ĐỒ ÁN CUỐI KỲ

NHẬP MÔN HỌC MÁY

Giảng viên: **LÊ ANH CƯỜNG**

Sinh viên: **TRẦN NGỌC DŨNG – 51800187**

NGUYỄN MINH ĐĂNG KHOA – 51800882

TRÌNH BÀY VỀ LÝ THUYẾT MẠNG RECURRENT NEURAL NETWORK

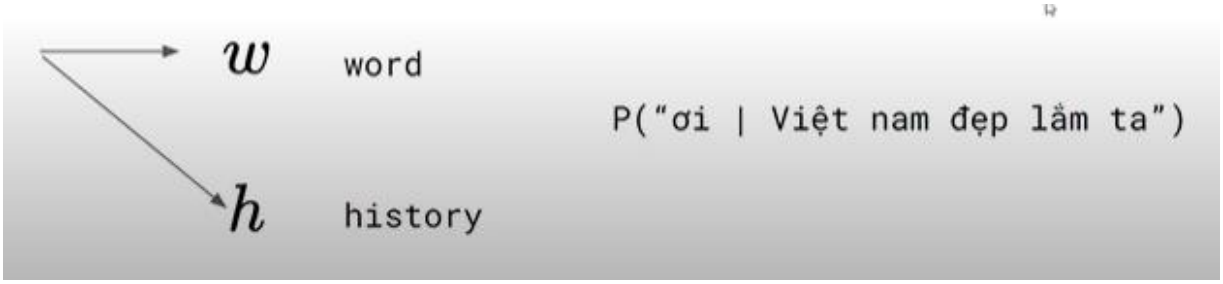
CHƯƠNG 1 – LÝ THUYẾT VỀ MẠNG RECURRENT NEURAL NETWORK

1.1 Mô hình ngôn ngữ

- Mô hình ngôn ngữ là một phân bố xác suất trên các tập văn bản. Nói đơn giản, mô hình ngôn ngữ có thể cho biết xác suất một câu (hoặc cụm từ) thuộc một ngôn ngữ là bao nhiêu. Đo lường xác xuất xảy ra 1 từ khi cho các từ phía trước.

- Ví dụ: Cho từ “tôi đi” sẽ dự đoán ra từ tiếp theo. Giả sử như từ “ngủ”.

- Tính P(w|h) với w là từ cần dự đoán và h là tập văn bản.

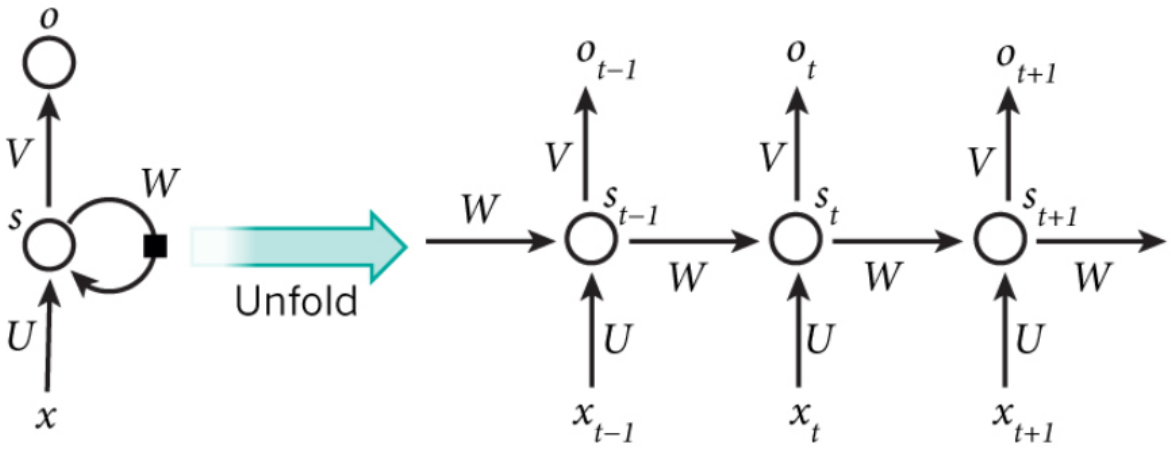


Hình 1.1.1 Ví dụ về mô hình ngôn ngữ

1.2 Mạng hồi quy RNN là gì?

- RNN ra đời với ý tưởng là sử dụng một bộ nhớ để lưu lại thông tin từ từ những bước tính toán xử lý trước để dựa vào nó có thể đưa ra dự đoán chính xác nhất cho bước dự đoán hiện tại.

- Trong các mạng nơ-ron truyền thống tất cả các đầu vào và cả đầu ra là độc lập với nhau. Tức là chúng không liên kết thành chuỗi với nhau. Nhưng các mô hình này không phù hợp trong rất nhiều bài toán. Ví dụ, nếu muốn đoán từ tiếp theo có thể xuất hiện trong một câu thì ta cũng cần biết các từ trước đó xuất hiện lần lượt thế nào. RNN thực hiện cùng một tác vụ cho tất cả các phần tử của một chuỗi với đầu ra phụ thuộc vào cả các phép tính trước đó. Nói cách khác, RNN có khả năng nhớ các thông tin được tính toán trước đó.



