DS505: INTRODUCTION TO DEEP LEARNING

P05: RECURRENT NEURAL NETWORK ¶

In [1]:
```
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
```

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 [5]:
```
# First, we get the data
dataset = pd.read_csv('AABA_2006-01-01_to_2018-01-01.csv', index_col='Date', pars
dataset.head()
```

Out[5]:

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

In [6]:
```
# Checking for missing values
training_set = dataset[:'2016'].iloc[:,1:2].values
test_set = dataset['2017':].iloc[:,1:2].values
```

In [7]:
```
# 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 [8]:
```
# Scaling the training set
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

In [10]:
```
# Since LSTMs store long term memory state, we create a data structure with 60 ti
# So for each element of training set, we have 60 previous training set elements
X_train = []
y_train = []
for i in range(60,2768):
    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 [11]:
```
# Reshaping X_train for efficient modelling
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

In [12]:
```
# 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]
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
85/85 [==============================] - 30s 129ms/step - loss: 0.0154
Epoch 2/50
85/85 [==============================] - 11s 126ms/step - loss: 0.0075
Epoch 3/50
85/85 [==============================] - 12s 137ms/step - loss: 0.0061
Epoch 4/50
85/85 [==============================] - 11s 132ms/step - loss: 0.0050
Epoch 5/50
85/85 [==============================] - 11s 124ms/step - loss: 0.0042
Epoch 6/50
85/85 [==============================] - 12s 142ms/step - loss: 0.0041
Epoch 7/50
85/85 [==============================] - 11s 125ms/step - loss: 0.0035
Epoch 8/50
85/85 [==============================] - 11s 129ms/step - loss: 0.0032
Epoch 9/50
85/85 [==============================] - 11s 134ms/step - loss: 0.0032
Epoch 10/50
85/85 [==============================] - 12s 138ms/step - loss: 0.0029
Epoch 11/50
85/85 [==============================] - 11s 132ms/step - loss: 0.0027
Epoch 12/50
85/85 [==============================] - 11s 124ms/step - loss: 0.0026
Epoch 13/50
85/85 [==============================] - 12s 142ms/step - loss: 0.0024
Epoch 14/50
85/85 [==============================] - 11s 129ms/step - loss: 0.0022
Epoch 15/50
85/85 [==============================] - 11s 128ms/step - loss: 0.0022
Epoch 16/50
85/85 [==============================] - 12s 141ms/step - loss: 0.0021
Epoch 17/50
85/85 [==============================] - 11s 130ms/step - loss: 0.0019
```

```
Epoch 18/50
85/85 [==============================] - 11s 130ms/step - loss: 0.0019
Epoch 19/50
85/85 [==============================] - 11s 132ms/step - loss: 0.0019
Epoch 20/50
85/85 [==============================] - 12s 137ms/step - loss: 0.0018
Epoch 21/50
85/85 [==============================] - 11s 129ms/step - loss: 0.0018
Epoch 22/50
85/85 [==============================] - 11s 131ms/step - loss: 0.0017
Epoch 23/50
85/85 [==============================] - 12s 143ms/step - loss: 0.0017
Epoch 24/50
85/85 [==============================] - 11s 131ms/step - loss: 0.0016
Epoch 25/50
85/85 [==============================] - 11s 130ms/step - loss: 0.0016
Epoch 26/50
85/85 [==============================] - 11s 134ms/step - loss: 0.0016
Epoch 27/50
85/85 [==============================] - 12s 137ms/step - loss: 0.0015
Epoch 28/50
85/85 [==============================] - 11s 131ms/step - loss: 0.0015
Epoch 29/50
85/85 [==============================] - 11s 128ms/step - loss: 0.0016
Epoch 30/50
85/85 [==============================] - 12s 144ms/step - loss: 0.0015
Epoch 31/50
85/85 [==============================] - 11s 130ms/step - loss: 0.0014
Epoch 32/50
85/85 [==============================] - 11s 130ms/step - loss: 0.0014
Epoch 33/50
85/85 [==============================] - 12s 136ms/step - loss: 0.0014
Epoch 34/50
85/85 [==============================] - 11s 135ms/step - loss: 0.0014
Epoch 35/50
85/85 [==============================] - 11s 132ms/step - loss: 0.0013
Epoch 36/50
85/85 [==============================] - 11s 129ms/step - loss: 0.0013
Epoch 37/50
85/85 [==============================] - 12s 146ms/step - loss: 0.0013
Epoch 38/50
85/85 [==============================] - 11s 124ms/step - loss: 0.0012
Epoch 39/50
85/85 [==============================] - 11s 127ms/step - loss: 0.0013
Epoch 40/50
85/85 [==============================] - 12s 144ms/step - loss: 0.0013
Epoch 41/50
85/85 [==============================] - 11s 134ms/step - loss: 0.0013
Epoch 42/50
85/85 [==============================] - 11s 130ms/step - loss: 0.0013
Epoch 43/50
85/85 [==============================] - 11s 125ms/step - loss: 0.0011
Epoch 44/50
85/85 [==============================] - 11s 134ms/step - loss: 0.0012
Epoch 45/50
85/85 [==============================] - 11s 129ms/step - loss: 0.0012
Epoch 46/50
```

```
85/85 [==============================] - 11s 130ms/step - loss: 0.0012
Epoch 47/50
85/85 [==============================] - 11s 125ms/step - loss: 0.0012
Epoch 48/50
85/85 [==============================] - 11s 129ms/step - loss: 0.0011
Epoch 49/50
85/85 [==============================] - 11s 126ms/step - loss: 0.0012
Epoch 50/50
85/85 [==============================] - 12s 141ms/step - loss: 0.0012
```

