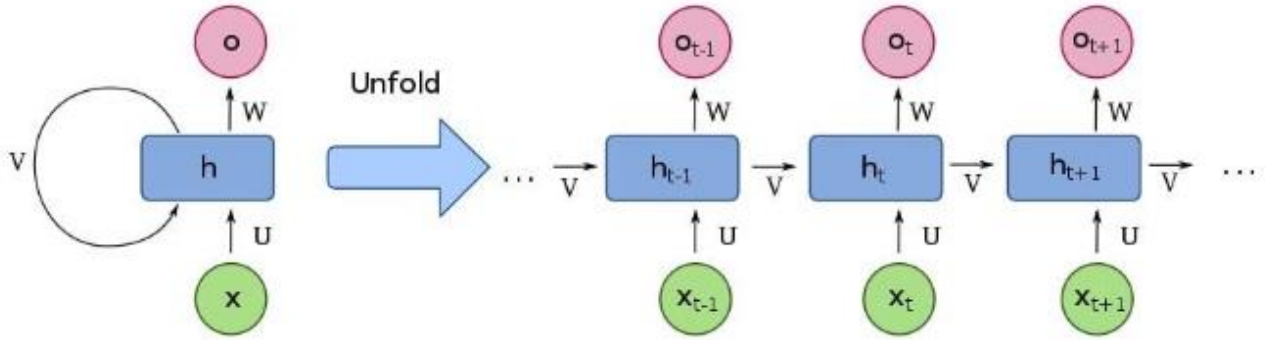
THỰC HÀNH 6 : RECURRENT NEURAL NETWORK

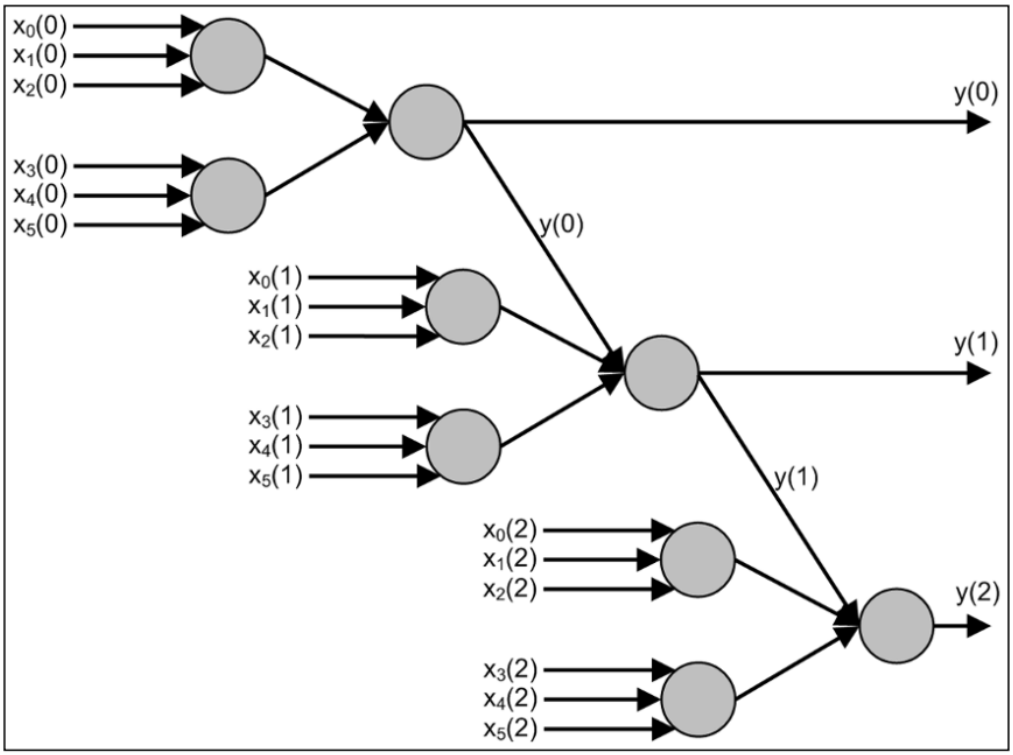
Sau bài thực hành này, sinh viên có khả năng:

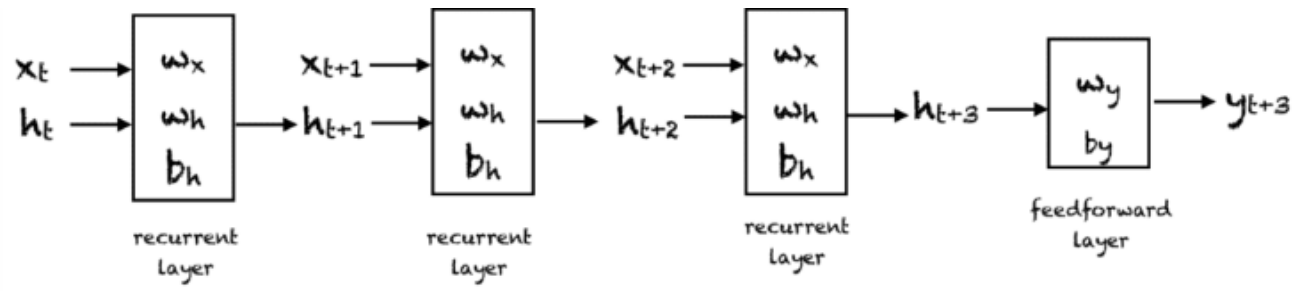
- Xây dựng được kiến trúc Recurrent Neural Network (RNN)
- Cài đặt mô hình Recurrent Neural Network bằng tensorflow
- Cài đặt ứng dụng thực tế sử dụng Recurrent Neural Network

1. GIỚI THIỆU

Recurrent Neural Network (RNN) là một mô hình học sâu xử lý chuỗi dữ liệu dạng feedback. Dữ liệu được gởi từ neuron trước đến neuron sau. Ví dụ: Muốn dự đoán từ tiếp theo của câu "Tôi là sinh viên của trường Đại học Văn ….". RNN nhận biết từ xuất hiện trước đó trong câu để dự báo từ tiếp theo. Mỗi neuron trong RNN có state. Các state này sẽ sử dụng kết quả state trước để tính state tiếp theo.







$x_t \in R$: Input ở thời điểm t. (giá trị, hoặc một từ nào đó)

$y_t \in R$: Output của RNN ở thời điểm t (có thể có nhiều output)

$h_t \in R^m$: là vector lưu dữ liệu của hidden state ở thời điểm t. Tham số *m* là số hidden unit. Vector h_0 khởi động bằng 0

$w_x \in R^m$: trọng số weight kết hợp với inputs ở tầng recurrent

$w_h \in R^{m\,x\,m}$: trọng số weight kết hợp với hidden state ở tầng recurrent

$w_y \in R^m$: trọng số weight kết hợp với hidden state ở output units

$b_y \in R^m$: trọng số bias kết hợp với tầng recurrent

$b_y \in R$: trọng số bias kết hợp với tầng feedforward

Các loại RNN:

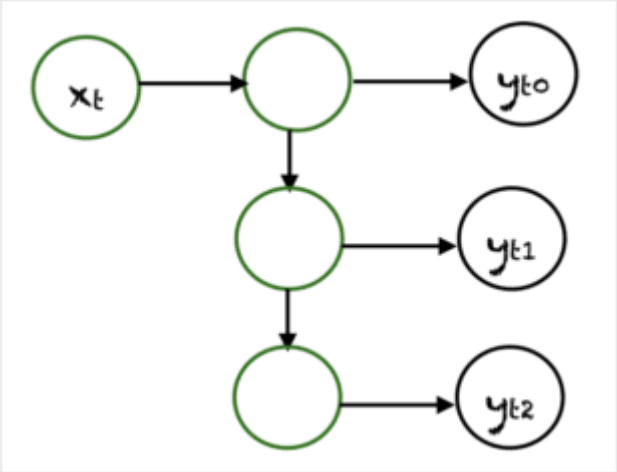
- One-one RNN
- One-many RNN
- Many-one RNN
- Many-many RNN

1.1. One-one RNN

One-one RNN (Tx = Ty = 1) là loại NN cơ bản với một input và một output.