Hình 1.2.1 Ví dụ về RNN

- Ví dụ như hình trên,việc tính toán bên trong RNN được thực hiện như sau:

- X_t là đầu vào tại bước t. Ví dụ, x1 là một vec-tơ one-hot tương ứng với từ thứ 2 của câu (*trai*).
- S_t là trạng thái ẩn tại bước t. Nó chính là bộ nhớ của mạng. s_t được tính toán dựa trên cả các trạng thái ẩn phía trước và đầu vào tại bước đó: $S_t = f(Ux_t + Ws_{t-1})$. Hàm f thường là một hàm phi tuyến tính như tang hyperbolic (tanh) hay ReLu. Để

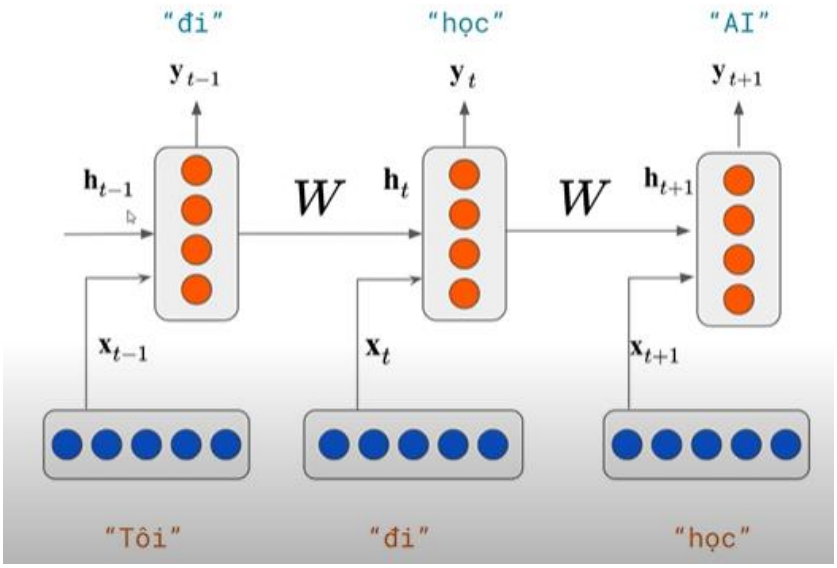
làm phép toán cho phần tử ẩn đầu tiên ta cần khởi tạo thêm s_{-1}, thường giá trị khởi tạo được gắn bằng 0.

- O_t là đầu ra tại bước t. Ví dụ, ta muốn dự đoán từ tiếp theo có thể xuất hiện trong câu thì O_t chính là một vec-tơ xác xuất các từ trong danh sách từ vựng của ta:
- $O_t=\text{softmax}(Vs_t)$

1.3 Ứng dụng của RNN?

1.3.1 Mô hình hóa ngôn ngữ và sinh văn bản

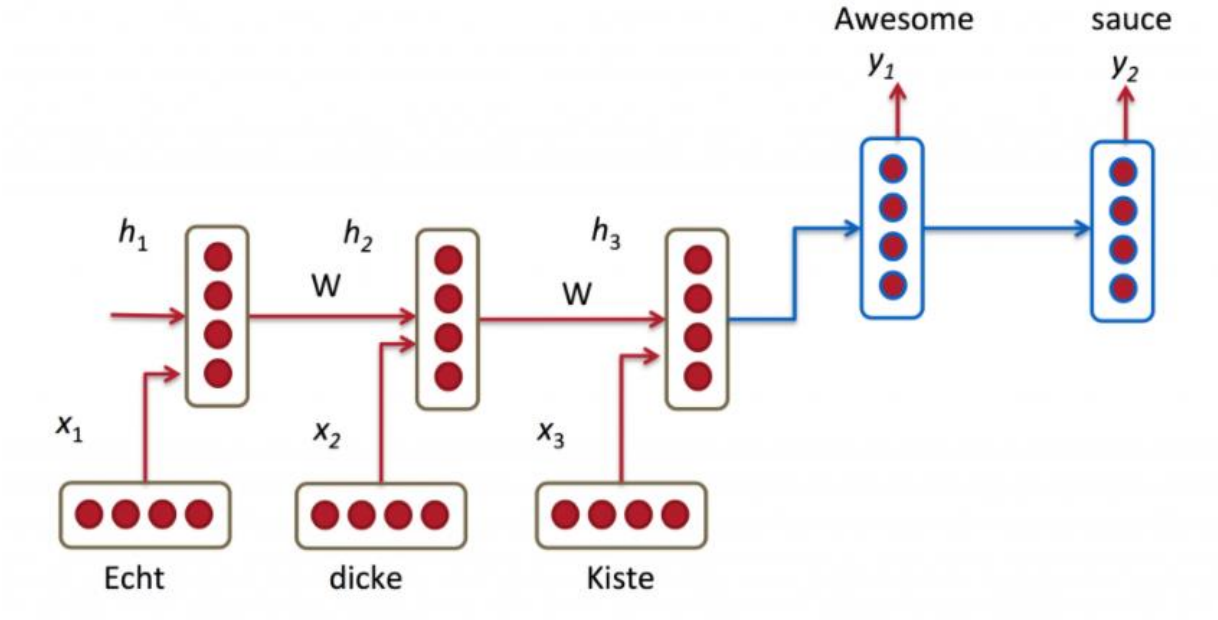
- Như đã trình bày bên trên, Mô hình ngôn ngữ cho phép ta dự đoán được xác xuất của một từ nào đó xuất hiện sau một chuỗi các từ đi liền trước nó. Do có khả năng ước lượng được độ tương tự của các câu nên nó còn được ứng dụng cho việc dịch máy. Một điểm lý thú của việc có thể dự đoán được từ tiếp theo là ta có thể xây dựng được một mô hình tự sinh từ cho phép máy tính có thể tự tạo ra các văn bản mới từ tập mẫu và xác xuất đầu ra của mỗi từ. Vậy nên, tùy thuộc vào mô hình ngôn ngữ mà ta có thể tạo ra được nhiều văn bản khác nhau khá là thú vị phải không. Trong mô hình ngôn ngữ, đầu vào thường là một chuỗi các từ (được mô tả bằng vec-tơ one-hot) và đầu ra là một chuỗi các từ dự đoán được. Khi huấn luyện mạng, ta sẽ gán *ot=xt*+1 vì ta muốn đầu ra tại bước t chính là từ tiếp theo của câu.



Hình 1.3.1 Mô hình hóa ngôn ngữ

1.3.2 Dịch máy

- Ngoài ra RNN còn có thể ứng dụng trong mô hình dịch máy, tương tự như mô hình hóa ngôn ngữ ở điểm là đầu vào là một chuỗi các từ trong ngôn ngữ nguồn (ngôn ngữ cần dịch - ví dụ là tiếng Việt). Còn đầu ra sẽ là một chuỗi các từ trong ngôn ngữ đích (ngôn ngữ dịch - ví dụ là tiếng Anh). Điểm khác nhau ở đây là đầu ra của ta chỉ xử lý sau khi đã xem xét toàn bộ chuỗi đầu vào. Vì từ dịch đầu tiên của câu dịch cần phải có đầy đủ thông tin từ đầu vào cần dịch mới có thể suy luận được.



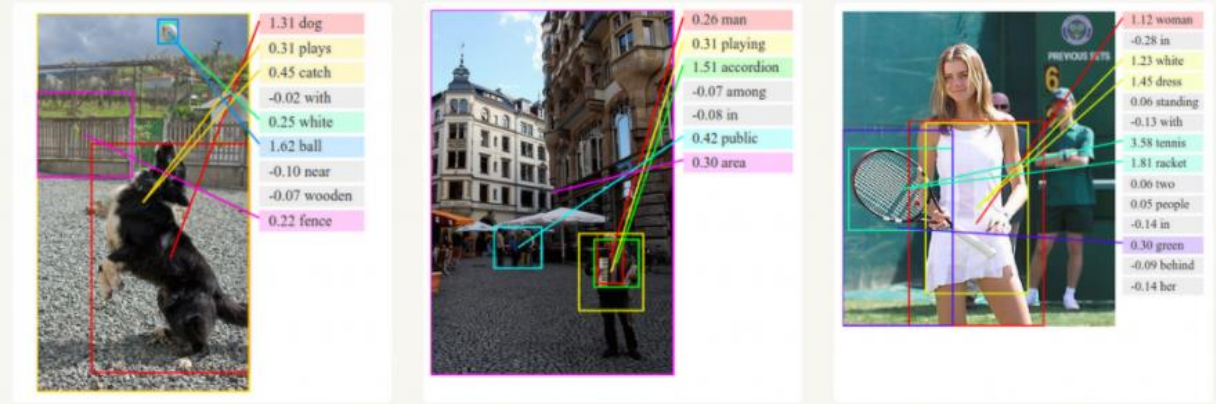
Hình 1.3.2 Dịch máy

1.3.3 Nhận dạng giọng nói

- RNN còn được ứng dụng trong việc nhận dạng giọng nói bằng việc Đưa vào một chuỗi các tín hiệu âm thanh, ta có thể dự đoán được chuỗi các đoạn ngữ âm đi kèm với xác xuất của chúng.

1.3.4 Mô tả hình ảnh

- RNN được sử dụng để tự động tạo mô tả cho các ảnh chưa được gán nhãn. Sự kết hợp này đã đưa ra được các kết quả khá kinh ngạc. Ví dụ như các ảnh dưới đây, các mô tả sinh ra có mức độ chính xác và độ tường tận khá cao.



Hình 1.3.4 Mô tả ảnh

1.4 Khả năng của RNN – LSTM

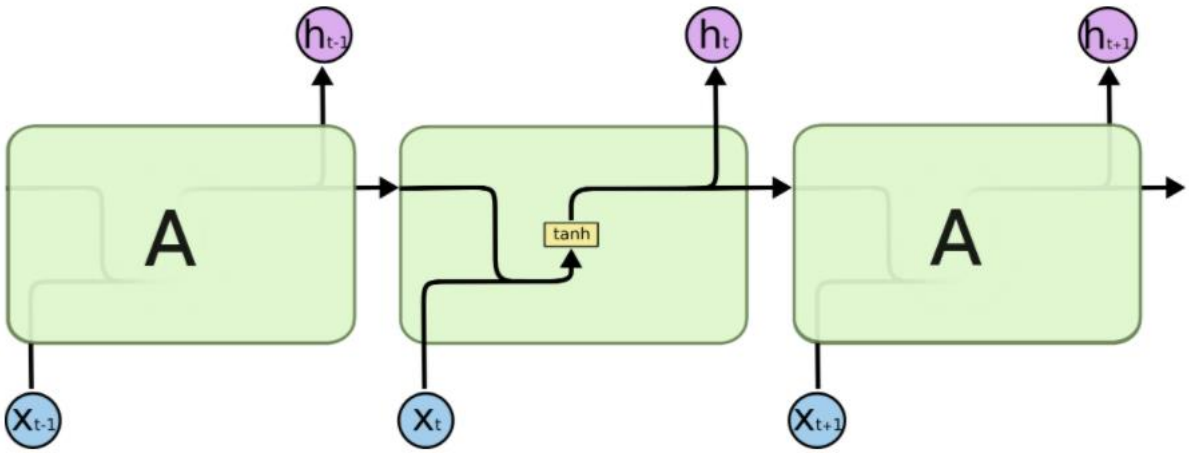
1.4.1 Tổng quan về LSTM

- Trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP - Natural Language Processing), đã ghi nhận được nhiều thành công của RNN cho nhiều vấn đề khác nhau. Mô hình phổ biến nhất được sử dụng của RNN là LSTM. LSTM (Long Short-Term Memory) thể hiện được sự ưu việt ở điểm có thể nhớ được nhiều bước hơn mô hình RNN truyền thống.

- Mạng bộ nhớ dài-ngắn (Long Short Term Memory networks), thường được gọi là LSTM - là một dạng đặc biệt của RNN, nó có khả năng học được các phụ thuộc xa. Chúng hoạt động cực kì hiệu quả trên nhiều bài toán khác nhau nên dần đã trở nên phổ biến như hiện nay.

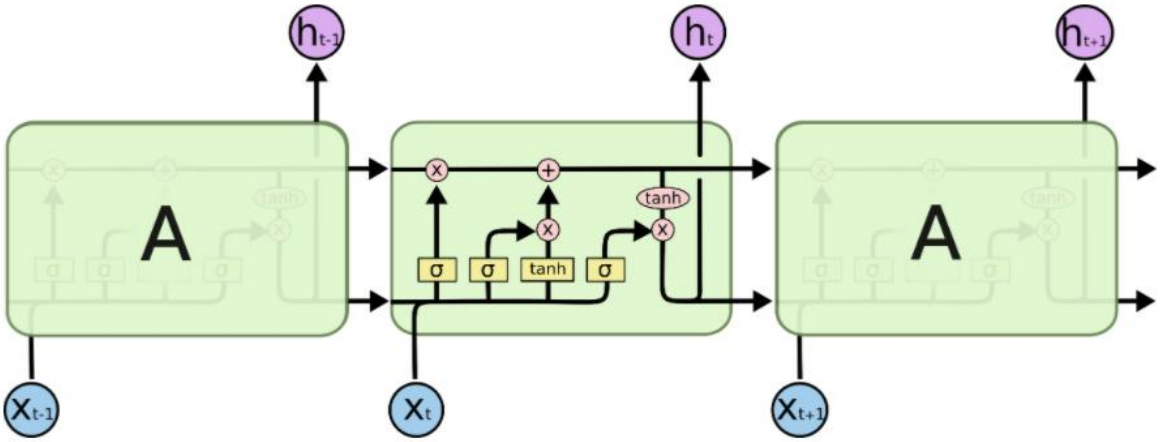
- LSTM được thiết kế để tránh được vấn đề phụ thuộc xa (long-term dependency). Việc nhớ thông tin trong suốt thời gian dài là đặc tính mặc định của chúng, chứ ta không cần phải huấn luyện nó để có thể nhớ được. Tức là ngay nội tại của nó đã có thể ghi nhớ được mà không cần bất kì can thiệp nào.

- Mọi mạng hồi quy đều có dạng là một chuỗi các mô-đun lặp đi lặp lại của mạng nơ-ron. Với mạng RNN chuẩn, các mô-dun này có cấu trúc rất đơn giản, thường là một tầng tanh.

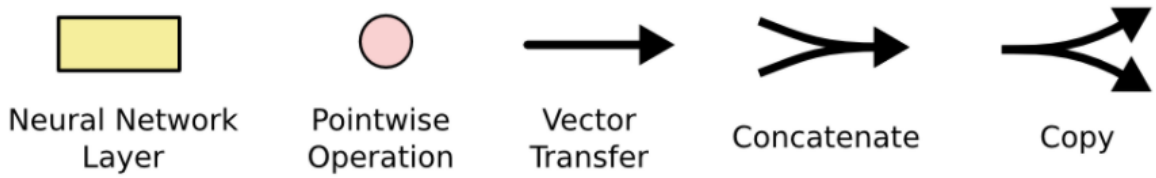


Hình 1.4.1 Module lặp lại trong RNN tiêu chuẩn chứa 1 lớp

- LSTM cũng có kiến trúc dạng chuỗi như vậy, nhưng các mô-đun trong nó có cấu trúc khác với mạng RNN chuẩn. Thay vì chỉ có một tầng mạng nơ-ron, chúng có tới 4 tầng tương tác với nhau một cách rất đặc biệt.



Hình 1.4.1 Module lặp lại trong RNN tiêu chuẩn chứa 4 lớp

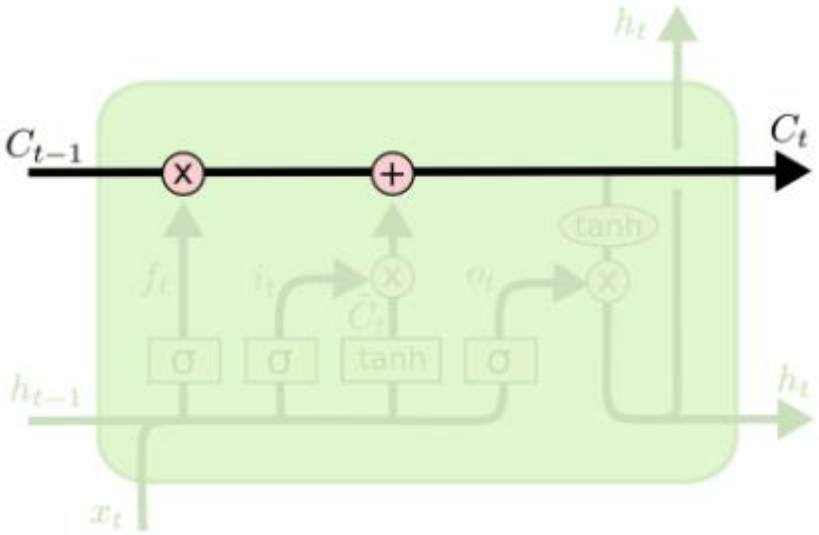


- Mỗi một đường mang một véc-tơ từ đầu ra của một nút tới đầu vào của một nút khác. Các hình trong màu hồng biểu diễn các phép toán như phép cộng véc-tơ chẳng hạn, còn các ô màu vàng được sử dụng để học trong các từng mạng nơ-ron. Các đường hợp nhau kí hiệu việc kết hợp, còn các đường rẽ nhánh ám chỉ nội dung của nó được sao chép và chuyển tới các nơi khác nhau.

1.4.2 Ý tưởng cốt lõi của LSTM

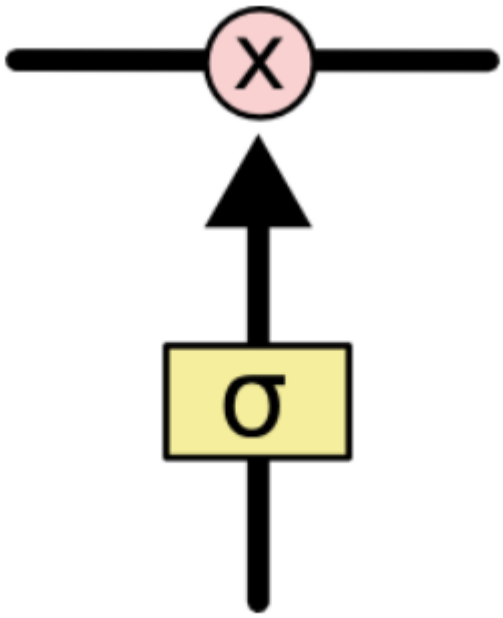
- Chìa khóa của LSTM là trạng thái tế bào (cell state) - chính đường chạy thông ngang phía trên của sơ đồ hình vẽ.

- Trạng thái tế bào là một dạng giống như băng truyền. Nó chạy xuyên suốt tất cả các mắt xích (các nút mạng) và chỉ tương tác tuyến tính đôi chút. Vì vậy mà các thông tin có thể dễ dàng truyền đi thông suốt mà không sợ bị thay đổi.



- LSTM có khả năng bỏ đi hoặc thêm vào các thông tin cần thiết cho trạng thái tế báo, chúng được điều chỉnh cẩn thận bởi các nhóm được gọi là cổng (gate).

- Các cổng là nơi sàng lọc thông tin đi qua nó, chúng được kết hợp bởi một tầng mạng sigmoid và một phép nhân.

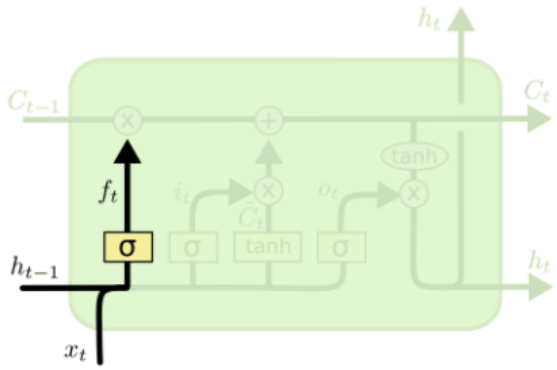


- Tầng sigmoid sẽ cho đầu ra là một số trong khoản $[0,1]$, mô tả có bao nhiêu thông tin có thể được thông qua. Khi đầu ra là 0 thì có nghĩa là không cho thông tin nào qua cả, còn khi là 1 thì có nghĩa là cho tất cả các thông tin đi qua nó.

- Một LSTM gồm có 3 cổng như vậy để duy trì và điều hành trạng thái của tế bào.

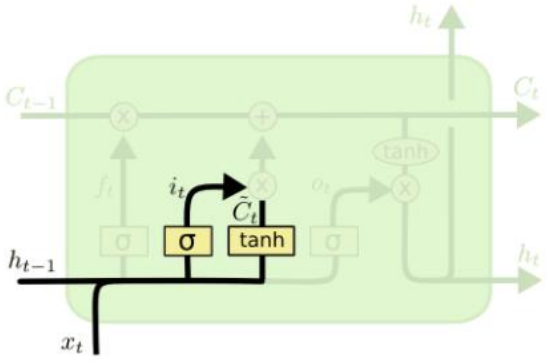
1.4.3 Bên trong LSTM

- Bước đầu tiên của LSTM là quyết định xem thông tin nào cần bỏ đi từ trạng thái tế bào. Quyết định này được đưa ra bởi tầng sigmoid - gọi là "tầng cổng quên" (forget gate layer). Nó sẽ lấy đầu vào là h_{t-1} và x_t rồi đưa ra kết quả là một số trong khoảng $[0,1]$ cho mỗi số trong trạng thái tế bào C_{t-1}. Đầu ra là1 thể hiện rằng nó giữ toàn bộ thông tin lại, còn 0 chỉ rằng toàn bộ thông tin sẽ bị bỏ đi.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

