LNMIIT
The LNM Institute of Information Technology

LSTM: Stock Market Prediction

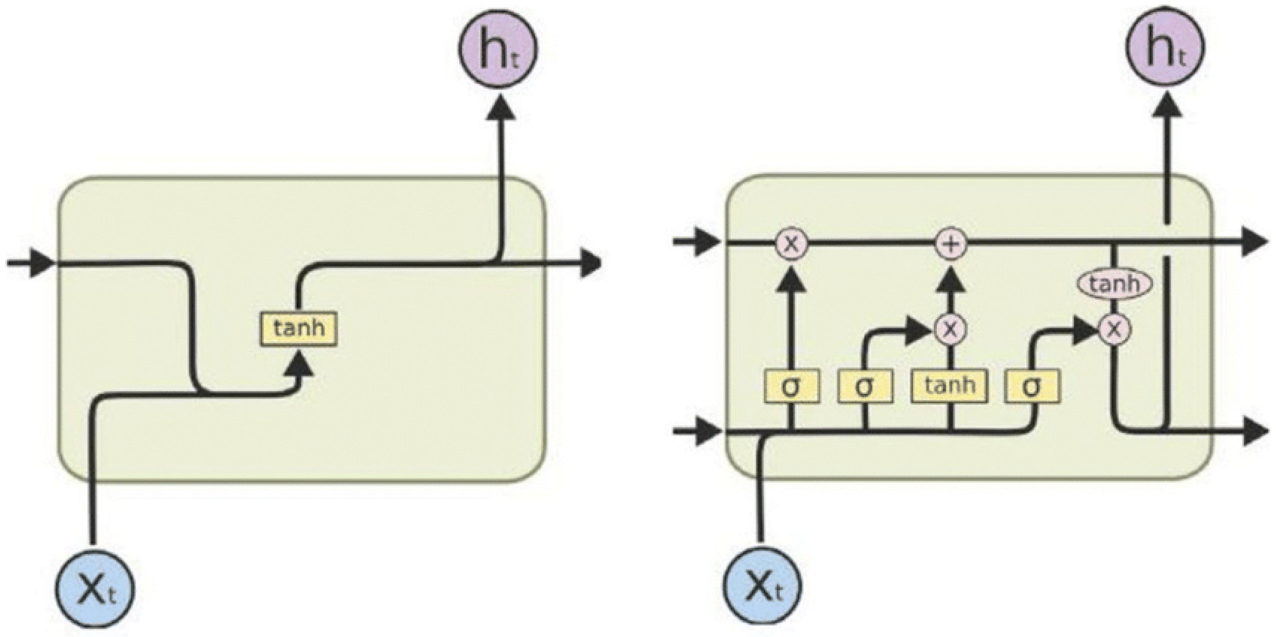
BTP REPORT

Samyak Goyal 18ucs205

Dushyant Gupta 18ucs175

Vatsal Mishra 18ucs060

Under Dr Animesh Chaturvedi



(a) RNN (b) LSTM

Overview

In this project we have used the concepts of **LSTM**(Long short term memory) and **RNN**(Recurrent Neural Networks) and tried to propose a model that predicts the opening price for a stock using the LSTM algorithm.

Introduction

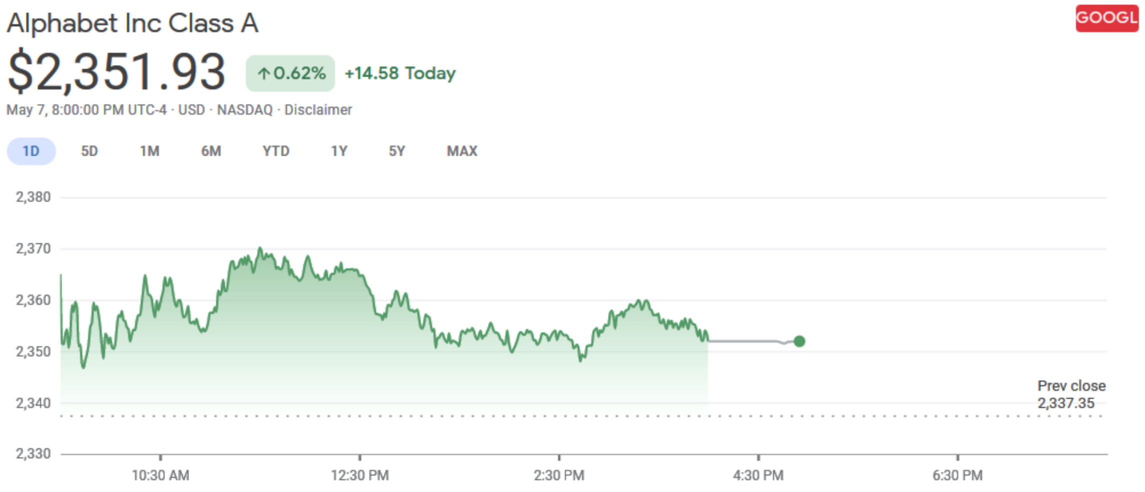
Stock market is a place where people buy/sell shares of publicly listed companies. It offers a platform to facilitate seamless exchange of shares. In simple terms, if A wants to sell shares of Reliance Industries, the stock market will help him to meet the buyer who is willing to buy Reliance Industries.

This project aims to build a machine learning model using the LSTM concept of RNN to predict the opening price for the Google stock for upcoming days based on the previous 90 days data for each day.

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.

LSTM: Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate.

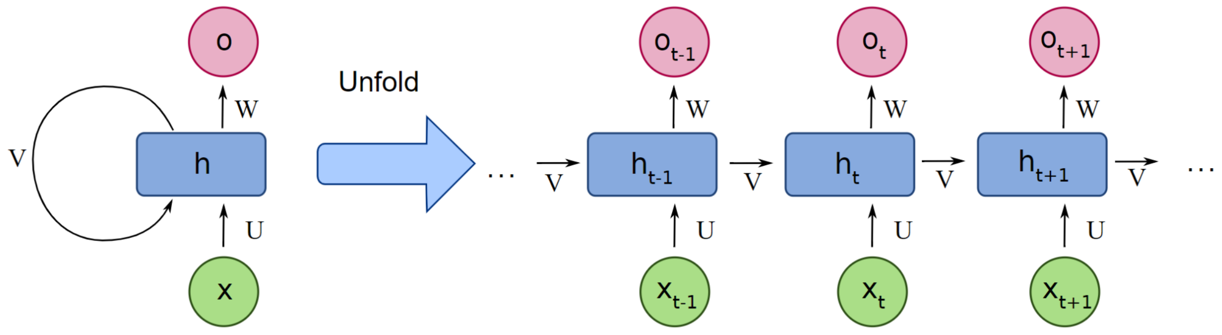
Graph for Google Stock Price



Algorithms Used

RNN

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs.

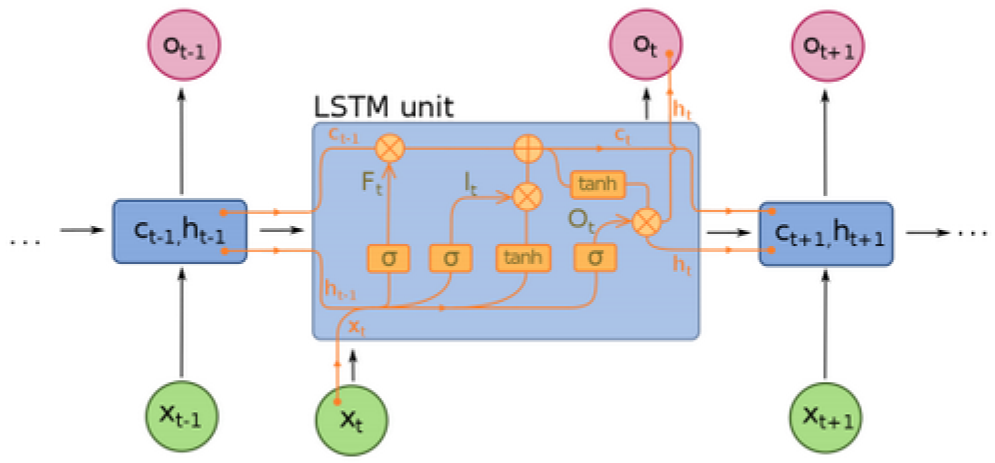


The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units. This is also called Feedback Neural Network (FNN).

LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.



In theory, classic (or "vanilla") RNNs can keep track of arbitrary long-term dependencies in the input sequences. The problem with vanilla RNNs is computational (or practical) in nature: when training a vanilla RNN using back-propagation, the gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers. RNNs using

LSTM units partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged. However, LSTM networks can still suffer from the exploding gradient problem.

Data Used

- We have imported our dataset from Yahoo Finance
    - (https://in.finance.yahoo.com/).
- We have used the stock of Google(symbol: GOOG) to train our model
    - (https://in.finance.yahoo.com/quote/GOOG?p=GOOG).