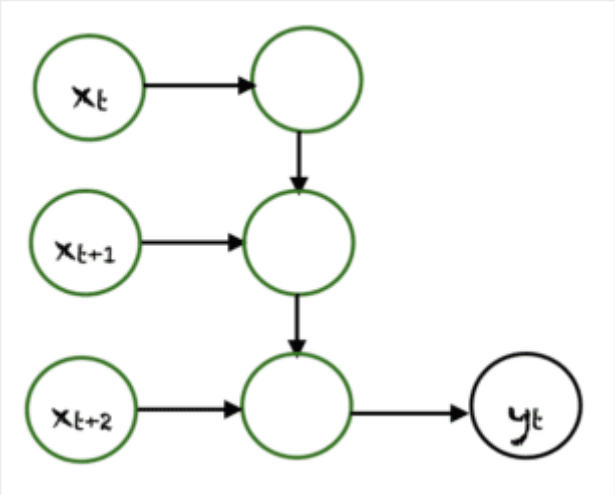
1.2. One-many RNN

One-many RNN (Tx = 1, Ty > 1) là loại NN với một input và nhiều output



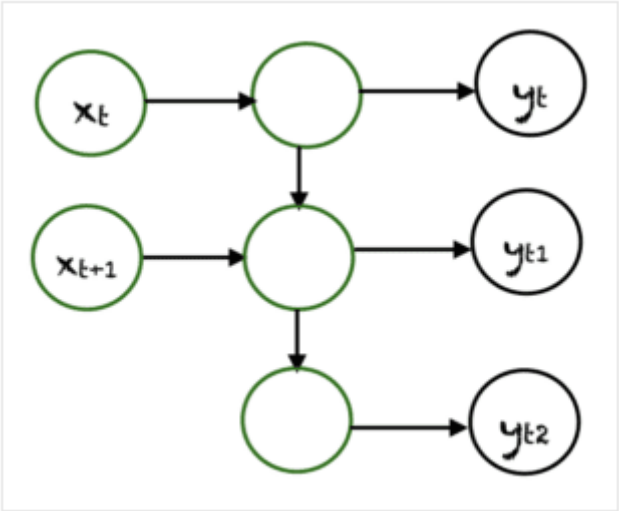
1.3. Many-one RNN

Many-one RNN (Tx > 1, Ty = 1) là loại NN với nhiều input và 1 output



1.4. Many-many RNN

Many-many RNN (Tx > 1, Ty > 1) là loại NN với nhiều input và nhiều output.



1.5. Ưu nhược điểm RNN

Ưu:

- Xử lý chuỗi dữ liệu
- Xử lý input với chiều dài khác nhau
- Lưu trữ thông tin quá khứ (past)

Nhược:

- Tính toán chậm
- Không nhận các input sau để dự báo
- Gặp vấn đề vanishing gradient. RNN sử dụng cập nhật weight gần với 0, làm cho NN không thể học weight mới.

2. CÀI ĐẶT RNN

2.1. Nạp thư viện

```
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
```

2.2. Tạo hàm định nghĩa mô hình RNN

```
[3]  #hidden_units: 2 blocks
     #dense_units : 1 block
     #input_shape: (time steps x features)
     #activation: linear f(x) = x
     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(dense_units, activation=activation[1]))
       model.compile(optimizer='adam', loss='mean_square_error')
       return model

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

2.3. Xem các tham số sinh bởi RNN

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

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

```
wx= [[1.0656968 0.6051017]]
wh= [[-0.6056522  -0.79572946]
 [-0.79572946  0.6056522 ]]
bh= [0. 0.]
wy= [[0.7070552]
 [0.7260777]]
by= [0.]
```

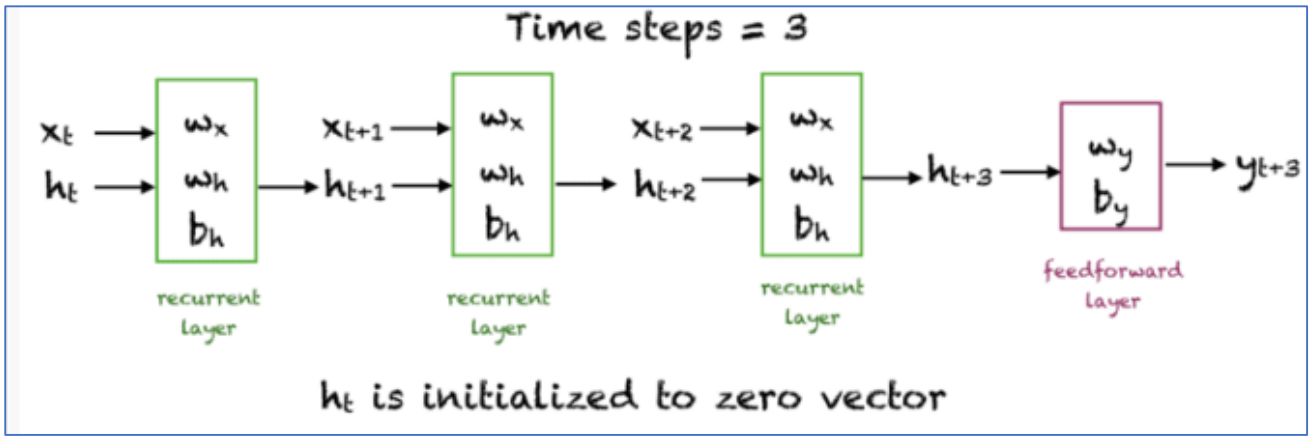
2.4. Tính vector hidden cho tầng recurrent

Ở mỗi thời điểm t, ta sử dụng giá trị mạng ở bước t để tính output của bước t + 1 với công thức sau:

$$h_{t+1} = f(x_t, h_t, w_x, w_h, b_h) = f(w_x x_t + w_h h_t + b_h)$$

Output y ở thời điểm t được tính

$$y_t = f(h_t, w_y) = f(w_y \cdot h_t + b_y)$$