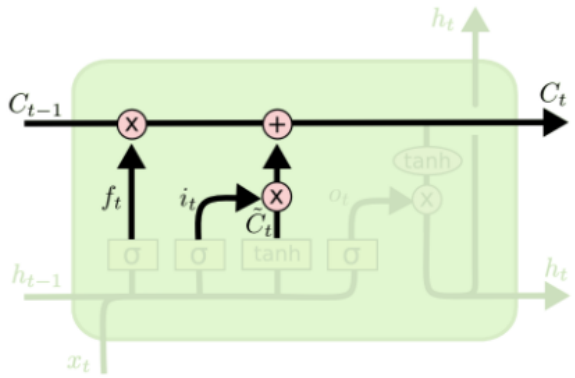
- Bước tiếp theo là quyết định xem thông tin mới nào ta sẽ lưu vào trạng thái tế bào. Việc này gồm 2 phần. Đầu tiên là sử dụng một tầng sigmoid được gọi là "tầng cổng vào" (input gate layer) để quyết định giá trị nào ta sẽ cập nhập. Tiếp theo là một tầng tanh tạo ra một véc-tơ cho giá trị mới C_t nhằm thêm vào cho trạng thái. Trong bước tiếp theo, ta sẽ kết hợp 2 giá trị đó lại để tạo ra một cập nhập cho trạng thái.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

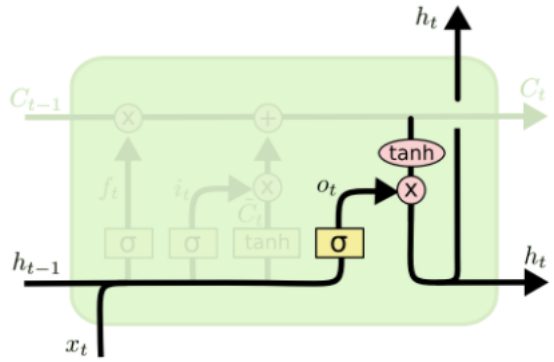
- Giờ là lúc cập nhập trạng thái tế bào cũ C_{t-1} thành trạng thái mới C_t. Ở các bước trước đó đã quyết định những việc cần làm, nên giờ ta chỉ cần thực hiện là xong.

- Ta sẽ nhân trạng thái cũ với f_t để bỏ đi những thông tin ta quyết định quên lúc trước. Sau đó cộng thêm $i_t * C_t\sim$. Trạng thái mơi thu được này phụ thuộc vào việc ta quyết định cập nhập mỗi giá trị trạng thái ra sao.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Cuối cùng, ta cần quyết định xem ta muốn đầu ra là gì. Giá trị đầu ra sẽ dựa vào trạng thái tế bào, nhưng sẽ được tiếp tục sàng lọc. Đầu tiên, ta chạy một tầng sigmoid để quyết định phần nào của trạng thái tế bào ta muốn xuất ra. Sau đó, ta đưa nó trạng thái tế bảo qua một hàm tanh để co giá trị nó về khoảng [-1, 1], và nhân nó với đầu ra của cổng sigmoid để được giá trị đầu ra ta mong muốn.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

1.4.4 Kết luận

- LSTM là một bước lớn trong việc sử dụng RNN. Ý tưởng của nó giúp cho tất cả các bước của RNN có thể truy vấn được thông tin từ một tập thông tin lớn hơn. Ví dụ, nếu bạn sử dụng RNN để tạo mô tả cho một bức ảnh, nó có thể lấy một phần ảnh để dự đoán mô tả từ tất cả các từ đầu vào.

CHƯƠNG 2 – DEMO & KẾT QUẢ

2.1 Import thư viện, Read Data

- Đầu tiên là import thư viện cần dùng.

Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt

import seaborn as sns
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.callbacks import EarlyStopping
```

- Tiếp theo là đọc Electric Production Data Set.

- Dataset gồm 2 cột, mộ cột là DATE và cột còn lại là Value (phần trăm tiêu thụ điện).

- Chuyển DATE sang dạng yyyy/MM/dd và để col DATE làm index trong df.

Read Data

```
file_path = "./Electric_Production.csv"
```

```
# Chuyển đổi sang dạng yyyy/MM/dd và để DATE làm index
df = pd.read_csv(file_path, index_col="DATE", parse_dates=True)
print(df.shape)
df.tail()
```

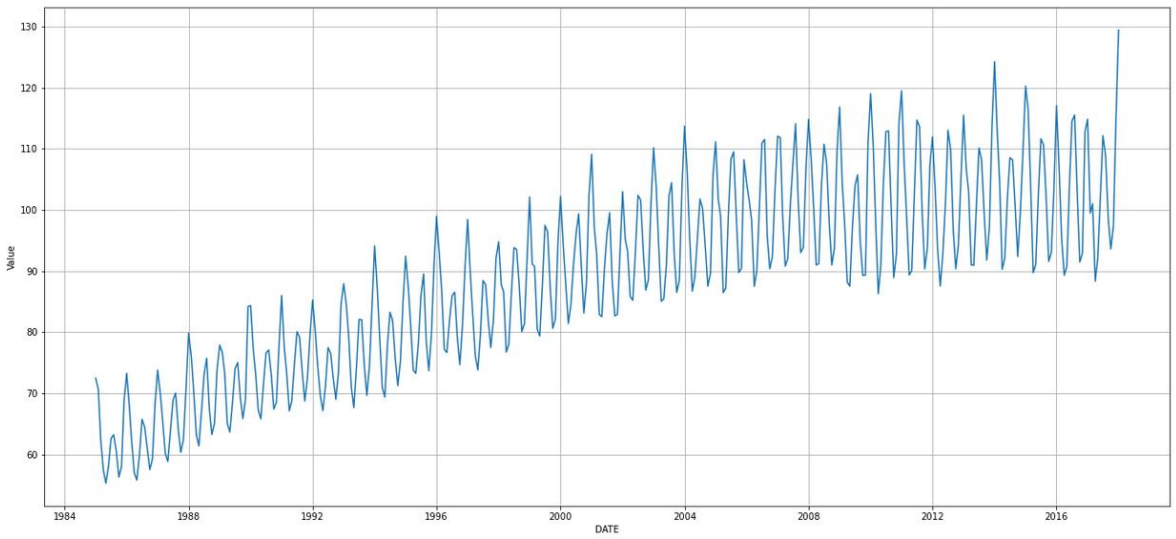
```
(397, 1)
```

DATE	Value
2017-09-01	98.6154
2017-10-01	93.6137
2017-11-01	97.3359
2017-12-01	114.7212
2018-01-01	129.4048

- Vẽ biểu đồ thể hiện Value qua từng tháng từ năm 1984 -> 2018.

```
# Vẽ thể hiện Value qua từng tháng từ năm 1984 -> 2018
plt.figure(figsize=(22,10))
plt.grid()
sns.lineplot(x=df.index, y='Value', data=df)
```

```
<AxesSubplot:xlabel='DATE', ylabel='Value'>
```



2.2 Preprocessing

- Chuyển đổi Value thành mảng numpy và sang dạng float32.

Preprocessing

```
# Chuyển đổi Value thành mảng numpy
dataset = df.values
```

```
# Đổi sang dạng float
dataset = dataset.astype("float32")
```

- Chuẩn hóa dataset bằng MinMaxScaler.

```
# Chuẩn hóa dataset
scaler = MinMaxScaler()
dataset = scaler.fit_transform(dataset)
```

- Với dataset này, chúng ta không thể sử dụng cách ngẫu nhiên để tách tập dữ liệu thành train và test vì chuỗi sự kiện rất quan trọng trong bài toán time series. Vì vậy, chúng ta sẽ lấy 80% giá trị đầu tiên cho train và còn lại để test.

```
# không thể sử dụng cách ngẫu nhiên để tách tập dữ liệu thành huấn luyện và kiểm tra
# vì chuỗi sự kiện rất quan trọng đối với chuỗi thời gian
# Tách 80% dataset làm train và 20% còn lại làm test
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

- Chúng ta không thể fit model như chúng ta thường khi chúng ta có X và Y. Chúng ta cần chuyển đổi dữ liệu của mình thành một thứ gì đó tương tự như X và Y. Bằng cách này, nó có thể được đào tạo trên một chuỗi thay vì điểm dữ liệu trực quan.

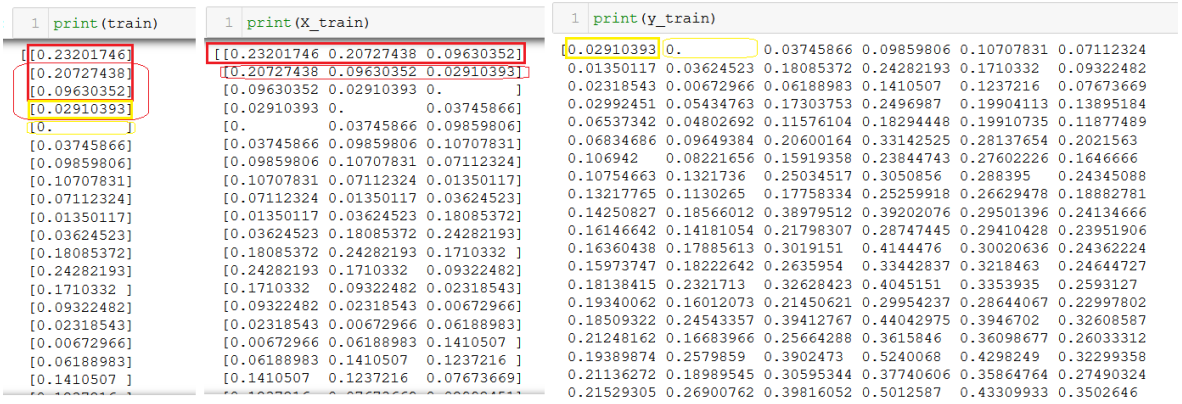
- Chúng ta sẽ chuyển đổi thành n số cột cho X trong đó cung cấp chuỗi số sau n cột đó - cột cuối cùng là Y, chúng ta cung cấp số tiếp theo trong chuỗi là n+1. Vì vậy, chúng ta sẽ chuyển đổi một mảng giá trị thành một ma trận tập dữ liệu.

- seq_size là timestep trước đó được sử dụng làm biến đầu vào để dự đoán khoảng thời gian tiếp theo.

