**Module 11**

**Name: Upadhyay Sachin Naresh**

**Batch id: AIWDEOB 300821**

**Topic: Recurrent neural network – LSTMs and GRUs**

**1.Write an LSTM program to predict next alphabet in the sequence “A B C D E F G H I J K L M N O P Q R S T U V W X Y Z”**

**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)

**2.Build a LSTM to predict the stock price for multivariate data(use multiple features).Use any online website or tiingo to get the dataset.**

Importing the Libraries required

import pandas as pd

import matplotlib.pyplot as plt import numpy as np

import math

from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import mean\_squared\_error from keras.models import Sequential

from keras.layers import Dense, Activation from keras.layers import LSTM

# Defining the function for pre processing the data

import numpy as np

**def** new\_dataset(dataset, step\_size): data\_X, data\_Y = [], []

**for** i **in** range(len(dataset)-step\_size-1): a = dataset[i:(i+step\_size), 0] data\_X.append(a)

data\_Y.append(dataset[i + step\_size, 0])

**return** np.array(data\_X), np.array(data\_Y) np.random.seed(7)

# Loading the data set

dataset = pd.read\_csv('C:/Users/usach/Desktop/AI assignments/Module 11/Apple\_Share\_Price.csv', usecols=[1,2,3,4])

dataset = dataset.reindex(index = dataset.index[::-1])

# Creating the index for flexibility

obs = np.arange(1, len(dataset) + 1, 1)

# Taking Different Indicators for Prediction

OHLC\_avg = dataset.mean(axis = 1)

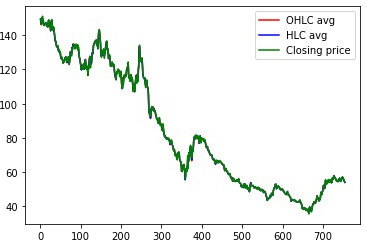
HLC\_avg = dataset[['High', 'Low', 'Close']].mean(axis = 1) close\_val = dataset[['Close']]

# Plotting all the Indicators in One Plot

plt.plot(obs, OHLC\_avg, 'r', label = 'OHLC avg') plt.plot(obs, HLC\_avg, 'b', label = 'HLC avg')

plt.plot(obs, close\_val, 'g', label = 'Closing price') plt.legend(loc = 'upper right')

plt.show()



# Preparing the time series data

OHLC\_avg = np.reshape(OHLC\_avg.values, (len(OHLC\_avg),1)) *# 1664*

scaler = MinMaxScaler(feature\_range=(0, 1)) OHLC\_avg = scaler.fit\_transform(OHLC\_avg)

from sklearn import preprocessing

# Splitting the data set into training and testing data

train\_OHLC = int(len(OHLC\_avg) \* 0.75) test\_OHLC = len(OHLC\_avg) - train\_OHLC

train\_OHLC, test\_OHLC = OHLC\_avg[0:train\_OHLC,:], OHLC\_avg[train\_OHLC:len(OHLC\_avg),:]

trainX, trainY = new\_dataset(train\_OHLC, 1) testX, testY = new\_dataset(test\_OHLC, 1)

trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1])) testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1])) step\_size = 1

# Defining the LSTM Model

model = Sequential()

model.add(LSTM(32, input\_shape=(1, step\_size), return\_sequences = True))

model.add(LSTM(16,activation = 'relu'))

model.add(Dense(1)) model.add(Activation('linear'))

# Compiling and training the model

model.compile(loss='mean\_squared\_error', optimizer='adam') model.fit(trainX, trainY, epochs=20, batch\_size=1, verbose=2)

Epoch 1/20

563/563 - 6s - loss: 0.0780

Epoch 2/20

563/563 - 1s - loss: 0.0017

Epoch 3/20

563/563 - 1s - loss: 0.0013

Epoch 4/20

563/563 - 1s - loss: 0.0011

Epoch 5/20

563/563 - 1s - loss: 8.9903e-04

Epoch 6/20

563/563 - 1s - loss: 7.2639e-04

Epoch 7/20

563/563 - 1s - loss: 5.4441e-04

Epoch 8/20

563/563 - 1s - loss: 4.5164e-04

Epoch 9/20

563/563 - 1s - loss: 4.0824e-04

Epoch 10/20

563/563 - 1s - loss: 3.4537e-04

Epoch 11/20

563/563 - 1s - loss: 3.4606e-04

Epoch 12/20

563/563 - 1s - loss: 3.3691e-04

Epoch 13/20

563/563 - 1s - loss: 3.2988e-04

Epoch 14/20

563/563 - 1s - loss: 4.2719e-04

Epoch 15/20

563/563 - 1s - loss: 3.1190e-04

Epoch 16/20

563/563 - 1s - loss: 3.9074e-04

Epoch 17/20

563/563 - 1s - loss: 3.1197e-04

Epoch 18/20

563/563 - 1s - loss: 3.2918e-04

Epoch 19/20

563/563 - 1s - loss: 3.5569e-04

Epoch 20/20

563/563 - 1s - loss: 3.8226e-04

<tensorflow.python.keras.callbacks.History at 0x23e402cf670>

# Predicting the vules using the defined model

trainPredict = model.predict(trainX) testPredict = model.predict(testX)

# De-Normalizing the data for plotting the values

trainPredict = scaler.inverse\_transform(trainPredict) trainY = scaler.inverse\_transform([trainY]) testPredict = scaler.inverse\_transform(testPredict) testY = scaler.inverse\_transform([testY])

# RMSE Value for the training Data

trainScore = math.sqrt(mean\_squared\_error(trainY[0], trainPredict[:,0]))

print('Train RMSE: %.2f' % (trainScore)) Train RMSE: 1.86

# RMSE Value for the Testing Data

testScore = math.sqrt(mean\_squared\_error(testY[0], testPredict[:,0])) print('Test RMSE: %.2f' % (testScore))

Test RMSE: 1.27

# Creating the similar dataset for plotting the training predictions

trainPredictPlot = np.empty\_like(OHLC\_avg) trainPredictPlot[:, :] = np.nan trainPredictPlot[step\_size:len(trainPredict)+step\_size, :] = trainPredict

# Creating the similar dataset for plotting the testing predictions

testPredictPlot = np.empty\_like(OHLC\_avg) testPredictPlot[:, :] = np.nan

testPredictPlot[len(trainPredict)+(step\_size\*2)+1:len(OHLC\_avg)-1, :]

= testPredict

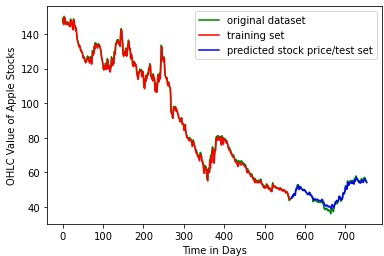
# De-Normalizing the main data set

OHLC\_avg = scaler.inverse\_transform(OHLC\_avg)

plt.plot(OHLC\_avg, 'g', label = 'original dataset') plt.plot(trainPredictPlot, 'r', label = 'training set') plt.plot(testPredictPlot, 'b', label = 'predicted stock price/test set')

plt.legend(loc = 'upper right') plt.xlabel('Time in Days')

plt.ylabel('OHLC Value of Apple Stocks') plt.show()



# Prediction of Future Values

last\_val = testPredict[-1] last\_val\_scaled = last\_val/last\_val

next\_val = model.predict(np.reshape(last\_val\_scaled, (1,1,1))) print ("Last Day Value:", np.asscalar(last\_val))

print ("Next Day Value:", np.asscalar(last\_val\*next\_val))

Last Day Value: 54.058773040771484 Next Day Value: 53.803123474121094

<ipython-input-25-a3575500764a>:4: DeprecationWarning: np.asscalar(a) is deprecated since NumPy v1.16, use a.item() instead

print ("Last Day Value:", np.asscalar(last\_val))

<ipython-input-25-a3575500764a>:5: DeprecationWarning: np.asscalar(a) is deprecated since NumPy v1.16, use a.item() instead

print ("Next Day Value:", np.asscalar(last\_val\*next\_val))