```
x = np.array([1, 2, 3])
# Reshape the input to the required sample_size x time_steps x features
x_input = np.reshape(x, (1, 3, 1))
y_predict_model = demo_RNN.predict(x_input)

m = 2
h0 = np.zeros(m)
h1 = np.dot(x[0], wx) + bh + h0
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)
print('h2:', h2)
print('h3:', h3)

print('Prediction of model:', y_predict_model)
print('Prediction from computation:', o3)
```

```
1/1 [==============================] - 0s 301ms/step
h1: [[-0.07785499  1.16493523]]
h2: [[-1.32229876  2.37683636]]
h3: [[-2.73799211  2.43383128]]
Prediction of model: [[-5.003797]]
Prediction from computation: [[-5.00379647]]
```

3. ÁP DỤNG RNN CHO ỨNG DỤNG THỰC TẾ

Phần này hướng dẫn áp dụng RNN cho bài toán dự báo chuỗi thời gian cho công ty bán mỹ phẩm dưỡng da. Bộ dữ liệu tên là Sunspots có thể tải tại địa chỉ sau https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-sunspots.csv

Bộ dữ liệu gồm 2 cột

```
"Month","Sunspots"
"1749-01",58.0
"1749-02",62.6
"1749-03",70.0
"1749-04",55.7
"1749-05",85.0
"1749-06",83.5
"1749-07",94.8
"1749-08",66.3
"1749-09",75.9
"1749-10",75.5
"1749-11",158.6
"1749-12",85.2
```

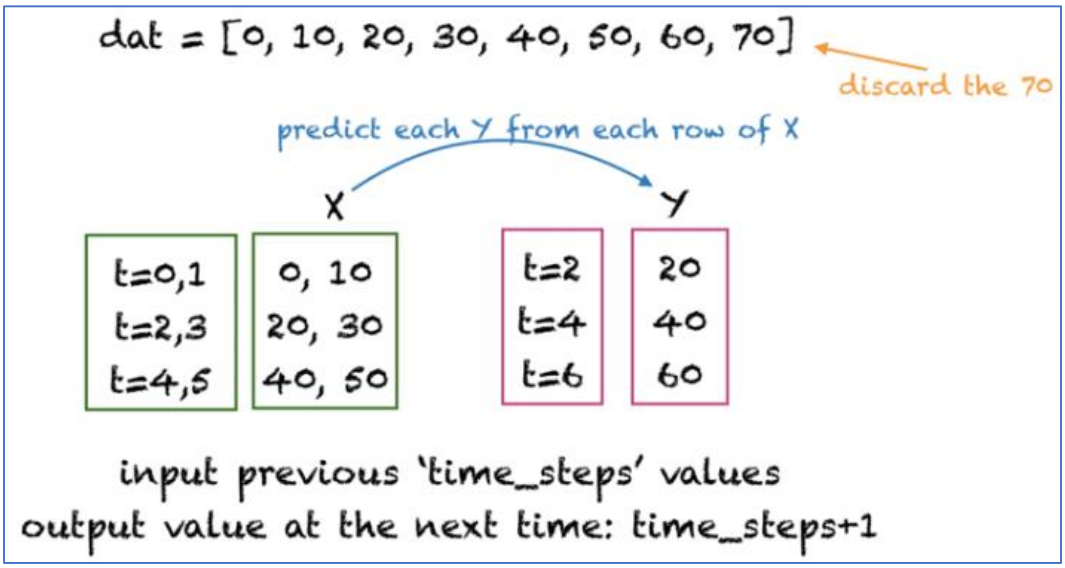
Bộ dữ liệu có tất cả 2820 dòng từ năm 1749 – 1983

3.1. Đọc dữ liệu

