Learning models as a tool to predict stock prices for individual traders. As with most, if not all, machine learning models, it will not predict with perfect accuracy but if it is used in conjunction with other strategies and indicators, it could help these traders decide where to place their money to maximize their return on investments. After some research I decided to try to use a Long-Short Term Memory network, which is a type of Recurrent Neural Network. This network will predict the stock prices of Microsoft (MSFT) and Apple (AAPL). The model will be trained to work with large cap companies since long-term investors will be more interested in those companies rather than medium or small cap stocks. With stock price prediction and a well-rounded risk management method, I believe that machine learning will be more popular with individual traders and not only banks, hedge funds and other trading firms.**

I. Introduction

STOCK price prediction has been around since the stock market has been created, with the likes of Benjamin Graham writing books on how to predict the market to make a good return on investment. It is still to this day a very heated debate on how to get the best return and it is a multi-billion-dollar industry. But the stock market is very hard to predict since there are so many factors at play, like politics, policies, weather events, company news, company financials and many others, and therefore it is a very interesting problem to solve. Since it is such a complex problem, maybe a Deep Learning network might be able to detect patterns and find the best possible guess at how much a stock will be priced at.

The interest in Machine Learning to solve complex problem across all industries have been growing, and the financial sector is one of these industries that has investing heavily in this area of research. Therefore, many have been trying to create networks to predict stock prices, funds like Renaissance and Sigma are hedge funds that have been using machine learning and statistical analysis to have the best and most informed decision when picking stocks to place in their portfolio and it has been paying off.

Many researches have also been trying to solve this problem with Machine Learning, using different architectures like a Hybrid model [3] Recurrent Neural Networks [4] and different kinds of Artificial Neural Networks [6]. There are also different approaches to the problem, if you are a long-term investor you don't care about the price movement each day, so the data that will be used will be of low frequency sample, for example the closing price of the stock between each week, instead of each day. If you are a day trader, you will care what the price movement of the stock is at every moment of the day.

After looking at papers from different researchers and looking at different Machine Learning websites I decided that this paper will be focusing on using a Long Short Term Memory (LSTM) Network, which is a type of Recurrent Neural Networks (RNN), where it uses LSTM cells instead of regular RNN ones. And it will try to predict the closing price of stock at the end of each trading day. How these cells work will be explained at a later section.

The rest of this paper is organized as follows. After the introduction, Section 2 overviews RNNs, and its issues, LSTM cells and how they work. Section 3 will show how the network is built, and what are its inputs and the expected output. Section 4 will talk about the results obtained. Section 5 will have the concluding remarks.

II. Background

Machine learning algorithms, like Support Vector Machines, Naïve Bayes Classifier, Linear Regression and Feedforward Neural Networks work well with data where each sample is independent of each other, but when it comes to data that is sequential in nature, where each sample inherently dependent on the previous sample like words on a sentence, or in this case price movement of a stock, these algorithms become useless since they cannot take into account the past samples to make a

.

prediction of what should come next.

Sequential data modeling comes in different forms [1], one-to-many, many-to-one or many-to-many. Price stock prediction can be considered as a many-to-many type, since it will take a sequence of inputs (price, volume, averages, etc.) and return a sequence of prices that are one time-step ahead of the input, and therefore it is delayed.

Recurrent Neural Networks are a type of Deep Neural Network where the output of a hidden-layer node loops back into itself and it is treated as an input, with a weight associated with it.

