

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/monthlysunspots.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)

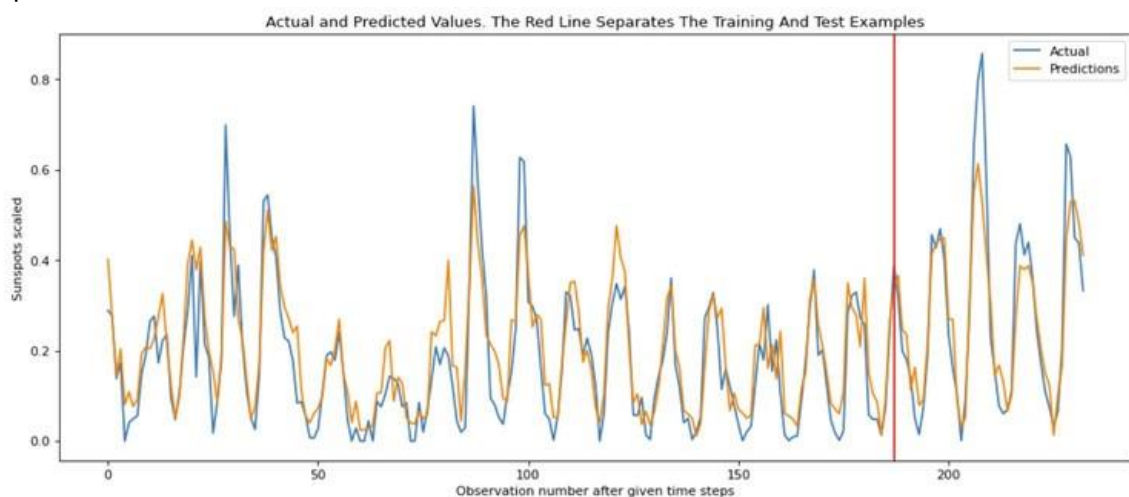
```



```

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:



```
187/187 - 1s - 4ms/step - loss: 0.0050
Epoch 11/20
187/187 - 1s - 4ms/step - loss: 0.0048
Epoch 12/20
187/187 - 1s - 4ms/step - loss: 0.0047
Epoch 13/20
187/187 - 1s - 4ms/step - loss: 0.0048
Epoch 14/20
187/187 - 1s - 4ms/step - loss: 0.0046
Epoch 15/20
187/187 - 1s - 4ms/step - loss: 0.0047
Epoch 16/20
187/187 - 1s - 4ms/step - loss: 0.0047
Epoch 17/20
187/187 - 1s - 4ms/step - loss: 0.0045
Epoch 18/20
187/187 - 1s - 4ms/step - loss: 0.0046
Epoch 19/20
187/187 - 1s - 4ms/step - loss: 0.0046
Epoch 20/20
187/187 - 1s - 4ms/step - loss: 0.0045
6/6  1s 56ms/step
2/2  0s 0s/step
Train RMSE: 0.070 RMSE
Test RMSE: 0.089 RMSE
```

RESULT:

A simple RNN has been successfully created using timeseries data.