# **Mạng noron hồi tiếp – RNN**

1. **Mạng noron hồi tiếp RNN**

- Đn: 1 mạng sử dụng tính toán hồi tiếp đgl RNN

- Input: là 1 chuỗi dl đầu vào có tính chất chuỗi hoặc dl liên quan đến time

- Output: cũng là một chuỗi dữ liệu, tương ứng với mỗi thời điểm trong chuỗi đầu vào.

* Nếu mạng RNN được sử dụng để dự đoán chuỗi thời gian tiếp theo, đầu ra có thể là một chuỗi giá trị dự đoán cho từng thời điểm trong tương lai.
* Trong các bài toán phân loại chuỗi, đầu ra có thể là một chuỗi các nhãn phân loại cho từng thời điểm trong chuỗi đầu vào.
* Tổng quát, RNN nhận dữ liệu đầu vào dưới dạng chuỗi và tạo ra đầu ra tương ứng với từng thời điểm trong chuỗi đầu vào.

1. **Mạng noron Hồi tiếp hiện đại**
   1. **Nút Hồi tiếp có Cổng (GRU)**

- Input và Output của GRU tương tự như RNN

- Một ô nhớ GRU có 2 cổng: Cổng xóa và Cổng cập nhật

* **Cổng xóa**: cho phép kiểm soát bn phần của trạng thái trc đây được giữ lại.

🡪 Hoạt động của cổng xóa: để giảm ảnh hưởng của các trạng thái trước đó, ta nhân Ht-1(trạng thái ẩn) với Rt (cổng xóa) theo từng phần tử

* **Cổng cập nhật**: cho phép kiểm soát bn phần của trạng thái mới sẽ giống trạng thái cũ.

🡪 Hoạt động: xđ mức độ giống nhau giữa trạng thái mới Ht và trạng thái cũ Ht-1, cũng như mức độ trạng thái ẩn tiềm năng được sử dụng

- **Tính chất của GRU**: có 2 tính chất nổi bật:

* Cổng xóa giúp nắm bắt các phụ thuộc ngắn hạn trong chuỗi time
* Cổng cập nhật giúp nắm bắt các phụ thuộc dài hạn trong chuỗi time

**- So sánh RNN và GRU**

* Cấu trúc: GRU giải quyết vde gradient vụn vặt và khả năng lưu trữ thông tin lâu hơn, cấu trúc đơn giản hơn RNN, chỉ sdung 2 cổng: xóa và cập nhật
* GRU hỗ trợ kiểm soát trạng thái ẩn, có các cơ chế học để quyết định khi nào nên cập nhật hay xóa trạng thái ẩn

🡪 Nếu ký tự đầu tiên có mức độ quan trọng cao, mô hình sẽ học để không cập nhật trạng thái ẩn sau lần quan sát đầu tiên

🡪 Học cách bỏ qua những quan sát tạm thời không liên quan, cũng như cách xóa trạng thái ẩn khi cần thiết

* 1. **Bộ nhớ Ngắn hạn Dài (LSTM)**

- Thiết kế 3 cổng:

* Cổng đầu ra (ouput gate): để đọc các thông tin từ ô nhớ
* Cổng đầu vào (input gate): để quyết định khi nào cần ghi dữ liệu vào ô nhớ
* Cổng quên (forget gate): để thiết lập lại nội dung chứa trong ô nhớ

**- So sánh giữa RNN và LSTM**

* Cấu trúc: LSTM phức tạp hơn gồm 3 cổng và ô tế bào (cell state) để điều chỉnh và lưu trữ thông tin.
* Khả năng lưu trữ thông tin của LSTM lâu hơn: nhờ sdung cổng quên nên nó có khả năng quyết định thông tin nào nên được lưu trữ và thông tin nào nên bị bỏ qua 🡪 giúp lưu tt quan trọng trong time dài và truyền nó qua các thời điểm tiếp theo
* Tính hiệu quả tính toán: LSTM có xu hướng tốn nhiều tham số hơn và tính toán phức tạp hơn so với RNN. Điều này có thể làm tăng thời gian huấn luyện và đòi hỏi nhiều tài nguyên tính toán hơn.