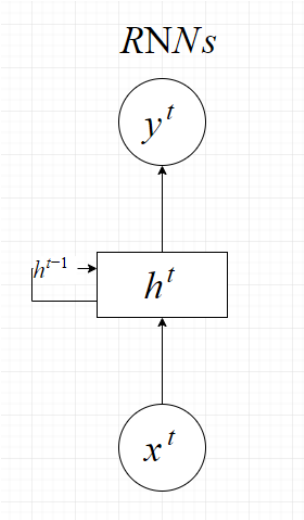

Fig. 1. Simple Recurrent Neural Network diagram

Fig. 1 shows how a one hidden layer neural network would look like. As it can be noticed, besides the *x* input to the hidden layer there is also an input from itself from the previous time-step. This recurrent edge, or loop allows the network to remember what the previous step was, giving the network a 'memory'.

$$a^{[h^t]} = \phi_h(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \quad (1)$$

The calculation of the activation function is very similar to feedforward networks as shown (1), but it now also has the linear combination of the output of the layer from the precious time step and its weights.

There is also the issue of how this network will learn, and this is where Backpropagation Through Time (BPTT) comes in, where the overall loss is the sum of all the losses from time 1 till current time T. Now the computation of the gradient will have a multiplicative factor which can cause the weights of a certain inputs in these hidden layers to vanish or it can cause the weights to have a huge value, causing problems when the training set becomes large enough [1]. Another problem that needs to be pointed out with RNNs is that long term dependencies will not work, if the price of a stock is dependent for some reason on the price of it 1 month ago, this neural network topology will not be able to account for that. This vanishing and exploding [1] gradient problem and the long-term dependency problem [9] are solved by using a LSTM cell, which replaces all the nodes in a hidden layer.

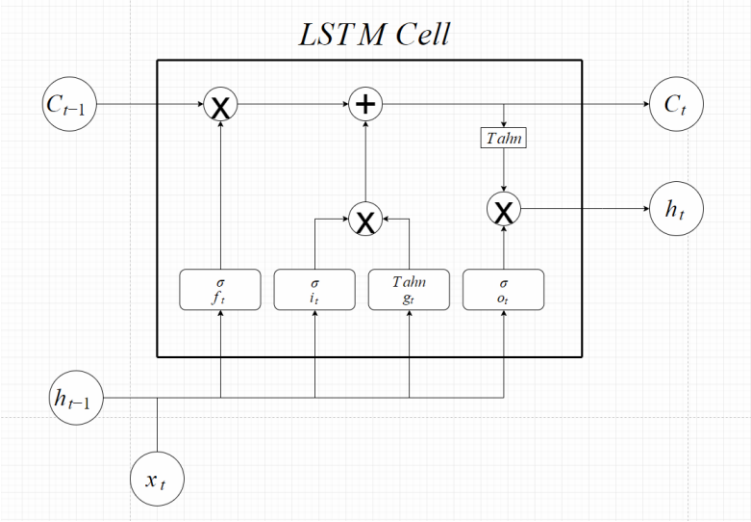

Fig. 2. LSTM cell diagram.

$$f_t = \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f) \quad (2)$$
$$i_t = \sigma(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i) \quad (3)$$
$$g_t = Tahn(W_{gx}x^{(t)} + W_{gh}h^{(t-1)} + b_g) \quad (4)$$
$$o_t = \sigma(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_t) \quad (5)$$
$$C^{(t)} = (C^{(t-1)} \odot f_t) \oplus (i_t \odot g_t) \quad (6)$$
$$h^{(t)} = o_t \odot Tahn(C^{(t)}) \quad (7)$$

Fig. 2 shows the diagram of a LSTM cell. The main feature of the LSTM cell is the cell state, which runs along the top of the cell (C_t, C_{t-1}). The cell state just takes the information from the previous state and brings to the current state, without changing too much between each iteration aside from some element wise operations shown by the circles with an X as multiplication and a + as summation.

The LSTM will change the cell state, but it is controlled by three "gates" which choose what information should be added or removed from it. When a new time step starts, the cell state will be element-wise multiplied by the output of the "forget gate" (f_t) which its output is calculated using (2), it uses the sigmoid activation function on the linear combination of the input *x* at time *t*, and its weights, plus the linear combination of the output of the cell from the previous time step h^{t-1} and its weights, plus the bias; the output will be a value between 0 or 1 for each attribute of the cell state, and that way choosing which information to keep and which one to get rid of.

