Artificial Intelligence and Machine Learning

Project: Stock Market Prediction

Group Members:
Jash Kabra: 200050054
Ojaswi Jain: 200050092
Sandeepan Naskar: 200050126
Subarno Nath Roy: 200050139

Introduction

In this project, we observed data from the American stock market, particularly focusing on stocks from the companies Google, IBM and Apple. Here, we analysed the chosen dataset and after due pre-processing, ran a few primitive models, followed by predicting future stock prices through the Long Short Term Memory (LSTM) method, including a few experiments.

We explored the following questions along the way:

- What was the change in price of the stock over time?
- What was the daily return of the stock on average?
- What was the moving average of the various stocks?
- How can we attempt to predict future stock behavior?
- How far into the future can we predict?

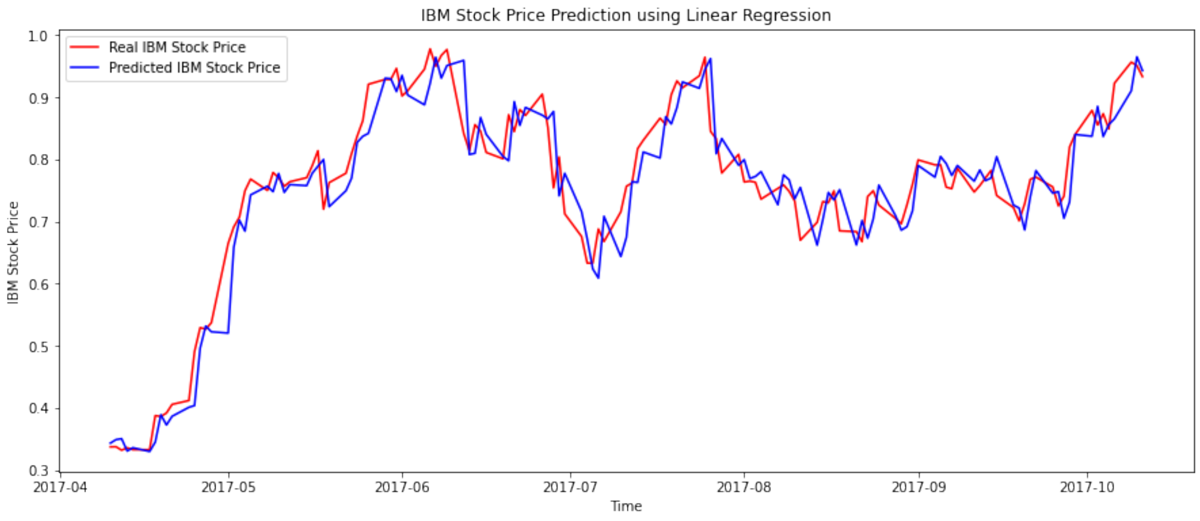
To explore these questions we started off with a basic Linear Regression model, followed by a Feed forward Neural Network and then eventually, a Long Short Term Memory (LSTM) Recurrent Neutral Network to check the stock prediction accuracies and compare them with respect to each other to find the best model for prediction.

For the training of a model, we take the previous n days of lookback data from the Input dataset and predicted the Stock Value on the next day. That is for trading, the prediction model must be run everyday with the new correct data provided after the market for the day is closed in order to predict the stock value for the next day.

We follow this training for each of the models mentioned above and elaborated below.

The Linear Regression Model

In the linear regression model we get a plot that *very* closely resembles the given curve. However, there's one caveat: the curve is slightly shifted along the positive x-axis. This implies that the linear regression model simply states yesterday's price as today's prediction - which is indeed a ridiculous proposition.

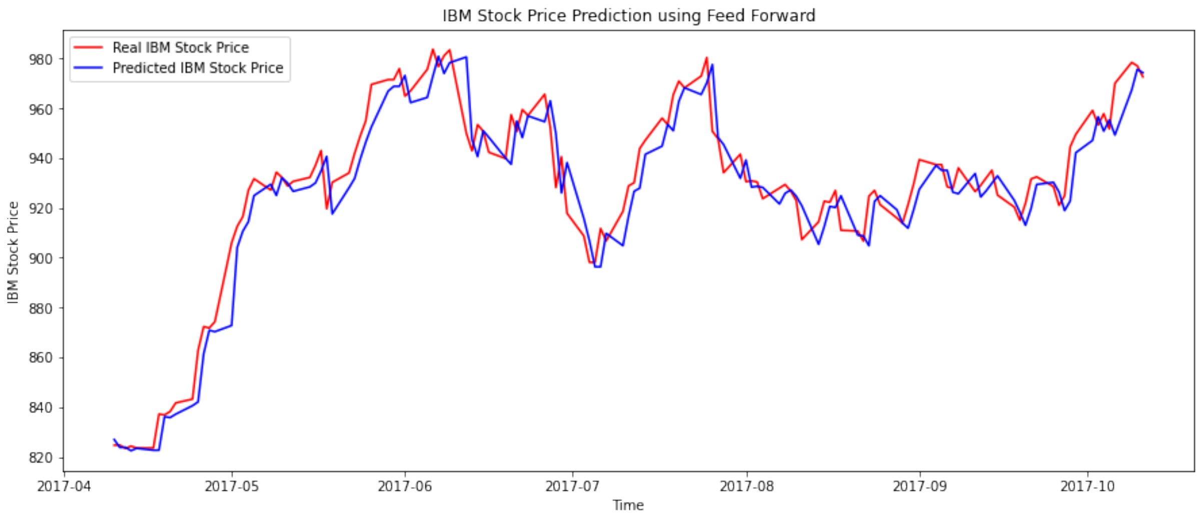


This idea yields very high accuracy and low errors since the curves are similar, but obviously this model would yield poor results when it comes to real-life trading.

As it turns out, the linear correlation between the last day's price and the current day's prediction turns out to be 0.999, which is somewhat fishy.

The Feed forward network

Again, using a simple neural network yields results extremely similar to the linear regression models, where today's predictions hinge strongly on yesterday's values. Shown below is a plot depicting the same.



In the above two models we can see that even if the stock value graphs fit pretty accurately, the prediction accuracy in the real-life stock market is still poor because the variable to be predicted is the change in stock value (whether positive or negative) and by how much (gradient of change) each day instead of just giving a prediction of the next day mostly from its immediate predecessor factoring in.
So, we need models which lay somewhat equal emphasis on the historical trends as sequences and not just isolated values. This is where **LSTM** steps in.

The Long Short Term Memory (LSTM) RNN Model

A Long Short-Term Memory (LSTM) recurrent neural network processes a variable-length sequence $x = (x1, x2, \ldots, xn)$ by incrementally adding new content into a single memory slot, with gates controlling the extent to which new content should be memorized, old content should be erased, and current content should be exposed. [1]

This applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.
For each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
&i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})\\
&f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})\\
&g_t = tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})\\
&o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})\\
&c_t = f_t \odot c_{t-1} + t_t \odot g_t\\
&h_t = o_t \odot tanh(c_t)
\end{aligned}
$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the layer at time $t-1$ or the initial hidden state at time 0, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the Hadamard product.

In a multilayer LSTM, the input $x_t^{(l)}$ of the l -th layer ($l >= 2$) is the hidden state $h_t^{(l-1)}$ of the previous layer multiplied by dropout $\delta_t^{(l-1)}$ where each $\delta_t^{(l-1)}$ is a Bernoulli random variable which is 0 with probability `dropout`.

If `proj_size > 0` is specified, LSTM with projections will be used. This changes the LSTM cell in the following way. First, the dimension of h_t will be changed from `hidden_size` to `proj_size` (dimensions of W_{hi} will be changed accordingly). Second, the output hidden state of each layer will be multiplied by a learnable projection matrix: $h_t = W_{hr}h_t$. Note that as a consequence of this, the output of LSTM network will be of different shape as well.

Inputs: input, (h_0, c_0)

- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first =False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence.

- h_0: tensor of shape $(D{*}\text{num\_layers}, H_{out})$ for unbatched input or $(D{*}\text{num\_layers}, N, H_{out})$ containing the initial hidden state for each element in the input sequence. Defaults to zeros if (h_0, c_0) is not provided.

