LP-5 DEEP LEARNING Practical 4

Recurrent neural network (RNN)

Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.

Name: **ONASVEE BANARSE**

Rollno: **09**

BE COMP 1

1. Import library

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

3. Data processing

3.0 import the data

In [2]:
```
dataset_train = pd.read_csv('Google_Stock_Price_Train.csv')
```

In [3]:
```
dataset_train.head()
```

Out[3]:

	Date	Open	High	Low	Close	Volume
0	01/03/2012	325.25	332.83	324.97	663.59	7,380,500
1	01/04/2012	331.27	333.87	329.08	666.45	5,749,400
2	01/05/2012	329.83	330.75	326.89	657.21	6,590,300
3	01/06/2012	328.34	328.77	323.68	648.24	5,405,900
4	01/09/2012	322.04	322.29	309.46	620.76	11,688,800

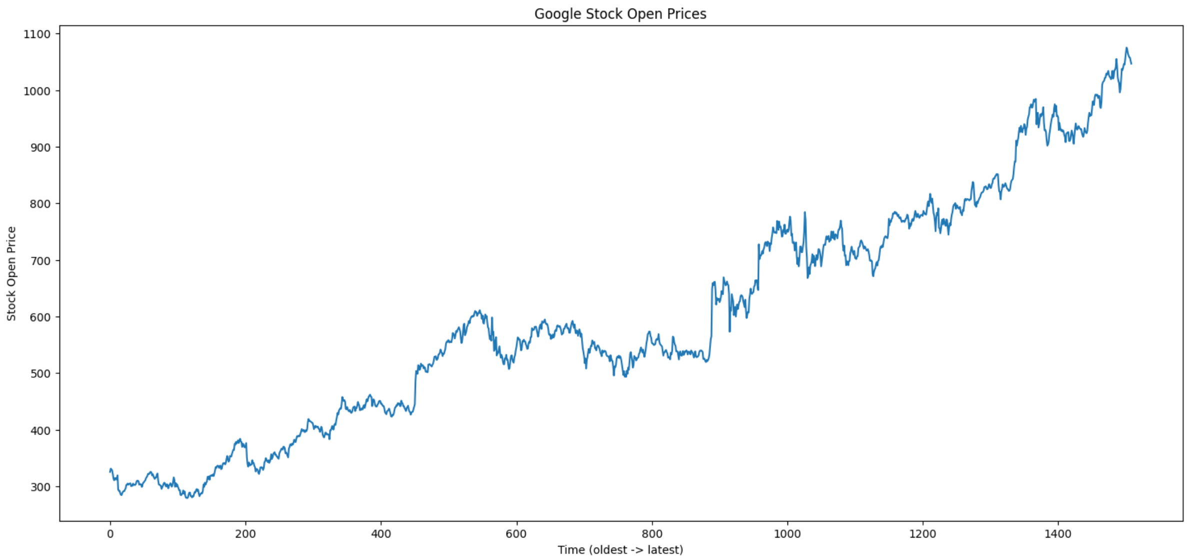
In [4]:
```
#keras only takes numpy array
training_set = dataset_train.iloc[:, 1: 2].values
```

In [5]:
```
training_set.shape
```

Out[5]: (1509, 1)

In [6]:
```
plt.figure(figsize=(18, 8))
plt.plot(dataset_train['Open'])
plt.title("Google Stock Open Prices")
plt.xlabel("Time (oldest -> latest)")
```

```
plt.ylabel("Stock Open Price")
plt.show()
```



3.1 Feature scaling

In [7]:
```
import os
if os.path.exists('config.py'):
    print(1)
else:
    print(0)
```

```
0
```

In [8]:
```
sc = MinMaxScaler(feature_range = (0, 1))
#fit: get min/max of train data
training_set_scaled = sc.fit_transform(training_set)
```

3.2 Data structure creation

- taking the reference of past 60 days of data to predict the future stock price.
- It is observed that taking 60 days of past data gives us best results.
- In this data set 60 days of data means 3 months of data.
- Every month as 20 days of Stock price.
- X train will have data of 60 days prior to our date and y train will have data of one day after our date