**- So sánh GRU và LSTM:**

* Cấu trúc: cả 2 đều là biến thể của RNN nhưng ctruc GRU đơn giản hơn với số cổng và lượng tham số ít hơn
* Tính toán: GRU đơn giản và ít phức tạp hơn, thường nhanh hơn trong qtrinh tính toán, đòi hỏi ít tài nguyên
* Khả năng lưu trữ: GRU đơn giản nhưng vẫn có khả năng lưu trữ thông tin quan trọng trong thời gian dài. Tuy nhiên, LSTM có cấu trúc phức tạp hơn và có khả năng lưu trữ thông tin lâu hơn và chính xác hơn trong một số trường hợp.

Tách chuỗi tt thành 1 tập chuỗi con

Reshape X thành ma trận 3 chiều

Cbi các tập train test, val

Xây dựng mô hình

Khai báo các tso dịch mô hình

Huấn luyện

Dùng mô hình đã huấn luyện dự báo tập X\_test

Có y dự báo r thì ss y dự báo vs y test để đánh giá chất lượng

# lstm for time series forecasting

from numpy import sqrt

import numpy as np

from numpy import asarray

from pandas import read\_csv

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM, SimpleRNN,GRU

from sklearn.metrics import r2\_score

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, AveragePooling2D,BatchNormalization

from sklearn.metrics import precision\_score, recall\_score, f1\_score

# split a univariate sequence into samples: cắt thành các chuỗi con

def split\_sequence(sequence, n\_steps, n\_pred, pred\_col):

#pred\_col: tên cột, STT cột dự đoán

    X, y = list(), list() #tập chuỗi con X, tập nhãn y

    for i in range(len(sequence) - (n\_steps + n\_pred) +1):

#y sẽ đi từ 0 -> độ dài của chuỗi trừ đi vtri của chuỗi cuối

Cùng (n\_steps + n\_pred. Nma trong python kbh n đi đến cận này,

Nó thường dừng trước cận 1 số nên ta +1

        # gather input and output parts of the pattern

        seq\_x = sequence[i: i+n\_steps, :]

#lấy từ vtri 0 -> vtri thứ i+n\_steps để ta lấy

đc chuỗi có độ dài nstep bắt đầu từ vtri thứ I,

và ta lấy all các cột, dấu : cuối

        seq\_y = sequence[i+n\_steps+n\_pred -1, pred\_col]

#tại dòng dự đoán, ta lấy gtri của cột dự đoán pred\_col

        X.append(seq\_x)

        y.append(seq\_y)

    return asarray(X), asarray(y)

# load the dataset

filename = "data\_tibau.csv"

all\_attributes = ['Wind\_Speed', 'Wind\_direction']

data = read\_csv(filename, index\_col=False, usecols=all\_attributes, encoding='utf-8')[all\_attributes]

print(data)

# retrieve the values: ép kiểu dl, dl chỉ lấy phần gtri chứ k có phần tiêu đề, độ dài k gian lưu trữ 32 bit

values = data.values.astype('float32')

print(values)

# specify the window size

n\_steps = 5

n\_pred = 2 #khoảng dự đoán

# split into samples

X, y = split\_sequence(values, n\_steps, n\_pred, 0)

#dự đoán cột wind speed nên điền tt là 0

print(X)

print(y)

# reshape into [samples, timesteps, features]

X = X.reshape((X.shape[0], X.shape[1], len(all\_attributes)))

#reshape X thành ma trận 3 chiều (số các chuỗi con, độ dài chuỗi con,

Số các thuộc tính)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, shuffle = False)

X\_train, X\_val, y\_train, y\_val =  train\_test\_split(X\_train,y\_train, test\_size=0.125, shuffle = False)

print(X\_train.shape, X\_test.shape , X\_val.shape, y\_train.shape, y\_test.shape, y\_val.shape)

# model LSTM

model\_lstm = Sequential()

model\_lstm.add(LSTM(150, activation='relu', kernel\_initializer='he\_normal', input\_shape=(X\_train.shape[1], X\_train.shape[2]))) #input\_shape: kích thước của chuỗi con

model\_lstm.add(Dense(100, activation='relu', kernel\_initializer='he\_normal'))