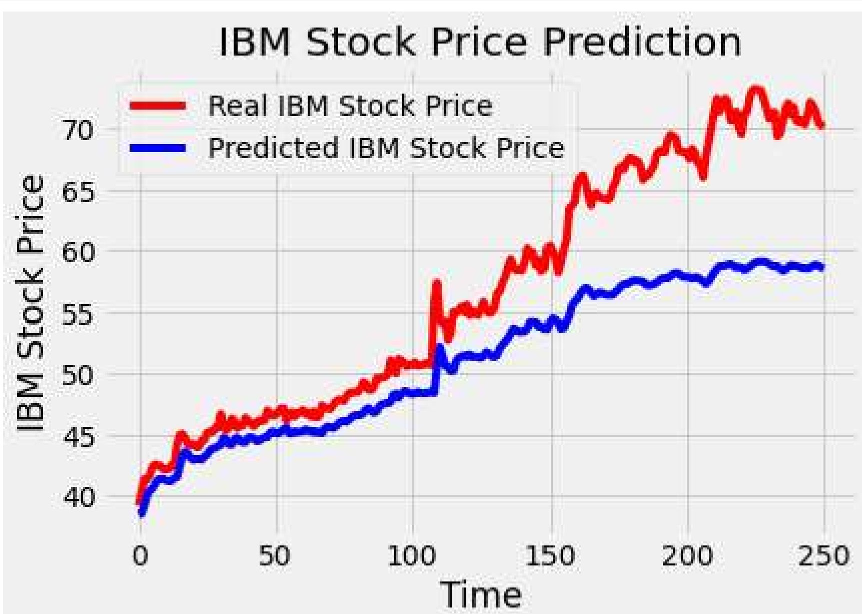
Out[12]:
```
<keras.callbacks.History at 0x2a50a81d6d0>
```

In [13]:
```
# 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 va
# 'High' attribute data for processing
dataset_total = pd.concat((dataset["High"][:'2016'],dataset["High"]['2017':]),axi
inputs = dataset_total[len(dataset_total)-len(test_set) - 60:].values
inputs = inputs.reshape(-1,1)
inputs  = sc.transform(inputs)
```

In [14]:
```
# 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)
```

```
8/8 [==============================] - 3s 49ms/step
```

In [15]:
```
# Visualizing the results for LSTM
plot_predictions(test_set,predicted_stock_price)
```



In [16]:
```
# Evaluating our model
return_rmse(test_set,predicted_stock_price)
```

```
The root mean squared error is 7.17938889158362.
```

In [ ]: