**STOCK PRICE PREDICTION USING**

**DEEP LEARNING**

TEAM MEMBER

**621721243040 : PRAVENRAJ D V**

**Phase 5 Submission document**

**Project Title**: **Stock Price Prediction**

**Phase 5**: **Project Documentation & Submission**

**Topic**: **In this section we will document the complete project and prepare it for submission.**

**WEBSITE TRAFFIC ANALYSIS**

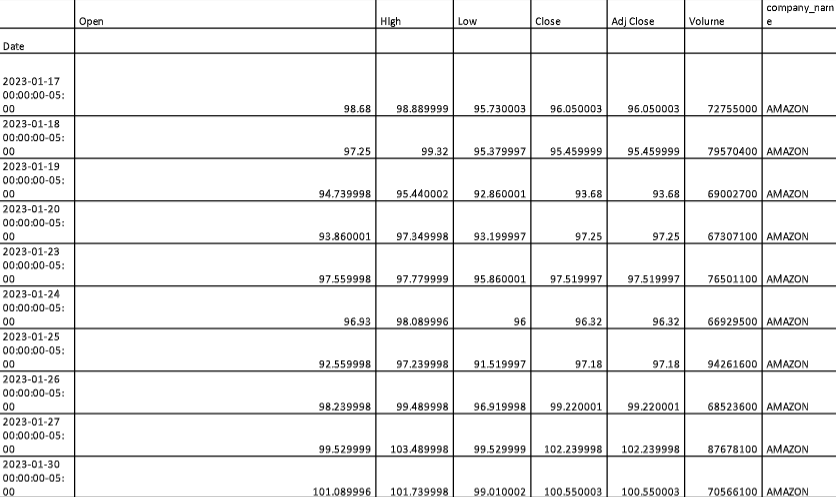
**INTRODUCTION:**

* Stock market is considered chaotic, complex, volatile and dynamic. Undoubtedly, its prediction is one of the most challenging tasks in time series forecasting. Moreover existing Artificial Neural Network (ANN) approaches fail to provide encouraging results.
* Meanwhile advances in machine learning have presented favourable results for speech recognition, image classification and language processing. Methods applied in digital signal processing can be applied to stock data as both are time series. Similarly, learning outcome of this paper can be applied to speech time series data.
* Deep learning for stock prediction has been introduced in this paper and its performance is evaluated on Google stock price multimedia data (chart) from NASDAQ.
* The objective of this paper is to demonstrate that deep learning can improve stock market forecasting accuracy. For this, (2D)2PCA + Deep Neural Network (DNN) method is compared with state of the art method 2-Directional 2-Dimensional Principal Component Analysis (2D)2PCA + Radial Basis Function Neural Network (RBFNN).

* It is found that the proposed method is performing better than the existing method RBFNN with an improved accuracy of 4.8% for Hit Rate with a window size of 20.
* Also the results of the proposed model are compared with the Recurrent Neural Network (RNN) and it is found that the accuracy for Hit Rate is improved by 15.6%. The correlation coefficient between the actual and predicted return for DNN is 17.1% more than RBFNN and it is 43.4% better than RNN.

Dataset Link: <https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset>

**Given Dataset:**



**Here's a list of tools and software commonly used in the process:**

**1.Office/Productivity Tools:**

* Microsoft Office (Word, Excel, PowerPoint, Outlook)
* Google Workspace (Docs, Sheets, Slides, Gmail)
* LibreOffice
* Apache OpenOffice

**2.Project Management:**

* Trello
* Asana
* Jira
* Monday.com
* Microsoft Project

**3.Collaboration and Communication:**

* Slack
* Microsoft Teams
* Zoom
* Skype
* Discord

**4.Design and Multimedia:**

* Adobe Creative Cloud (Photoshop, Illustrator, InDesign, Premiere Pro)
* Canva
* GIMP (GNU Image Manipulation Program)
* Sketch
* CorelDRAW

**5.Development and Programming:**

* Visual Studio Code
* IntelliJ IDEA
* Git (Version Control)
* GitHub
* Sublime Text

**6.Data Analysis and Visualization:**

* Tableau
* Power BI
* RStudio
* Python (with libraries like NumPy, pandas, Matplotlib)
* Excel (for basic data analysis)

**7.CAD and 3D Modeling:**

* AutoCAD
* SolidWorks
* Blender
* Rhino
* Tinkercad

**8.Web Development:**

* HTML/CSS editors (e.g., Visual Studio Code, Sublime Text)
* Adobe Dreamweaver
* WordPress
* Bootstrap
* Wix

**9.Video Editing and Production:**

* Adobe Premiere Pro
* Final Cut Pro
* iMovie
* DaVinci Resolve
* Camtasia

**10.3D Printing and Modeling:**

* Ultimaker Cura
* Meshmixer
* MatterControl
* Simplify3D

**11.Simulation and Modeling:**

* MATLAB
* Simulink
* COMSOL
* ANSYS
* SolidWorks Simulation

**12.Gaming Development:**

* Unity
* Unreal Engine
* Godot Engine
* GameMaker Studio
* RPG Maker

**13.Cybersecurity and Network Monitoring:**

* Wireshark
* Nmap
* Metasploit
* Snort
* SolarWinds

**14.Data Science and Machine Learning:**

* Python (with libraries like TensorFlow, scikit-learn)
* Jupyter Notebook
* R
* KNIME
* RapidMiner

**15.Content Management Systems (CMS):**

* WordPress
* Joomla
* Drupal
* Magento (for e-commerce)
* Shopify (for e-commerce)

**16.Customer Relationship Management (CRM):**

* Salesforce
* HubSpot
* Zoho CRM
* Microsoft Dynamics 365
* Pipedrive

**17.Accounting and Financial Software:**

* QuickBooks
* Xero
* FreshBooks
* Wave
* SAP Business One

**18.Human Resources and HCM:**

* Workday
* BambooHR
* ADP Workforce Now
* Zenefits
* Gusto

**19.Video Conferencing and Webinar:**

* Zoom
* WebEx
* GoToMeeting
* Adobe Connect
* BigBlueButton

**20. Customer Support and Helpdesk:**

* Zendesk
* Freshdesk
* Intercom
* Help Scout
* Jira Service Management

1.DESIGN THINKING AND PRESENT IN FORMOF DOCUMENT

**MY UNDERSTANDING :**

My understanding on this project **“STOCK PRICE PREDICTION”** involves using historical data, market analysis, and various models to forecast future stock price movements. It aids investors and traders in making informed decisions about buying or selling stocks.

**AIM OF MY PROJECT :**

The primary aims of stock price prediction are to assist users in making investment decisions, managing risk, and enhancing trading strategies.

**OBJECTIVES OF MY PROJECT:**

1. Audience Understanding
2. Facilitate Informed Investment Decisions
3. Risk Mitigation
4. Optimized Portfolio Management
5. Long-Term Planning
6. Enhance Trading Strategies
7. Improve Returns
8. Research and Insights

**AUDIENCE UNDERSTANDING:**

Stock price predictions to your audience's knowledge level, whether they are individual investors, professional traders, or policymakers.

**FACILITATE INFORMED INVESTMENT DECISIONS:**

The primary objective is to provide investors with insights and information to make well-informed decisions regarding buying, holding, or selling stocks.

**RISK MITIGATION:**

By predicting stock price movements, investors can identify potential risks and take measures to mitigate them, reducing the likelihood of financial losses.

**OPTIMIZE PORTFOLIO MANAGEMENT:**

Stock price predictions aid in optimizing investment portfolios by helping investors allocate assets effectively to achieve their financial goals.

**ENHANCE TRADING STRATEGIES**:

Traders use stock price predictions to refine their trading strategies, enabling them to make more precise entries and exits in the market.

**IMPROVE RETURNS**:

By making accurate predictions, investors and traders aim to improve the overall returns on their investments and trading activities.

**LONG-TERM PLANNING**:

Investors can use predictions to make long-term financial plans, such as retirement planning or saving for major life events.

**RESEARCH AND INSIGHTS**:

Analysts and researchers aim to gain insights into market dynamics, helping them understand the forces driving stock price movements.

2.DESIGN INTO INNOVATION

**Introduction:**

* Stock price prediction is a complex field with no guaranteed accuracy. Many factors can impact stock prices that are difficult to quantify accurately.
* Therefore, investors should exercise caution and use stock price predictions as a tool in conjunction with other research and analysis when making investment decisions.
* Predicting stock prices is a complex and challenging task, often relying on various data sources, algorithms and models. Stock price prediction involves forecasting the future value of a particular stock or a stock market index.
* It is important to note that stock price prediction is inherently uncertain, as stock markets are influenced by a multitude of factors, including economic conditions, company performance, investor sentiment and geopolitical.

**Content for Project Phase 2:**

Consider exploring more advanced deep learning techniques like CNN-LSTM or attention mechanisms for improved accuracy in predicting stock prices

**Data source:**

A good data source for stock price prediction using machine learning and artificial intelligence should be Accurate, Complete the geographic area of interest, Accessible.

**Dataset Link:** <https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Open | High | Low | Close | Adj Close | Volume |
| 09-08-2019 | 138.61 | 139.38 | 136.46 | 137.71 | 136.7875 | 23466700 |
| 12-08-2019 | 137.07 | 137.86 | 135.24 | 135.79 | 134.8804 | 20476600 |
| 13-08-2019 | 136.05 | 138.8 | 135 | 138.6 | 137.6716 | 25154600 |
| 14-08-2019 | 136.36 | 136.92 | 133.67 | 133.98 | 133.5256 | 32527300 |
| 15-08-2019 | 134.39 | 134.58 | 132.25 | 133.68 | 133.2267 | 28074400 |
| 16-08-2019 | 134.88 | 136.46 | 134.72 | 136.13 | 135.6684 | 24449100 |
| 19-08-2019 | 137.85 | 138.55 | 136.89 | 138.41 | 137.9406 | 24355700 |
| 20-08-2019 | 138.21 | 138.71 | 137.24 | 137.26 | 136.7945 | 21170800 |
| 21-08-2019 | 138.55 | 139.49 | 138 | 138.79 | 138.3193 | 14970300 |
| 22-08-2019 | 138.66 | 139.2 | 136.29 | 137.78 | 137.3128 | 18697000 |
| 23-08-2019 | 137.19 | 138.35 | 132.8 | 133.39 | 132.9377 | 38508600 |
| 26-08-2019 | 134.99 | 135.56 | 133.9 | 135.45 | 134.9907 | 20312600 |
| 27-08-2019 | 136.39 | 136.72 | 134.66 | 135.74 | 135.2797 | 23102100 |
| 28-08-2019 | 134.88 | 135.76 | 133.55 | 135.56 | 135.1003 | 17393300 |
| 29-08-2019 | 137.25 | 138.44 | 136.91 | 138.12 | 137.6516 | 20168700 |
| 30-08-2019 | 139.15 | 139.18 | 136.27 | 137.86 | 137.3925 | 23940100 |
| 03-09-2019 | 136.61 | 137.2 | 135.7 | 136.04 | 135.5787 | 18869300 |
| 04-09-2019 | 137.3 | 137.69 | 136.48 | 137.63 | 137.1633 | 17995900 |
| 05-09-2019 | 139.11 | 140.38 | 138.76 | 140.05 | 139.5751 | 26101800 |
| 06-09-2019 | 140.03 | 140.18 | 138.2 | 139.1 | 138.6283 | 20824500 |
| 09-09-2019 | 139.59 | 139.75 | 136.46 | 137.52 | 137.0537 | 25773900 |
| 10-09-2019 | 136.8 | 136.89 | 134.51 | 136.08 | 135.6185 | 28903400 |
| 10-10-2019 | 138.49 | 139.67 | 138.25 | 139.1 | 138.6283 | 17654600 |
| 11-10-2019 | 140.12 | 141.03 | 139.5 | 139.68 | 139.2063 | 25446000 |
| 14-10-2019 | 139.69 | 140.29 | 139.52 | 139.55 | 139.0768 | 13304300 |
| 15-10-2019 | 140.06 | 141.79 | 139.81 | 141.57 | 141.0899 | 19695700 |
| 16-10-2019 | 140.79 | 140.99 | 139.53 | 140.41 | 139.9338 | 20751600 |
| 17-10-2019 | 140.95 | 141.42 | 139.02 | 139.69 | 139.2163 | 21460600 |
| 18-10-2019 | 139.76 | 140 | 136.56 | 137.41 | 136.944 | 32273500 |
| 21-10-2019 | 138.45 | 138.5 | 137.01 | 138.43 | 137.9605 | 20078200 |
| 22-10-2019 | 138.97 | 140.01 | 136.26 | 136.37 | 135.9075 | 27431000 |
| 23-10-2019 | 136.88 | 137.45 | 135.61 | 137.24 | 136.7746 | 29844600 |
| 24-10-2019 | 139.39 | 140.42 | 138.67 | 139.94 | 139.4654 | 37029300 |
| 25-10-2019 | 139.34 | 141.14 | 139.2 | 140.73 | 140.2527 | 25959700 |
| 28-10-2019 | 144.4 | 145.67 | 143.51 | 144.19 | 143.701 | 35280100 |
| 29-10-2019 | 144.08 | 144.5 | 142.65 | 142.83 | 142.3456 | 20589500 |
| 30-10-2019 | 143.52 | 145 | 142.79 | 144.61 | 144.1196 | 18496600 |
| 31-10-2019 | 144.9 | 144.93 | 142.99 | 143.37 | 142.8838 | 24605100 |
| 01-11-2019 | 144.26 | 144.42 | 142.97 | 143.72 | 143.2326 | 33128400 |
| 04-11-2019 | 144.83 | 145 | 144.16 | 144.55 | 144.0598 | 16912000 |
| 05-11-2019 | 144.97 | 145.02 | 143.91 | 144.46 | 143.9701 | 18250200 |
| 06-11-2019 | 144.37 | 144.52 | 143.2 | 144.06 | 143.5715 | 16575800 |
| 07-11-2019 | 143.84 | 144.88 | 143.77 | 144.26 | 143.7708 | 17786700 |
| 08-11-2019 | 143.98 | 145.99 | 143.76 | 145.96 | 145.465 | 16732700 |
| 11-11-2019 | 145.34 | 146.42 | 144.73 | 146.11 | 145.6145 | 14362600 |
| 12-11-2019 | 146.28 | 147.57 | 146.06 | 147.07 | 146.5713 | 18641600 |
| 13-11-2019 | 146.74 | 147.46 | 146.28 | 147.31 | 146.8104 | 16919200 |
| 14-11-2019 | 147.02 | 148.41 | 147 | 148.06 | 147.5579 | 19729800 |
| 15-11-2019 | 148.93 | 149.99 | 148.27 | 149.97 | 149.4614 | 23485700 |
| 18-11-2019 | 150.07 | 150.55 | 148.98 | 150.34 | 149.8302 | 21534000 |
| 19-11-2019 | 150.88 | 151.33 | 150.2 | 150.39 | 149.88 | 23935700 |
| 20-11-2019 | 150.31 | 150.84 | 148.46 | 149.62 | 149.62 | 25696800 |
| 21-11-2019 | 149.4 | 149.8 | 148.5 | 149.48 | 149.48 | 18576100 |
| 22-11-2019 | 150.07 | 150.3 | 148.82 | 149.59 | 149.59 | 15901800 |
| 25-11-2019 | 150 | 151.35 | 149.92 | 151.23 | 151.23 | 22420900 |
| 26-11-2019 | 151.36 | 152.42 | 151.32 | 152.03 | 152.03 | 24620100 |
| 27-11-2019 | 152.33 | 152.5 | 151.52 | 152.32 | 152.32 | 15184400 |
| 29-11-2019 | 152.1 | 152.3 | 151.28 | 151.38 | 151.38 | 11977300 |
| 02-12-2019 | 151.81 | 151.83 | 148.32 | 149.55 | 149.55 | 27418400 |
| 03-12-2019 | 147.49 | 149.43 | 146.65 | 149.31 | 149.31 | 24066000 |
| 04-12-2019 | 150.14 | 150.18 | 149.2 | 149.85 | 149.85 | 17574700 |
| 05-12-2019 | 150.05 | 150.32 | 149.48 | 149.93 | 149.93 | 17869100 |
| 06-12-2019 | 150.99 | 151.87 | 150.27 | 151.75 | 151.75 | 16403500 |
| 09-12-2019 | 151.07 | 152.21 | 150.91 | 151.36 | 151.36 | 16687400 |
| 10-12-2019 | 151.29 | 151.89 | 150.76 | 151.13 | 151.13 | 16476100 |
| 11-12-2019 | 151.54 | 151.87 | 150.33 | 151.7 | 151.7 | 18856600 |
| 12-12-2019 | 151.65 | 153.44 | 151.02 | 153.24 | 153.24 | 24612100 |
| 13-12-2019 | 153 | 154.89 | 152.83 | 154.53 | 154.53 | 23845400 |
| 16-12-2019 | 155.11 | 155.9 | 154.82 | 155.53 | 155.53 | 24144200 |
| 17-12-2019 | 155.45 | 155.71 | 154.45 | 154.69 | 154.69 | 25425600 |
| 18-12-2019 | 154.3 | 155.48 | 154.18 | 154.37 | 154.37 | 24129200 |
| 19-12-2019 | 154 | 155.77 | 153.75 | 155.71 | 155.71 | 24958900 |
| 20-12-2019 | 157.35 | 158.49 | 156.29 | 157.41 | 157.41 | 53477500 |
| 23-12-2019 | 158.12 | 158.12 | 157.27 | 157.41 | 157.41 | 17718200 |
| 24-12-2019 | 157.48 | 157.71 | 157.12 | 157.38 | 157.38 | 8989200 |
| 26-12-2019 | 157.56 | 158.73 | 157.4 | 158.67 | 158.67 | 14520600 |
| 27-12-2019 | 159.45 | 159.55 | 158.22 | 158.96 | 158.96 | 18412800 |
| 30-12-2019 | 158.99 | 159.02 | 156.73 | 157.59 | 157.59 | 16348400 |
| 31-12-2019 | 156.77 | 157.77 | 156.45 | 157.7 | 157.7 | 18369400 |
| 02-01-2020 | 158.78 | 160.73 | 158.33 | 160.62 | 160.62 | 22622100 |
| 03-01-2020 | 158.32 | 159.95 | 158.06 | 158.62 | 158.62 | 21116200 |
| 06-01-2020 | 157.08 | 159.1 | 156.51 | 159.03 | 159.03 | 20813700 |
| 07-01-2020 | 159.32 | 159.67 | 157.33 | 157.58 | 157.58 | 18017762 |

Stock price prediction using deep learning, specifically with recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, has gained popularity due to their ability to capture sequential data patterns.

Here's a step-by-step guide on how to approach stock price prediction using deep learning:

**1.Data Collection and Preprocessing:**

* Collect historical stock price data, including open, close, high, and low prices, trading volumes, and other relevant features.
* Gather additional data sources such as news sentiment, economic indicators, and company financial reports.
* Preprocess the data by cleaning, handling missing values, and normalizing or scaling it for uniformity.

**2. Time Series Data Preparation:**

* Organize the data into time series sequences, with each sequence containing a fixed number of time steps. For example, you can use daily closing prices over a specified period.
* Create target labels for each sequence, representing the stock price at a future time step. This can be done by shifting the price data by one or more time steps forward.

**3.Feature Engineering:**

* Create relevant features that may impact stock prices, such as moving averages, relative strength index (RSI), and volatility measures.
* Incorporate external factors like news sentiment scores, economic indicators (e.g., GDP, interest rates), and company-specific data (e.g., earnings reports).

**4. Model Architecture:**

* Build an RNN or LSTM-based deep learning model. These models are well-suited for sequential data like time series.
* Design the model with an input layer, one or more LSTM layers, and an output layer.
* Adjust the number of hidden units, dropout rates, and other hyperparameters based on experimentation and validation results.

**5.Training:**

* Split the data into training and testing sets to evaluate model performance.
* Train the LSTM model on historical data, using past stock prices and relevant features to learn sequential patterns.
* Implement mini-batch training to speed up convergence.

**6.Model Evaluation:**

* Use appropriate evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) to assess the model's accuracy.
* Consider additional metrics such as Sharpe ratio or profitability metrics to gauge the effectiveness of trading strategies based on predictions.

**7.Hyperparameter Tuning:**

* Fine-tune model hyperparameters to optimize prediction performance. This may involve adjusting learning rates, the number of LSTM layers, or the batch size.

**8.Validation and Backtesting:**

* Validate the model's performance on out-of-sample data to ensure it generalizes well to unseen market conditions.
* Conduct backtesting to evaluate the profitability of trading strategies based on the predictions. This step is critical for assessing real-world viability.

**9.Continuous Monitoring and Updating:**

* Stock market conditions change over time. Continuously monitor the model's performance and retrain it periodically to adapt to changing market dynamics.

**10.Risk Management:**

* Implement risk management strategies to minimize potential losses from incorrect predictions. This includes setting stop-loss orders and portfolio diversification.

**11.Ethical and Regulatory Considerations:**

* Ensure compliance with ethical guidelines and regulations governing financial markets and algorithmic trading to avoid potential legal issues.

**Program:**

**Stock Price prediction**

**I will use LSTMs for predicting the price of stocks of IBM for the year 2017**

**In [1]:**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional

from keras.optimizers import SGD

import math

from sklearn.metrics import mean\_squared\_error

**Using TensorFlow backend.**

**In [2]:**

*# Some functions to help out with*

def plot\_predictions(test,predicted):

plt.plot(test, color='red',label='Real IBM Stock Price')

plt.plot(predicted, color='blue',label='Predicted IBM Stock Price')

plt.title('IBM Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('IBM Stock Price')

plt.legend()

plt.show()

def return\_rmse(test,predicted):

rmse = math.sqrt(mean\_squared\_error(test, predicted))

print("The root mean squared error is **{}**.".format(rmse))

**In [3]:**

*# First, we get the data*

dataset = pd.read\_csv('../input/IBM\_2006-01-01\_to\_2018-01-01.csv', index\_col='Date', parse\_dates=['Date'])

dataset.head()

**Out [3]:**

****

**In [4]:**

*# Checking for missing values*

training\_set = dataset[:'2016'].iloc[:,1:2].values

test\_set = dataset['2017':].iloc[:,1:2].values

**In [5]:**

*# We have chosen 'High' attribute for prices. Let's see what it looks like*

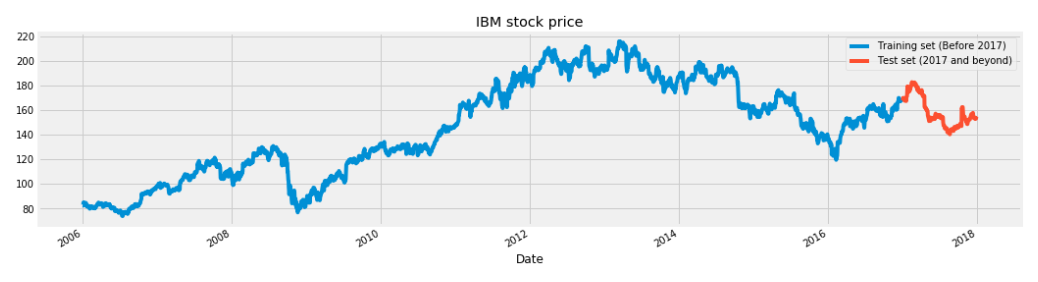
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)

dataset["High"]['2017':].plot(figsize=(16,4),legend=True)

plt.legend(['Training set (Before 2017)','Test set (2017 and beyond)'])

plt.title('IBM stock price')

plt.show()



**In [6]:**

*# Scaling the training set*

sc = MinMaxScaler(feature\_range=(0,1))

training\_set\_scaled = sc.fit\_transform(training\_set)

**In [7]:**

*# Since LSTMs store long term memory state, we create a data structure with 60 timesteps and 1 output*

*# So for each element of training set, we have 60 previous training set elements*

X\_train = []

y\_train = []

for i in range(60,2769):

X\_train.append(training\_set\_scaled[i-60:i,0])

y\_train.append(training\_set\_scaled[i,0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

**In [8]:**

*# Reshaping X\_train for efficient modelling*

X\_train = np.reshape(X\_train, (X\_train.shape[0],X\_train.shape[1],1))

**In [9]:**

*# The LSTM architecture*

regressor = Sequential()

*# First LSTM layer with Dropout regularisation*

regressor.add(LSTM(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1)))

regressor.add(Dropout(0.2))

*# Second LSTM layer*

regressor.add(LSTM(units=50, return\_sequences=True))

regressor.add(Dropout(0.2))

*# Third LSTM layer*

regressor.add(LSTM(units=50, return\_sequences=True))

regressor.add(Dropout(0.2))

*# Fourth LSTM layer*

regressor.add(LSTM(units=50))

regressor.add(Dropout(0.2))

*# The output layer*

regressor.add(Dense(units=1))

*# Compiling the RNN*

regressor.compile(optimizer='rmsprop',loss='mean\_squared\_error')

*# Fitting to the training set*

regressor.fit(X\_train,y\_train,epochs=50,batch\_size=32)

Epoch 1/50

2709/2709 [==============================] - 43s 16ms/step - loss: 0.0266

Epoch 2/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0094

Epoch 3/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0085

Epoch 4/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0067

Epoch 5/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0063

Epoch 6/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0055

Epoch 7/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0047

Epoch 8/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0046

Epoch 9/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0041

Epoch 10/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0039

Epoch 11/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0037

Epoch 12/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0034

Epoch 13/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0033

Epoch 14/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0032

Epoch 15/50

2709/2709 [==============================] - 41s 15ms/step - loss: 0.0029

Epoch 16/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0028

Epoch 17/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0028

Epoch 18/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0026

Epoch 19/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0026

Epoch 20/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0025

Epoch 21/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0023

Epoch 22/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0024

Epoch 23/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0023

Epoch 24/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0023

Epoch 25/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0021

Epoch 26/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0022

Epoch 27/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0020

Epoch 28/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0020

Epoch 29/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0020

Epoch 30/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0021

Epoch 31/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0020

Epoch 32/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0019

Epoch 33/50

2709/2709 [==============================] - 40s 15ms/step - loss: 0.0018

Epoch 34/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0018

Epoch 35/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0019

Epoch 36/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0018

Epoch 37/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0017

Epoch 38/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0017

Epoch 39/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0016

Epoch 40/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0016

Epoch 41/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0015

Epoch 42/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0017

Epoch 43/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0016

Epoch 44/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0015

Epoch 45/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0015

Epoch 46/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0015

Epoch 47/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0014

Epoch 48/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0015

Epoch 49/50

2709/2709 [==============================] - 39s 15ms/step - loss: 0.0015

Epoch 50/50

2709/2709 [==============================] - 39s 14ms/step - loss: 0.0014

**Out [9]:**

<keras.callbacks.History at 0x7f146640dd68>

**In [10]:**

*# Now to get the test set ready in a similar way as the training set.*

*# The following has been done so forst 60 entires of test set have 60 previous values which is impossible to get unless we take the whole*

*# 'High' attribute data for processing*

dataset\_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axis=0)

inputs = dataset\_total[len(dataset\_total)-len(test\_set) - 60:].values

inputs = inputs.reshape(-1,1)

**inputs = sc.transform(inputs)**

**In [11]:**

*# Preparing X\_test and predicting the prices*

X\_test = []

for i **in** range(60,311):

X\_test.append(inputs[i-60:i,0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0],X\_test.shape[1],1))

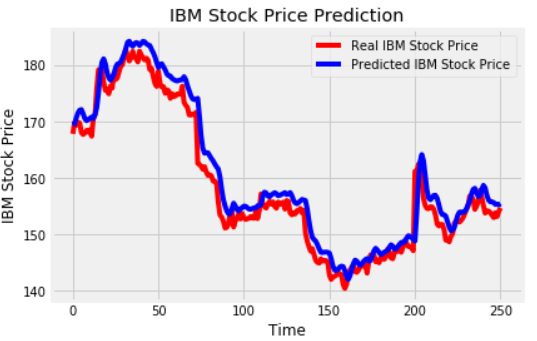
predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price

**In [12]:**

*# Visualizing the results for LSTM*

plot\_predictions(test\_set,predicted\_stock\_price)



**In [13]:**

*# Evaluating our model*

return\_rmse(test\_set,predicted\_stock\_price)

The root mean squared error is 2.8685362362359834.

LSTM is not the only kind of unit that has taken the world of Deep Learning by a storm. We have **Gated Recurrent Units(GRU)**. It's not known, which is better: GRU or LSTM because they have comparable performances. GRUs are easier to train than LSTMs.

**Gated Recurrent Units:**

**In [14]:**

*# The GRU architecture* regressorGRU = Sequential() *# First GRU layer with Dropout regularisation* regressorGRU.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1), activation='tanh')) regressorGRU.add(Dropout(0.2)) *# Second GRU layer* regressorGRU.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1), activation='tanh')) regressorGRU.add(Dropout(0.2)) *# Third GRU layer* regressorGRU.add(GRU(units=50, return\_sequences=True, input\_shape=(X\_train.shape[1],1), activation='tanh')) regressorGRU.add(Dropout(0.2)) *# Fourth GRU layer* regressorGRU.add(GRU(units=50, activation='tanh')) regressorGRU.add(Dropout(0.2)) *# The output layer* regressorGRU.add(Dense(units=1)) *# Compiling the RNN* regressorGRU.compile(optimizer=SGD(lr=0.01, decay=1e-7, momentum=0.9, nesterov=False),loss='mean\_squared\_error') *# Fitting to the training set* regressorGRU.fit(X\_train,y\_train,epochs=50,batch\_size=150)

Epoch 1/50

2709/2709 [==============================] - 10s 4ms/step - loss: 0.1382

Epoch 2/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0623

Epoch 3/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0381

Epoch 4/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0150

Epoch 5/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0051

Epoch 6/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0040

Epoch 7/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0036

Epoch 8/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0034

Epoch 9/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0034

Epoch 10/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0034

Epoch 11/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0032

Epoch 12/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0031

Epoch 13/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0031

Epoch 14/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0030

Epoch 15/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0027

Epoch 16/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0030

Epoch 17/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0028

Epoch 18/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0027

Epoch 19/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0025

Epoch 20/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0027

Epoch 21/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0025

Epoch 22/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0026

Epoch 23/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0028

Epoch 24/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0025

Epoch 25/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0025

Epoch 26/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0026

Epoch 27/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0024

Epoch 28/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0023

Epoch 29/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0025

Epoch 30/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0023

Epoch 31/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0024

Epoch 32/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0024

Epoch 33/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0022

Epoch 34/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0022

Epoch 35/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0022

Epoch 36/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0022

Epoch 37/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0023

Epoch 38/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0023

Epoch 39/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0023

Epoch 40/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0023

Epoch 41/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0022

Epoch 42/50

2709/2709 [==============================] - 7s 3ms/step - loss: 0.0022

Epoch 43/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0022

Epoch 44/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0021

Epoch 45/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0021

Epoch 46/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0022

Epoch 47/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0021

Epoch 48/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0022

Epoch 49/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0021

Epoch 50/50

2709/2709 [==============================] - 8s 3ms/step - loss: 0.0020

**Out [14]:**

<keras.callbacks.History at 0x7f1458231b38>

**In [15]:**

*# Preparing X\_test and predicting the prices*

X\_test = []

for i **in** range(60,311):

X\_test.append(inputs[i-60:i,0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0],X\_test.shape[1],1))

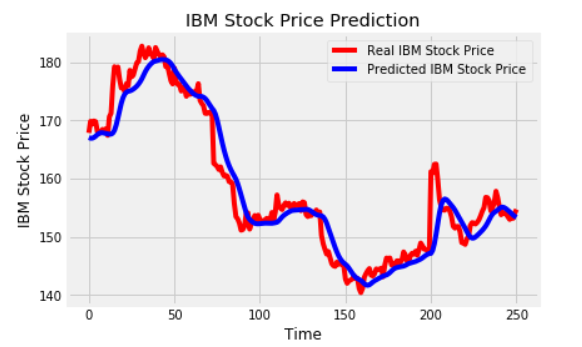
GRU\_predicted\_stock\_price = regressorGRU.predict(X\_test)

GRU\_predicted\_stock\_price = sc.inverse\_transform(GRU\_predicted\_stock\_price)

**In [16]:**

*# Visualizing the results for GRU*

plot\_predictions(test\_set,GRU\_predicted\_stock\_price)



**In [17]:**

*# Evaluating GRU*

return\_rmse(test\_set,GRU\_predicted\_stock\_price)

The root mean squared error is 3.253068340009998.

**Sequence Generation:**

**In [18]:**

*# Preparing sequence data*

initial\_sequence = X\_train[2708,:]

sequence = []

for i in range(251):

new\_prediction = regressorGRU.predict(initial\_sequence.reshape(initial\_sequence.shape[1],initial\_sequence.shape[0],1))

initial\_sequence = initial\_sequence[1:]

initial\_sequence = np.append(initial\_sequence,new\_prediction,axis=0)

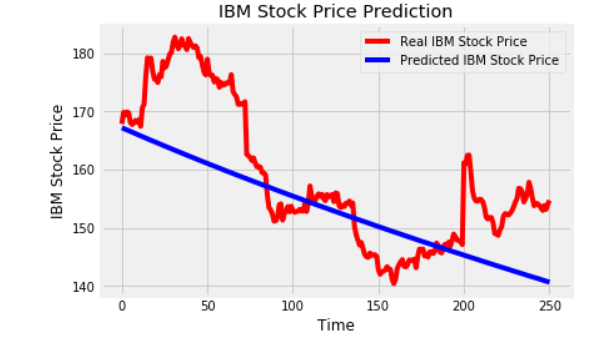
sequence.append(new\_prediction)

sequence = sc.inverse\_transform(np.array(sequence).reshape(251,1))

**In [19]:**

*# Visualizing the sequence*

plot\_predictions(test\_set,sequence)

****

**In [20]:**

*# Evaluating the sequence*

return\_rmse(test\_set,sequence

The root mean squared error is 9.651351091589397.

**CONCLUSION:**

* So, GRU works better than LSTM in this case. Bidirectional LSTM is also a good way so make the model stronger. But this may vary for different data sets. **Applying both LSTM and GRU together gave even better results.**
* while deep learning can be a valuable tool for stock price prediction, it should be used as part of a broader investment strategy and considered alongside other fundamental and technical analysis methods.
* Accurate predictions require careful data preprocessing, model selection, and continuous monitoring, and even the best models may not guarantee consistent profits due to the inherent uncertainty and complexity of financial markets.

3.BUILD LOADING AND PREPROCESSING THE DATASET

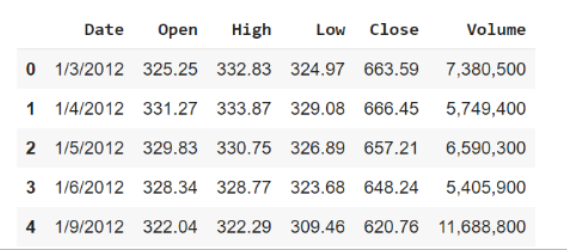
**Introduction:**

Stock Price Prediction using machine learning helps you discover the future value of company stock and other financial assets traded on an exchange. The entire idea of predicting stock prices is to gain significant profits.

**LSTMs** work in a three-step process:

* The first step in **LSTM** is to decide which information to be omitted from the cell in that particular time step. It is decided with the help of a sigmoid function. It looks at the previous state (ht-1) and the current input xt and computes the function.
* There are two functions in the second layer. The first is the sigmoid function, and the second is the tanh function. The sigmoid function decides which values to let through (0 or 1). The tanh function gives the weightage to the values passed, deciding their level of importance from -1 to 1.
* The third step is to decide what will be the final output. First, you need to run a sigmoid layer which determines what parts of the cell state make it to the output. Then, you must put the cell state through the tanh function to push the values between -1 and 1 and multiply it by the output of the sigmoid gate.

**Given data set:**



**Google Stock Price Prediction Using LSTM**

1. **Import the Libraries.**

#import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

1. **Load the Training Dataset.**

There are five columns. The Open column tells the price at which a stock started trading when the market opened on a particular day. The Close column refers to the price of an individual stock when the stock exchange closed the market for the day. The High column depicts the highest price at which a stock traded during a period. The Low column tells the lowest price of the period. Volume is the total amount of trading activity during a period of time.

dataset\_train = pd.read\_csv(“Google\_Stock\_Price\_Train.csv”)

dataset\_train.head()

1. **Use the Open Stock Price Column to Train Your Model.**

training\_set=dataset\_train.iloc[:,1:2].values

print(training\_set)

print(training\_set.shape)

[[325.25]

[331.27]

[329.83]

……

[793.7]

[783.33]

[782.75]]

(1258,1)

**4. Normalizing the Dataset.**

From sklearn.preprocesssing import minmaxscalar

Scalar=minmaxscalar(feature\_range=(0,1))

Scaled\_training\_set

array([[0.0.8581368],

[0.09701243],

[0.09433366],

…..,

[0.95725128],

[0.93576041],

[0.93688146]])

**5. Creating X\_train and y\_train Data Structures**

x\_train=[]

y\_train=[]

for i in range(60,1258):

x\_train.append(scaled\_trained\_set[i-60:i,0])

y\_train.append(scaled\_trained\_set[i,0])

x\_train=np.array(x\_train)

y\_train=np.array(y\_train)

print(x\_train.shape)

print(y\_train.shape)

(1198,60)  
 (1198,)

**6. Reshape the Data**

x\_train=np.reshape(x\_train,(x\_train.shape[0],x\_train.shape[1],1))

x\_train.shape

(1198,60,1)

**7. Building the Model by Importing the Crucial Libraries and Adding Different Layers to LSTM.**

from keras.models import sequential

from keras.layers import LSTM|

from keras.layers import dense

from keras.layers import dropout

regressor =Sequential( )

regressor.add(LSTM(units = 50, return\_sequences= True, input\_shape = (X\_traín.shape[1], 1)))

regressor. add(Dropout(0 . 2) )

regressor.add(LSTM(units = 50, return\_sequences=True)

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences= True) )

regressor. add(Dropout(0. 2) )

regressor.add(LSTM(units = 50))

regressor. add(Dropout(0 . 2) )

regressor.add(Dense(units=1))

**8. Fitting the Model**

regressor.compile(optimizer=’adam’,loss=’mean\_squared\_error’)

regressor.fit(x\_train,y\_train,epochs=100,batch\_size=32)

epoch1/100

38/38 [====================] – 11s 114s/step – loss:0.1011

epoch2/100

38/38 [====================] – 4s 117s/step – loss:0.0061

epoch3/100

38/38 [====================] – 4s 118s/step – loss:0.0063

**9. Extracting the Actual Stock Prices of Jan-2017**

dataset\_test=pd.read\_csv(“google\_stock\_price\_test.csv”)

actual\_stock price = dataset\_test.iloc[:,1:2].values

**10. Preparing the Input for the Model**

dataset\_total=pd.concat((dataset\_train[‘open’],dataset\_test[‘open’]),axis=0)

inputs=dataset\_total[len(dataset\_total)-len(dataset\_test)-60:].values

inputs=inputs.reshape(-1,1)

inputs=scalar.transform(inputs)

x\_test=[]

for i in range(60,80):

x\_test.append(inputs[i-60:I,0])

x\_test=np.array(x\_test)

x\_test=np.reshape(x\_test,(x\_test.shape[0],x\_test.shape[1],1))

**11. Predicting the Values for Jan 2017 Stock Prices.**

predicted\_stock\_price=regressor.predict(x\_test)

predicted\_stock\_price=scalar.inverse\_transform(ptredicted\_stock\_price)

**12. Plotting the Actual and Predicted Prices for Google Stocks**

plt.plot(actual\_stock\_price,color=’red’,label=’actual google stock price’)

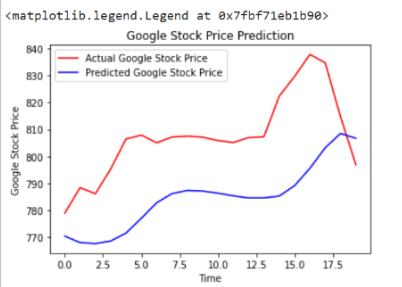
plt.plot(predicted\_stock\_price,color=’blue’,label=’predicted google stock price’)

plt.title(‘google stock price prediction’)

plt.xlabel(‘time’)

plt.ylabel(‘google stock price’)

plt.legend()



**Conclusion:**

The stock market plays a remarkable role in our daily lives. It is a significant factor in a country's GDP growth. In this tutorial, you learned the basics of the stock market and how to perform stock price prediction using machine learning.

4.BUILDING THE STOCK PRICE PREDICTION MODEL BY FEATURE ENGINEERING,MODEL TRAINING,EVALUATION.

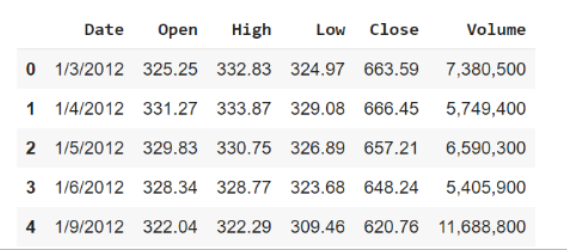
**Introduction:**

Stock Price Prediction using machine learning helps you discover the future value of company stock and other financial assets traded on an exchange. The entire idea of predicting stock prices is to gain significant profits.

**LSTMs** work in a three-step process:

* The first step in **LSTM** is to decide which information to be omitted from the cell in that particular time step. It is decided with the help of a sigmoid function. It looks at the previous state (ht-1) and the current input xt and computes the function.
* There are two functions in the second layer. The first is the sigmoid function, and the second is the tanh function. The sigmoid function decides which values to let through (0 or 1). The tanh function gives the weightage to the values passed, deciding their level of importance from -1 to 1.
* The third step is to decide what will be the final output. First, you need to run a sigmoid layer which determines what parts of the cell state make it to the output. Then, you must put the cell state through the tanh function to push the values between -1 and 1 and multiply it by the output of the sigmoid gate.

**Given data set:**



**Google Stock Price Prediction Using LSTM**

1. **Import the Libraries.**

#import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

1. **Load the Training Dataset.**

There are five columns. The Open column tells the price at which a stock started trading when the market opened on a particular day. The Close column refers to the price of an individual stock when the stock exchange closed the market for the day. The High column depicts the highest price at which a stock traded during a period. The Low column tells the lowest price of the period. Volume is the total amount of trading activity during a period of time.

dataset\_train = pd.read\_csv(“Google\_Stock\_Price\_Train.csv”)

dataset\_train.head()

1. **Use the Open Stock Price Column to Train Your Model.**

training\_set=dataset\_train.iloc[:,1:2].values

print(training\_set)

print(training\_set.shape)

[[325.25]

[331.27]

[329.83]

……

[793.7]

[783.33]

[782.75]]

(1258,1)

**4. Normalizing the Dataset.**

From sklearn.preprocesssing import minmaxscalar

Scalar=minmaxscalar(feature\_range=(0,1))

Scaled\_training\_set

array([[0.0.8581368],

[0.09701243],

[0.09433366],

…..,

[0.95725128],

[0.93576041],

[0.93688146]])

**5. Creating X\_train and y\_train Data Structures**

x\_train=[]

y\_train=[]

for i in range(60,1258):

x\_train.append(scaled\_trained\_set[i-60:i,0])

y\_train.append(scaled\_trained\_set[i,0])

x\_train=np.array(x\_train)

y\_train=np.array(y\_train)

print(x\_train.shape)

print(y\_train.shape)

(1198,60)  
 (1198,)

**6. Reshape the Data**

x\_train=np.reshape(x\_train,(x\_train.shape[0],x\_train.shape[1],1))

x\_train.shape

(1198,60,1)

**7. Building the Model by Importing the Crucial Libraries and Adding Different Layers to LSTM.**

from keras.models import sequential

from keras.layers import LSTM|

from keras.layers import dense

from keras.layers import dropout

regressor =Sequential( )

regressor.add(LSTM(units = 50, return\_sequences= True, input\_shape = (X\_traín.shape[1], 1)))

regressor. add(Dropout(0 . 2) )

regressor.add(LSTM(units = 50, return\_sequences=True)

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences= True) )

regressor. add(Dropout(0. 2) )

regressor.add(LSTM(units = 50))

regressor. add(Dropout(0 . 2) )

regressor.add(Dense(units=1))

**8. Fitting the Model**

regressor.compile(optimizer=’adam’,loss=’mean\_squared\_error’)

regressor.fit(x\_train,y\_train,epochs=100,batch\_size=32)

epoch1/100

38/38 [====================] – 11s 114s/step – loss:0.1011

epoch2/100

38/38 [====================] – 4s 117s/step – loss:0.0061

epoch3/100

38/38 [====================] – 4s 118s/step – loss:0.0063

**9. Extracting the Actual Stock Prices of Jan-2017**

dataset\_test=pd.read\_csv(“google\_stock\_price\_test.csv”)

actual\_stock price = dataset\_test.iloc[:,1:2].values

**10. Preparing the Input for the Model**

dataset\_total=pd.concat((dataset\_train[‘open’],dataset\_test[‘open’]),axis=0)

inputs=dataset\_total[len(dataset\_total)-len(dataset\_test)-60:].values

inputs=inputs.reshape(-1,1)

inputs=scalar.transform(inputs)

x\_test=[]

for i in range(60,80):

x\_test.append(inputs[i-60:I,0])

x\_test=np.array(x\_test)

x\_test=np.reshape(x\_test,(x\_test.shape[0],x\_test.shape[1],1))

**11. Predicting the Values for Jan 2017 Stock Prices.**

predicted\_stock\_price=regressor.predict(x\_test)

predicted\_stock\_price=scalar.inverse\_transform(ptredicted\_stock\_price)

**12. Plotting the Actual and Predicted Prices for Google Stocks**

plt.plot(actual\_stock\_price,color=’red’,label=’actual google stock price’)

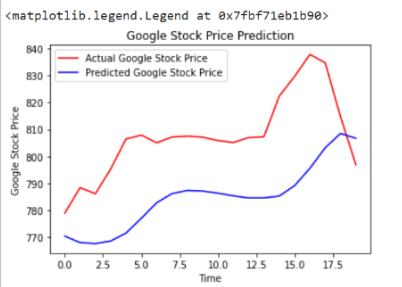
plt.plot(predicted\_stock\_price,color=’blue’,label=’predicted google stock price’)

plt.title(‘google stock price prediction’)

plt.xlabel(‘time’)

plt.ylabel(‘google stock price’)

plt.legend()



**Feature engineering:**

Feature engineering is a critical step in developing predictive models for stock price movements using deep learning techniques. Properly engineered features can help your model capture meaningful patterns and relationships in historical price data. Here are some feature engineering ideas for stock price prediction using deep learning:

**1.Price and Volume Data:**

* Historical prices: Include features like daily open, high, low, and close prices.
* Trading volume: Add daily trading volume as a feature.
* Price returns: Calculate daily returns, such as percentage change in closing prices.

**2.Moving Averages:**

* Simple moving averages (SMA): Calculate the SMA over various time periods (e.g., 10 days, 50 days, 200 days).
* Exponential moving averages (EMA): Calculate the EMA over different time frames.

**3.Volatility Measures:**

* Historical volatility: Compute the historical volatility using metrics like the standard deviation of returns over a specified window.
* Bollinger Bands: Create features based on Bollinger Bands to capture price volatility.

**4.Technical Indicators:**

* Relative Strength Index (RSI): A momentum oscillator that measures the speed and change of price movements.
* Moving Average Convergence Divergence (MACD): A trend-following momentum indicator.
* Stochastic Oscillator: A momentum indicator comparing a particular closing price to a range of its prices over a specific period.

**5.Lagged Features:**

* Include lagged versions of the target variable and other relevant features, which can help capture autocorrelation in the time series data.

**6.Calendar and Economic Events:**

* Include features related to important calendar events like holidays, earnings reports, and economic indicators like interest rates, GDP, etc.

**7.Market Sentiment:**

* Incorporate sentiment analysis of news articles, social media, or financial reports related to the stock or the market in general.

**8.Market Index Data:**

* Include features related to broader market indices (e.g., S&P 500, NASDAQ) to capture overall market trends and sentiment.

**9.Fundamental Data:**

* If available, include fundamental data like earnings per share (EPS), price-to-earnings (P/E) ratio, and other financial metrics.

**10.Seasonal Patterns:**

* Identify and include features that capture seasonal patterns if relevant to the stock or industry.

**11.Correlations and Cross-Correlations:**

* Calculate and include correlations between your target stock and related stocks, sectors, or market indices.

**12.Time of Day and Day of Week:**

* Incorporate features that capture intraday patterns and day-of-week effects on stock prices.

**13.Feature Scaling:**

* Normalize or standardize your features to ensure that deep learning models can work effectively.

**14.Feature Selection:**

* Use techniques like feature importance analysis or dimensionality reduction to select the most relevant features and eliminate noise.

**15.Custom Features:**

* Experiment with creating custom features specific to the industry or stock you are analyzing.

**16.Data Preprocessing:**

* Handle missing data and outliers appropriately, as these can significantly affect model performance.

**17.Sequential Data:**

* If using recurrent neural networks (RNNs) or Long Short-Term Memory (LSTM) models, structure your data into sequences of historical prices and indicators.

**Model Training:**

Training a deep learning model for stock price prediction is a complex task that involves several steps, from data preparation to model architecture selection, training, and evaluation. Here's a step-by-step guide on how to train a deep learning model for stock price prediction:

**1.Data Collection and Preprocessing:**

* Collect historical stock price data, including features you've engineered during the feature engineering stage.
* Split the data into training, validation, and test sets. A common split might be 70% for training, 15% for validation, and 15% for testing.
* Normalize or standardize the data to ensure that all features have the same scale. This is especially important for deep learning models.

**2.Sequence Generation (for Time Series Models):**

* If you're using RNNs, LSTMs, or CNNs for time series data, create sequences of historical data with corresponding target values. For example, if you're using daily data, a sequence might consist of the past N days' data, and the target would be the next day's price.

**3.Selecting the Model Architecture:**

* Choose a suitable deep learning architecture for your task. Common choices for stock price prediction include:
* Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) for sequential data.
* Convolutional Neural Networks (CNNs) for analyzing image-based financial charts.
* Feedforward Neural Networks (FNNs) for tabular data or simpler time series predictions.
* Experiment with different architectures to find the one that works best for your dataset.

**4.Loss Function and Metrics:**

* Select an appropriate loss function for regression tasks, such as Mean Squared Error (MSE) or Mean Absolute Error (MAE).
* Choose evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), or others that are relevant to your specific problem.

**5.Model Training:**

* Train your deep learning model using the training dataset. During training, you'll aim to minimize the chosen loss function.
* Implement mini-batch training to improve convergence speed and prevent overfitting.
* Monitor training progress by tracking loss and metrics on the validation set.
* Implement techniques to prevent overfitting, such as dropout, early stopping, and regularization.

**6.Hyperparameter Tuning:**

* Experiment with different hyperparameters, including learning rate, batch size, the number of hidden layers, and the number of neurons in each layer, to optimize model performance.
* Use techniques like grid search or random search to find the best combination of hyperparameters.

**7.Regularization:**

* Implement regularization techniques such as L1 and L2 regularization to prevent overfitting.

**8.Feature Importance Analysis:**

* Analyze feature importance to understand which features are the most informative for the model.

**9.Model Evaluation:**

* Evaluate the model's performance on the test dataset using the chosen evaluation metrics. This step will give you an idea of how well your model generalizes to unseen data.

**10.Backtesting and Real-World Application:**

* If you plan to use your model for trading or investment, conduct backtesting to see how it would have performed in a real-world setting.
* Implement a strategy for trading based on model predictions and evaluate its performance.

**11.Monitoring and Model Maintenance:**

* Continuously monitor the model's performance and retrain it as necessary, as stock market dynamics can change over time.

**12.Deployment:**

* If your model performs well in a real-world setting, you can deploy it for live predictions.

**Evaluating:**

Evaluating the performance of a deep learning model for stock price prediction is crucial to determine how well the model is performing and whether it's providing meaningful insights for trading or investment decisions. Here are some common evaluation techniques and metrics for assessing the performance of your stock price prediction model:

**1.Mean Absolute Error (MAE):**

* MAE measures the average absolute difference between the predicted and actual stock prices. It provides a straightforward understanding of prediction accuracy. A lower MAE indicates better performance.

**2.Mean Squared Error (MSE):**

* MSE measures the average squared difference between the predicted and actual prices. It punishes larger errors more severely than MAE. Lower MSE values indicate better performance, but they might be less intuitive to interpret due to the squared nature of the metric.

**3.Root Mean Squared Error (RMSE):**

* RMSE is the square root of the MSE, which gives a measure of the average error in the same units as the target variable (stock prices). It is more interpretable than MSE.

**4.Mean Absolute Percentage Error (MAPE):**

* MAPE calculates the percentage difference between predicted and actual prices, which is particularly useful when you want to understand the relative error in terms of percentage. It's especially relevant for comparing predictions across different stocks with varying price levels.

**5.Directional Accuracy (Hit Ratio):**

* This metric evaluates whether the model correctly predicts the direction (up or down) of stock price movements. It can be a useful measure for trading strategies.

**6.Sharpe Ratio and Other Financial Metrics:**

* If you plan to use your model for trading, you can evaluate its performance using financial metrics like the Sharpe ratio, which considers risk-adjusted returns.

**7.Backtesting:**

* Backtesting involves simulating trading or investment decisions based on model predictions and assessing how well these decisions would have performed historically. Backtesting can provide insights into the model's practical utility.

**8.Cross-Validation:**

* Implement cross-validation techniques, such as k-fold cross-validation, to assess the model's stability and generalization performance.

**9.Out-of-Sample Testing:**

* It's important to evaluate your model's performance on a separate, unseen dataset (test set) to ensure it generalizes well to new data. This is especially important in time series forecasting.

**10.Benchmark Comparison:**

* Compare the model's performance to that of a simple benchmark, such as a random walk (predicting the next price as the current price) or a basic moving average strategy, to assess whether the model adds value beyond simple approaches.

**11.Confidence Intervals:**

* Calculate confidence intervals around your predictions to quantify the uncertainty of your model. This is particularly important when making financial decisions based on predictions.

**12.Visual Inspection:**

* Plot the actual and predicted stock prices over time to visually assess how well the model captures price movements and trends.

**13.Post-Analysis:**

* Perform post-analysis to understand the causes of errors or outliers in the model's predictions. This can help refine your model and your trading or investment strategies.

**Conclusion:**

The stock market plays a remarkable role in our daily lives. It is a significant factor in a country's GDP growth. In this tutorial, you learned the basics of the stock market and how to perform stock price prediction using machine learning.

**ADVANTAGES:**

**1. Non-linearity Handling:** Deep learning models, such as neural networks, can capture non-linear relationships in stock price data, which are often challenging for traditional linear models to represent.

**2.** **Feature Extraction:** Deep learning models can automatically learn and extract relevant features from raw data, reducing the need for manual feature engineering.

**3.** **Temporal Patterns:** Deep learning models can effectively capture temporal patterns in stock price time series data, including short-term and long-term trends, seasonality, and irregular patterns.

**4. Handling Large Datasets:** Deep learning models can handle large volumes of historical stock price data, which is crucial for learning complex patterns and making accurate predictions.

**5.Ensemble Models:** Deep learning can be combined with other machine learning techniques to create ensemble models that leverage the strengths of both approaches, potentially improving prediction accuracy.

**6. Sequential Data Handling:**Deep learning models like recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks are well-suited for handling sequential data, making them suitable for time series forecasting.

**7. Flexibility:**Deep learning models can be adapted to various data sources, including textual news sentiment analysis, social media data, and economic indicators, allowing for a more comprehensive analysis.

**8. Real-time Prediction:** Deep learning models can be used for real-time stock price prediction, which is essential for high-frequency trading and making quick investment decisions.

**9.Improved Generalization:** Deep learning models are capable of generalizing well from historical data, allowing them to adapt to changing market conditions to some extent.

**10.Model Adaptation:** Deep learning models can be fine-tuned or retrained to adapt to new market information, ensuring that they stay relevant in dynamic financial markets.

**11. Risk Management:** Deep learning models can assist in risk management by providing more accurate estimates of volatility and potential market shocks.

**Disadvantages:**

1. **Data Sensitivity:** Deep learning models, especially neural networks, are highly sensitive to the quality and quantity of data. Noisy or incomplete data can lead to inaccurate predictions.
2. **Overfitting:** Deep learning models are prone to overfitting, especially when dealing with limited historical data. Overfit models may perform well on the training data but fail to generalize to unseen data.
3. **Hyperparameter Sensitivity:** Tuning hyperparameters in deep learning models can be challenging, and small changes in hyperparameters can significantly affect the model's performance. This requires extensive experimentation.
4. **Complexity:** Deep learning models are complex and require substantial computational resources, making them less accessible for individual investors or small firms with limited computing power.
5. **Interpretability:** Deep learning models are often considered "black boxes" because they provide limited insight into why a particular prediction was made. This lack of interpretability can be a drawback in financial decision-making, where understanding the rationale behind predictions is crucial.
6. **Noisy Data and Market Anomalies:** Financial markets are inherently noisy, and they can be influenced by various anomalies, making it difficult for models to distinguish genuine trends from noise.
7. **Market Efficiency:** The Efficient Market Hypothesis (EMH) suggests that stock prices already incorporate all available information, making it challenging to gain a consistent edge in predicting prices using historical data alone.
8. **Lack of Causality:** Correlation does not imply causation. Deep learning models might identify patterns in historical data, but these patterns may not have a causal relationship with future stock price movements.
9. **Non-Stationarity:** Financial markets are subject to changing conditions, and the statistical properties of stock prices can change over time. Deep learning models may struggle to adapt to non-stationary data.
10. **Risk of Loss:** Relying solely on deep learning models for investment decisions can lead to significant financial losses. Models are not foolproof and should be used as part of a diversified investment strategy.
11. **Model Complexity:** Deep learning models can be overly complex for the task of stock price prediction. Simpler models may perform just as well or better, especially when dealing with limited data.
12. **Regulatory and Ethical Considerations:** The use of deep learning for financial prediction may raise regulatory and ethical concerns, especially when it involves algorithmic trading, market manipulation, or insider trading.
13. **Model Updating:** Deep learning models may require frequent updates and retraining to adapt to changing market conditions, which can be resource-intensive.

**Conclusion:**

In conclusion, deep learning has both advantages and disadvantages when it comes to stock price prediction. It offers the potential to capture complex non-linear patterns in historical data, adapt to temporal trends, and handle large datasets. However, it also comes with challenges such as data sensitivity, overfitting, hyperparameter tuning, complexity, and the "black box" nature of models.

While deep learning can be a valuable tool for financial analysis, it should be used in conjunction with other methods and approaches, including fundamental analysis, technical analysis, and risk management strategies. Stock price prediction is inherently uncertain, and past performance is not always indicative of future results. Therefore, it's essential to approach it with caution and not rely solely on deep learning models for investment decisions.

Prepared By:

**PRAVENRAJ D V**