Machine Learning Nanodegree

Capstone Project (Kushal Sharma)

Project Overview

Stock market prediction is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. A stock market, equity market or share market is the aggregation of buyers and sellers of stocks (also called shares), which represent ownership claims on businesses; these may include securities listed on a public stock exchange as well as those only traded privately. The successful prediction of a stock's future price could yield significant profit.

History has shown that the price of stocks and other assets is an important part of the dynamics of economic activity, and can influence or be an indicator of social mood. An economy where the stock market is on the rise is considered to be an up-and-coming economy. The stock market is often considered the primary indicator of a country's economic strength and development.

With the advent of the digital computer, stock market prediction has since moved into the technological realm. The most prominent technique involves the use of artificial neural networks(ANNs) and Genetic Algorithms. The most common form of ANN in use for stock market prediction is the feed forward network utilizing the backward propagation of errors algorithm to update the network weights. These networks are commonly referred to as Backpropagation networks. Another form of ANN that is more appropriate for stock prediction is the time recurrent neural network (RNN) or time delay neural network (TDNN).

Problem Statement

In this project the challenge is to analyse the data of a given stock we fetch from an API and predict the return of a stock, given the history of the past few days/months/years. Provided N-day window of time, days N-2, N-1, N, N+1. We have returns in days N-2, N-1, N, and we have to predict the returns in the coming day N+1.

The goal of this project is to reasonably predict the price of individual stocks one day in future, provided the historical data for that stock. The tasks involved are the following:

1. Download and preprocess the historic data for a given stock.
2. Create a LSTM model using Keras and TensorFlow and train the model.
3. Evaluate results on test dataset.
4. Predict the value of stock one day in future.

Metrics

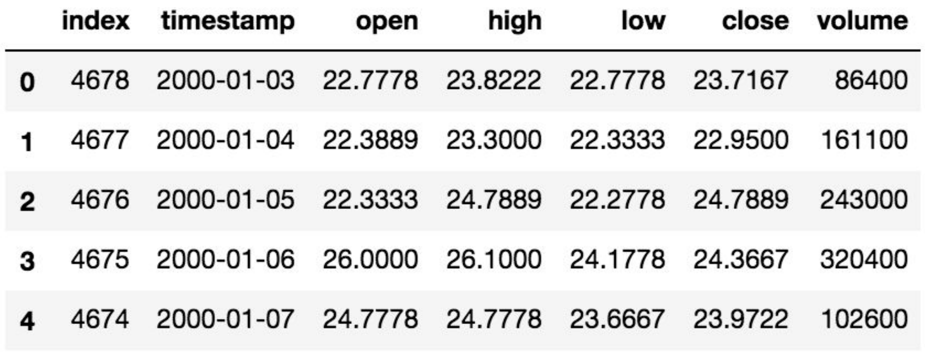
The evaluation metric that we will be using in this project is root-mean-square error (RMSE). The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) (or sometimes root-mean-squared error) is a frequently used measure of the differences between values (sample values) predicted by a model or an estimator and the values observed.

$$\mathrm{RMSD}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{\mathrm{E}((\hat{\theta} - \theta)^2)}.$$

RMSD is always non-negative, and a value of 0 (never achieved in practice) would indicate a perfect fit to the data. In general, a lower RMSD is better than a higher one. Also, this metric will allow to compare the model's performance over single data points and very large subsets of data equally well.

Data Exploration

The dataset used is taken from alphavantage.co . The query for data is constructed using two variables i.e. ticker and exchange. Ticker is the code name of the stock we need to get the data for (example TECHM for Tech Mahindra), exchange is the name of the exchange we want to fetch data for (example NSE for National Stock Exchange). The above example will return us the data for Tech Mahindra separated at day intervals for all the years. For this project we will use the data for ONGC (Oil and Natural Gas Limited) from NSE (National Stock Exchange). The data will cover stock prices from 1998-03-24 to 2018-08-07 in INR.