- c_0: tensor of shape $(D{*}num\_layers, H_{cell})$ for unbatched input or $(D{*}\text{num\_layers}, N, H_{cell})$ containing the initial cell state for each element in the input sequence. Defaults to zeros if (h_0, c_0) is not provided.

where:

$$\begin{aligned}
N &= \text{batch size} \\
L &= \text{sequence length} \\
D &= 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} &= \text{input\_size} \\
H_{cell} &= \text{hidden\_size} \\
H_{out} &= \text{proj\_size if proj\_size} > 0 \text{ otherwise hidden\_size}
\end{aligned}$$

Outputs: output, (h_n, c_n)

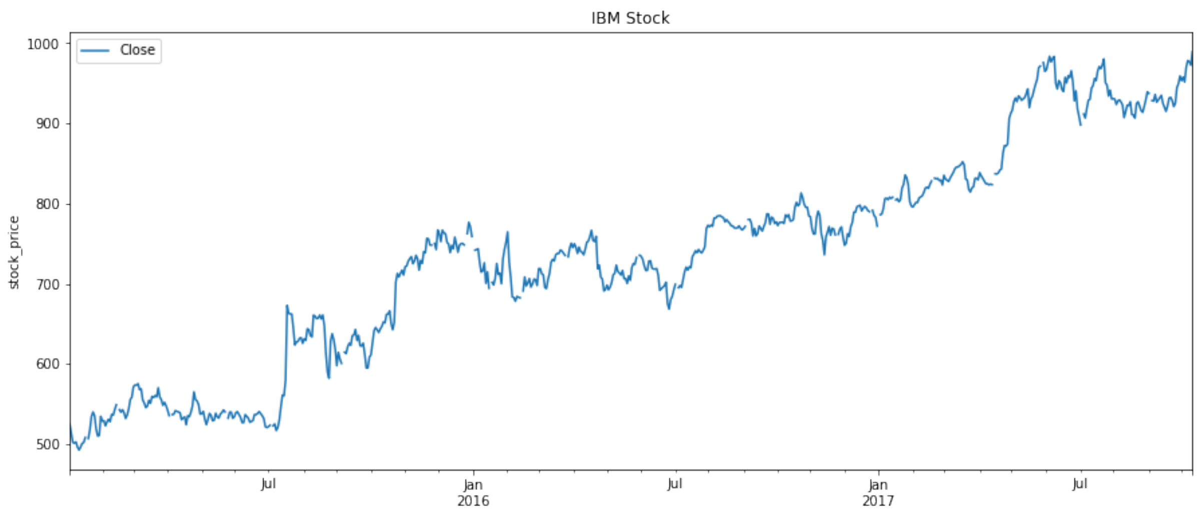
- **output**: tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features (h_t) from the last layer of the LSTM, for each t. When `bidirectional =True`, *output* will contain a concatenation of the forward and reverse hidden states at each time step in the sequence.

- h_n: tensor of shape $(D{*}\text{num\_layers}, H_{out})$ for unbatched input or $(D{*}\text{num\_layers}, N, H_{out})$ containing the final hidden state for each element in the sequence. When `bidirectional =True`, h_n will contain a concatenation of the final forward and reverse hidden states, respectively.

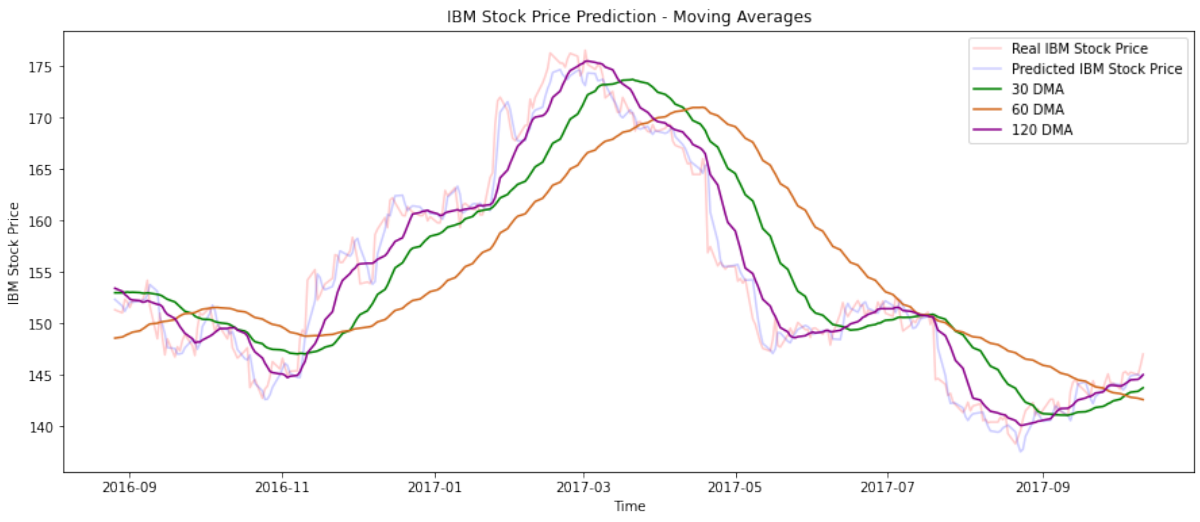
- c_n: tensor of shape $(D{*}\text{num\_layers}, H_{cell})$ for unbatched input or $(D{*}\text{num\_layers}, N, H_{cell})$ containing the final cell state for each element in the sequence. When `bidirectional =True`, c_n will contain a concatenation of the final forward and reverse cell states, respectively.

