LSTM Price Prediction

Import Necessary Libraries

```
[1] import pandas as pd
    import numpy as np
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.model_selection import train_test_split
    from keras.models import Sequential
    from keras.layers import Dense, LSTM, Dropout
    import matplotlib.pyplot as plt
```

The required libraries for solving the problem and training the neural network are imported.

Load dataset from Google Drive!

```
[44] from google.colab import drive
     drive.mount('/content/gdrive')
     !ls
     df = pd.read_csv('gdrive/MyDrive/NeuralNetworks/Datasets/AABA_2006-01-01_to_2018-01-01.csv', index_col='Date', parse_dates=['Date'])
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
gdrive  sample_data
```

The dataset is downloaded from google drive and is read.

Show Head of Dataset

```
[3] df.head()
```

Date	Open	High	Low	Close	Volume	Name
2006-01-03	39.69	41.22	38.79	40.91	24232729	AABA
2006-01-04	41.22	41.90	40.77	40.97	20553479	AABA
2006-01-05	40.93	41.73	40.85	41.53	12829610	AABA
2006-01-06	42.88	43.57	42.80	43.21	29422828	AABA
2006-01-09	43.10	43.66	42.82	43.42	16268338	AABA

As was asked in the question, the head of the dataset is shown.

Split Train and Test datasets

Train : 2006 - 2016 Test : 2017 - Now

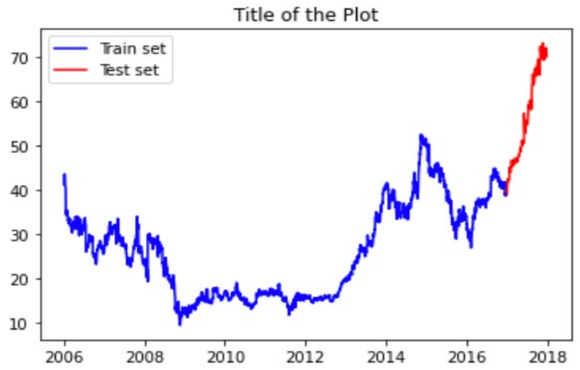
```
[4] train_set = df[:'2016'].iloc[:,1:2].values
    test_set = df['2017':].iloc[:,1:2].values
```

The dataset is split into two parts: training and testing parts. The data that runs from 2006 through 2016 is the training data, and the data that runs since 2017 is the testing data. We use the data 60 days before for training.

First view of dataset(train/test values)

```
[5]  def magic_plot(plot1=None, plot2=None, title='Title of the Plot', label1='label1', label2=None, color1='b', color2=None):
         plt.title(title)
         plt.plot(plot1, label=label1, color=color1)
         plt.plot(plot2, label=label2, color=color2)
         plt.legend()
         plt.show()

     magic_plot(plot1=df["High"][:'2016'], plot2=df["High"]['2017':], label1='Train set', label2='Test set', color2='r')
```



using the “magic_plot” function, the dataset that is spilt into training and testing parts is plotted in green and red in the graph.

Pre Processing

Normalize dataset values to (0,1)

```
[6]  PERIOD = 60
     scaler = MinMaxScaler()
     normalized_train_set = scaler.fit_transform(train_set)
     normalized_test_set = scaler.fit_transform(test_set)
```

```
[7]  X_train = []
     y_train = []

     for i in range(PERIOD, len(df[:'2016'])):
         X_train.append(normalized_train_set[i-PERIOD:i, 0])
         y_train.append(normalized_train_set[i, 0])

     X_train = np.array(X_train)
     y_train = np.array(y_train)

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

Using the “sklearn” library, the training and testing data are normalized between 0 and 1.

Train and valuation split

```
[8]  X_train, X_train_val, y_train, y_train_val = train_test_split(X_train, y_train, test_size=0.30, random_state=0)
```

Using the “sklearn” library, the training and testing data are separated in the ratio of 0/7 and 0/3.

In the following pages, you can see the types of neural network models that are trained with LSTM architecture.

Model 1: changes in the number of units or the number of neurons in each layer.

Model 1 with 30 neurons in each layer

Model 1 with 30 neurons in each layer

Model 2: changes in the number of neural network layers

Model 2 with 5 layers

Model 2 with 6 layers

Model 3: changes in the sample package size (Batch Size) while training the neural network

Model 3 with the package size 32

Model 3 with the package size 16

Training

Model 1

(Variable Units-Neurons)

Train Model 1 (30 Neurons)

```
[9] model1_30 = Sequential()

    model1_30.add(LSTM(units=30, return_sequences=True))
    model1_30.add(Dropout(0.5))

    model1_30.add(LSTM(units=30, return_sequences=True))
    model1_30.add(Dropout(0.5))

    model1_30.add(LSTM(units=30))
    model1_30.add(Dropout(0.5))

    model1_30.add(Dense(units=1))

    model1_30.compile(optimizer='adam',loss='mean_squared_error')

    history1_30 = model1_30.fit(
        X_train,
        y_train,
        epochs=40,
        batch_size=32,
        validation_data=(X_train_val, y_train_val)
        )
```

```
Epoch 11/40
60/60 [==============================] - 1s 14ms/step - loss: 0.0058 - val_loss: 0.0012
Epoch 12/40
60/60 [==============================] - 1s 14ms/step - loss: 0.0060 - val_loss: 0.0014
Epoch 13/40
60/60 [==============================] - 1s 14ms/step - loss: 0.0056 - val_loss: 0.0011
Epoch 14/40
60/60 [==============================] - 1s 14ms/step - loss: 0.0049 - val_loss: 0.0011
```

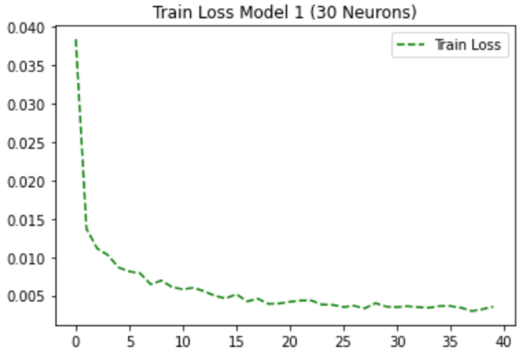
Training the neural network model 1 with 30 neurons per layer

In order to halter overfitting in each layer from happening, the code “Dropout” is used

The optimizer’s type is Adam, the number of repetitions is 40, and the sample package size is 32.

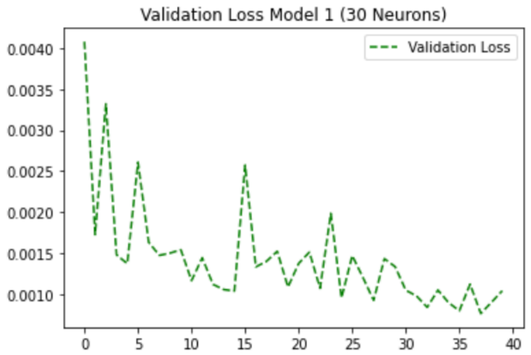
Train Loss Model 1 (30 Neurons)

```
[11] plt.title('Train Loss Model 1 (30 Neurons)')
     plt.plot(history1_30.history['loss'], '--', label='Train Loss', color='g')
     plt.legend()
     plt.show()
```



Validation Loss Model 1 (30 Neurons)

```
plt.title('Validation Loss Model 1 (30 Neurons)')
plt.plot(history1_30.history['val_loss'], '--', label='Validation Loss', color='g')
plt.legend()
plt.show()
```



In these two figures, the error's fluctuations on data validation while training and testing are shown.

Prediction with Model 1 (30 Neurons)

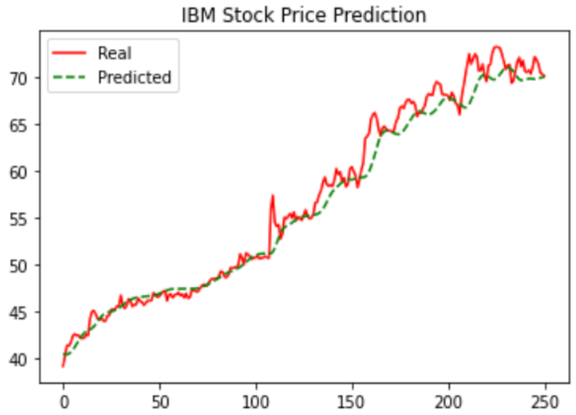
```
[13] df2 = pd.concat((df["High"][:'2016'], df["High"]['2017':]), axis=0)
     inputs = df2[len(df2) - len(test_set) - PERIOD:].values
     inputs = inputs.reshape(-1,1)
     inputs = scaler.transform(inputs)

     X_test = []
     for i in range(PERIOD, len(df['2017':]) + PERIOD):
         X_test.append(inputs[i-PERIOD:i, 0])

     X_test = np.array(X_test)
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
     predict = model1_30.predict(X_test)
     predict = scaler.inverse_transform(predict)

     plt.plot(test_set, color='r',label='Real')
     plt.plot(predict, '--', color='g',label='Predicted')
     plt.title('IBM Stock Price Prediction')
     plt.legend()
     plt.show()
```

```
8/8 [==============================] - 1s 6ms/step
```



In this figure, the model's expected price diagram is shown.

