Name: Nithin Dsouza
Batch ID: 05062021-9AM (weekend)
Topic: Recurrent Neural network-LSTMs and GRUs

Problem1

Importing required libraries

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
```

Fix random seed for reproducibility

```
numpy.random.seed(10)
```

Define the raw dataset as given in problem statement

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Create mapping of characters to integers (0-25) and the reverse

```
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
```

Prepare the dataset of input to output pairs encoded as integers

```
seq_length = 1
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
 seq_in = alphabet[i:i + seq_length]
 seq_out = alphabet[i + seq_length]
 dataX.append([char_to_int[char] for char in seq_in])
 dataY.append(char_to_int[seq_out])
 print(seq_in, '->', seq_out)
```

Reshape X to be [samples, time steps, features]

```
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
# normalize
X = X / float(len(alphabet))
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
```

Create and fit the model

```
model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=500, batch_size=1, verbose=2)
```

Summarize performance of the model

```
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
```

Demonstrate some model predictions

```
for pattern in dataX:
 x = numpy.reshape(pattern, (1, len(pattern), 1))
 x = x / float(len(alphabet))
 prediction = model.predict(x, verbose=0)
 index = numpy.argmax(prediction)
 result = int_to_char[index]
 seq_in = [int_to_char[value] for value in pattern]
 print(seq_in, "->", result)
```

Problem 2
Dataset is APPLE Company Sock Price Prediction Data

Required libraries

```
import pandas as pd
import  os
```

Loading dataset

```
df=pd.read_csv(os.getcwd() + os.path.sep + ‘AAPL.csv’)
df.head()
df.tail()

df1=df.reset_index()['close']
df1

import matplotlib.pyplot as plt
plt.plot(df1)

import numpy as np
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))

print(df1)
```

Splitting dataset into train and test split

```
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]

training_size,test_size

train_data
```

```
import numpy
```

Convert an array of values into a dataset matrix

```
def create_dataset(dataset, time_step=1):
  dataX, dataY = [], []
  for i in range(len(dataset)-time_step-1):
    a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
    dataX.append(a)
    dataY.append(dataset[i + time_step, 0])
  return numpy.array(dataX), numpy.array(dataY)

 # reshape into X=t,t+1,t+2,t+3 and Y=t+4
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)

print(X_train.shape), print(y_train.shape)

print(X_test.shape), print(ytest.shape)
```

Reshape input to be [samples, time steps, features] which is required for LSTM

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

Create the Stacked LSTM model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM

model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')

model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=100,batch_size=64,verbose=1)
import tensorflow as tf

tf.__version__
```

Lets Do the prediction and check performance metrics

```
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

Transformback to original form

```
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

Calculate RMSE performance metrics

```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

Test Data RMSE

```
math.sqrt(mean_squared_error(ytest,test_predict))
```

Plotting

```
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()

len(test_data)

x_input=test_data[341:].reshape(1,-1)
x_input.shape


temp_input=list(x_input)
temp_input=temp_input[0].tolist()

temp_input
```

Demonstrate prediction for next 10 days

```
from numpy import array

lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(temp_input)>100):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1
print(lst_output)
```

Predicting nexture future 30days

```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)

import matplotlib.pyplot as plt

len(df1)

plt.plot(day_new,scaler.inverse_transform(df1[1158:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))

df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[1200:])

df3=scaler.inverse_transform(df3).tolist()

plt.plot(df3)
```