Ex No: 6 A RECURRENT NEURAL NETWORK

Aim:

To build a recurrent neural network with Keras/TensorFlow.

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Program:

# Parameter split\_percent defines the ratio of training examples def get\_train\_test(url, split\_percent=0.8): df = read\_csv(url, usecols=[1], engine='python') data = np.array(df.values.astype('float32')) scaler = MinMaxScaler(feature\_range=(0, 1)) data = scaler.fit\_transform(data).flatten() n = len(data)

# Point for splitting data into train and test split = int(n\*split\_percent) train\_data = data[range(split)] test\_data = data[split:] return train\_data, test\_data, data

sunspots\_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly- sunspots.csv' train\_data, test\_data, data = get\_train\_test(sunspots\_url)

# Prepare the input X and target Y def get\_XY(dat, time\_steps):

# Indices of target array

Y\_ind = np.arange(time\_steps, len(dat), time\_steps)

Y = dat[Y\_ind] # Prepare X rows\_x = len(Y)

X = dat[range(time\_steps\*rows\_x)] X = np.reshape(X, (rows\_x, time\_steps, 1)) return X, Y

time\_steps = 12 trainX, trainY = get\_XY(train\_data, time\_steps) testX, testY = get\_XY(test\_data, time\_steps)

model = create\_RNN(hidden\_units=3, dense\_units=1, input\_shape=(time\_steps,1),

activation=['tanh', 'tanh'])

model.fit(trainX, trainY, epochs=20, batch\_size=1, verbose=2) def print\_error(trainY, testY, train\_predict, test\_predict):

# Error of predictions train\_rmse = math.sqrt(mean\_squared\_error(trainY, train\_predict)) test\_rmse = math.sqrt(mean\_squared\_error(testY, test\_predict))

# Print RMSE print('Train RMSE: %.3f RMSE' % (train\_rmse))

print('Test RMSE: %.3f RMSE' % (test\_rmse))

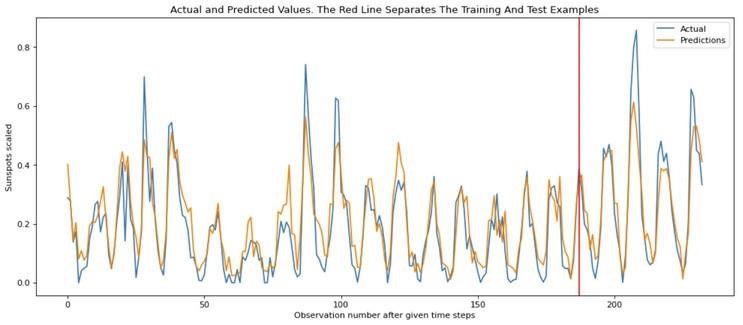
# make predictions train\_predict = model.predict(trainX) test\_predict = model.predict(testX)

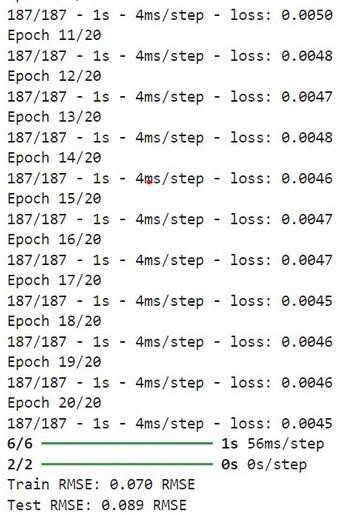
# Mean square error print\_error(trainY, testY, train\_predict, test\_predict)

# Plot the result def plot\_result(trainY, testY, train\_predict, test\_predict): actual = np.append(trainY, testY) predictions = np.append(train\_predict, test\_predict) rows = len(actual) plt.figure(figsize=(15, 6), dpi=80) plt.plot(range(rows), actual) plt.plot(range(rows), predictions) plt.axvline(x=len(trainY), color='r') plt.legend(['Actual', 'Predictions']) plt.xlabel('Observation number after given time steps') plt.ylabel('Sunspots scaled') plt.title('Actual and Predicted Values. The Red Line Separates The Training And Test Examples')

plot\_result(trainY, testY, train\_predict, test\_predict)

Output:





RESULT:

A simple RNN has been successfully created using timeseries data.