DL_LAB_8.1

```
from pandas import read_csv
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import math
import matplotlib.pyplot as plt
```

```
def create_RNN(hidden_units, dense_units, input_shape, activation):
    model = Sequential()
    model.add(SimpleRNN(hidden_units, input_shape=input_shape,
    activation=activation[0]))
    model.add(Dense(units=dense_units, activation=activation[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

```
demo_model = create_RNN(2, 1, (3,1), activation=['linear', 'linear'])
```

```
def __init__(feature_range=(0, 1), *, copy=True, clip=False):
    #Open in tab View source
    #Transform features by scaling each feature to a given range.
    ##This estimator scales and translates each feature individually such
    #that it is in the given range on the training set, e.g. between
    #zero and one.
    #The transformation is given by:
    X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
    X_scaled = X_std * (max - min) + min
    #where min, max = feature_range.
    #This transformation is often used as an alternative to zero mean,
```

```
wx = demo_model.get_weights()[0]
wh = demo_model.get_weights()[1]
bh = demo_model.get_weights()[2]
wy = demo_model.get_weights()[3]
by = demo_model.get_weights()[4]
```

```
print('wx = ', wx, ' wh = ', wh, ' bh = ', bh, ' wy =', wy, 'by = ', by)
```

```
wx =  [[ 0.8626205  -0.69797206]]  wh =  [[ 0.51309454  0.8583321 ]
 [ 0.8583321  -0.51309454]]  bh =  [0. 0.]  wy = [[-1.1464654 ]
 [ 0.75158226]] by =  [0.]
```

We'll input x for three time steps and let the network generate an output. The values of the hidden units at time steps 1, 2, and 3 will be computed.

```
x = np.array([1, 2, 3])
```

```
x_input = np.reshape(x,(1, 3, 1))
y_pred_model = demo_model.predict(x_input)
```

```
1/1 [==============================] - 0s 231ms/step
```

```
m = 2
h0 = np.zeros(m)
h1 = np.dot(x[0], wx) + h0 + bh
h2 = np.dot(x[1], wx) + np.dot(h1,wh) + bh
h3 = np.dot(x[2], wx) + np.dot(h2,wh) + bh
o3 = np.dot(h3, wy) + by
```

```
print('h1 = ', h1,'h2 = ', h2,'h3 = ', h3)
```

```
h1 =  [[ 0.86262047 -0.69797206]] h2 =  [[ 1.56875498 -0.29740362]] h3 =  [[ 3.13750997 -0.59480725]]
```

```
print("Prediction from network ", y_pred_model)
print("Prediction from our computation ", o3)
```

```
Prediction from network  [[-4.0440936]]
Prediction from our computation  [[-4.04409326]]
```

```
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
```

```
def __init__(feature_range=(0, 1), *, copy=True, clip=False):
#Open in tab View source
#Transform features by scaling each feature to a given range.
#This estimator scales and translates each feature individually such
#that it is in the given range on the training set, e.g. between
#zero and one.
#The transformation is given by:
    X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
    X_scaled = X_std * (max - min) + min
#where min, max = feature_range.
#This transformation is often used as an alternative to zero mean,
```

Prepare the input X and target Y

```
def get_XY(dat, time_steps):
    Y_ind = np.arange(time_steps, len(dat), time_steps)
    Y = dat[Y_ind]
    rows_x = len(Y)
    X = dat[range(time_steps*rows_x)]
    X = np.reshape(X, (rows_x, time_steps, 1))
    return X, Y
```

```
def create_RNN(hidden_units, dense_units, input_shape, activation):
    model = Sequential()
    model.add(SimpleRNN(hidden_units, input_shape=input_shape, activation=activation[0]))
    model.add(Dense(units=dense_units, activation=activation[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

Error of predictions

```
def print_error(trainY, testY, train_predict, test_predict):

    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))
```

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')


sunspots_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-sunspots.csv'
time_steps = 12
train_data, test_data, data = get_train_test(sunspots_url)
trainX, trainY = get_XY(train_data, time_steps)
testX, testY = get_XY(test_data, time_steps)
```

Create model and train

```
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)
```

```
Epoch 1/20
187/187 - 1s - loss: 0.0364 - 1s/epoch - 5ms/step
Epoch 2/20
187/187 - 0s - loss: 0.0225 - 287ms/epoch - 2ms/step
Epoch 3/20
187/187 - 0s - loss: 0.0168 - 280ms/epoch - 1ms/step
Epoch 4/20
187/187 - 0s - loss: 0.0135 - 281ms/epoch - 2ms/step
Epoch 5/20
187/187 - 0s - loss: 0.0112 - 281ms/epoch - 2ms/step
Epoch 6/20
187/187 - 0s - loss: 0.0096 - 285ms/epoch - 2ms/step
Epoch 7/20
187/187 - 0s - loss: 0.0084 - 291ms/epoch - 2ms/step
Epoch 8/20
187/187 - 0s - loss: 0.0076 - 269ms/epoch - 1ms/step
Epoch 9/20
187/187 - 0s - loss: 0.0068 - 268ms/epoch - 1ms/step
Epoch 10/20
187/187 - 0s - loss: 0.0063 - 285ms/epoch - 2ms/step
Epoch 11/20
187/187 - 0s - loss: 0.0058 - 279ms/epoch - 1ms/step
Epoch 12/20
187/187 - 0s - loss: 0.0056 - 285ms/epoch - 2ms/step
Epoch 13/20
187/187 - 0s - loss: 0.0052 - 262ms/epoch - 1ms/step
Epoch 14/20
187/187 - 0s - loss: 0.0050 - 286ms/epoch - 2ms/step
Epoch 15/20
187/187 - 0s - loss: 0.0048 - 270ms/epoch - 1ms/step
Epoch 16/20
187/187 - 0s - loss: 0.0047 - 273ms/epoch - 1ms/step
Epoch 17/20
187/187 - 0s - loss: 0.0046 - 268ms/epoch - 1ms/step
Epoch 18/20
187/187 - 0s - loss: 0.0045 - 276ms/epoch - 1ms/step
Epoch 19/20
187/187 - 0s - loss: 0.0044 - 263ms/epoch - 1ms/step
Epoch 20/20
187/187 - 0s - loss: 0.0043 - 264ms/epoch - 1ms/step
<keras.callbacks.History at 0x7f2c29104760>
```

make predictions

```
train_predict = model.predict(trainX)
test_predict = model.predict(testX)
```

```
6/6 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 5ms/step
```

Print error

```
print_error(trainY, testY, train_predict, test_predict)
```

```
Train RMSE: 0.066 RMSE
Test RMSE: 0.089 RMSE
```

Plot result

```
plot_result(trainY, testY, train_predict, test_predict)
```