In [9]:
```
## 60 timesteps and 1 output
X_train = []
y_train = []
for i in range(60, len(training_set_scaled)):
    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 [10]:
```
X_train.shape
```

Out[10]:
```
(1449, 60)
```

In [11]:
```
y_train.shape
```

Out[11]:
```
(1449,)
```

3.3 Data reshaping

In [12]:
```
X_train = np.reshape(X_train, newshape =
                     (X_train.shape[0], X_train.shape[1], 1))
```

1. Number of stock prices - 1449
2. Number of time steps - 60
3. Number of Indicator - 1

In [13]:
```
X_train.shape
```

Out[13]:
```
(1449, 60, 1)
```

4. Create & Fit Model

4.1 Create model

In [14]:
```
regressor = Sequential()
#add 1st lstm layer
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shap
regressor.add(Dropout(rate = 0.2))

##add 2nd lstm layer: 50 neurons
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(rate = 0.2))

##add 3rd lstm layer
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(rate = 0.2))

##add 4th lstm layer
regressor.add(LSTM(units = 50, return_sequences = False))
regressor.add(Dropout(rate = 0.2))

##add output layer
regressor.add(Dense(units = 1))
```

In [39]:
```
regressor.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 60, 50)            10400     
                                                                 
 dropout (Dropout)           (None, 60, 50)            0         
                                                                 
 lstm_1 (LSTM)               (None, 60, 50)            20200     
                                                                 
 dropout_1 (Dropout)         (None, 60, 50)            0         
                                                                 
 lstm_2 (LSTM)               (None, 60, 50)            20200     
                                                                 
 dropout_2 (Dropout)         (None, 60, 50)            0         
                                                                 
 lstm_3 (LSTM)               (None, 50)                20200     
                                                                 
 dropout_3 (Dropout)         (None, 50)                0         
                                                                 
 dense (Dense)               (None, 1)                 51        
                                                                 
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

4.2 Model fit

In [ ]:
```
regressor.fit(x = X_train, y = y_train, batch_size = 32, epochs = 100)
```

4.3 Model evaluation

4.3.1 Read and convert

In [28]:
```
dataset_test = pd.read_csv('Google_Stock_Price_Test.csv')
```

In [29]:
```
dataset_test.head()
```

Out[29]:

	Date	Open	High	Low	Close	Volume
0	02/01/2018	1048.339966	1066.939941	1045.229980	1065.000000	1237600
1	03/01/2018	1064.310059	1086.290039	1063.209961	1082.479980	1430200
2	04/01/2018	1088.000000	1093.569946	1084.001953	1086.400024	1004600
3	05/01/2018	1094.000000	1104.250000	1092.000000	1102.229980	1279100
4	08/01/2018	1102.229980	1111.270020	1101.619995	1106.939941	1047600

In [30]:
```
#keras only takes numpy array
real_stock_price = dataset_test.iloc[:, 1: 2].values
real_stock_price.shape
```

Out[30]:
```
(125, 1)
```

4.3.2 Concat and convert

In [ ]:
```
#vertical concat use 0, horizontal uses 1
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']),
                          axis = 0)
##use .values to make numpy array
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 60:].values
```

4.3.3 Reshape and scale

In [32]:
```
#reshape data to only have 1 col
inputs = inputs.reshape(-1, 1)

#scale input
inputs = sc.transform(inputs)
```

In [33]:
```
len(inputs)
```

Out[33]:
```
185
```

4.3.4 Create test data strucutre

In [34]:
```
X_test = []
for i in range(60, len(inputs)):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
#add dimension of indicator
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

In [35]:
```
X_test.shape
```

Out[35]:
```
(125, 60, 1)
```

4.3.5 Model prediction

In [36]:
```
predicted_stock_price = regressor.predict(X_test)
```

```
4/4 [==============================] - 0s 20ms/step
```

In [37]:
```
#inverse the scaled value
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

4.3.6 Result visualization

In [38]:
```
##visualize the prediction and real price
plt.plot(real_stock_price, color = 'red', label = 'Real price')
plt.plot(predicted_stock_price, color = 'blue', label = 'Predicted price')

plt.title('Google price prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
```


Google price prediction
1175
1150
1125
1100
1075
1050
1025
1000
Price
0
20
40
60
80
100
120
Time
Real price
Predicted price

In [ ]: