PREDICTING STOCK PRICE USING DEEP LEARNING

TEAM MEMBER

621721243052 : THIRUMALAI G

Phase 2 Submission Document

Project: Stock Price Prediction



**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*

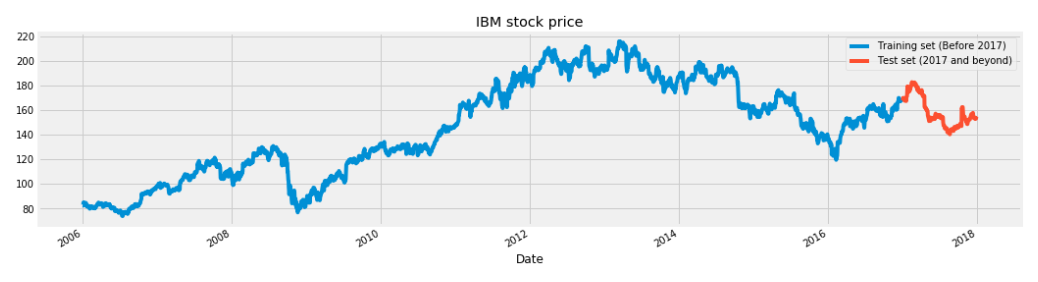
dataset["High"][:'2016'].plot(figsize=(16,4),legend=True)

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))

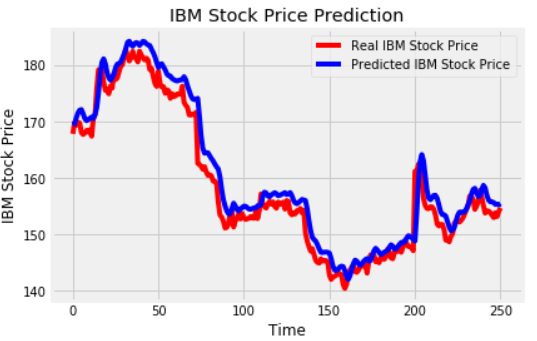
predicted\_stock\_price = regressor.predict(X\_test)

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))

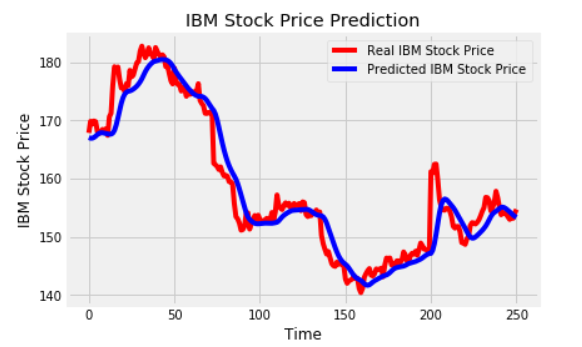
GRU\_predicted\_stock\_price = regressorGRU.predict(X\_test)

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)

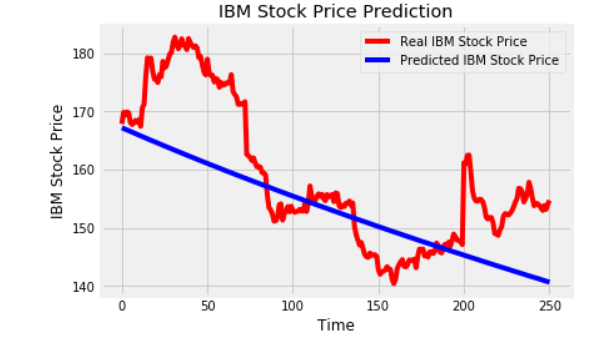
sequence.append(new\_prediction)

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.