The next step in creating the current cell state, is decided by "input gate" (i_t) and the "input node" (g_t). The input gate, shown in (3), decides which values need to be updated in the cell state, and it has the same activation function as the forget gate. The input node, generates a new cell state, using the same linear combinations described for the forget layer, but it uses a

hyperbolic tangent activation function instead, shown in (4). The input gate and the input node are then multiplied and added to the cell state, which then is considered the current cell state, shown in (6).

The output of the LSTM cell will be a variation of the cell state, since the current cell state will be pushed through a hyperbolic tangent activation function, which forces the values to be between -1 and 1. And the output of the activation function will be multiplied by the "output gate" (o_t) which decided which values of the modified cell state should be outputted.

III. Proposed Method

The model will be created using Python and the TensorFlow library. This library makes it easier for machine learning developers to create models by providing functions and classes that take care of creating common Machine Learning algorithms, including the LSTM cell. To use the library, first a TensorFlow graph is created, which sets how the flow of information is going to go through the network, the hyper parameters are created, and it also sets the functions that are going to be used to learn the weights of each of the cell gates and node, for each of the layers. After the graph is completed a TensorFlow session is then used to feed the training data to the model so the weights can be calculated, and it is also used to make the model predict the values of the testing data.

A few hyperparameters will be configured to try to get the best possible result, they are the following. The number of LSTM layers that are going to be stacked. The number of hidden units in each of the LSTM cells. The learning rate of the model, which will also decay as the model learns more and more. And lastly, the batch size, also called the sliding window [5], which is a grouping of a x number of days that predict the next x number of days and they do not overlap.

The input data of the model will be comprised of adjusted closing price of the stock, the adjusted volume of the stock, a 200-day simple moving average, a 50-day simple moving average, the Relative Strength Index (RSI) of the stock [7], the middle, upper and lower bands of the Bollinger Bands indicator [8] the difference between the upper and lower bands and lastly the Moving Average Convergence Divergence (MACD) indicator. The RSI, the Bollinger Bands and MACD are calculated from the initial data found in the Quandl database.

For this paper the model will be trained and tested with the stocks of Microsoft and Apple. The data was found in the Quandl database, and it already had the adjusted prices and volumes. The adjustment is made since throughout time stock prices change due to stock splits, dividends and distributions, which may cause the impression that what changed the stock price was caused by regular market movements when in fact it was artificially caused by those factors.

The RSI, Bollinger Bands and MACD will also use the adjusted stock prices and volumes. The RSI is an indicator for stocks that are being overbought or oversold [7]. The RSI is calculated in two steps, the first step can be considered an initialization which will then be used to get the actual RSI value.

$$RSI = 100 - \frac{100}{1+RS} \tag{8}$$

$$RS = \frac{Average\ Gain}{Average\ Loss} \tag{9}$$

$$Av.Gain = Sum\ of\ Gains\ in\ past\ 14\ days/14 \tag{10}$$

$$Av.Loss = Sum\ of\ Losses\ in\ past\ 14\ days/14 \tag{11}$$

$$Av.Gain = \frac{(previous\ Av.Gain)\times 13 + current\ Gain}{14} \tag{12}$$

$$Av.Loss = \frac{(previous\ Av.Loss)\times 13 + current\ Loss}{14} \tag{13}$$

To get the RSI value (8) will be used, and RS will be populated with (10) and (11), after the first 14-day period, the subsequent RSI values will be calculated using RS populated with (12) and (13).

The RSI values will be oscillating between 0 and 100, and it is considered that when the RSI reaches the value 70 and it overbought, and the price should start decreasing, and when the value is at 30, the stock is oversold, and the price should start increasing.

The Bollinger Bands is another indicator, but it comprises of three different values, shown in (14), (15) and (16), where SMA means a simple moving average and it is regularly set to 20 time-steps, and since the database provides daily numbers, it will be considered and 20-day moving average. The upper and lower bands are the SMA plus and minus the two times the 20-day price standard deviation.

$$Middle\ Band = SMA \tag{14}$$

$$Upper\ Band = SMA + (20day\ \sigma\ \times 2) \tag{15}$$

$$Lower\ Band = SMA - (20day\ \sigma\ \times 2) \tag{16}$$

This indicator is a volatility indicator as the bands narrows if the price of the stock is more stable or the bands widen as the price of the stock becomes more volatile. There are many uses for Bollinger Bands and you can find more in-depth explanation on reference [8]. This widening or narrowing is going to be captured by calculating the difference between the upper and lower bands.

The MACD indicator is like the RSI in terms that it indicates the momentum of the stock, but it differs since it takes the trend of the stock into account. To use this indicator the MACD line is created by (17) and then a signal line (18) is added to be used as a reference against the MACD line. Where EMA means an exponential moving average. By using the exponential moving average instead if a simple moving average, the indicator gives more importance to the prices that are more recent.

$$MACD = 12day\ EMA - 26day\ EMA \tag{17}$$

$$Signal = 9day\ EMA \tag{18}$$

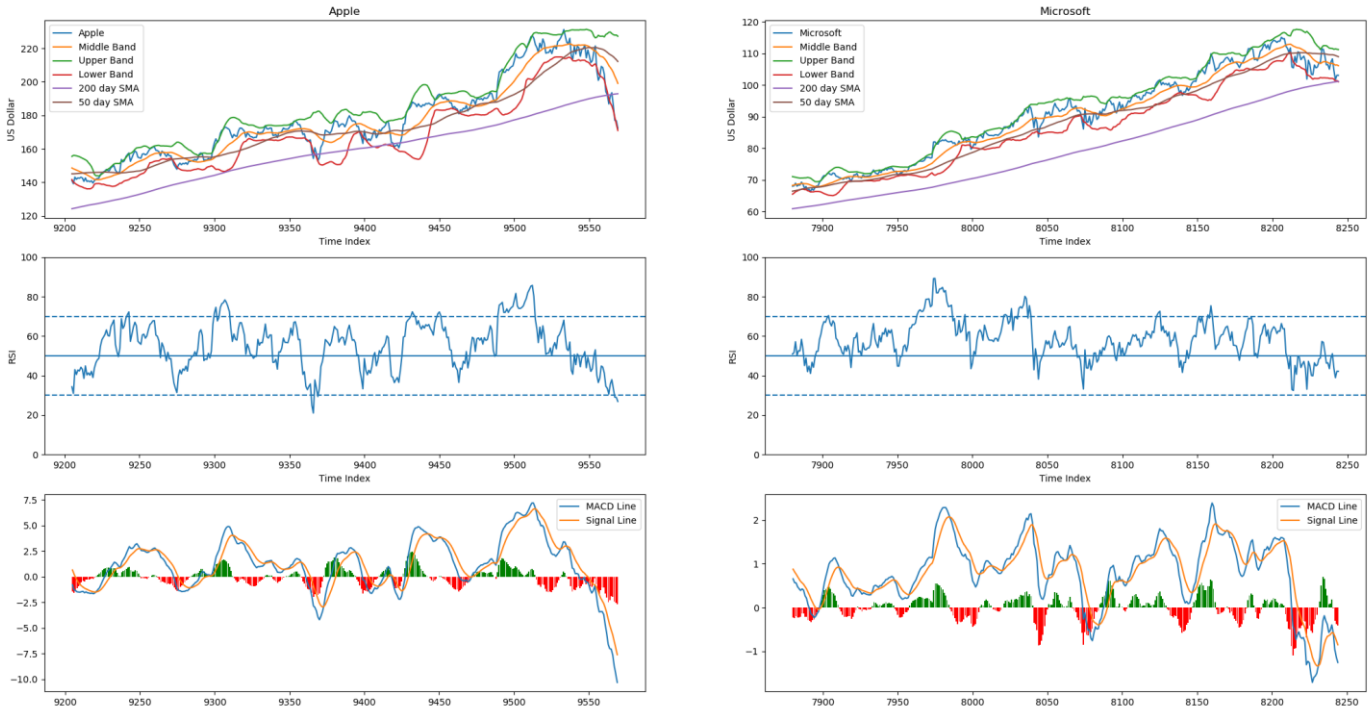

Fig. 3. Dataset visualization (Green and Red histogram in MACD graph is the difference between signal and MACD line, but not used as a feature).

After all indicators are calculated and placed in the dataset, it can be visualized. This visualization is shown in Fig. 3.

The values shown in Fig. 3 are scaled using a Min Max Scaler that transforms the data from high values to values between 0 and 1. The scaling adjusts the values based on a non-overlapping window of size 365, i.e. adjusting the prices of each year to be values ranging from 0 to 1. This adjustment is made because as prices increases through time, the neural network won't be trained in these new values and won't be able to predict them. Another reason is to make sure that early stock prices, which are very small, don't lose their impact.

IV. Results

After running a few different learning models with different parameters for both Microsoft and Apple stocks a few hyperparameters did not seem to help past a certain point and were fixed. These were: The number of epochs, 10 epochs were enough to get the model to fit to the training data and the MSE line was flattened; the learning rate, 0.0001 was found to be the best number; and finally, the learning rate decay, which was set to 0.75.

The last parameters remaining were the batch size, the unrolling size, the number of hidden layers and the size of those hidden layers. For the Apple database, the best combination seemed to have been a batch size of 5, an unrolling size of 5, three hidden layers, each layer to have the size of 128, 256, and 128 respectively from input to output. The mean squared error of the training of the model with the parameters shown above can be seen in Fig. 4.

There is a sharp down curve at the very beginning of training, from the 0 iteration to around 200, and after that the curve settles after 6000 iterations.

Fig. 4. Mean Square Error Graph of AAPL training

To make sure these hyperparameters were a good choice the average Mean Square Error obtained from the Test dataset was compared to other combinations of hyperparameters. The resulting prediction for the Apple stock is shown in Fig. 5 top figure, which also shows the actual predictions values, and the model seems to be predicting the trend very well and it leads the big falling prices that happen, which would be very useful for a trader to know. Fig. 5 bottom graph shows the Mean Squared Error of the predicted values, which they jump right as our model indicates that the price will go down steeply.