```
[ ]  from keras import engine
     def get_train_test(url, split_percent=0.8):
       df = pd.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)

       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)
```

3.2. Chuẩn bị dữ liệu cho RNN

Chuyển đổi dữ liệu sang total_samples x time_steps x features



Ta đặt time_steps = 2

Hàm get_XY nhận tham số mảng 1 chiều và chuyển thành mảng input X và mục tiêu Y. Sử dụng 12 time_steps để lấy 12 tháng.

```
def get_XY(dat, time_steps):
  #indices target arrays
  y_ind = np.arange(time_steps, len(dat), time_steps)
  Y = dat[y_ind]

  #prepare X
  row_x = len(Y)
  X = dat[range(time_steps * row_x)]
  X = np.reshape(X, (row_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)
```

3.3. Tạo RNN model và huấn luyện

```
[30] model = create_RNN(hidden_units=3, dense_units=1, input_shape=(time_steps,1), activation=['tanh', 'tanh'])
     model_fit = model.fit(trainX, trainY, epochs=20, verbose=1)

     Epoch 1/20
     6/6 [==============================] - 1s 5ms/step - loss: 0.0088 - accuracy: 0.0428
     Epoch 2/20
     6/6 [==============================] - 0s 4ms/step - loss: 0.0086 - accuracy: 0.0428
     Epoch 3/20
     6/6 [==============================] - 0s 4ms/step - loss: 0.0082 - accuracy: 0.0428
     Epoch 4/20
     6/6 [==============================] - 0s 4ms/step - loss: 0.0080 - accuracy: 0.0428
     Epoch 5/20
     6/6 [==============================] - 0s 5ms/step - loss: 0.0079 - accuracy: 0.0428
```

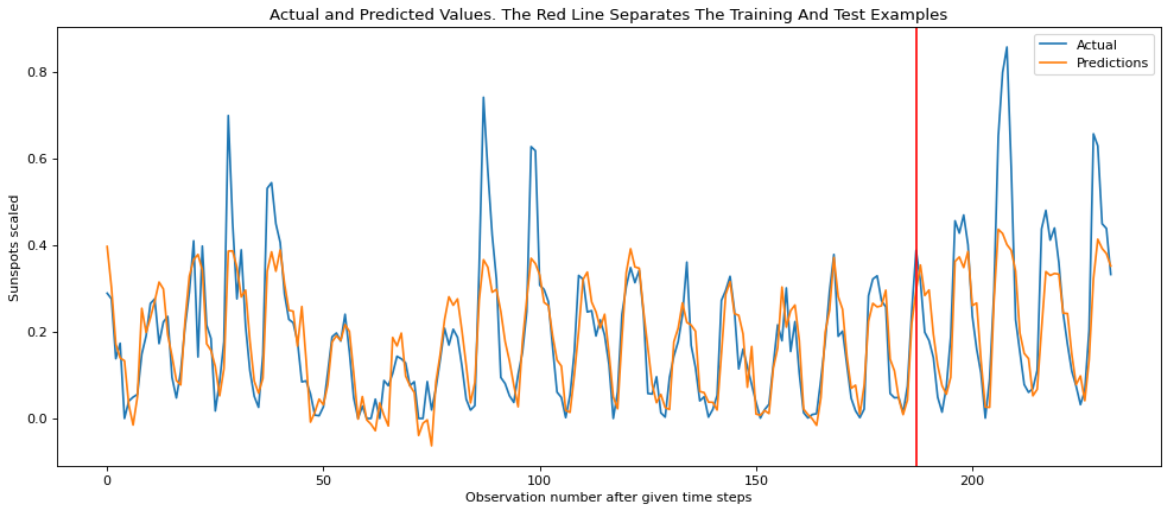
3.4. Hàm tính hiệu quả model

```
[31] 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)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>
6/6 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 7ms/step
Train RMSE: 0.079 RMSE
Test RMSE: 0.134 RMSE
```

3.5. Đánh giá kết quả

```
# 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')
    plt.show()
plot_result(trainY, testY, train_predict, test_predict)
```



THỰC HÀNH 6 : RECURRENT NEURAL NETWORK