Change in price of the stock over time

Here we present the change in closing stock price over time over the entire dataset time-period for IBM stocks. The trend looks as below:



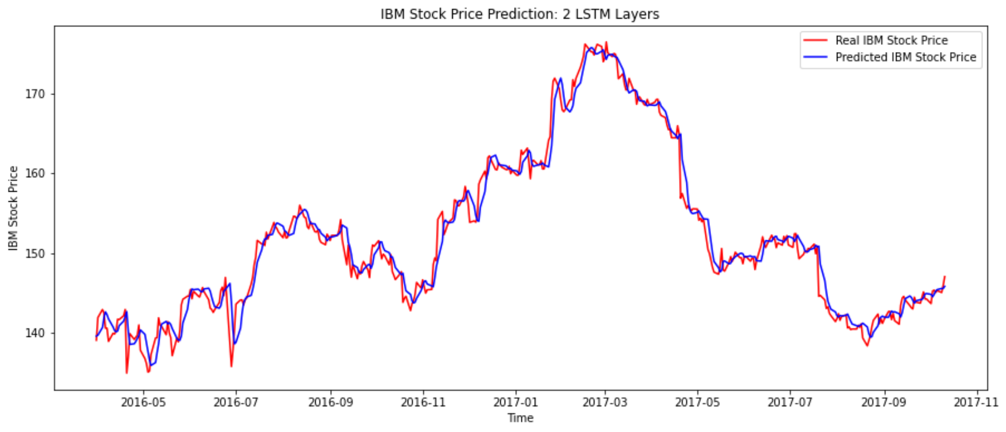
Comparison with Moving Average

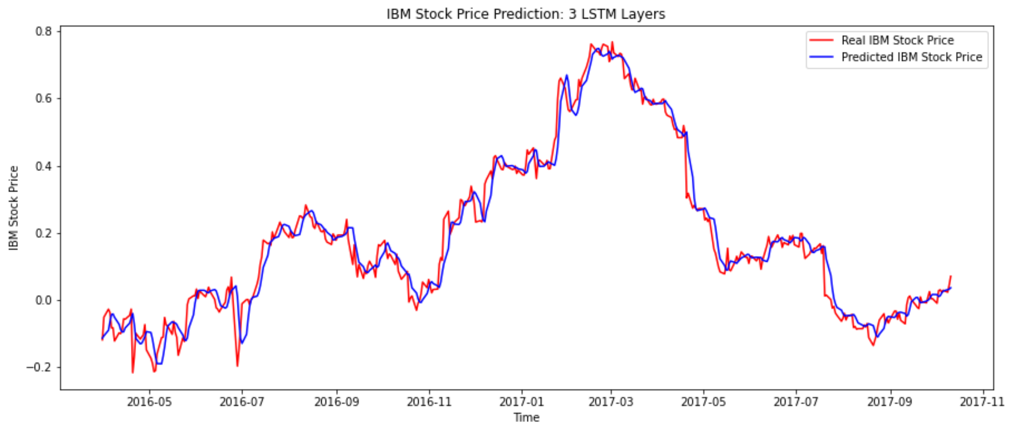


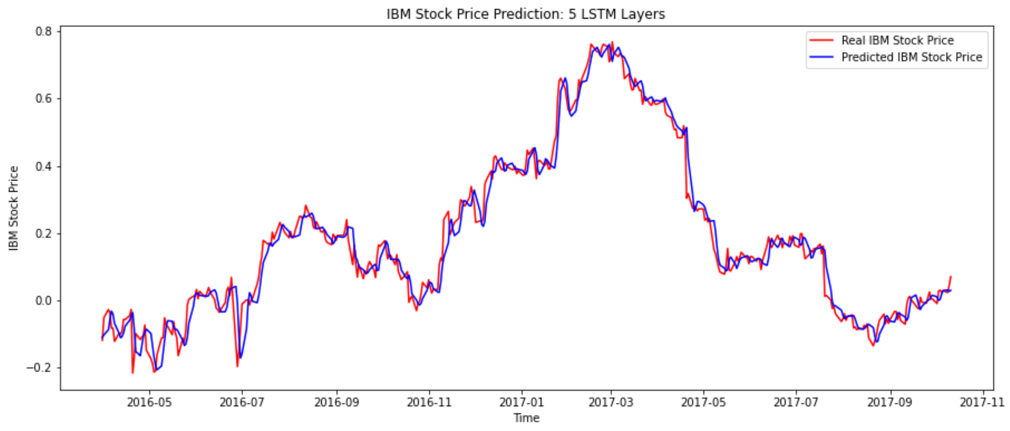
IBM Stock Prediction Graphs

(i) With respect to the number of layers in the LSTM RNN model

- Higher number of layers fit better on the training data, yielding lower losses, but after a saturation, there is no improvement in test results on adding more layers.
- More the layers, higher the training period as expected.

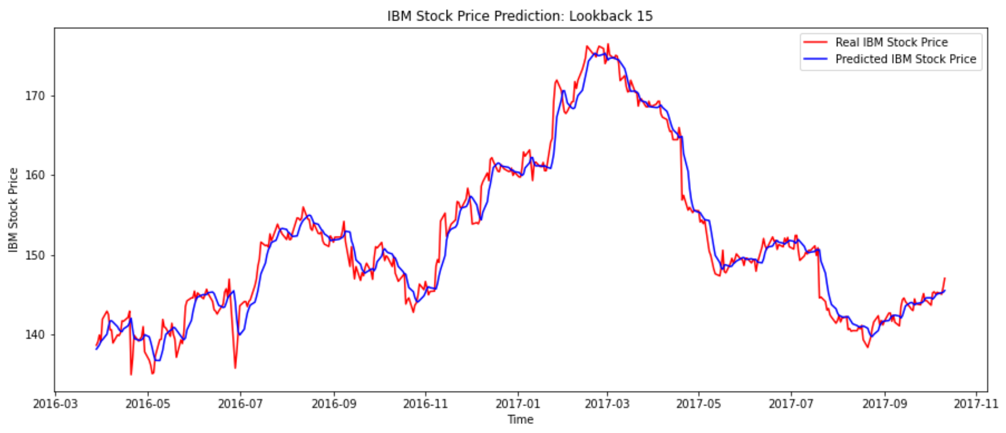


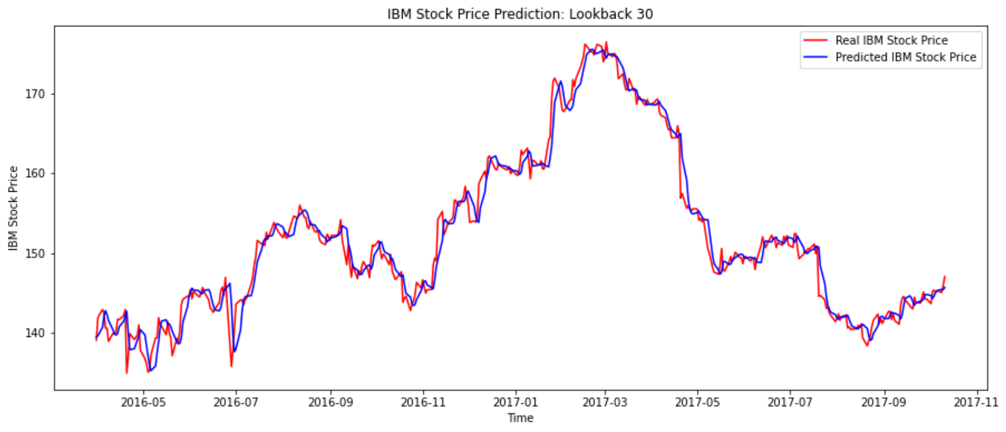


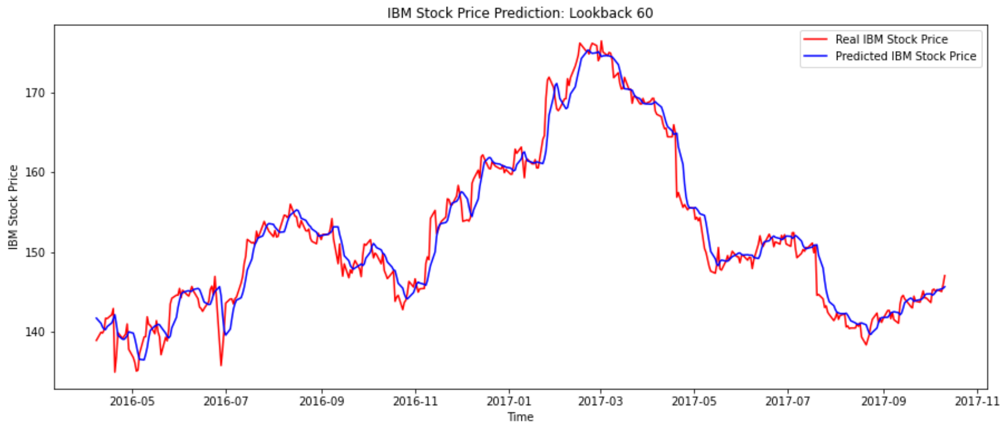


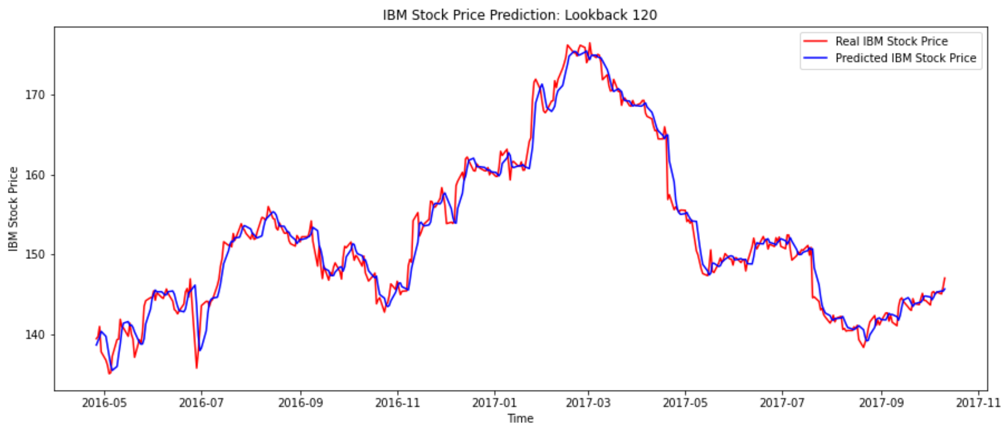
(ii) With respect to the amount of lookback (in days) we are using for prediction

- Higher lookback implies a higher training time. since taking into account the long term trends increases the complexity of the model. Such models seem to be empirically more accurate for stable stocks.
- A smaller lookback model trains faster, and lays higher emphasis on recent perturbations. This seems to suit stocks with higher volatility trends.
- As per our observations, models with lookback 30 and 120 gave extremely similar performances.