```
[14] mse = 0
     for i in range(len(test_set)):
         mse += (test_set[i] - predict[i]) ** 2

     mse / len(test_set)
```

```
array([2.45258993])
```

This function is used to estimate the MSE error in each model.

The MSE error for this model is shown

MSE = 2/45

Train Model 1 (50 Neurons)

```
[15] model1_50 = Sequential()

     model1_50.add(LSTM(units=50, return_sequences=True))
     model1_50.add(Dropout(0.5))

     model1_50.add(LSTM(units=50, return_sequences=True))
     model1_50.add(Dropout(0.5))

     model1_50.add(LSTM(units=50))
     model1_50.add(Dropout(0.5))

     model1_50.add(Dense(units=1))

     model1_50.compile(optimizer='rmsprop',loss='mean_squared_error')

     history1_50 = model1_50.fit(
         X_train,
         y_train,
         epochs=40,
         batch_size=32,
         validation_data=(X_train_val, y_train_val)
         )
```

44s

```
Epoch 12/40
60/60 [==============================] - 1s 15ms/step - loss: 0.0047 - val_loss: 0.0070
Epoch 13/40
60/60 [==============================] - 1s 15ms/step - loss: 0.0040 - val_loss: 0.0049
Epoch 14/40
60/60 [==============================] - 1s 14ms/step - loss: 0.0043 - val_loss: 0.0046
Epoch 15/40
60/60 [==============================] - 1s 15ms/step - loss: 0.0036 - val_loss: 0.0022
Epoch 16/40
60/60 [==============================] - 1s 14ms/step - loss: 0.0037 - val_loss: 8.2472e-04
Epoch 17/40
```

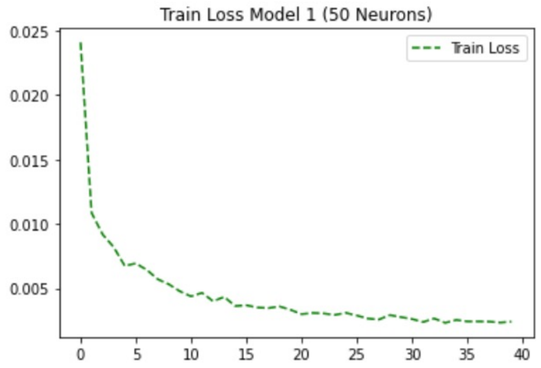
Training the neural network model 1 with 50 neurons per layer

In order to halter overfitting in each layer from happening, the code “Dropout” is used.

The optimizer’s type is Adam, the number of repetitions is 40, and the sample package size is 32.

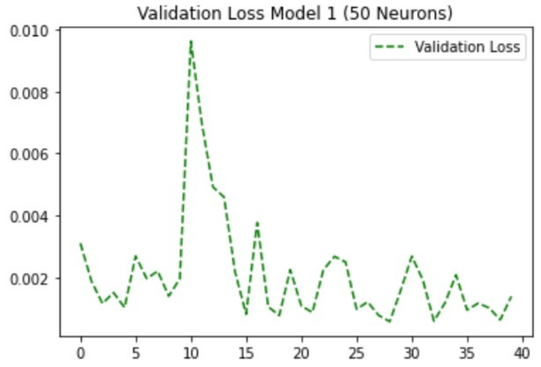
Train Loss Model 1 (50 Neurons)

```
[16] plt.title('Train Loss Model 1 (50 Neurons)')
     plt.plot(history1_50.history['loss'], '--', label='Train Loss', color='g')
     plt.legend()
     plt.show()
```



Validation Loss Model 1 (50 Neurons)

```
[17] plt.title('Validation Loss Model 1 (50 Neurons)')
     plt.plot(history1_50.history['val_loss'], '--', label='Validation Loss', color='g')
     plt.legend()
     plt.show()
```



In these two figures, the error's fluctuations on data validation while training and testing are shown.

Prediction with Model 1 (50 Neurons)

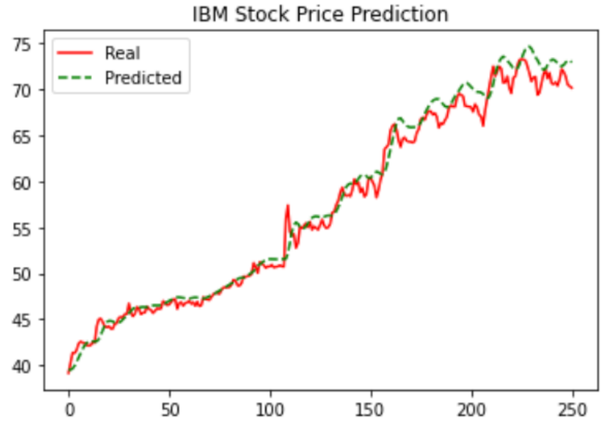
```
[18] df2 = pd.concat((df["High"][:'2016'], df["High"]['2017':]), axis=0)
     inputs = df2[len(df2) - len(test_set) - PERIOD:].values
     inputs = inputs.reshape(-1,1)
     inputs = scaler.transform(inputs)

     X_test = []
     for i in range(PERIOD, len(df['2017':]) + PERIOD):
         X_test.append(inputs[i-PERIOD:i, 0])

     X_test = np.array(X_test)
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
     predict = model1_50.predict(X_test)
     predict = scaler.inverse_transform(predict)

     plt.plot(test_set, color='r',label='Real')
     plt.plot(predict, '--', color='g',label='Predicted')
     plt.title('IBM Stock Price Prediction')
     plt.legend()
     plt.show()
```

```
8/8 [==============================] - 1s 6ms/step
```



In this figure, the model's expected price diagram is shown.

Mean Squared Error for Model 1 (50 Neurons)

```
[19] mse = 0
     for i in range(len(test_set)):
         mse += (test_set[i] - predict[i]) ** 2

     mse / len(test_set)

     array([1.91632002])
```

This function is used to estimate the MSE error in each model.

The MSE error for this model is shown

MSE = 1/96

This shows that as the number of neurons with other constant situations increases, the model's error decreases.

Model 2

(Variable Number of Layers)

Train Model 2 (5 Layers)

```
[20] model2_5 = Sequential()

     model2_5.add(LSTM(units=50, return_sequences=True))
     model2_5.add(Dropout(0.5))

     model2_5.add(LSTM(units=50, return_sequences=True))
     model2_5.add(Dropout(0.5))

     model2_5.add(LSTM(units=50, return_sequences=True))
     model2_5.add(Dropout(0.5))

     model2_5.add(LSTM(units=50, return_sequences=True))
     model2_5.add(Dropout(0.5))

     model2_5.add(LSTM(units=50))
     model2_5.add(Dropout(0.5))

     model2_5.add(Dense(units=1))

     model2_5.compile(optimizer='rmsprop',loss='mean_squared_error')

     history2_5 = model2_5.fit(
         X_train,
         y_train,
         epochs=50,
         batch_size=32,
         validation_data=(X_train_val, y_train_val)
         )
```

```
60/60 [==============================] - 1s 23ms/step - loss: 0.0045 - val_loss: 8.6602e-04
Epoch 22/50
60/60 [==============================] - 1s 23ms/step - loss: 0.0044 - val_loss: 0.0011
Epoch 23/50
60/60 [==============================] - 1s 23ms/step - loss: 0.0040 - val_loss: 0.0011
Epoch 24/50
60/60 [==============================] - 1s 23ms/step - loss: 0.0041 - val_loss: 0.0017
Epoch 25/50
60/60 [==============================] - 1s 22ms/step - loss: 0.0042 - val_loss: 0.0010
Epoch 26/50
```

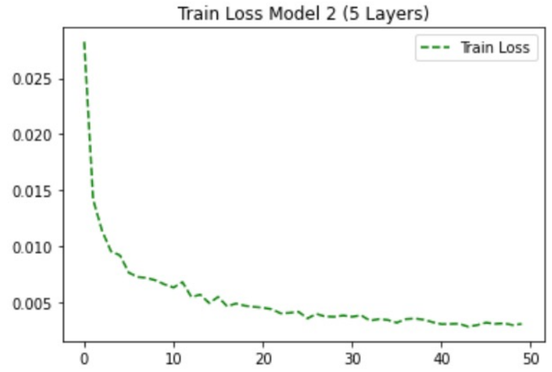
Training the neural network model 2 with 5 layers

In order to halter overfitting in each layer from happening, the code “Dropout” is used.

The optimizer’s type is rmsprop, the number of repetitions is 40, and the sample package size is 32.

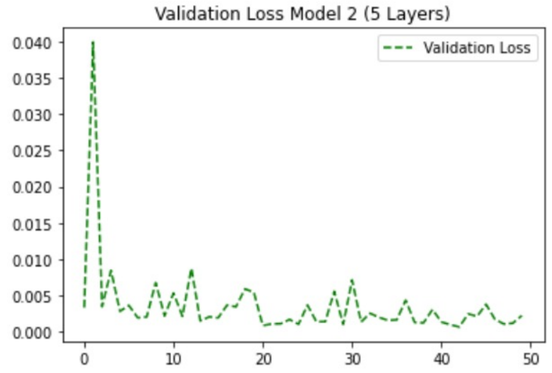
Train Loss Model 2 (5 Layers)

```
[21] plt.title('Train Loss Model 2 (5 Layers)')
     plt.plot(history2_5.history['loss'], '--', label='Train Loss', color='g')
     plt.legend()
     plt.show()
```



Validation Loss Model 2 (5 Layers)

```
[22] plt.title('Validation Loss Model 2 (5 Layers)')
     plt.plot(history2_5.history['val_loss'], '--', label='Validation Loss', color='g')
     plt.legend()
     plt.show()
```



In this two figures, the error's fluctuations on data validation while training and testing are shown.

Prediction with Model 2 (5 Layers)

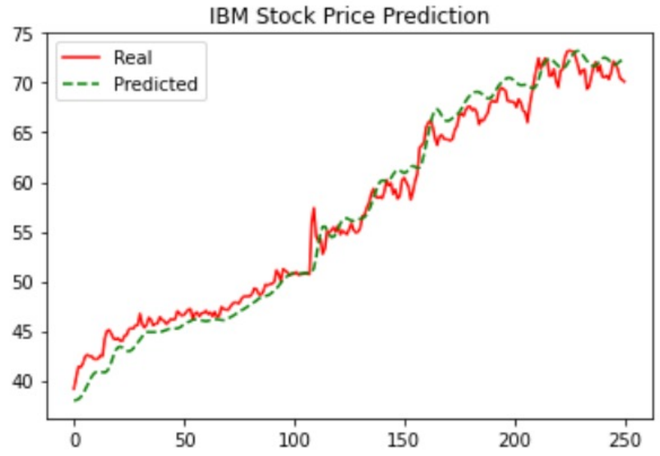
```
[23] df2 = pd.concat((df["High"][:'2016'], df["High"]['2017':]), axis=0)
     inputs = df2[len(df2) - len(test_set) - PERIOD:].values
     inputs = inputs.reshape(-1,1)
     inputs = scaler.transform(inputs)

     X_test = []
     for i in range(PERIOD, len(df['2017':]) + PERIOD):
         X_test.append(inputs[i-PERIOD:i, 0])

     X_test = np.array(X_test)
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
     predict = model2_5.predict(X_test)
     predict = scaler.inverse_transform(predict)

     plt.plot(test_set, color='r',label='Real')
     plt.plot(predict, '--', color='g',label='Predicted')
     plt.title('IBM Stock Price Prediction')
     plt.legend()
     plt.show()
```

```
8/8 [==============================] - 2s 10ms/step
```



In this figure, the model's expected price diagram is shown.

```
[24] mse = 0
     for i in range(len(test_set)):
         mse += (test_set[i] - predict[i]) ** 2

     mse / len(test_set)
```

0s

```
array([2.58759116])
```

This function is used to estimate the MSE error in each model.

The MSE error for this model is shown

MSE = 2/58

Train Model 2 (6 Layers)

```
[40] model2_6 = Sequential()

     model2_6.add(LSTM(units=50, return_sequences=True))
     model2_6.add(Dropout(0.5))

     model2_6.add(LSTM(units=50, return_sequences=True))
     model2_6.add(Dropout(0.5))

     model2_6.add(LSTM(units=50, return_sequences=True))
     model2_6.add(Dropout(0.5))

     model2_6.add(LSTM(units=50, return_sequences=True))
     model2_6.add(Dropout(0.5))

     model2_6.add(LSTM(units=50, return_sequences=True))
     model2_6.add(Dropout(0.5))

     model2_6.add(LSTM(units=50))
     model2_6.add(Dropout(0.5))

     model2_6.add(Dense(units=1))

     model2_6.compile(optimizer='rmsprop',loss='mean_squared_error')

     history2_6 = model2_6.fit(
         X_train,
         y_train,
         epochs=50,
         batch_size=32,
         validation_data=(X_train_val, y_train_val)
         )
```

```
Epoch 22/50
60/60 [==============================] - 1s 25ms/step - loss: 0.0051 - val_loss: 0.0011
Epoch 23/50
60/60 [==============================] - 1s 24ms/step - loss: 0.0044 - val_loss: 9.4309e-04
Epoch 24/50
60/60 [==============================] - 2s 25ms/step - loss: 0.0047 - val_loss: 0.0017
Epoch 25/50
60/60 [==============================] - 1s 25ms/step - loss: 0.0044 - val_loss: 0.0027
Epoch 26/50
60/60 [==============================] - 1s 24ms/step - loss: 0.0043 - val_loss: 0.0027
Epoch 27/50
60/60 [==============================] - 1s 24ms/step - loss: 0.0042 - val_loss: 0.0063
```

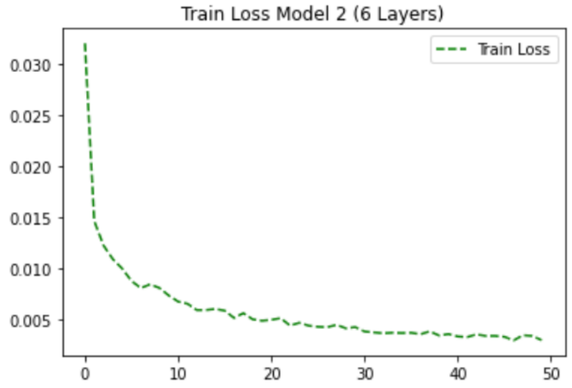
Training the neural network model 2 with 6 layers

In order to halter overfitting in each layer from happening, the code “Dropout” is used.

The optimizer’s type is rmsprop, the number of repetitions is 40, and the sample package size is 32.

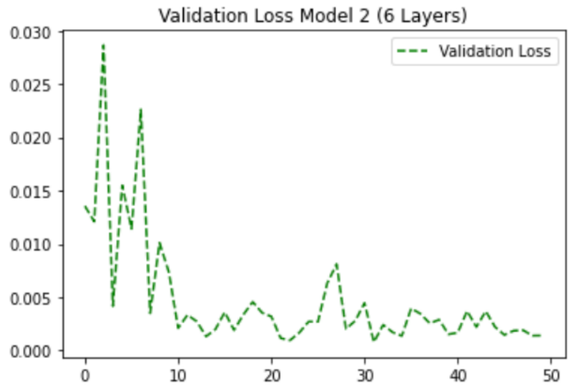
Plot Train Loss Model 2(6 Layers)