The parameters that were imported were:

- ★ Date: The date of the instance of the stock
- ★ Open: The opening price of the stock for that particular day
- ★ High: The highest price achieved by the stock for that particular day
- ★ Low: The lowest price achieved by the stock for that particular day
- ★ Close: The closing price of the stock for that particular day
- ★ Volume: The volume of the stock

The Dataset imported was from 20th August 2004 to 31st December 2020. The downloaded csv file can be seen here:

https://github.com/samyakgoyal/Stock-Prediction-Using-LSTM/blob/main/GOOG.csv

We divided the dataset in chunks of 90 days and then trained the model on them. The training was done on the data from 20th August 2004 to 31st December 2018, the testing was done on the data from 1st January 2019 to 1st January 2021.

First of all, the columns Date and Adj close were dropped since they were not providing any additional information. Then the remaining features were scaled using MinMaxScaler imported from sklearn.preprocessing. Then the dataset was broken into chunks of 90 days and then the model was trained to predict the opening price for the stock.

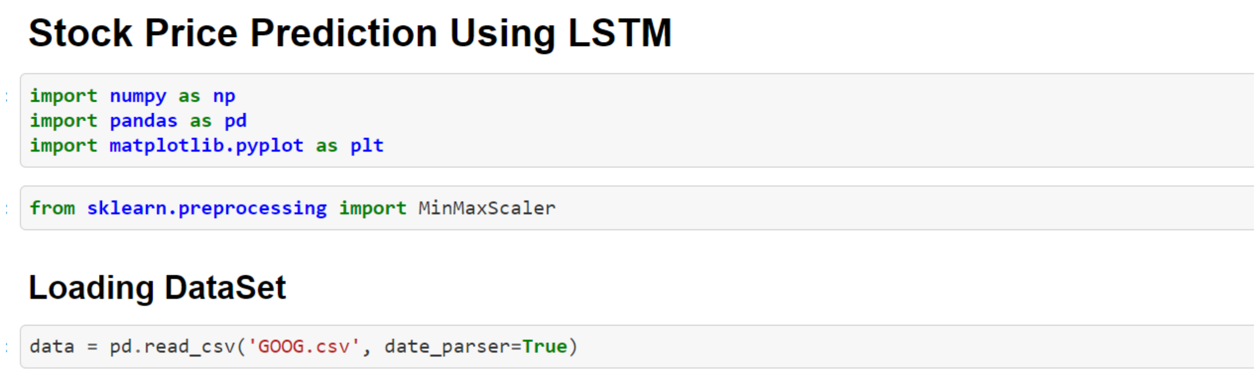
Code and Model

First of all the following libraries were imported:

Numpy and pandas : For mathematical and Array manipulation

Matplotlib.pyplot : To plot the graph

Then the dataset was loaded



Stock Price Prediction Using LSTM

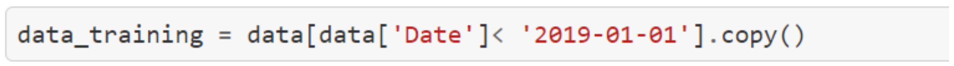
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import MinMaxScaler
```

Loading DataSet

```
data = pd.read_csv('GOOG.csv', date_parser=True)
```

Splitting the dataset into training and test set



```
data_training = data[data['Date']< '2019-01-01'].copy()
```

```
data_test = data[data['Date']>= '2019-01-01'].copy()
data_test
```

Dropping the columns that were not adding any value

```
training_data = data_training.drop(['Date', 'Adj Close'], axis=1)
training_data
```

Feature Scaling the data to normalize the range of independent variables

```
sc = MinMaxScaler(feature_range = (0, 1))
training_data_scaled = sc.fit_transform(training_data)
```

Creating Data with Timestamps: Because our model is trained on the previous 90 days data to predict the upcoming stock price.

```
X_train = []
y_train = []
for i in range(90, training_data_scaled.shape[0]):
    X_train.append(training_data_scaled[i-90:i, 0])
    y_train.append(training_data_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

Building the LSTM model

First of all the Sequential model was imported from tensorflow.keras because LSTM is a sequential model and we need to combine our layers in a sequential order.

Dense and LSTM layers were imported from tensorflow.keras because the LSTM layers provide the basic block for the LSTM model and the Dense layer is used at the end to summarize the model.

Then Dropout rate is also imported to avoid the case of underfitting / overfitting.

First of all the Sequential model was defined, then 4 LSTM layers were added.

Then Different units are used to vary the layers, the activation function is chosen as ‘relu’ because it provides the best result, also the Dropout is used to avoid the case of overfitting.

Then a Dense layer was added.

At last the model was compiled using ‘Adam’ optimiser(provided best result) and the loss was calculated as ‘mean squared error’ (provided the best insight).

Then our dataset was fit into the model and the model was trained for 11 epochs with batch size 32.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
```

```
regressor = Sequential()

regressor.add(LSTM(units = 100, activation='relu', return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.25))

regressor.add(LSTM(units = 140, activation='relu', return_sequences = True))
regressor.add(Dropout(0.35))

regressor.add(LSTM(units = 90, activation='relu', return_sequences = True))
regressor.add(Dropout(0.35))

regressor.add(LSTM(units = 120))
regressor.add(Dropout(0.25))

regressor.add(Dense(units = 1))
```

```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regressor.fit(X_train, y_train, epochs = 11, batch_size = 32 )
```

Test Dataset was prepared:

```
past_90_days=data_training.tail(90)
```

```
df= past_90_days.append(data_test, ignore_index= True )
```

```
df= df.drop(['Date', 'Adj Close'], axis=1)
```

```
inputs= sc.transform(df)
```

```
X_test = []
y_test = []
for i in range(90, inputs.shape[0]):
    X_test.append(inputs[i-90:i, 0])
    y_test.append(inputs[i, 0])
X_test, y_test = np.array(X_test), np.array(y_test)
```

```
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
y_pred = regressor.predict(X_test)
```

This step was added to revert back the Feature scaling for plotting the Graph.

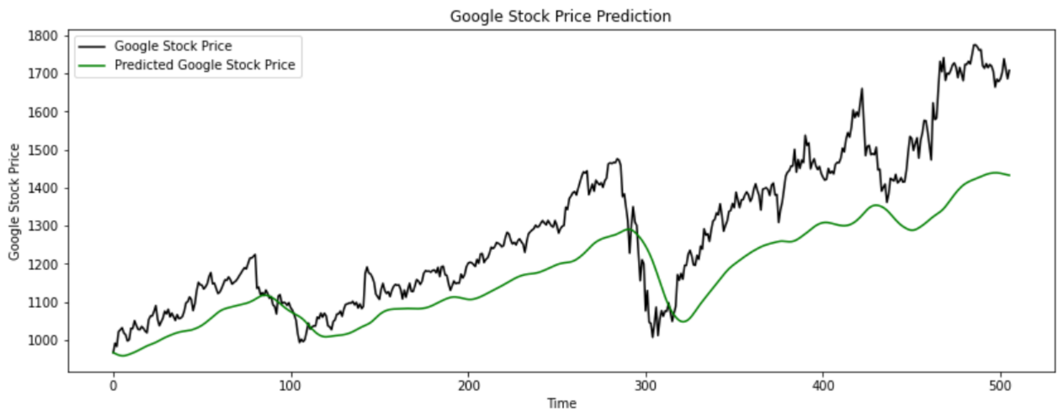
```
scale=1/sc.scale_[0]
```

```
y_pred= y_pred*scale
y_test= y_test*scale
```

At last the graph was plotted to compare the accuracy of the model:

The black line depicts the actual cost and the green line depicts the cost of the stock predicted by our model.

```
plt.figure(figsize=(14,5))
plt.plot(y_test, color = 'black', label = 'Google Stock Price')
plt.plot(y_pred, color = 'green', label = 'Predicted Google Stock Price')
plt.title('Google Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```

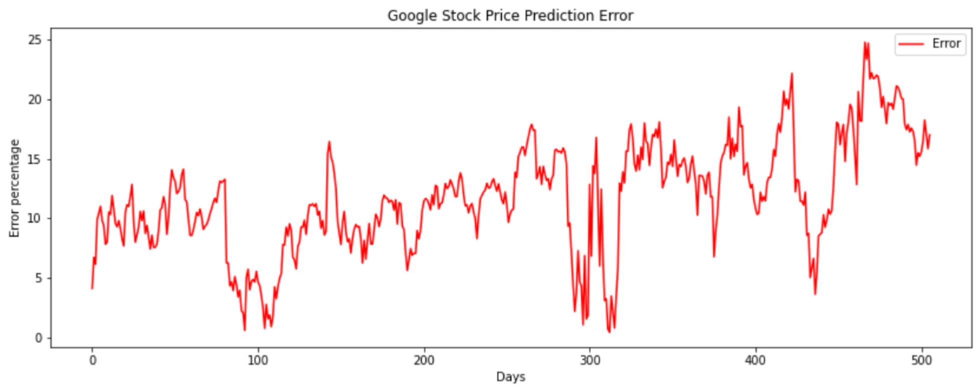


Then the error was calculated through averaging the difference between predicted and actual values

```
error=[]
sum_of_error=0
for i in range(len(y_pred)):
    error_percentage=100*abs(y_pred[i][0]-y_test[i])/y_test[i]
    error.append(error_percentage)
    sum_of_error+=error_percentage
avg_error=sum_of_error/len(y_pred)
avg_error
```