The same process was performed to find the best

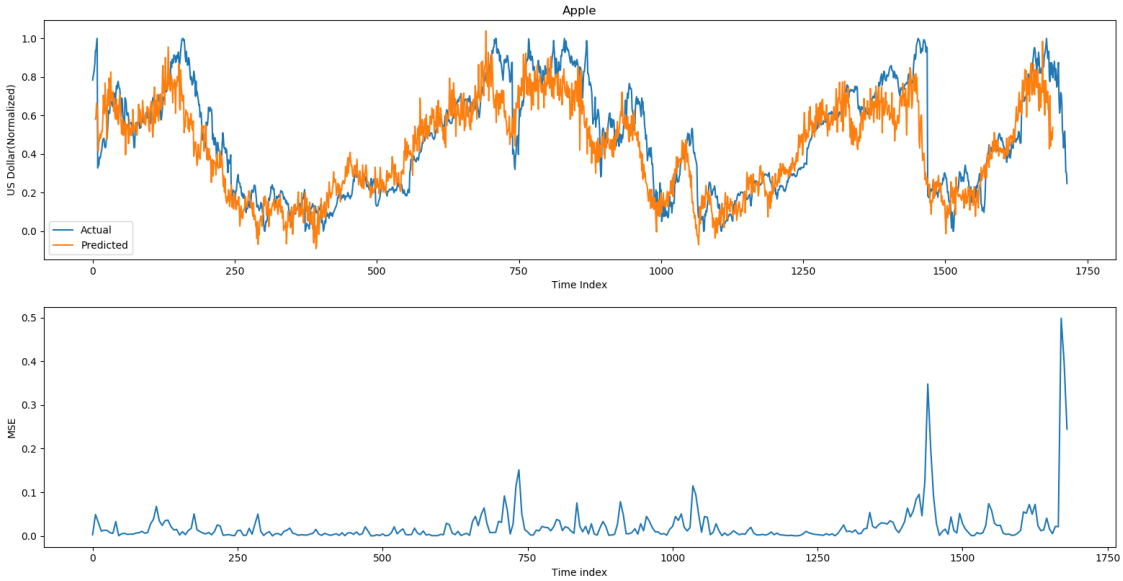

Fig. 5. Normalized stock price for actual and predicted values for Apple(Top) Mean Squared Error of predictions for Apple (Bottom)

hyperparameters for the Microsoft stock price dataset. After running different combinations of hyperparameters, the best combination was found to be: unrolling size of 10; batch size of 5; three hidden layers; and each layer to have the size of 128, 256, and 128 respectively from input to output. These hyperparameters are very similar to Apple's.

The same results that were extracted for both the training and test datasets from the apple dataset, were also extracted from the Microsoft dataset. The training Mean Squared Error can be seen in Fig. 6. Fig. 6 shows a similar pattern that we found with the Apple dataset, where the error quickly drops in the first couple of hundred iterations and then it stays very steady for the rest of the training periods.

Fig. 7 shows the predicted and actual prices of the test dataset, with its respective Mean Squared Error graph. Fig. 7 also shows similarities to the prediction made on the Apple dataset, the model seems to be anticipating the great downturns of the stock.

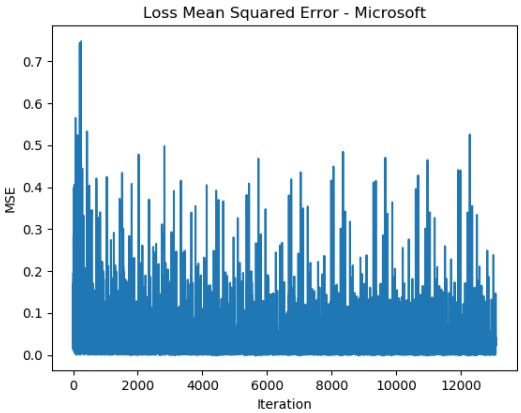

Fig. 6. Mean Squared Error Graph for Microsoft Dataset

These were the results found with this model. To truly find out if the model works to help traders, it would have to undergo a backtesting where the model will make decisions of when to buy or sell the stock and see what the returns would be.

V. Conclusion

This project was a great learning experience in how to think of a problem to solve, how to leverage machine learning to solve the problem, how to build a machine learning solution, how to test the model created and how to visualize and show your ideas and results to other people.

Although the model seems to be working well, there are a lot of things that could be added to improve it and make it more robust. Sentiment analysis could be a great addition since, what really moves the prices of stocks are not efficiencies or deficiencies in the market but rather people making decisions and therefore there must be a level of emotion to it that could be tapped by sentiment analysis on tweets or news articles. Or adding more indicators might better line up the predictions with the actual price instead of being ahead of it.

This model could be used by individual traders to help make decisions or even let the model make the decisions for them, a lot of Wall Street trading firms are moving to Machine Learning and algorithmic trading to better returns to their clients and simple models like mine can help put the same power that Wall Street firms have in the hands of individual traders.