# model\_lstm.add(BatchNormalization())

# model\_lstm.add(Dropout(0.2))

model\_lstm.add(Dense(50, activation='relu', kernel\_initializer='he\_normal'))

model\_lstm.add(Dense(1))

#khai báo hàm mất mát

#mse: cộng all bình phương sai số giữa dự đoán và thực tế, sau đấy chia TB

model\_lstm.compile(optimizer='adam', loss='mse', metrics=['mae'])

#huấn luyện model: truyền vào X\_train, y\_train,

Epochs: lặp 150 lần, batch\_size: mỗi 1 lần cập nhật trọng số sdung 32 mẫu dl. Nếu = 0 thì huấn luyện hết r ms đưa ra kqua, nếu = 1 thì sau mỗi lần epochs nó sẽ đưa ra kqua

model\_lstm.fit(X\_train, y\_train, epochs=150, batch\_size=32, verbose=1, validation\_data=(X\_val, y\_val))

#đánh giá chất lượng mô hình

mse\_lstm, mae\_lstm = model\_lstm.evaluate(X\_test, y\_test, verbose=0)

y\_pred\_lstm = model\_lstm.predict(X\_test)

y\_pred\_lstm=y\_pred\_lstm.reshape(y\_pred\_lstm.shape[0])

#RNN

model\_rnn= Sequential()

model\_rnn.add(SimpleRNN(100, activation='relu', kernel\_initializer='he\_normal', input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model\_rnn.add(Dense(100, activation='relu', kernel\_initializer='he\_normal'))

# model\_rnn.add(BatchNormalization())

# model\_rnn.add(Dropout(0.2))

model\_rnn.add(Dense(50, activation='relu', kernel\_initializer='he\_normal'))

model\_rnn.add(Dense(1))

model\_rnn.compile(optimizer='adam', loss='mse', metrics=['mae'])

model\_rnn.fit(X\_train, y\_train, epochs=150, batch\_size=32, verbose=1, validation\_data=(X\_val, y\_val))

mse\_rnn, mae\_rnn = model\_rnn.evaluate(X\_test, y\_test, verbose=0)

y\_pred\_rnn = model\_rnn.predict(X\_test)

y\_pred\_rnn=y\_pred\_rnn.reshape(y\_pred\_rnn.shape[0])

#GRU

model\_gru = Sequential()

model\_gru.add(GRU(100, activation='relu', kernel\_initializer='he\_normal', input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model\_gru.add(Dense(100, activation='relu', kernel\_initializer='he\_normal'))

# model\_gru.add(BatchNormalization())

# model\_gru.add(Dropout(0.2))

model\_gru.add(Dense(50, activation='relu', kernel\_initializer='he\_normal'))

model\_gru.add(Dense(1))

model\_gru.compile(optimizer='adam', loss='mse', metrics=['mae'])

model\_gru.fit(X\_train, y\_train, epochs=150, batch\_size=32, verbose=1, validation\_data=(X\_val, y\_val))

mse\_gru, mae\_gru = model\_gru.evaluate(X\_test, y\_test, verbose=0)

y\_pred\_gru = model\_gru.predict(X\_test)

y\_pred\_gru=y\_pred\_gru.reshape(y\_pred\_gru.shape[0])

#Đánh giá mô hình

#LSTM

print("LSTM")

print('MSE: %.2f' % mse\_lstm)

print('RMSE: %.2f' % sqrt(mse\_lstm))

print('MAE: %.2f' % mae\_lstm)

print('R2: %.2f' % r2\_score(y\_test, y\_pred\_lstm))

print()

#RNN

print("RNN")

print('MSE: %.2f' % mse\_rnn)

print('RMSE: %.2f' % sqrt(mse\_rnn))

print('MAE: %.2f' % mae\_rnn)

print('R2: %.2f' % r2\_score(y\_test, y\_pred\_rnn))

print()

#GRU

print("GRU")

print('MSE: %.2f' % mse\_gru)

print('RMSE: %.2f' % sqrt(mse\_gru))

print('MAE: %.2f' % mae\_gru)

print('R2: %.2f' % r2\_score(y\_test, y\_pred\_gru))