```
[41] plt.title('Train Loss Model 2 (6 Layers)')
     plt.plot(history2_6.history['loss'], '--', label='Train Loss', color='g')
     plt.legend()
     plt.show()
```



Plot Validation Loss Model 2 (6 Layers)

```
plt.title('Validation Loss Model 2 (6 Layers)')
plt.plot(history2_6.history['val_loss'], '--', label='Validation Loss', color='g')
plt.legend()
plt.show()
```



In these two figures, the error's fluctuations on data validation while training and testing are shown.

Prediction with Model (6 Layers) 's)

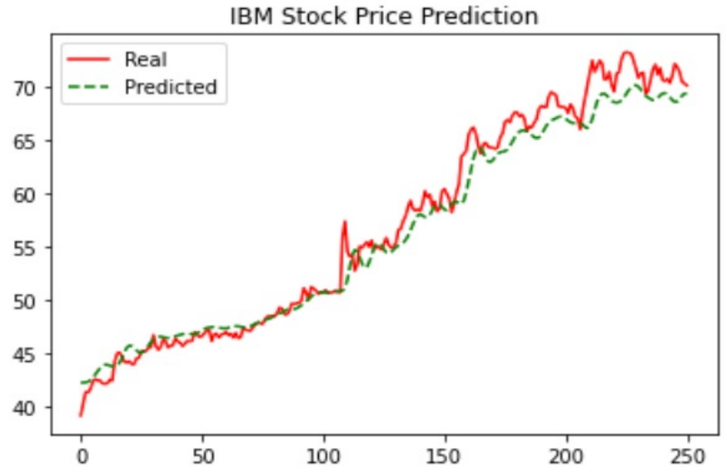
```
[43] df2 = pd.concat((df["High"][:'2016'], df["High"]['2017':]), axis=0)
     inputs = df2[len(df2) - len(test_set) - PERIOD:].values
     inputs = inputs.reshape(-1,1)
     inputs = scaler.transform(inputs)

     X_test = []
     for i in range(PERIOD, len(df['2017':]) + PERIOD):
         X_test.append(inputs[i-PERIOD:i, 0])

     X_test = np.array(X_test)
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
     predict = model2_6.predict(X_test)
     predict = scaler.inverse_transform(predict)

     plt.plot(test_set, color='r',label='Real')
     plt.plot(predict, '--', color='g',label='Predicted')
     plt.title('IBM Stock Price Prediction')
     plt.legend()
     plt.show()
```

```
8/8 [==============================] - 2s 11ms/step
```



In this figure, the model's expected price diagram is shown.

```
[44] mse = 0
     for i in range(len(test_set)):
         mse += (test_set[i] - predict[i]) ** 2

     mse / len(test_set)

     array([3.01583832])
```

This function is used to estimate the MSE error in each model.

The MSE error for this model is shown

MSE = 3/01

This shows that as the number of layers increases, the error increases slightly. However,

It is possible that the result change with another training.

Model 3

(Variable batch size)

Train Model 3 (Batch Size = 32)

```
[30] model3_32 = Sequential()

    model3_32.add(LSTM(units=50, return_sequences=True))
    model3_32.add(Dropout(0.5))

    model3_32.add(LSTM(units=50, return_sequences=True))
    model3_32.add(Dropout(0.5))

    model3_32.add(LSTM(units=50, return_sequences=True))
    model3_32.add(Dropout(0.5))

    model3_32.add(LSTM(units=50))
    model3_32.add(Dropout(0.5))

    model3_32.add(Dense(units=1))

    model3_32.compile(optimizer='rmsprop',loss='mean_squared_error')

    history3_32 = model3_32.fit(
        X_train,
        y_train,
        epochs=40,
        batch_size=32,
        validation_data=(X_train_val, y_train_val)
        )
```