The last step was to plot the graph of error percentage for better visualization

```
plt.figure(figsize=(14,5))
plt.plot(error, color = 'red', label = 'Error')
plt.title('Google Stock Price Prediction Error')
plt.xlabel('Days')
plt.ylabel('Error percentage')
plt.legend()
plt.show()
```



The model can be found here:

Google collab: https://colab.research.google.com/drive/1JQWRwMvq8pG2t72Q-CFoD6zn0ur7Bil2?authuser=1

Github Repository:

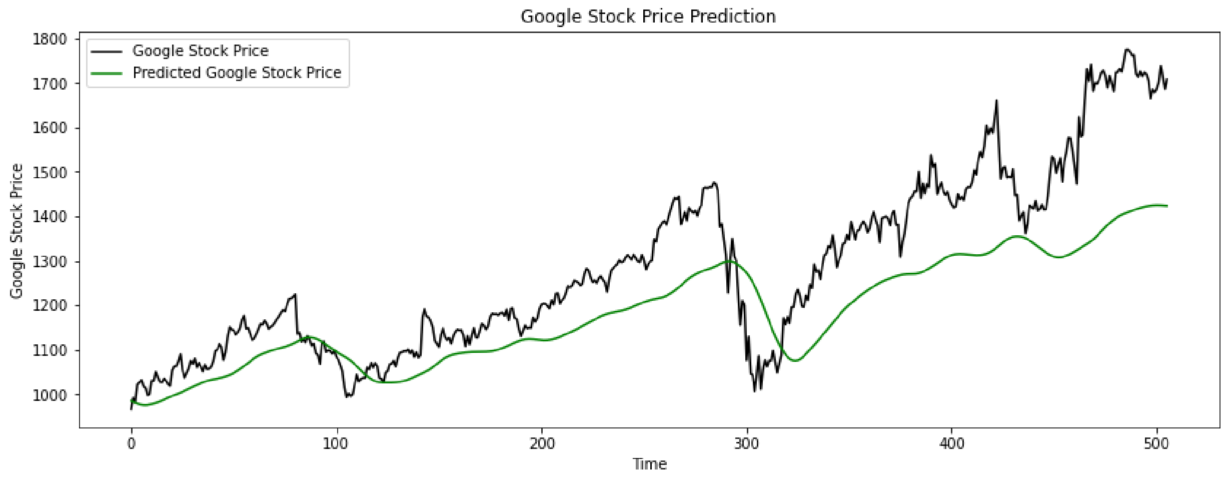
https://github.com/samyakgoyal/Stock-Prediction-Using-LSTM/blob/main/Final%20Improved.ipynb

Findings

While building the regressor for our model, we have to set some parameters like the number of epochs to train our model, The units which are used to vary the layers in LSTM, The Dropout rate which is set for managing underfitting / overfitting.

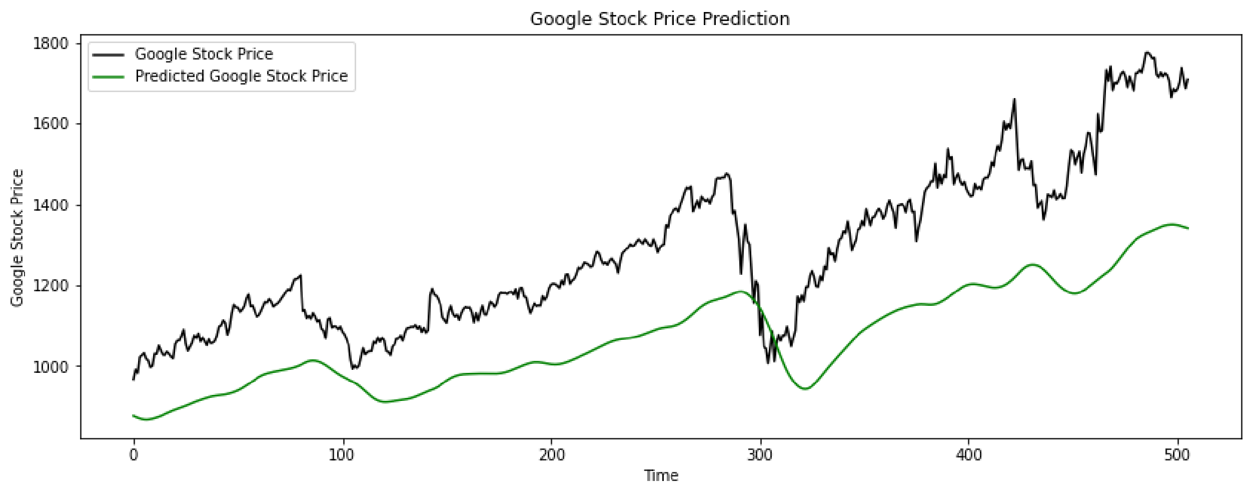
Now we tweaked these values to get the optimum results (Least error percentage).

Epoch	10
1st layer unit	50
Dropout Rate	0.2
2nd layer unit	70
Dropout Rate	0.3
3rd layer unit	90
Dropout Rate	0.4
4th layer unit	120
Dropout Rate	0.2
Avg. Error %	15.59%



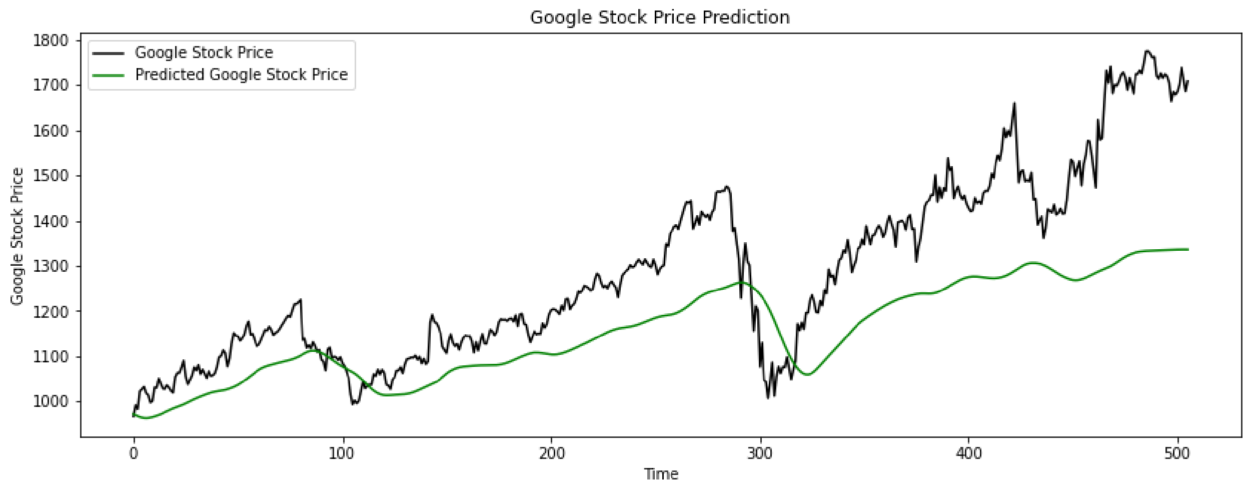
First we change the units:

Epoch	10
1st layer unit	100
Dropout Rate	0.2
2nd layer unit	70
Dropout Rate	0.3
3rd layer unit	90
Dropout Rate	0.4
4th layer unit	120
Dropout Rate	0.2
Avg. Error %	16.43%



Then we increased Epochs for better results:

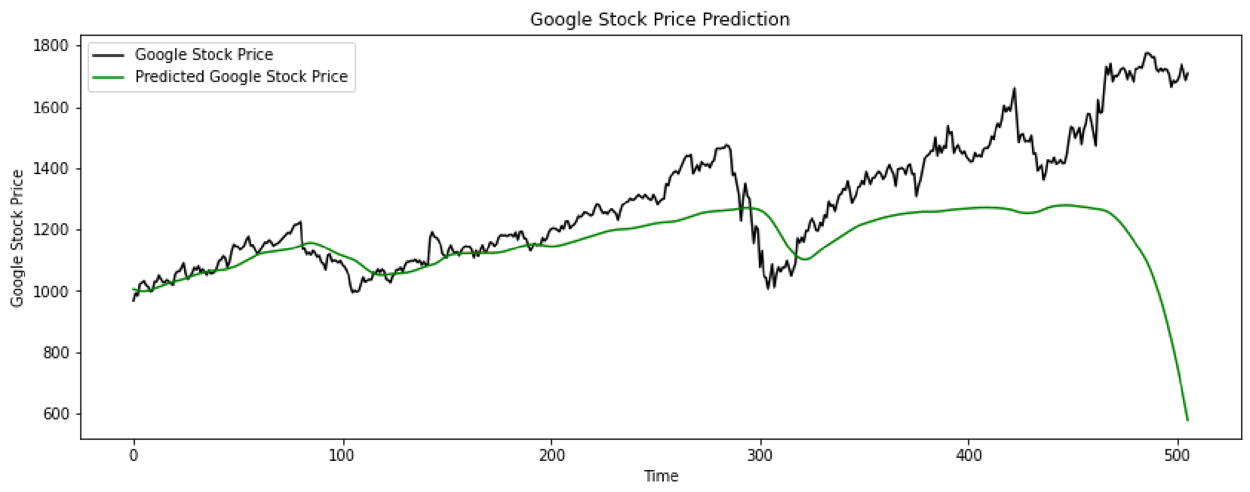
Epoch	11
1st layer unit	100
Dropout Rate	0.2
2nd layer unit	70
Dropout Rate	0.3
3rd layer unit	90
Dropout Rate	0.4
4th layer unit	120
Dropout Rate	0.2
Avg. Error %	10.05%



Further increasing the Epochs:

Epoch	25
1st layer unit	100
Dropout Rate	0.2
2nd layer unit	70
Dropout Rate	0.3
3rd layer unit	90
Dropout Rate	0.4
4th layer unit	120
Dropout Rate	0.2

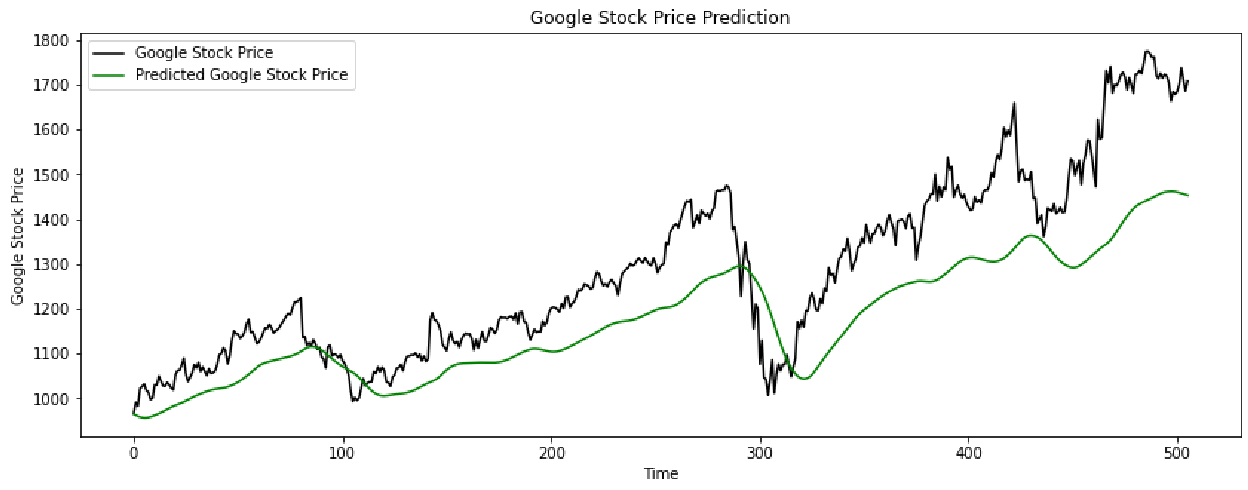
Avg. Error %	9.83%



Here we see models with higher epochs have too much deviation with respect to larger intervals. Now fixing epochs =11 , we now change the Dropout Rates:

Epoch	11
1st layer unit	100
Dropout Rate	0.2
2nd layer unit	140
Dropout Rate	0.3
3rd layer unit	90
Dropout Rate	0.4
4th layer unit	120