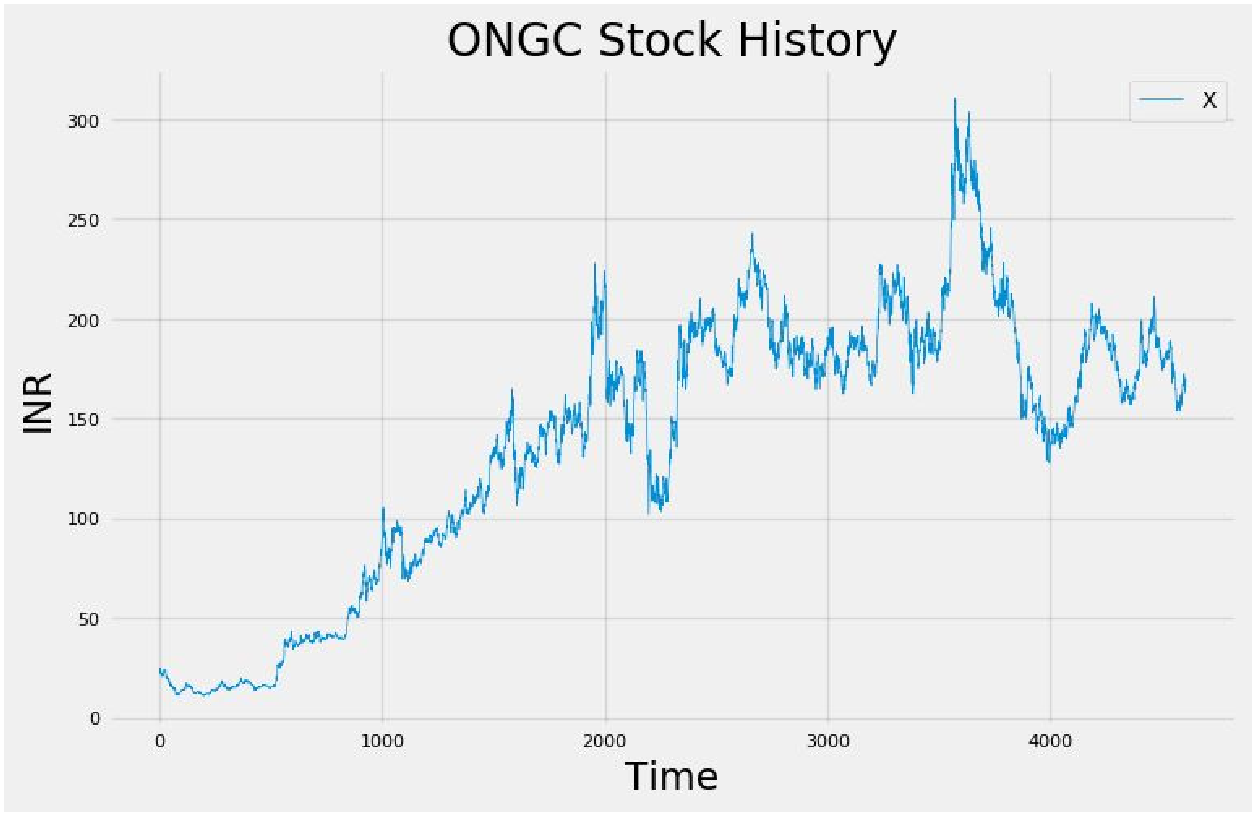


	index	timestamp	open	high	low	close	volume
0	4678	2000-01-03	22.7778	23.8222	22.7778	23.7167	86400
1	4677	2000-01-04	22.3889	23.3000	22.3333	22.9500	161100
2	4676	2000-01-05	22.3333	24.7889	22.2778	24.7889	243000
3	4675	2000-01-06	26.0000	26.1000	24.1778	24.3667	320400
4	4674	2000-01-07	24.7778	24.7778	23.6667	23.9722	102600

The response for the above data call contains the date, open, high, low, close and volume for the stock. There are total of 4678 data points as of 2018-08-21 for ONGC stock. Open is the opening price of the stock on that day. High is the highest price of the stock on that day. Low is the lowest price of the stock on that day. Close is the closing price for the stock on that day. Volume is the volume of the stock traded on that day.