```
Epoch 12/40
60/60 [==============================] - 1s 18ms/step - loss: 0.0054 - val_loss: 0.0081
Epoch 13/40
60/60 [==============================] - 1s 17ms/step - loss: 0.0050 - val_loss: 0.0011
Epoch 14/40
60/60 [==============================] - 1s 18ms/step - loss: 0.0046 - val_loss: 0.0012
Epoch 15/40
60/60 [==============================] - 1s 20ms/step - loss: 0.0048 - val_loss: 0.0053
Epoch 16/40
60/60 [==============================] - 1s 18ms/step - loss: 0.0043 - val_loss: 0.0012
Epoch 17/40
60/60 [==============================] - 1s 18ms/step - loss: 0.0041 - val_loss: 0.0017
Epoch 18/40
60/60 [==============================] - 1s 21ms/step - loss: 0.0042 - val_loss: 0.0021
Epoch 19/40
60/60 [==============================] - 1s 17ms/step - loss: 0.0037 - val_loss: 0.0014
```

Training the neural network model 3 with sample package size 32.

In order to halter overfitting in each layer from happening, the code “Dropout” is used.

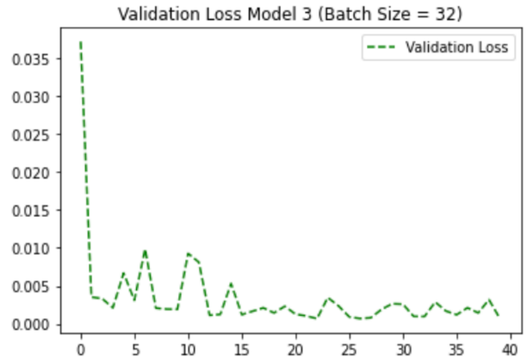
The optimizer’s type is rmsprop and the number of repetitions is 40.

Train Loss Model 3 (Batch Size = 32)

```
[32] plt.title('Train Loss Model 3 (Batch Size = 32)')
     plt.plot(history3_32.history['loss'], '--', label='Train Loss', color='g')
     plt.legend()
     plt.show()
```

Validation Loss Model 3 (Batch Size = 32)

```
plt.title('Validation Loss Model 3 (Batch Size = 32)')
plt.plot(history3_32.history['val_loss'], '--', label='Validation Loss', color='g')
plt.legend()
plt.show()
```



In these two figures, the error's fluctuations on data validation while training and testing are shown.

Prediction With Model 3 (Batch Size = 32)

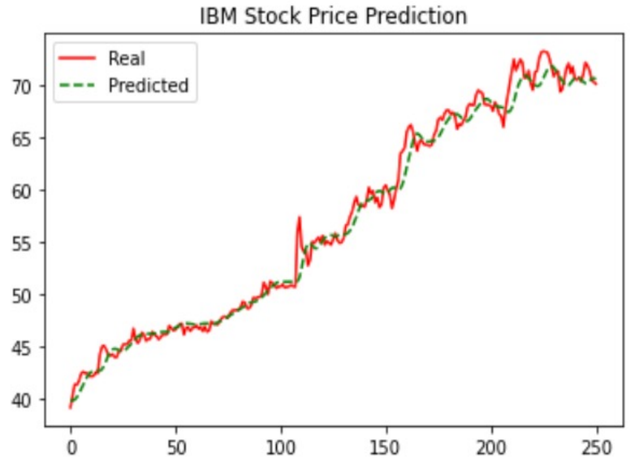
```
[31] df2 = pd.concat((df["High"][:'2016'], df["High"]['2017':]), axis=0)
     inputs = df2[len(df2) - len(test_set) - PERIOD:].values
     inputs = inputs.reshape(-1,1)
     inputs = scaler.transform(inputs)

     X_test = []
     for i in range(PERIOD, len(df['2017':]) + PERIOD):
         X_test.append(inputs[i-PERIOD:i, 0])

     X_test = np.array(X_test)
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
     predict = model3_32.predict(X_test)
     predict = scaler.inverse_transform(predict)

     plt.plot(test_set, color='r',label='Real')
     plt.plot(predict, '--', color='g',label='Predicted')
     plt.title('IBM Stock Price Prediction')
     plt.legend()
     plt.show()
```

```
8/8 [==============================] - 1s 7ms/step
```



In this figure, the model's expected price diagram is shown.

```
[34] mse = 0
     for i in range(len(test_set)):
         mse += (test_set[i] - predict[i]) ** 2

     mse / len(test_set)
```

```
array([1.35620713])
```

This function is used to estimate the MSE error in each model.

The MSE error for this model is shown

MSE = 1/35

Train Model 3 (Batch Size = 16)

```
model3_16 = Sequential()

model3_16.add(LSTM(units=50, return_sequences=True))
model3_16.add(Dropout(0.5))

model3_16.add(LSTM(units=50, return_sequences=True))
model3_16.add(Dropout(0.5))

model3_16.add(LSTM(units=50, return_sequences=True))
model3_16.add(Dropout(0.5))

model3_16.add(LSTM(units=50))
model3_16.add(Dropout(0.5))

model3_16.add(Dense(units=1))

model3_16.compile(optimizer='rmsprop',loss='mean_squared_error')

history3_16 = model3_16.fit(
    X_train,
    y_train,
    epochs=40,
    batch_size=16,
    validation_data=(X_train_val, y_train_val)
    )
```

```
Epoch 12/40
119/119 [==============================] - 2s 19ms/step - loss: 0.0042 - val_loss: 8.3002e-04
Epoch 13/40
119/119 [==============================] - 2s 19ms/step - loss: 0.0039 - val_loss: 7.8406e-04
Epoch 14/40
119/119 [==============================] - 2s 19ms/step - loss: 0.0036 - val_loss: 0.0016
Epoch 15/40
119/119 [==============================] - 2s 19ms/step - loss: 0.0035 - val_loss: 9.5607e-04
Epoch 16/40
119/119 [==============================] - 2s 19ms/step - loss: 0.0033 - val_loss: 0.0034
Epoch 17/40
119/119 [==============================] - 2s 19ms/step - loss: 0.0037 - val_loss: 0.0015
```

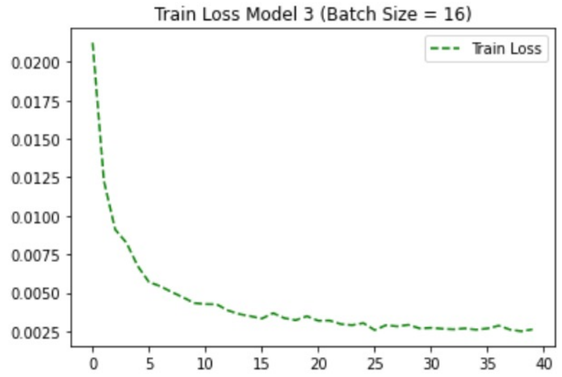
Training the neural network model 3 with sample package size 16.

In order to halter overfitting in each layer from happening, the code “Dropout” is used.

The optimizer’s type is rmsprop and the number of repetitions is 40.

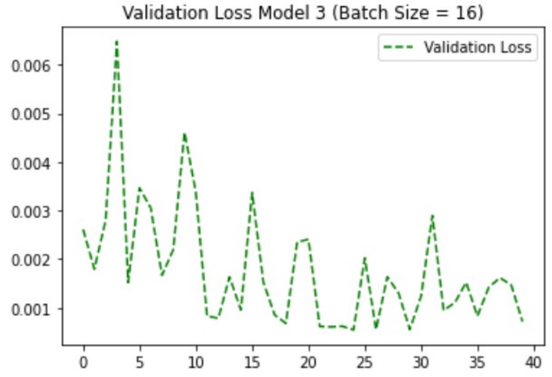
Train Loss Model 3 (Batch Size = 16)

```
[37] plt.title('Train Loss Model 3 (Batch Size = 16)')
     plt.plot(history3_16.history['loss'], '--', label='Train Loss', color='g')
     plt.legend()
     plt.show()
```



Validation Loss Model 3 (Batch Size = 16)

```
[38] plt.title('Validation Loss Model 3 (Batch Size = 16)')
     plt.plot(history3_16.history['val_loss'], '--', label='Validation Loss', color='g')
     plt.legend()
     plt.show()
```



In these two figures, the error's fluctuations on data validation while training and testing are shown.

Prediction with Model 3 (Batch Size = 16)

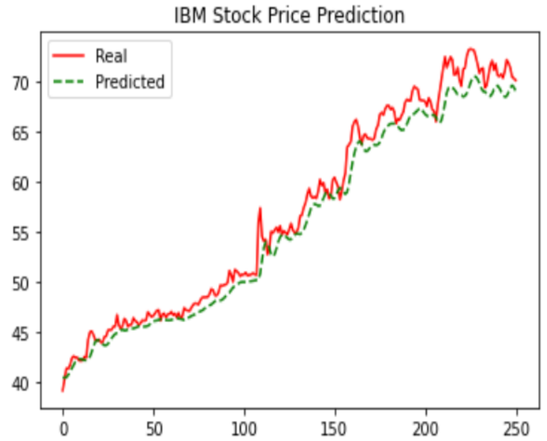
```
[36] df2 = pd.concat((df["High"][:'2016'], df["High"]['2017':]), axis=0)
     inputs = df2[len(df2) - len(test_set) - PERIOD:].values
     inputs = inputs.reshape(-1,1)
     inputs = scaler.transform(inputs)

     X_test = []
     for i in range(PERIOD, len(df['2017':]) + PERIOD):
         X_test.append(inputs[i-PERIOD:i, 0])

     X_test = np.array(X_test)
     X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
     predict = model3_16.predict(X_test)
     predict = scaler.inverse_transform(predict)

     plt.plot(test_set, color='r',label='Real')
     plt.plot(predict, '--', color='g',label='Predicted')
     plt.title('IBM Stock Price Prediction')
     plt.legend()
     plt.show()
```

```
8/8 [==============================] - 1s 6ms/step
```



In this figure, the model's expected price diagram is shown.

```
[39] mse = 0
     for i in range(len(test_set)):
         mse += (test_set[i] - predict[i]) ** 2

     mse / len(test_set)

     array([2.92198569])
```

This function is used to estimate the MSE error in each model.

The MSE error for this model is shown

MSE = 2/92

The errors of the last two models have very slight difference. The packages with size 32 have a better result. However, It is possible that the result change by another training.