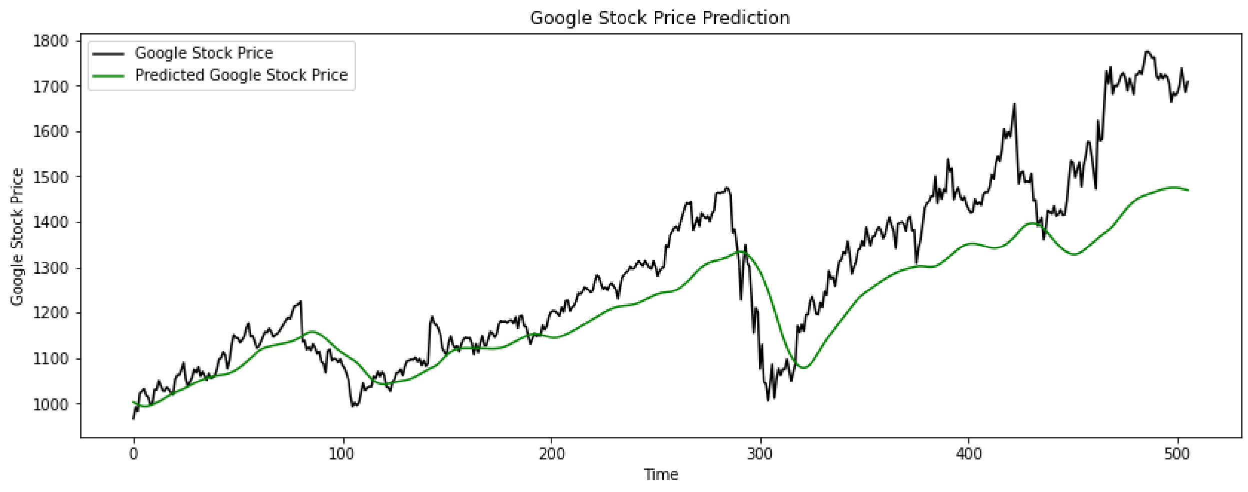
Dropout Rate	0.2
Avg. Error %	8.87%

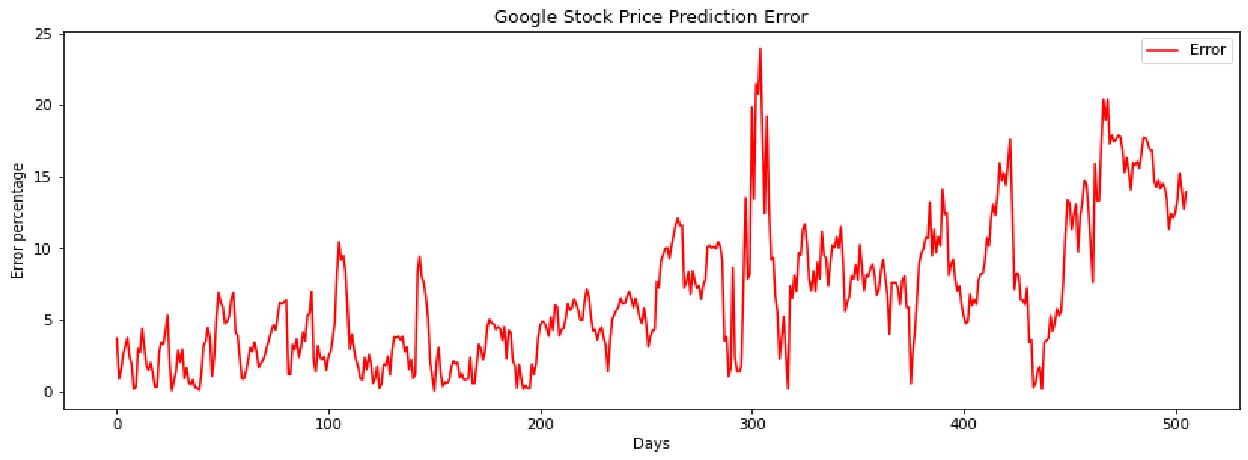


Now with the changed dropout rates, we see the average error is just 6.59%.

Epoch	11
1st layer unit	100
Dropout Rate	0.25
2nd layer unit	140
Dropout Rate	0.35
3rd layer unit	90
Dropout Rate	0.35
4th layer unit	120

Dropout Rate	0.25
Avg. Error %	6.59%





This model without any observable underitting / overitting gives only 6.59 % average error rate or **93.41% Accuracy result.**

More of the experimented models with various parameters combinations can be found here.

Resources

https://in.finance.yahoo.com/

https://www.youtube.com/watch?v=5tvmMX8r_OM&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI

https://in.finance.yahoo.com/quote/GOOG?p=GOOG

https://paperswithcode.com/method/memory-network , https://arxiv.org/pdf/1410.3916v11.pdf

https://en.wikipedia.org/wiki/Recurrent_neural_network

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce https://en.wikipedia.org/wiki/Long_short-term_memory

https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://keras.io/api/layers/recurrent_layers/lstm/\

https://www.tutorialspoint.com/python/index.htm