

Machine Learning Engineer Nanodegree

Capstone Project

Predict Stock Prices using Deep Learning

Navneet Latawa

28 Sep 2017

Definition

In this MLND capstone project, I will be attempting to predict future stock prices for selective stock symbols. I will take historical stock price data for specific Stock symbols from yahoo finance and use deep learning techniques to train a model. This model will then be used to predict prices for next couple of days.

As a reference, I will be using Machine Learning techniques taught in Udacity's Machine Learning NanoDegree and Machine Learning for Trading course.

Python language and Keras⁶ library will be used for application development.

Domain Background

Predicting stock prices has long been a subject of interest for traders, mathematicians, statisticians and lately technologists. This has generated a lot of interest as one will be able to reap enormous profits if one is able to predict stock prices and stock market movements reflect, to some extent, the economic state of the country also. This has also been the field of research and interest as theoretical background of financial theory does not support stock market prediction.

Theoretically, there are two major hypotheses rejecting the predictability of future stock prices.

1. **Efficient Market Hypothesis**
2. **Random Walk Theory**

Efficient Market Hypothesis¹

The efficient market hypothesis is an investment theory, proposed by Eugene Fama, which suggests that stock prices reflect all currently available information and so it is not possible to predict stock prices. This means that stock prices adjust quickly to any recently released information and past stock price history (which is publicly available) is also already reflected in the stock prices making it impossible to predict future prices. There are three versions of the efficient market hypothesis (EMH), all based on varying assumptions of price efficiency. The **weak** form of EMH claims that the prices of publicly-traded assets already reflect all available information, and past prices are of little value in predicting future trends. The **semi-strong** version of EMH holds that while prices are efficient, they react instantaneously to new information, while the **strong** version of EMH maintains that asset prices reflect not just public knowledge, but private insider information as well.

Random Walk Theory²

The **random walk theory hypothesis** is a financial theory which states that stock market prices evolve as per the random walk and thus cannot be predicted. The Random Walk Theory term was popularized by the 1973 book, *A Random Walk Down Wall Street*, by Burton Malkiel, and this was used earlier in Eugene Fama's 1965 article "Random Walks In Stock Market Prices".

There are both proponents and opponents to these hypotheses.

Proponents believe that it is pointless to search for undervalued stocks or to try to predict trends in the market through either fundamental or technical analysis. They conclude that, because of the randomness of the market, investors could do better by investing in a low-cost, passive portfolio. A study done by Morningstar Inc found out that 25% of the top-performing active managers are able to consistently outperform their passive manager counterparts.

Opponents believe that the market is predictable to some degree. Martin Weber, a leading researcher in behavioral finance, has performed many tests and studies on finding trends in the stock market. In one of his key studies, he found that stocks with high price increases in the first five years tended to become under-performers in the following five years. He cited this as a key contributor and contradictor to the random walk hypothesis. Other opponents give examples of investors such as Warren Buffett who has consistently beaten the market over long periods of time. Some give reference to events such as the 1987 stock market crash, when the Dow Jones Industrial Average (DJIA) fell by over 20% in a single day, as evidence that stock prices can seriously deviate from their fair values.

Prediction methodologies fall into three broad categories - Fundamental analysis, Technical analysis and Technological methods.

Fundamental analysis

In FA, basis of analysis is company's past performance. Fundamental analysis is built on the belief that human society needs capital to make progress and if a company operates well, it should be rewarded with additional capital which results in a surge in stock price. Fundamental analysis is widely used by fund managers as it is the most reasonable, objective and made from publicly available information like financial statement analysis. Following indicators related to past performance are used for FA.

Technical analysis

Technical analysis is not based on company's fundamentals. In TA, the future price of a stock is calculated solely on the basis of the (potential) trends of the past price (a form of time series analysis). Numerous patterns are employed such as the head and shoulders or cup and saucer. Alongside the patterns, statistical techniques are used such as the exponential moving average (EMA). Candle stick patterns are believed to be first developed by Japanese rice merchants, and nowadays widely used by technical analysts.

Technological methods

With the advent of the digital computer and recent developments in very high computing capacity, stock market prediction has evolved to a great extent. The algorithmic trading techniques such as Artificial Neural Networks (ANNs) and Genetic Algorithms are used to predict stock prices.

Personal motivation

Finance field has quite intrigued me since last couple of years. Seeing conflicting views about predicting stock prices, I myself wanted to check whether it is really possible to predict prices. During my MLND, I saw quite interesting implementations of Machine Learning techniques and wanted to utilize Deep Learning techniques to predict stock prices. Further motivation was to be able to apply the algorithm in real time to see if it will really result in stock market profits.

Problem Statement

The problem to be solved in this project is to predict the future stock prices for next couple (5, 10, 20) of days.

This problem will be structured as a supervised learning problem where the input dataset will be framed in such a way that it will have input and matching output values. A sliding window technique will be used to traverse input dataset and matching output. This will be considered a regression type of supervised learning as the output will be a continuous real value.

The input dataset will be the historical daily prices(of the chosen Stock symbol) dataset downloaded from the Yahoo finance website⁸. Adjusted Close price for each day will be considered as an input. The input values indexed on date will be treated as a time series(sequence). This dataset will be split into training, validation and test datasets. The training dataset will further be traversed using a sliding window technique. A sliding window with a fixed number of Adj Close values will be the input values and fixed number(depending upon the number of values to be predicted) of values immediately after the sliding window in sequence will be the matching output values. Validation dataset will be used to test model during training and Testing dataset will be used to test the final model.

A model will be developed matching input window values to the output values using Deep Learning. After developing the model using training dataset, stock price will be predicted for the period starting where the training dataset period ends. The predicted values can be compared against the testing dataset values for evaluation. The closeness of predicted values to actual values can be measured using Kera's evaluate method or using RMSE method.

Datasets and Inputs

Data for the project is taken from Yahoo finance website⁸. Historical data of following stock tickers is taken – S&P500, MSFT, XOM, MYL, WMT, PFE, IBM, AAPL, GOOG.

All available historical data for these stocks from yahoo finance website⁸ will be taken. Different stocks have different periods of data available.

Start and End dates for each stock are as shown below. End date may change if more data is available at the time of submission of project.

Stock Symbol	Start Date	End Date
MSFT	1986-03-13	2017-07-07
XOM	1980-12-12	2017-07-07
MYL	1980-12-12	2017-07-07
WMT	1980-12-12	2017-07-07
PFE	1980-12-12	2017-07-07
IBM	1980-12-12	2017-07-07
SP500	1980-12-12	2017-07-07
AAPL	1980-12-12	2017-07-07
GOOG	2004-08-19	2017-07-07

The historical data will be collected for following events – Open, Close, High, Low, Adj Close, Volume. Here, we will use only Adj Close⁵ price of the stocks. Adj Close price is similar to daily close price, only difference is that it takes care of past splits, reverse splits, dividends, rights offerings also.

Final dataset will contain Adj Close column with Date as the index. Dataset will be cleaned for any “null” values and it will be made sure that Adj Close data only have float values. Pandas dataframe dropna function will be used to drop any rows containing undefined data.

After data cleanup, it will be normalized so that its values are scaled between -1 and 1. It will help in convergence of machine learning algorithm and it will also help in bringing different stocks to the same scale.

Datasets for each stock will be split into training and test datasets in the ratio of 60/40 or 80/20.

A fixed window of data values from sequence will be taken as inputs and the a fixed number(depending upon the number of predicted values) of values just after the input window is taken as outputs. Window is rolled forward repeatedly to traverse the input dataset to get inputs and matching outputs.

Although chances of testing data information being trickled to training are negligible, as testing is done after model is fully trained using Keras

To avoid testing dataset information trickle down to training dataset -

- Testing data is used after model is fully trained

- Training and testing will be done on different stock symbols.

Solution Statement

In this project, attempt will be made to predict the prices using Deep Learning techniques of Recurrent Neural Networks. Recurrent Neural Network model LSTM (Long short term Memory) will be used to predict the stock prices.

Evaluation Metrics

Predicted prices will be compared to Actual historical prices.

- Keras model.evaluate method will be used to find the training and test errors.
- MSE (Mean Square Error) will be calculated for predicted and actual values.

MSE - Mean Squared Error¹⁰

MSE of a predictor measures the average of the squares of the errors or deviations of the predictor from what is predicted.

If \hat{Y} is a vector of n predictions and Y is a vector of observed values corresponding to the inputs to the function which generated the predictions, then the MSE of the predictor can be estimated by¹⁰

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$$

As this project involves comparing two number series, MSE seems to be an appropriate function to compare values. It is one of the simpler methods to compare series.

Analysis

Project is implemented using programming language Python and neural network API Keras is used for deep learning.

Data Exploration and Preprocessing

Input Data

- Data is downloaded from finance.yahoo.com⁸ for selected tickers (S&P500, IBM, MSFT, WMT, PFE, XOM, MYL etc.) for date ranges (1980-12-12 to 2017-07-07)
- Data is read using pandas dataframe.
- Data is indexed using date value.

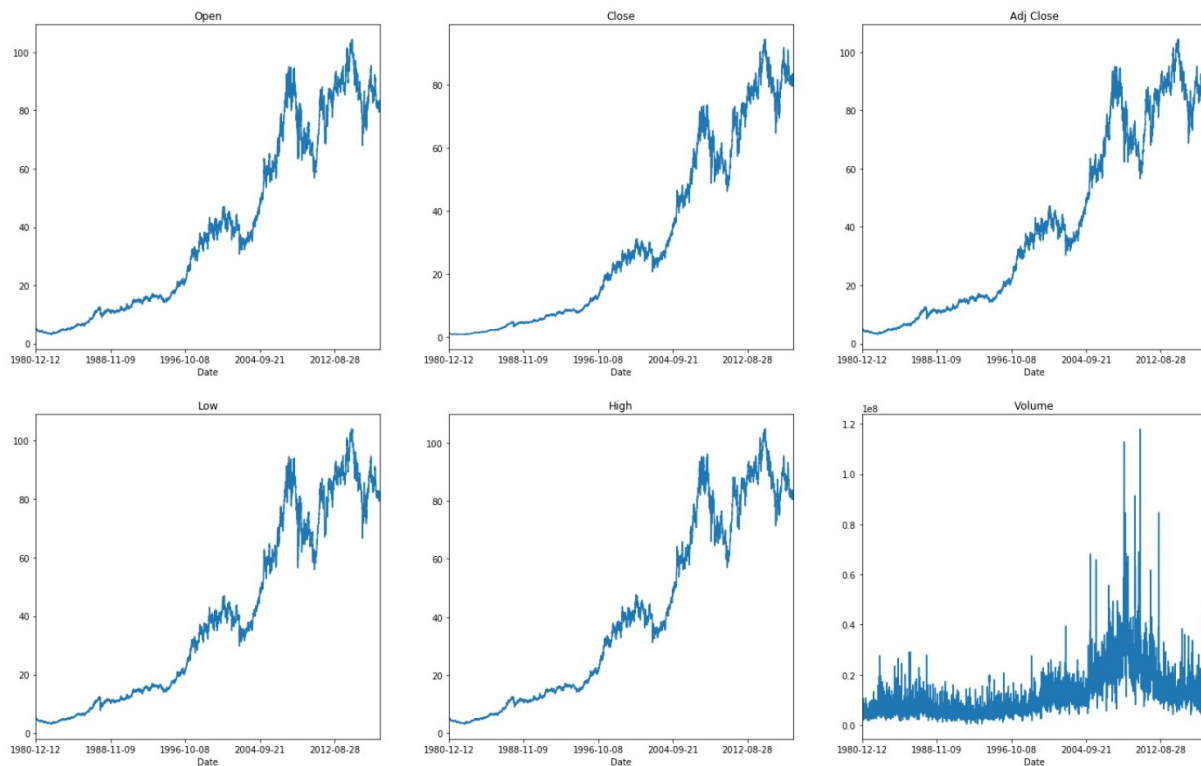
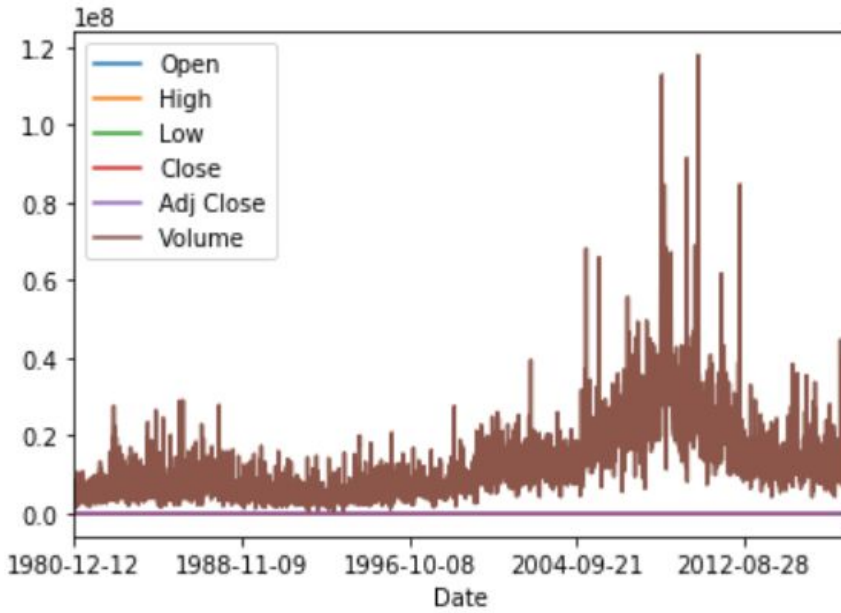
Data Cleanup

- Nulls and blanks will be removed from the data.

Feature Selection

- Input features - Open, Close, Low, High, Close, AdjClose and Volume - are explored for SP500 dataset.

	Open	High	Low	Close	Adj Close	Volume
Date						
1980-12-12	4.882813	4.953125	4.882813	1.116966	4.921875	4892800
1980-12-15	4.921875	4.992188	4.921875	1.122285	4.945313	2446400
1980-12-16	4.945313	5.125000	4.945313	1.157744	5.101563	4040000
1980-12-17	5.148438	5.289063	5.148438	1.196749	5.273438	3816000
1980-12-18	5.273438	5.359375	5.210938	1.182565	5.210938	5790400

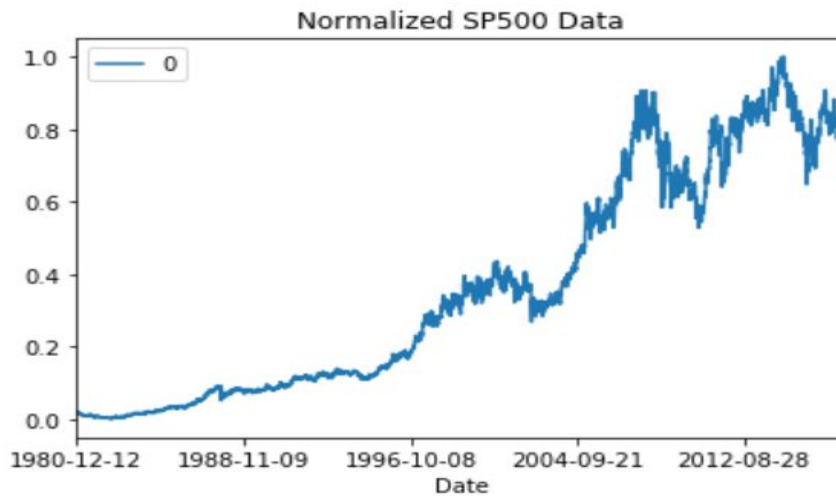


As Open, Close, Low, High, Close and AdjClose follow almost same pattern, so only one of these AdjClose is selected for analysis.

Daily Return for AdjClose data was also explored but it did not seem to add additional value to the results. So it was not included in the final analysis.

Normalize

- AdjClose data is normalized using sklearn MinMaxScaler so that values are between 0 and 1. Different Scalers are used for each Stock symbol so that these scalers can then be used later to recover original values.



Train\Validation\Test

- Data is split in three separate sets(60:20:20) for training, validation testing and testing. This is done so that there is no information leak from training to testing dataset.

Supervised Form Data

- For supervised learning, data is transformed so that it represents input-output dependency relationship. Various window sizes(5, 10, 25, 50, 75, 100) for input data are used for finding best fit.

Each set of data (Train, Validation and Test) is separately prepared to be in supervised form.

SP500 training data first 5 rows using Window Size = 5

	lag_1	lag_2	lag_3	lag_4	lag_5	output
0	0.013007	0.012960	0.011986	0.011497	0.011403	0.013305
1	0.013305	0.013007	0.012960	0.011986	0.011497	0.014189
2	0.014189	0.013305	0.013007	0.012960	0.011986	0.013985
3	0.013985	0.014189	0.013305	0.013007	0.012960	0.014232
4	0.014232	0.013985	0.014189	0.013305	0.013007	0.014525

Algorithms and Techniques

Algorithm

The problem to predict future prices is being framed as a supervised learning problem where output is dependent upon a number of features. Here, to start with an assumption is made that price of a stock is dependent upon previous days' values. Then a neural network based deep learning LSTM RNN model is used to capture this dependency relationship and to predict the future prices.

As artificial neural networks have the ability to identify various patterns and relationships among data, these are used in this project to identify dependency of stock prices on past stock prices.

Artificial Neural Networks³

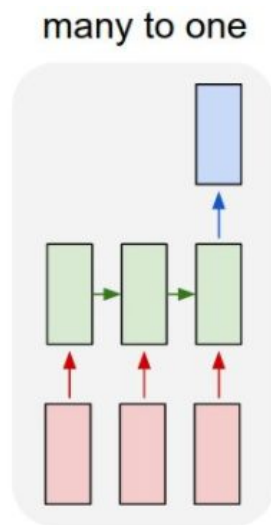
Artificial neural networks are computing systems inspired by the biological neural networks constituting human brains. A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use.¹² An ANN is a collection of connected units (logical units) called artificial neurons. In a neural network, thousands or millions of neurons, are organized in layers. Each layer may perform different kinds of transformations on their inputs. ANN learn by forward propagation (matching inputs to outputs) and backpropagation (matching outputs to inputs).

RNN (Recurrent Neural Network)⁴

A **recurrent neural network (RNN)** is a class of artificial neural network in which neurons have directed cycle connections. Unlike feedforward neural networks(where neurons are connected in only one direction from input to output), RNNs have backward connections also. RNNs can use their internal memory to process arbitrary sequences of inputs.

Recurrent neural networks (RNNs) are connectionist models that capture the dynamics of sequences via cycles in the network of nodes. Unlike standard feedforward neural networks, recurrent networks retain a state that can represent information from an arbitrarily long context window.¹³ This helps RNNs in identifying patterns in sequences in a better way than the standard feedforward neural networks.

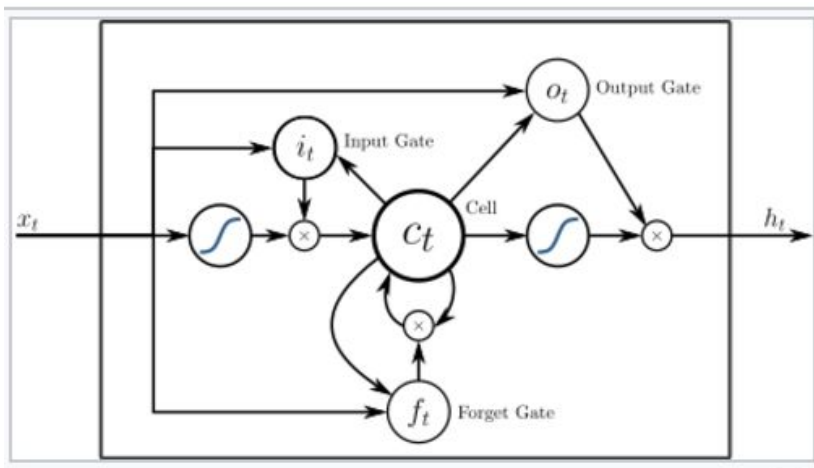
The model used in this project is many to one model as shown in fig¹⁴ below.



LSTM (Long Short Term Memory) Network⁴

Long short-term memory (LSTM) is specific type of RNN that attempts to remember and forget previously identified patterns. LSTMs have “forget” gates to discard unwanted patterns. LSTMs try to remove the vanishing or exploding gradient problems and this helps LSTM learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier.

Fig Ref:¹⁵



Benchmark Model

Multiple benchmark models will be used for comparison:

- 1) As we are using existing data and we have test data available for which actual values are already available. We can test our model generated predicted values with these actual values and can see how accurate our model is.
- 2) Use Linear Regression prediction Model⁹
- 3) Find\develop prediction model using KNN(K nearest neighbor) regression model and run the model on same data as to be used on Deep Learning model. KNN model will be used as a benchmark.
- 4) Use Support vector Machine(RBF and Poly) methods

Implementation

After doing Data Reading, Data CleanUp, Normalizing, Data Splitting and Supervised transformation for SP500 dataset as explained earlier, Data modelling and fitting is done.

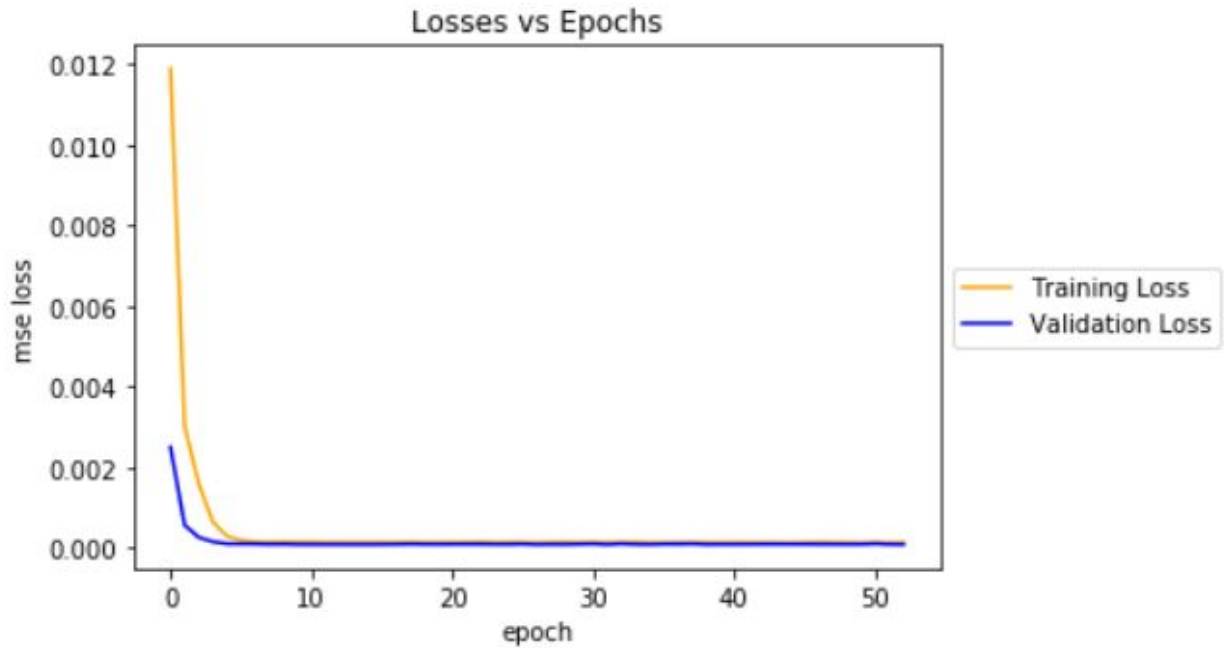
Data Modelling and Fitting

- Keras API is used for model training and prediction testing.
- Keras sequential model⁷ is used for training.
- Two LSTM layers are added to this sequential model.
- Dropout layers with dropout rate of 0.2 is added after each LSTM layer to avoid data overfitting. Overfitting happens when training error keeps on decreasing but testing error starts rising after reaching bottom.
- Final linear layer with single output is added at the end of model.
- Adagrad Optimizer⁷ (provided by Keras API) is used to optimize the model.
- Model is compiled with batch_size = 256 and default optimizer parameters.
- A high number(300) is assigned to epochs but best model is saved based on validation_loss and model fit is Early stopped if there is no improvement for 20 epochs.
- Model fitting results in following mse losses.

Model Summary

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, None, 50)	10400
dropout_1 (Dropout)	(None, None, 50)	0
lstm_2 (LSTM)	(None, 256)	314368
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257
activation_1 (Activation)	(None, 1)	0
Total params: 325,025		
Trainable params: 325,025		
Non-trainable params: 0		

Losses during model training



Model Tuning

Various parameter values were tried before reaching the final model.

Window_sizes = 5, 10, 25, 50, 75, 100, 150

Batch_sizes = 64, 128, 256, 512, 1024

Layers - LSTM layers upto 5

Dropout rates - 0.2 to 0.8

Different Layers - Convolutional Layers, Bidirectional LSTM units, GRUs

Optimizers - Adagrad (Learning rates - 0.001, 0.005, 0.008, 0.01), RMSProp, Adam

Model parameters optimal values

Window Size = 5 [out of 5, 10, 25, 50, 75, 100, 150]

Batch Size = 256 [out of 64, 128, 256, 512, 1024]

LSTM = 2 Layers with 50 and 256 units out of [(5,256), (100, 256), (50, 128), (50, 256), (50,512)]

DropOut = 0.1 [out of No Dropout, 0.1, 0.2, 0.5, 0.8]

Optimizers = Adagrad [out of Adagrad, RMSProp, Adam]

Adagrad Learning Rate = 0.005 [out of 0.001, 0.005, 0.008, 0.01, 0.05]

Results

Model Testing and Prediction

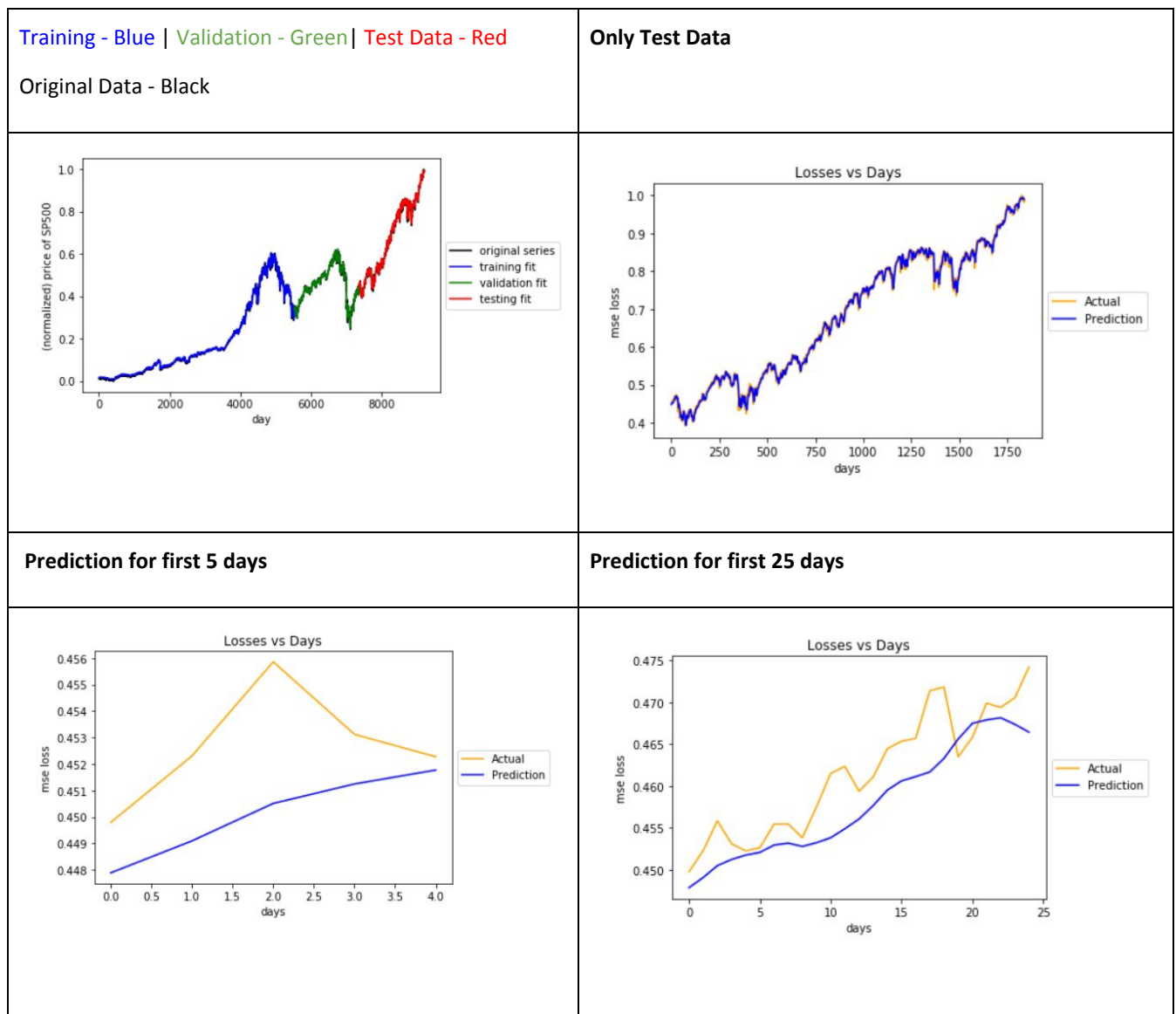
- After model fitting and testing, prediction is made for training and testing data to calculate training and testing errors. Keras Predict⁷ and evaluate functions will be used to predict and evaluate⁷.
- Testing is done first on SP500 Testing dataset.
- For testing model robustness, testing is on other stock datasets (IBM, MSFT, WMT, PFE, XOM, MYL, AAPL, GOOG).

Test, Validation and Prediction Errors - SP500

Training error = $4.07788668582e-05$

validation error = $7.53448343746e-05$

Testing error = 0.000100467985182



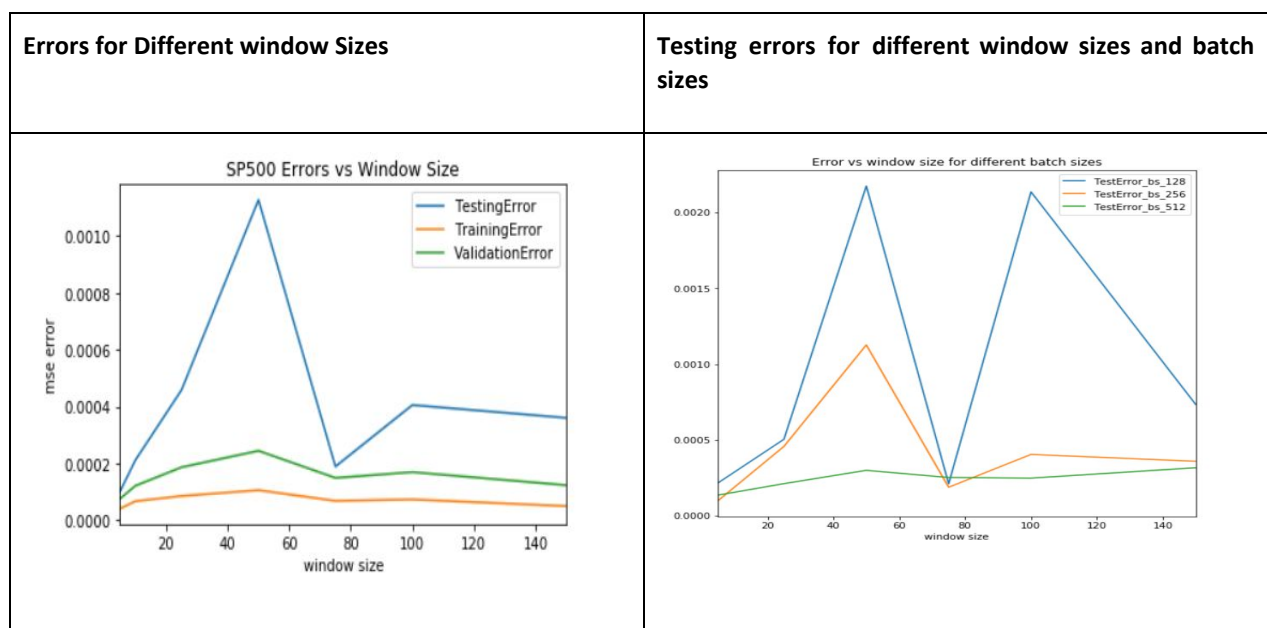
Model was trained and tested using different window sizes. Following table shows that the Errors are lowest for window_size = 5.

Window_Size	Test_Data_Size	SP500_Training_Error	SP500 Validation Error	SP500 Test Error
5	1834	0.00004078	0.00007534	0.00010047
10	1834	0.00006683	0.00012114	0.00021127
25	1834	0.00008527	0.00018637	0.00045830
50	1834	0.00010575	0.00024435	0.00112720
75	1834	0.00006819	0.00014890	0.00018894
100	1834	0.00007315	0.00016924	0.00040543
150	1834	0.00004967	0.00012329	0.00036009

Testing Errors for Different Window_Sizes and Batch Sizes

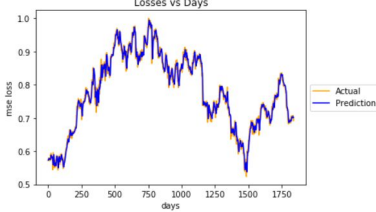
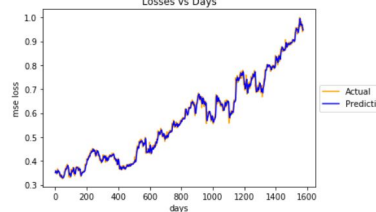
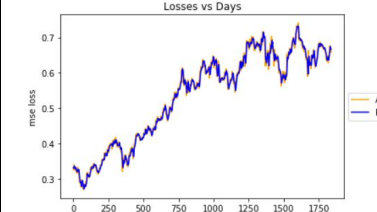

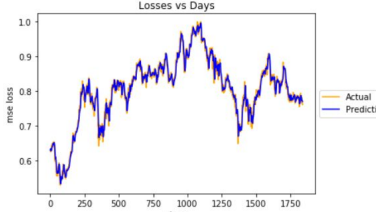


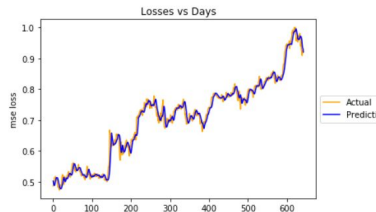
Left graph shows that Window size = 5 gives the lowest error.

Model was trained using different batch sizes (128, 256, 512) and testing errors below show that the batch size of 256 for window size = 5 gives lowest error.



Model Robustness

Model testing on other Stocks' Test Data - For checking robustness, trained model was tested for other stocks also. Following results show that model performs quite well for other stocks also. As this data is not previously seen by the model, it depicts model's robustness in predicting future values.

IBM Testing error = 0.000257458367624	MSFT Testing error = 0.000179582325707	PFE Testing error = 0.000104524871926
		
WMT Testing error = 0.000148269985709	XOM Testing error = 0.000225811280732	MYL Testing error = 0.000338295471863
		
AAPL Testing error = 0.000207465062985	GOOG Testing error = 0.000326079152949	
		

Benchmark

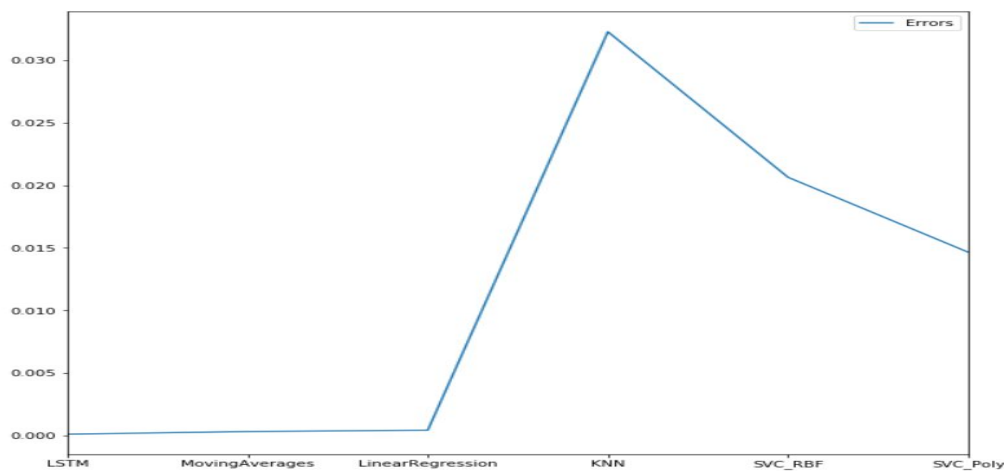
Following benchmark models are considered^[1]

- Moving Averages
- Linear Regression
- KNN
- Support vector machines (RBF)
- Support vector machines (Poly)

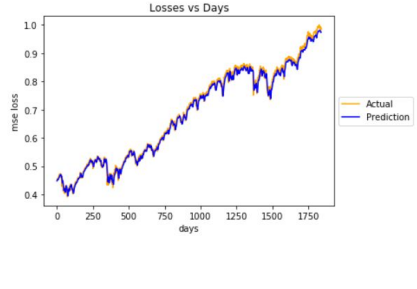
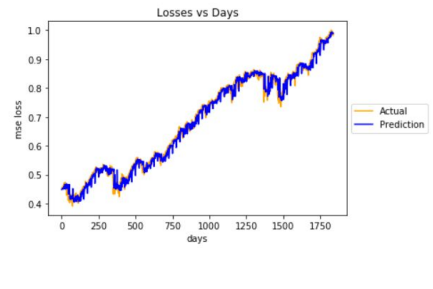
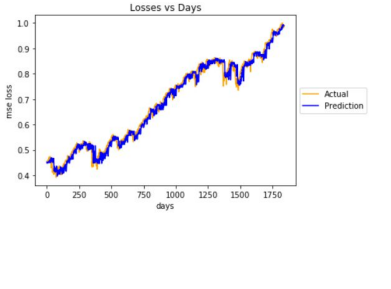
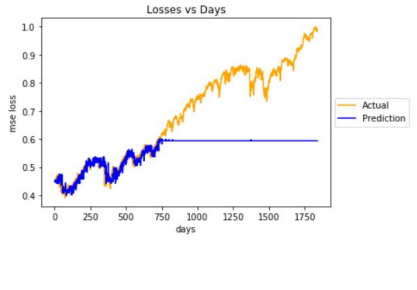
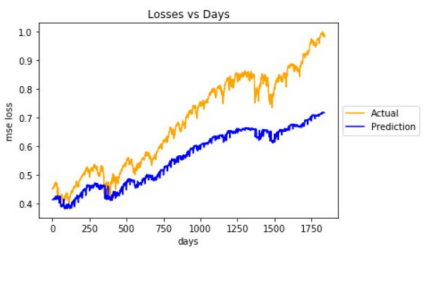
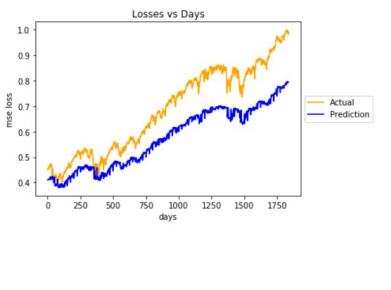
Testing results shown below for these benchmarks show that only Linear Regression and Moving Averages are close to the performance obtained with Deep Learning LSTM model and the selected LSTM model for SP500 gives best performance.

Window Size	SP500 Test Error	IBM Test Error	Moving Averages	Linear Regression	KNN	SVC RBF	SVC Poly
5	0.00010047	0.00025746	0.00031200	0.00041431	0.03227614	0.02063950	0.01464979
10	0.00021127	0.00042141	0.00029952	0.00054327	0.03261012	0.02372918	0.01130500
25	0.00045830	0.00073806	0.00038604	0.00088376	0.03746455	0.04741630	0.00838335
50	0.00112720	0.00159768	0.00055406	0.00188684	0.03814208	0.07295037	0.00999640
75	0.00018894	0.00046954	0.00070063	0.00041257	0.03899267	0.08549924	0.01116770
100	0.00040543	0.00072450	0.00091074	0.00072633	0.04193859	0.09606416	0.01167342
150	0.00012329	0.00036009	0.00063898	0.00145170	0.00072645	0.05042111	0.11851730

Testing Errors for Different Models



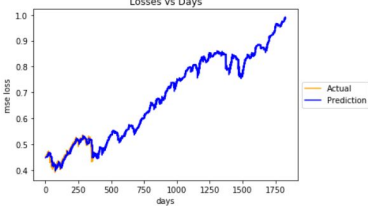
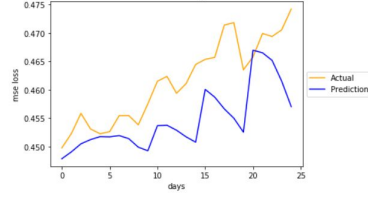
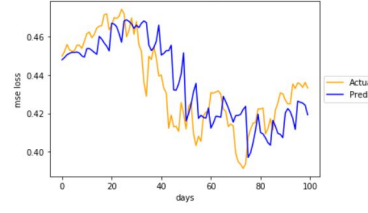
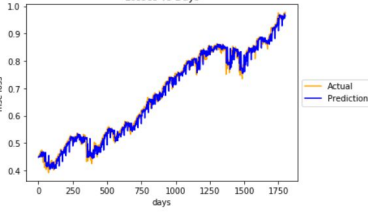
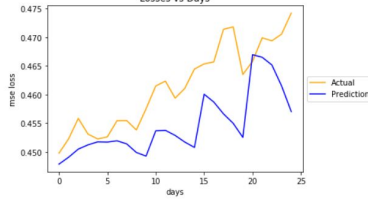
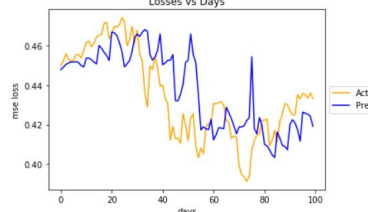
Benchmarks Visualizations

<p>LSTM Model</p> <p>Testing error = 0.00010047</p>	<p>Moving Averages</p> <p>Testing error = 0.00031200</p>	<p>Linear Regression</p> <p>Testing error = 0.00041431</p>
 <p>Losses vs Days</p> <p>This plot shows the mean squared error (mse) loss over 1750 days for the LSTM model. The y-axis represents 'mse loss' from 0.4 to 1.0, and the x-axis represents 'days' from 0 to 1750. The 'Actual' loss (orange line) and 'Prediction' (blue line) are nearly identical, showing a fluctuating upward trend from approximately 0.45 to 0.95.</p>	 <p>Losses vs Days</p> <p>This plot shows the mse loss over 1750 days for the Moving Averages model. The y-axis represents 'mse loss' from 0.4 to 1.0, and the x-axis represents 'days' from 0 to 1750. The 'Actual' loss (orange line) and 'Prediction' (blue line) are nearly identical, showing a fluctuating upward trend from approximately 0.45 to 0.95.</p>	 <p>Losses vs Days</p> <p>This plot shows the mse loss over 1750 days for the Linear Regression model. The y-axis represents 'mse loss' from 0.4 to 1.0, and the x-axis represents 'days' from 0 to 1750. The 'Actual' loss (orange line) and 'Prediction' (blue line) are nearly identical, showing a fluctuating upward trend from approximately 0.45 to 0.95.</p>
<p>KNN</p> <p>Testing error = 0.03227614</p>	<p>SVM(Kernel=rbf)</p> <p>Test Error = 0.02063950</p>	<p>SVM(Kernel=poly)</p> <p>Test Error = 0.01464979</p>
 <p>Losses vs Days</p> <p>This plot shows the mse loss over 1750 days for the KNN model. The y-axis represents 'mse loss' from 0.4 to 1.0, and the x-axis represents 'days' from 0 to 1750. The 'Actual' loss (orange line) shows a significant upward trend, reaching nearly 1.0 by day 1750. The 'Prediction' (blue line) is a flat horizontal line at approximately 0.58, indicating a complete failure to track the actual loss.</p>	 <p>Losses vs Days</p> <p>This plot shows the mse loss over 1750 days for the SVM(Kernel=rbf) model. The y-axis represents 'mse loss' from 0.4 to 1.0, and the x-axis represents 'days' from 0 to 1750. The 'Actual' loss (orange line) shows a significant upward trend, reaching nearly 1.0 by day 1750. The 'Prediction' (blue line) follows the general trend but remains significantly lower than the actual loss, ending around 0.7.</p>	 <p>Losses vs Days</p> <p>This plot shows the mse loss over 1750 days for the SVM(Kernel=poly) model. The y-axis represents 'mse loss' from 0.4 to 1.0, and the x-axis represents 'days' from 0 to 1750. The 'Actual' loss (orange line) shows a significant upward trend, reaching nearly 1.0 by day 1750. The 'Prediction' (blue line) follows the general trend but remains significantly lower than the actual loss, ending around 0.8.</p>

Limitations

There is a limitation with this model that if we use predicted values for predicting further in future then the errors pile up. So this model can be used for predicting prices only in the near future (for a couple of days). As number of days for prediction increase, the prediction errors increase.

Below graphs show that if we use predicted values in future predictions, the predictions fall apart and correct results are only upto a few days. This happens because errors accumulate and trickles down to future values

Predicting 5 values at a time and using these 5 values to predict future values		
Whole Test Data	First 25 predicted values	First 100 predicted values
		
Predicting 25 values at a time and using these 25 values to predict future values		
Whole Test Data	First 25 predicted values	First 100 predicted values
		

Conclusions¹

Prediction of future stock prices has been quite challenging and tedious task. After spending lot of time on framing the data for supervised learning, then model selection and tuning, it has been possible to build a model performing close(rather better) to the model in practice in industry. The Deep Learning LSTM model has worked than the benchmarked models.

Feature selection showed that there is enough information in previous Adj Close prices to predict the future values. Using other features such as High, Low , Open, Close and calculated features DailyReturn did not add any value to the model.

Different window sizes (previous days values) experimentation also showed that 5 days data carried enough information to predict future values and adding additional days data did not add to the performance.

Batch sizes for model training also affected performance and although using larger batch sizes made the model run faster, but it resulted in low model accuracy.

The problem of predicting stock prices actually involves identifying patterns in sequences and LSTMs are designed to predict sequence patterns. LSTMs have been successfully used in identifying patterns in images, speech recognition and some attempt has been made to predict future time series. The results from this project show that LSTMs can be used successfully in predicting future prices.

Reflections

There were a lot of challenges faced before reaching the unbelievable results.

First challenge was how to frame the problem i.e, how to extract the dependency patterns in the data. The dependency was expected to be feature related and then time dimension also needed to be added as time dependent future prices were to be predicted.

- Time dependency was added as taking any day price and assuming that it depended upon previous days' values. Then challenge was to take how many days in the past. I started with 5 days windows and tested model with window sizes upto 150. Increased window sizes required complex training model design(having more LSTM layers) and much longer time for training but not much improvement on errors\fitting.
- Feature Selection involved trying different input values (Open, Close, High, Low, Adj Close and Daily Return). Open, Close, High, Low, AdjClose looked similar in growth pattern and this manifested in not showing improvement in fitting. DailyReturn looked different from other features but even it did not add any improvement. Although I could have tried Volume also but seeing not much improvement in results with DailyReturn, only AdjClose was finally chosen.

Second major challenge was model design and tuning - Different options mentioned below were tried.

- Additional LSTM layers
- Large number of LSTM units in each layer (I tried even more than 1000 units in 1 LSTM layer)
- GRUs in place of LSTM
- Different values for Drop-outs
- Batch Sizes of 64, 128, 256, 512, 1024
- Keras optimizers - Adagrad, RMSProp, Adam
- Optimizer parameter values - although Keras suggested not to change the parameter values - I did try different Learning Rates and Decay values and found out that LR= 0.005 gave better results than the default LR = 0.01.

Third challenge was avoiding information leakage from Training to Validation to Testing Datasets. Originally I had transformed data from sequences to supervised form before splitting data into Training, Validation and Testing datasets. This was causing data overlap between Training and Validation datasets and between Validation and Testing datasets. I fixed this problem by first splitting the datasets and then transforming each dataset separately to

Training, Validation and Testing datasets. This helped in avoiding information leakage from Training to Validation and From Validation to Testing datasets.

Another coding challenge was about normalizing data. For Open, Close, Low, High, AdjClose normalization was to be done between 0 and 1 but as I was trying DailyReturn which could have been negative also, so I chose normalization between -1 and 1 and I kept this for all features even when I had removed DailyReturn from analysis. I realized it during code walk through later when I was not getting good results.

Using Deep Learning model with trying all of the above different options was very time consuming and it required a lot of patience.

Improvements

- 1) Try other features such as Volume of trading for data dependency
- 2) Although I tried various models with different number of layers, different number of LSTM units per layer, tried Bidirectional LSTMs (to check current price dependency not only on past values but future values also), tried combination of Convolution layers and Bidirectional and simple LSTM units but these did not give better results. GAN(Generative Adversarial Networks) can be tried for predicting stock prices.
- 3) Removing stationarity - on a long term basis , there is a trend of increasing values of stock prices. It can be tried to remove this trend and then predict the prices.
- 4) For predicting a fixed number of days - for example 5 days, we may need to modify the model to show output to be 5 dimensional, so that 5 values are predicted at the same time. We will need to frame this problem as LSTM many to many sequence problem where input is a number of days of input data and output is fixed number of days of output.

References

- 1) https://en.wikipedia.org/wiki/Stock_market_prediction
- 2) https://en.wikipedia.org/wiki/Random_walk_hypothesis
- 3) https://en.wikipedia.org/wiki/Artificial_neural_network
- 4) https://en.wikipedia.org/wiki/Recurrent_neural_network
- 5) http://www.investopedia.com/terms/a/adjusted_closing_price.asp
- 6) <https://keras.io/>
- 7) <https://keras.io/models/model/>
- 8) <https://finance.yahoo.com/>
- 9) <https://www.quantstart.com/articles/Forecasting-Financial-Time-Series-Part-1>
- 10) https://en.wikipedia.org/wiki/Mean_squared_error
- 11) <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- 12) https://people.eecs.berkeley.edu/~akar/IITK_website/EE671/report_stock.pdf
- 13) <https://arxiv.org/pdf/1506.00019.pdf>
- 14) <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- 15) https://en.wikipedia.org/wiki/Long_short-term_memory