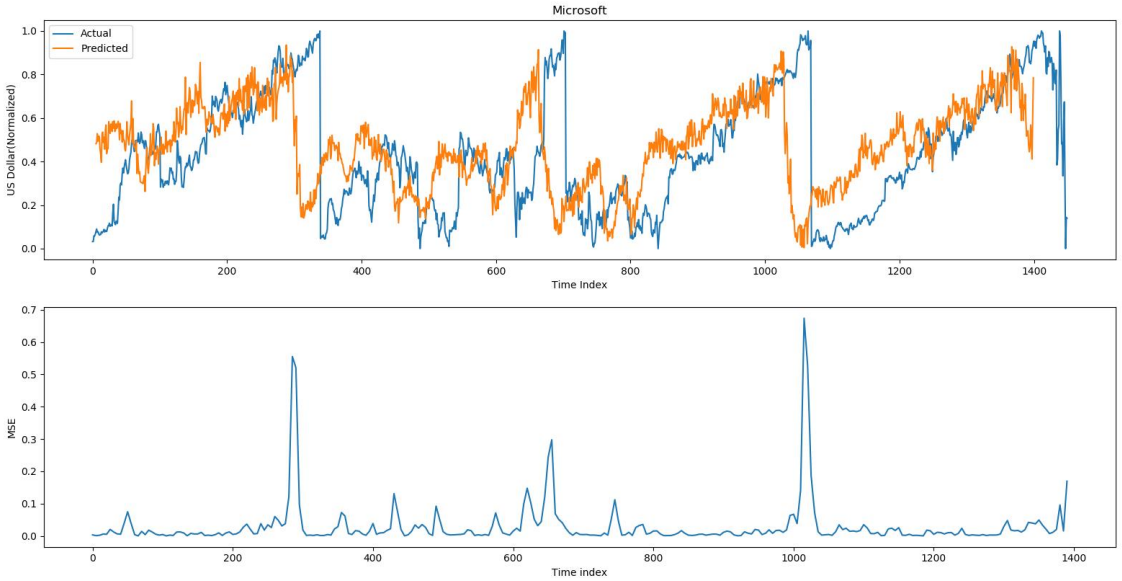

Fig. 7. Normalized stock price for actual and predicted values for Microsoft(Top) Mean Squared Error of predictions for Microsoft (Bottom)

REFERENCES

[1] Sebastian Raschka & Vahid Mirjalili, "Modeling Sequential Data using Recurrent Neural Networks" in *Python Machine Learning, 2*th ed. Birmingham, U.K.: Packt, 2017, ch. *16*, pp. *421–451*.

[2] Thushan Ganegedara. "Stock Market Predictions with LSTM in Python," DataCamp. May 3rd, 2018. [Online] Available: https://www.datacamp.com/community/tutorials/lstm-python-stock-market

[3] A. Rather, A. Agarwal and V. Sastry, "Recurrent neural network and a hybrid model for prediction of stock returns", Expert Systems with Applications, vol. 42, no. 6, pp. 3234-3241, 2015.

[4] C. Jeenanunta, R. Chaysiri and L. Thong, "Stock Price Prediction With Long Short-Term Memory Recurrent Neural Network", 2018 International Conference on Embedded Systems and Intelligent Technology & International Conference on Information and Communication Technology for Embedded Systems (ICESIT-ICICTES), 2018.

[5] L. Weng, "Predict Stock Prices Using RNN: Part 1", Lilianweng.github.io, 2018. [Online]. Available: https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html. [Accessed: 12- Nov- 2018].

[6] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Systems with Applications*, vol. 83, pp. 187–205, Oct. 2017.

[7] J. A. Jagerson, "Relative Strength Index - RSI," *Investopedia*, 20-Nov-2018. [Online]. Available: https://www.investopedia.com/terms/r/rsi.asp. [Accessed: 26-Nov-2018].

[8] "Ultimate Guide to Bollinger Bands," *TraderHQ.com*. [Online]. Available: https://traderhq.com/ultimate-guide-to-bollinger-bands/. [Accessed: 26-Nov-2018].

[9] "Understanding LSTM Networks," *Understanding LSTM Networks -- colah's blog*. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 26-Nov-2018].