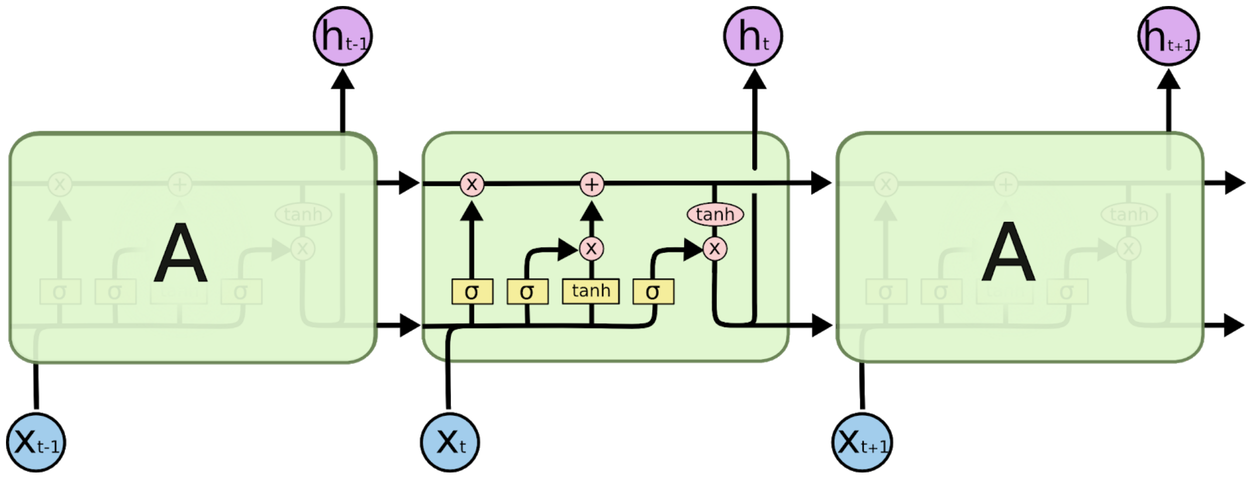
Exploratory Visualization

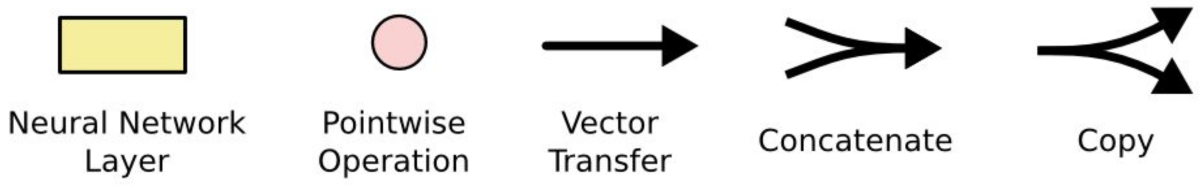
The plot below shows the Adj. Close price of the stock in Indian National Rupees from 1998 to 2018.



Algorithms and Techniques

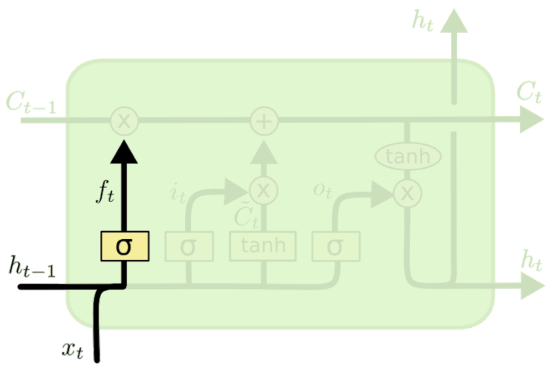
For this project we will use LSTM model. A typical LSTM network is comprised of different memory blocks called cells. There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates.





Forget gate

Forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter. This is required for optimizing the performance of the LSTM network.

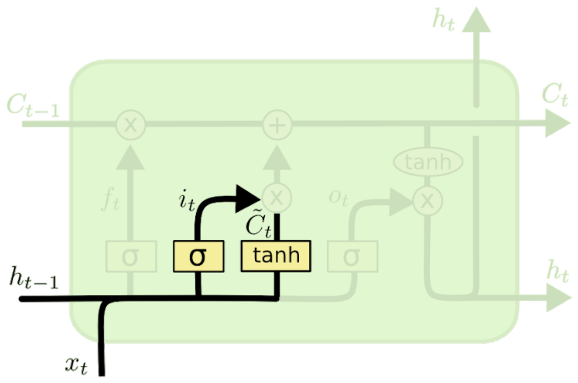


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

h_t-1 is the hidden state from the previous cell or the output of the previous cell and x_t is the input at that particular time step. The given inputs are multiplied by the weight matrices and a bias is added. After this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. If a ‘0’ is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely. Similarly, a '1' means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

Input gate

The input gate is responsible for the addition of information to the cell state. First it regulates what values need to be added to the cell state by involving a sigmoid function.

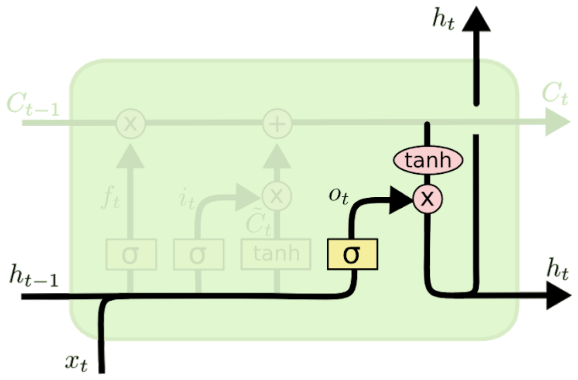


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

This is similar to the forget gate and acts as a filter for all the information from h_t-1 and x_t. Then it creates a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state. This is done using the tanh function, which outputs values from -1 to +1. Lastly, the value of the regulatory filter (the sigmoid gate) is multiplied to the created vector (the tanh function) and then this information is added to the cell state via addition operation.

Output gate

The output gate selects useful information from the current cell state and show it as an output. It creates a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Then it makes a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function. Lastly it multiplies the value of this regulatory filter to the vector created using the tanh function, and sending it out as a output along with to the hidden state of the next cell.

Benchmark

The benchmark model used for this problem statement will be Prophet model by facebook. Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit. The original solution model and benchmark model will both be evaluated by comparing the predicted price with the original price at the given dates to predict. One metric that we will use will be root mean squared error of the original and predicted values.

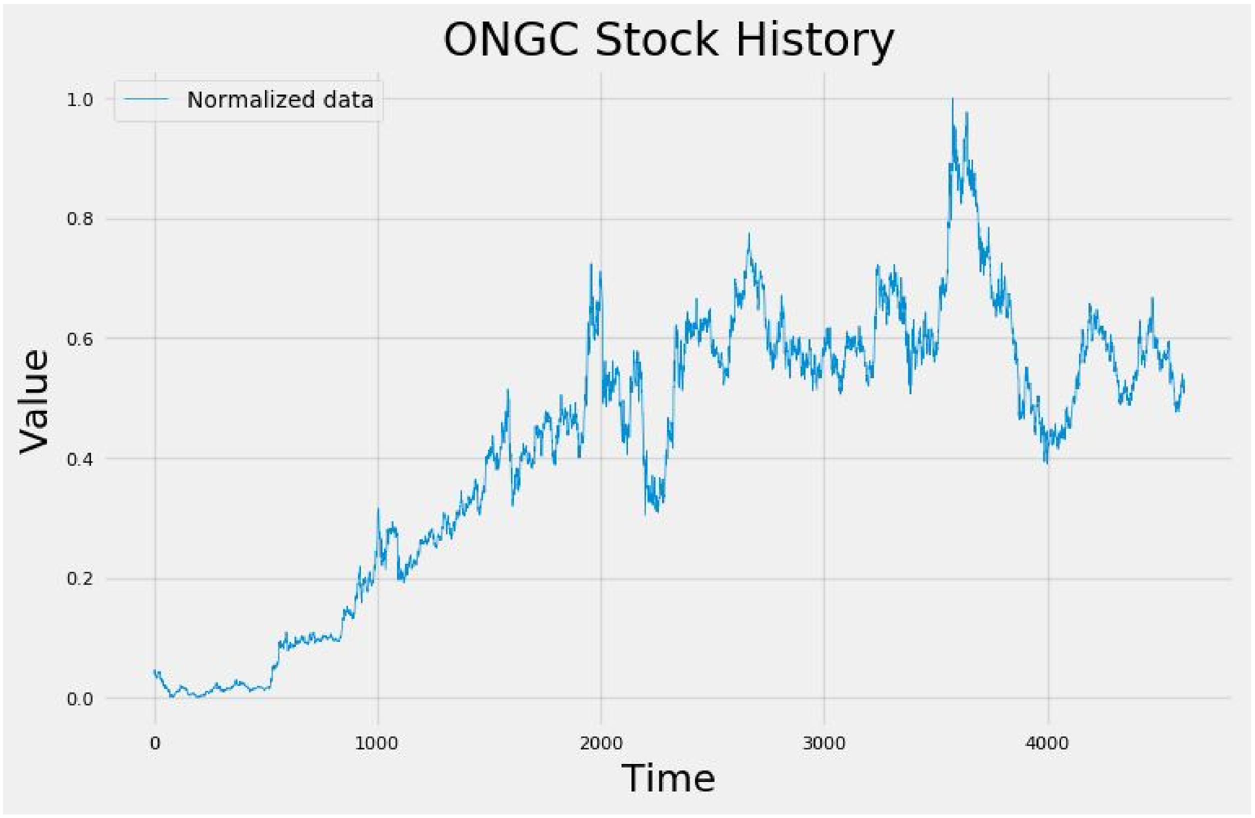
We made a simple prophet model and trained it with the same train data set as with our original model. The data set was split 75-25 as train and test data sets. Predictions were made on both the data sets and compared to the original value using the root mean squared error (RMSE). Following are the results for the prophet model we will compare our model with :

Train Score: 13.11 RMSE
Test Score: 39.45 RMSE

Data Preprocessing

We used the data from alphavantage.co (its a free api service for financial data). In this project we used data for ONGC from National Stock Exchange (NSE). The api call returns the data for ONGC from 2000-01-03 to 2018-08-21 with total of 4678 data points as of

2018-08-21. We will first check the data for nan values or zero values and remove those rows from our data frame so that the data is consistent and does not have outliers.



The data is then reversed and we reset the index of our data frame. LSTMs are sensitive to the scale of the input data, specifically when the sigmoid (default) or tanh activation functions are used. We rescale the data to the range of 0-to-1. This is also called normalizing. We will normalize the dataset using the MinMaxScaler preprocessing class from the scikit-learn library. The only feature we will use in this project is the close value of the stock.

Implementation

After we model our data and estimate the skill of our model on the training dataset, we need to get an idea of the skill of the model on new unseen data. For a normal classification or regression problem, we would do this using cross validation.

With time series data, the sequence of values is important. A simple method that we used is to split the ordered dataset into train and test datasets. We will calculate the index of the split point and separates the data into the training datasets with 75% of the observations that we can use to train our model, leaving the remaining 25% for testing the model.

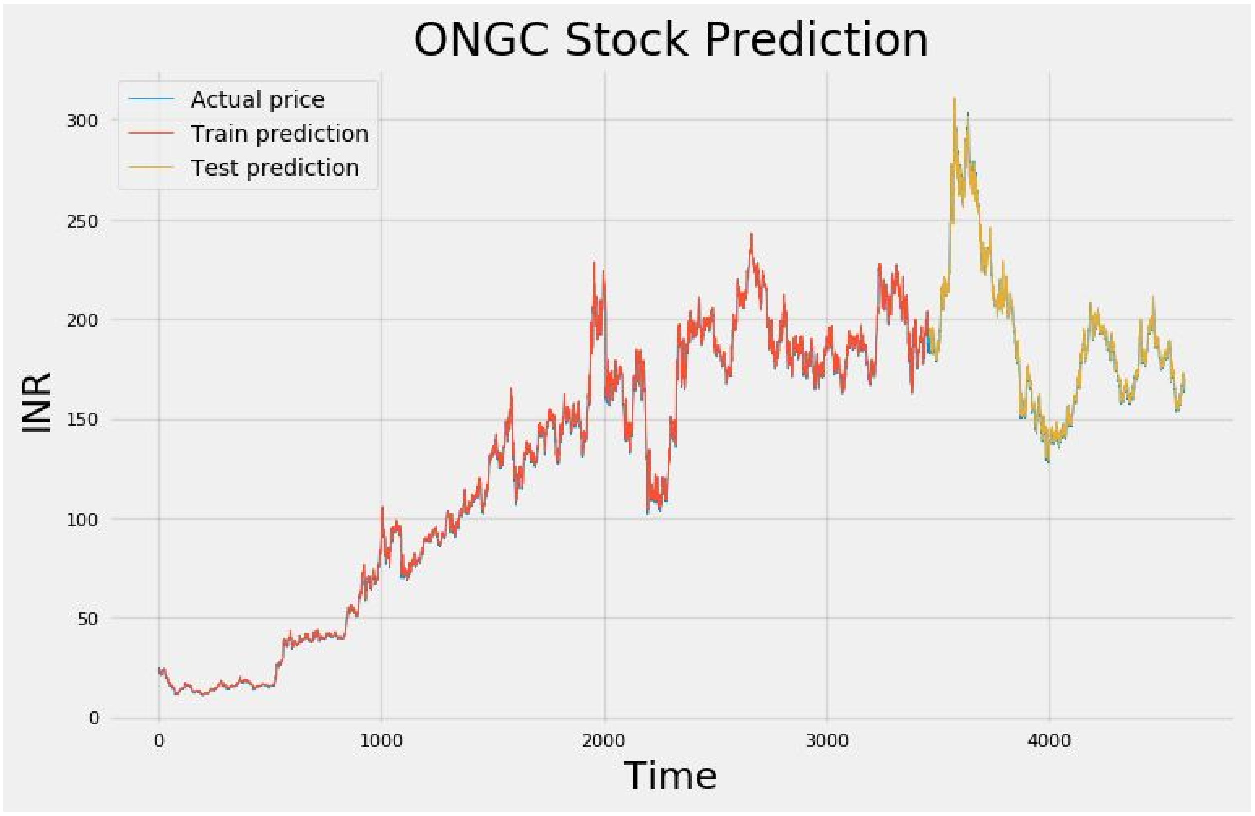
We create a create data set function that takes two arguments: the dataset, which is a NumPy array that we want to convert into a dataset, and the look_back, which is the number of previous time steps to use as input variables to predict the next time period, in this case defaulted to 1. This default will create a dataset where X is the price if the stock at a given time (t) and Y is the price of the stock at the next time (t + 1).

The LSTM network expects the input data (X) to be provided with a specific array structure in the form of: [samples, time steps, features]. Currently, the data is in the form: [samples, features] and we are framing the problem as one time step for each sample. We will transform the prepared train and test input data into the expected structure using numpy.reshape()

Then we build the LSTM network. The network has a visible layer with 1 input, two hidden layer with 256 and 128 LSTM blocks or neurons each and an output layer that makes a single value prediction. The linear activation function is used for the LSTM blocks. We fit the model with training data. The network is trained for 1 epochs and a batch size of 1 is used. We used 'mean_squared_error' as the loss function and 'adagrad' as optimizer.

Once the model is fit, we can estimate the performance of the model on the train and test datasets. This will give us a point of comparison for the benchmark model. We invert the predictions before calculating error scores to ensure that performance is reported in the same units as the original data (price of stock in INR).

Finally, we generate predictions using the model for both the train and test dataset to get a visual indication of the skill of the model. Because of how the dataset was prepared, we must shift the predictions so that they align on the x-axis with the original dataset. Once prepared, the data is plotted, showing the original dataset in blue, the predictions for the training dataset in red, and the predictions on the unseen test dataset in yellow.



Refinement

We adjusted the parameters for the models to acquire improved solution. The following parameters were explored and tuned during the process of finding the best parameters for the LSTM model :

1. Look Back : With different look back intervals for our model, the best suited options were 5 and 20. Looking at the loss convergence for both the values 5 seemed as a better value. The observations were made on both these values as the stock market is open 5 days a week or approx 20 days a month.
2. LSTM blocks or neurons : Observed improvement in model by increasing the number of blocks or neurons from 32 and 16 to 256 and 128 respectively.
3. Epochs : increasing the number of epochs from 1 to 100 improved the model.
4. Model optimizer : There were a lot of optimizers to test on like SGD, adam, adagrad etc. Observations were made for many different optimizers and adagrad seemed the best one to use for this model.

There was a huge improvement in both train and test score after making the above refinements the results are compared below :

Before :
Train Score: 5.00 RMSE
Test Score: 8.16 RMSE

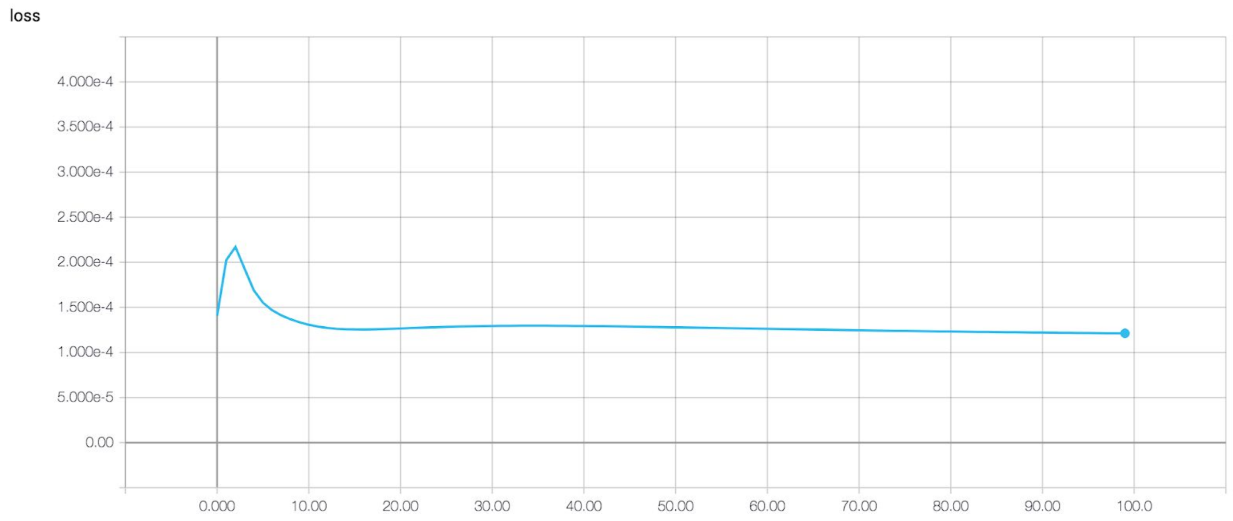
After :
Train Score: 4.29 RMSE
Test Score: 5.84 RMSE

Model Evaluation and Validation

The model has been trained on 75 percent of the total data i.e 4676 data points. The LSTM model is made using two LSTM layers with 256 and 128 neurons. It uses Adaptive Gradient (adagrad) as optimizer with default parameters and minimizes the mean square error loss.

```
Epoch 1/100
 - 22s - loss: 5.2961e-04
Epoch 2/100
 - 21s - loss: 2.2413e-04
Epoch 3/100
 - 21s - loss: 1.9420e-04
Epoch 4/100
 - 22s - loss: 1.7294e-04
Epoch 5/100
 - 22s - loss: 1.5683e-04
```

The figure below shows how the loss in minimized during the 100 epochs and by the end of 100 epochs the it is getting difficult for the model to minimize the loss.



The model is trained for 100 epochs and achieved a RMSE value of 4.29 on train data set and 5.84 on test data set. The test data set is completely new data for the model and RMSE value near to zero indicates that the model performed very well on the unseen data.

Justification

We made a simple prophet model and trained it with the same train data set as with our original LSTM model. Predictions were made on both train and test data sets and compared to the original values using the root mean squared error (RMSE). Following are the results for the prophet model compared with our LSTM model :

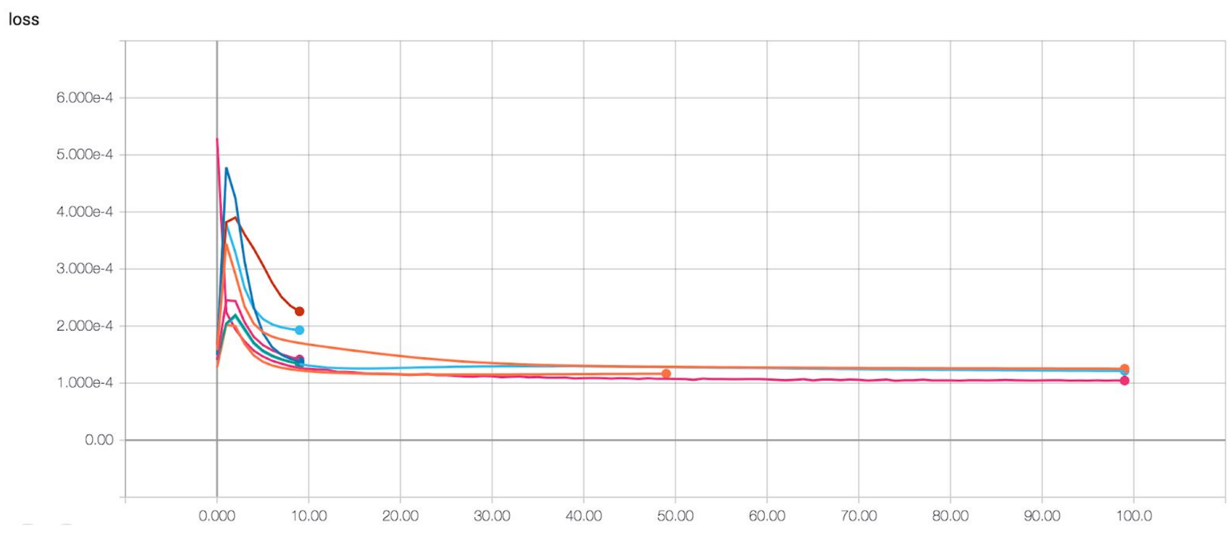
Prophet model :
Train Score: 13.11 RMSE
Test Score: 39.45 RMSE

LSTM model :
Train Score: 4.29 RMSE
Test Score: 5.84 RMSE

Looking at the RMSE scores, our LSTM model outperforms the prophet model by a huge factor in both train and test scores.

Free-Form Visualization

The graph below shows the mean square error loss for the model over different epochs.



The graph helped to visualize the loss of different configurations and parameters. It helped to pick the best parameters for tuning and thus training the model for minimizing RMSE on test data.

Reflection

The idea of the project was to predict the price of a given stock in future given the historic data of the stock. We were able to achieve the goal by building a LSTM network that can be trained on the historic data and give us a prediction for the coming day.

There were many challenges that I came across. Choosing the appropriate model for time series data was most important and crucial. Another challenging part was to prepare the data set such that it fits the input of our LSTM model and tuning the parameters for the model to minimize the loss.

Improvement

There are a lot of improvements the can be made on this model. To point out a few :

1. Model tweaking : There are still a lot of parameter settings the needs to be explored and can give significant improvements.
2. Another way to improve on the model will be having sentiment analysis of the stock in the market using Google Trends and provide that as a feature to the model.

References

- http://papers.nips.cc/paper/5956-scheduled-sampling-for-sequence-prediction-with-recurrent-neural-networks.pdf
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://facebook.github.io/prophet/
- https://en.wikipedia.org/wiki/Root-mean-square_deviation
- https://en.wikipedia.org/wiki/Long_short-term_memory
- https://en.wikipedia.org/wiki/Stock_market_prediction
- https://www.alphavantage.co/
- https://www.tensorflow.org/
- https://keras.io/