**Lab 8: To study and implement LSTM using Keras**

The Long Short-Term Memory network or LSTM network is a type of recurrent neural network used in deep learning. In this lab, you will discover how to develop LSTM networks in Python using the Keras deep learning library to address a demonstration time-series prediction problem

import numpy

import matplotlib.pyplot as plt

import pandas

import math

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

# load the dataset

dataframe = pandas.read\_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)

dataset = dataframe.values

dataset = dataset.astype('float32')

dataset

# normalize the dataset

scaler = MinMaxScaler(feature\_range=(0, 1))

dataset = scaler.fit\_transform(dataset)

dataset

train\_size = int(len(dataset) \* 0.67)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]

print(len(train), len(test))

# convert an array of values into a dataset matrix

def create\_dataset(dataset, look\_back=1):

dataX, dataY = [], []

for i in range(len(dataset)-look\_back-1):

a = dataset[i:(i+look\_back), 0]

dataX.append(a)

dataY.append(dataset[i + look\_back, 0])

return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t and Y=t+1

look\_back = 5

trainX, trainY = create\_dataset(train, look\_back)

testX, testY = create\_dataset(test, look\_back)

trainX

# reshape input to be [samples, time steps, features]

trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))

testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network

model = Sequential()

model.add(LSTM(4, input\_shape=(1, look\_back)))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(trainX, trainY, epochs=100, batch\_size=1, verbose=2)

# make predictions

trainPredict = model.predict(trainX)

testPredict = model.predict(testX)

# invert predictions

trainPredict = scaler.inverse\_transform(trainPredict)

trainY = scaler.inverse\_transform([trainY])

testPredict = scaler.inverse\_transform(testPredict)

testY = scaler.inverse\_transform([testY])

# calculate root mean squared error

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

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

testScore = math.sqrt(mean\_squared\_error(testY[0], testPredict[:,0]))

print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting

trainPredictPlot = numpy.empty\_like(dataset)

trainPredictPlot[:, :] = numpy.nan

trainPredictPlot[look\_back:len(trainPredict)+look\_back, :] = trainPredict

# shift test predictions for plotting

testPredictPlot = numpy.empty\_like(dataset)

testPredictPlot[:, :] = numpy.nan

testPredictPlot[len(trainPredict)+(look\_back\*2)+1:len(dataset)-1, :] = testPredict

# plot baseline and predictions

plt.plot(scaler.inverse\_transform(dataset))

plt.plot(trainPredictPlot)

plt.plot(testPredictPlot)

plt.show()