IBM Stock Price Prediction: Lookback 15
Real IBM Stock Price
Predicted IBM Stock Price
IBM Stock Price
170
160
150
140
2016-03
2016-05
2016-07
2016-09
2016-11
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11
Time


IBM Stock Price Prediction: Lookback 30
Real IBM Stock Price
Predicted IBM Stock Price
IBM Stock Price
170
160
150
140
2016-05
2016-07
2016-09
2016-11
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11
Time


IBM Stock Price Prediction: Lookback 60
Real IBM Stock Price
Predicted IBM Stock Price
IBM Stock Price
170
160
150
140
2016-05
2016-07
2016-09
2016-11
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11
Time



Long-Term experimentation with LSTM

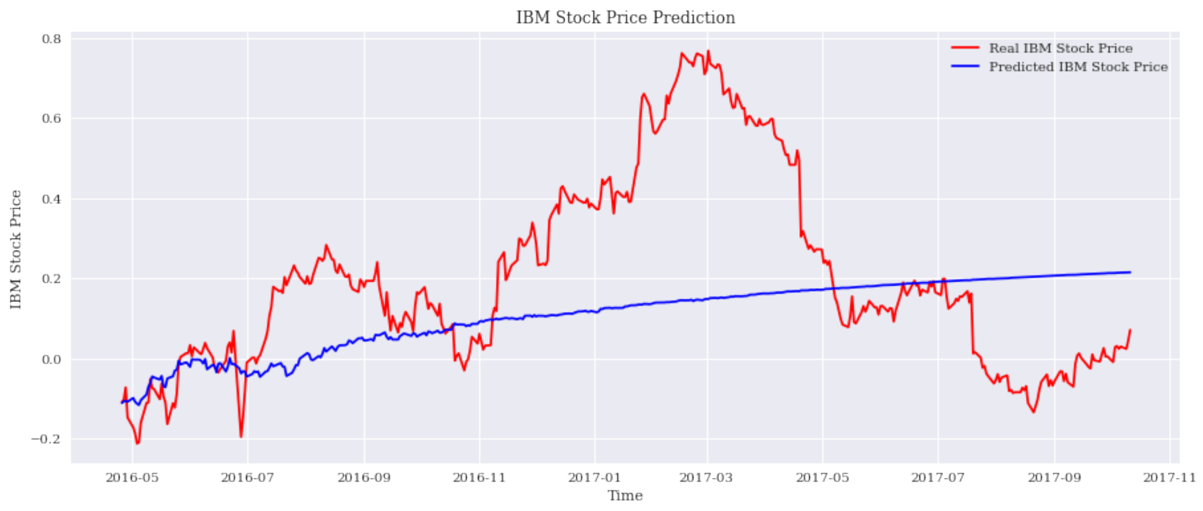
Another of our experiments involved building upon the results from the LSTM model to predict stock values well into the future.

We begin with taking in an initial sequence, and yielding appropriate outputs. These outputs are appended to the initial input sequence to obtain a new input sequence, which is then fed to the model to yield new outputs, and so on.

However, this attempt was an abject failure. This can be attributed to our initial model not being completely accurate, thus leading to cumulative build up of error, which is only amplified with time.

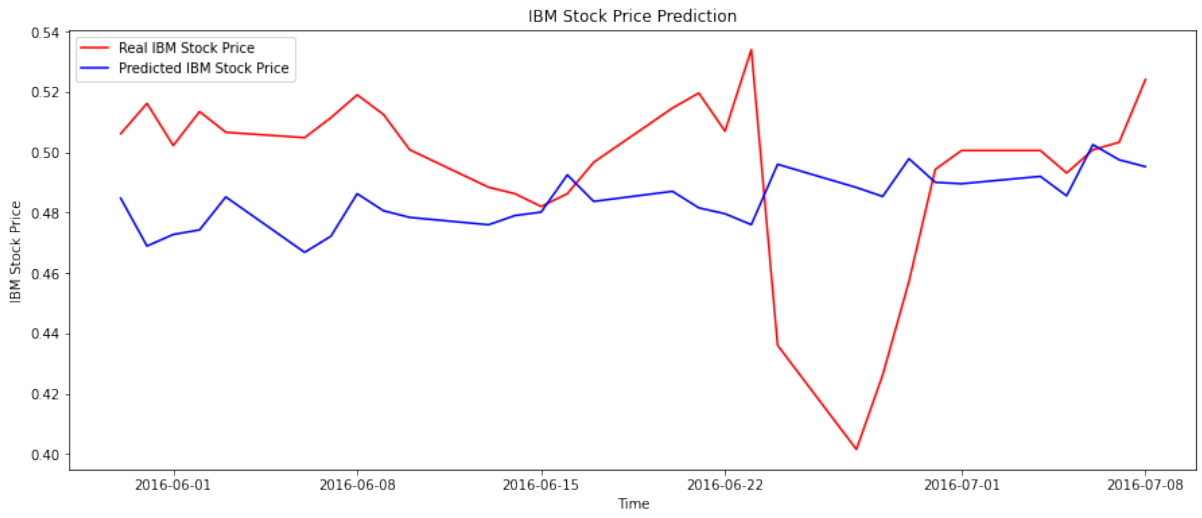
Still, the initial curve suggests that with an extremely finely tuned model, it is perhaps possible to emulate future predictions.

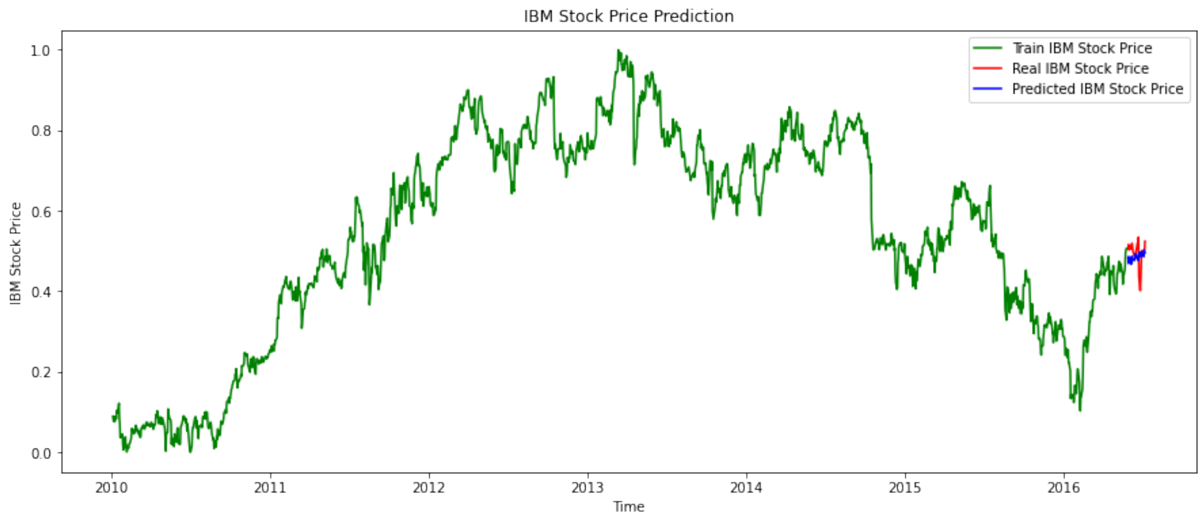
Attached below is a plot depicting the general idea:



Instead of this, we have attempted to create a sequence-to-sequence translation model using the same LSTM architecture as before, replacing the last fully connected layer to have the number of neurons as the output as we will be making a prediction on (30 days in this case). We have taken sum of RMSE across the 30-days as our loss function. It gives better output than the naive model, but there is still a lot of scope for improvement.

Here is our output for this model:





We can see that it is a big improvement over the naive model. It is also apparent that this model was unable to predict extreme perturbations in the stock prices. These can be predicted if other factors including news and sentiments are taken into account. These factors are beyond the scope of this particular project.

We still have a long way to go and our models have loads of scope for improvement. If stock market price prediction was so easy, we would all be millionaires :P.

References

[1] L. D. Jianpeng Cheng and M. Lapata, “Long short-term memory-networks for machine reading,” https://arxiv.org/pdf/1601.06733.pdf?source=post_page---------------------------, 2016.

[2] T. Zakaryan, “Predicting stock price using lstm model, pytorch,” https://www.kaggle.com/code/taronzakaryan/predicting-stock-price-using-lstm-model-pytorch/, 2020.

[3] F. Sayah, “Stock market analysis + prediction using lstm,” https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm/notebook/, 2022.

[4] S. Jariwala, “Nsepy library documentation,” https://nsepy.xyz/, 2020.

[5] J. W. Smith, “Stock prediction using recurrent neural networks,” https://towardsdatascience.com/stock-prediction-using-recurrent-neural-networks-c03637437578, 2019.