- Tạo tập dữ liệu trong đó X là số lượng % điện tiêu thụ tại một thời điểm nhất định (t, t-1, t-2). Y là số lượng % điện tiêu thụ tại thời thời điểm tiếp theo (t + 1).

```
# chuyển đổi mảng giá trị thành một ma trận tập dữ liệu
# trong đó X là số lượng % điện tiêu thụ tại một thời điểm nhất định (t, t-1, t-2)
# Y là số lượng % điện tiêu thụ tại thời thời điểm tiếp theo (t + 1)
def create_dataset(dataset, seq_size=1):
    dataX, dataY = [], []

    for i in range(len(dataset) - seq_size - 1):
        window = dataset[i:(i + seq_size), 0]
        dataX.append(window)
        dataY.append(dataset[i + seq_size, 0])
    return np.array(dataX), np.array(dataY)
```

```
seq_size = 3 # Số timesteps
X_train, y_train = create_dataset(train, seq_size)
X_test, y_test = create_dataset(test, seq_size)
```



```
print(train)
[[0.23201746]
 [0.20727438]
 [0.09630352]
 [0.02910393]
 [0.        ]
 [0.03745866]
 [0.09859806]
 [0.10707831]
 [0.07112324]
 [0.01350117]
 [0.03624523]
 [0.18085372]
 [0.24282193]
 [0.1710332 ]
 [0.09322482]
 [0.02318543]
 [0.00672966]
 [0.06188983]
 [0.1410507 ]
```

```
print(X_train)
[[0.23201746 0.20727438 0.09630352]
 [0.20727438 0.09630352 0.02910393]
 [0.09630352 0.02910393 0.        ]
 [0.02910393 0.         0.03745866]
 [0.         0.03745866 0.09859806]
 [0.03745866 0.09859806 0.10707831]
 [0.09859806 0.10707831 0.07112324]
 [0.10707831 0.07112324 0.01350117]
 [0.07112324 0.01350117 0.03624523]
 [0.01350117 0.03624523 0.18085372]
 [0.03624523 0.18085372 0.24282193]
 [0.18085372 0.24282193 0.1710332 ]
 [0.24282193 0.1710332  0.09322482]
 [0.1710332  0.09322482 0.02318543]
 [0.09322482 0.02318543 0.00672966]
 [0.02318543 0.00672966 0.06188983]
 [0.00672966 0.06188983 0.1410507 ]
 [0.06188983 0.1410507  0.1237216 ]
 [0.1410507  0.1237216  0.07673669]
```

```
print(y_train)
[0.02910393 0.         0.03745866 0.09859806 0.10707831 0.07112324
 0.01350117 0.03624523 0.18085372 0.24282193 0.1710332  0.09322482
 0.02318543 0.00672966 0.06188983 0.1410507  0.1237216  0.07673669
 0.02992451 0.05434763 0.17303753 0.2496987  0.19904113 0.13895184
 0.06537342 0.04802692 0.11576104 0.18294448 0.19910735 0.11877489
 0.06834686 0.09649384 0.20600164 0.33142525 0.28137654 0.2021563
 0.106942   0.08221656 0.15919358 0.23844743 0.27602226 0.1646666
 0.10754663 0.1321736  0.25034517 0.3050856  0.288395   0.24345088
 0.13217765 0.1130265  0.17758334 0.25259918 0.26629478 0.18882781
 0.14250827 0.18566012 0.38979512 0.39202076 0.29501396 0.24134666
 0.16146642 0.14181054 0.21798307 0.28747445 0.29410428 0.23951906
 0.16360438 0.17885613 0.3019151  0.4144476  0.30020636 0.24362224
 0.15973747 0.18222642 0.2635954  0.33442837 0.3218463  0.24644727
 0.18138415 0.2321713  0.32628423 0.4045151  0.3353935  0.2593127
 0.19340062 0.16012073 0.21450621 0.29954237 0.28644067 0.22997802
 0.18509322 0.24543357 0.39412767 0.44042975 0.3946702  0.32608587
 0.21248162 0.16683966 0.25664288 0.3615846  0.36098677 0.26033312
 0.19389874 0.2579859  0.3902473  0.5240068  0.4298249  0.32299358
 0.21136272 0.18989545 0.30595344 0.37740606 0.35864764 0.27490324
 0.21529305 0.26900762 0.39816052 0.5012587  0.43309933 0.3502646
```

- So sánh X_train và tập train, chúng ta có thể thấy các giá trị X = tại t, t+1 và t+2. Trong đó Y là giá trị theo sau: t+3 (vì kích thước chuỗi của chúng tôi là 3).

2.3 Feed Forward Neural Networks

2.3.1 Define Model 1

- Khởi tạo model bằng Sequential() của Keras.

- Sau đó, xác định Dense Layer với output_dim=64, input_dim=seq_size, activation=sigmoid.

- Lớp Dense cuối là kết quả đầu ra, chúng ta chỉ có 1 đầu ra là giá trị y - giá trị tiếp theo.

- Model compile với loss là mean_squared _error, optimizer là adam.

Feed Forward Neural Networks

MODEL 1

```
# hidden layer = 64
model_FFNN_1 = Sequential()
model_FFNN_1.add(Dense(64, input_dim=seq_size, activation="sigmoid"))
model_FFNN_1.add(Dense(1)) # kết quả đầu ra chỉ có 1 là Y (giá trị tiếp theo)
model_FFNN_1.compile(loss="mean_squared_error", optimizer="adam")
model_FFNN_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
```

- Tiếp theo dùng EarlyStopping – training sẽ stop khi training không cải thiện model với monitor là loss và patience = 3 (Sau 3 epochs mà loss không cải thiện thì stop training).

```
earlyStopping = EarlyStopping(monitor='loss', patience=3)
```

2.3.2 Training Model 1

- EPOCHS = 100, BATCH_SIZE = 64

```
EPOCHS = 100
BATCH_SIZE = 64

history_FFNN_1 = model_FFNN_1.fit(X_train, y_train,
                                  epochs=EPOCHS,
                                  batch_size=BATCH_SIZE,
                                  callbacks=[earlyStopping],
                                  validation_data=(X_test, y_test),
                                  verbose=2)
```

```
5/5 - 0s - loss: 0.0069 - val_loss: 0.0116
Epoch 57/100
5/5 - 0s - loss: 0.0069 - val_loss: 0.0112
Epoch 58/100
5/5 - 0s - loss: 0.0069 - val_loss: 0.0116
Epoch 59/100
5/5 - 0s - loss: 0.0069 - val_loss: 0.0117
Epoch 60/100
5/5 - 0s - loss: 0.0069 - val_loss: 0.0118
Epoch 61/100
5/5 - 0s - loss: 0.0069 - val_loss: 0.0113
Epoch 62/100
5/5 - 0s - loss: 0.0068 - val_loss: 0.0116
Epoch 63/100
5/5 - 0s - loss: 0.0069 - val_loss: 0.0119
Epoch 64/100
5/5 - 0s - loss: 0.0068 - val_loss: 0.0113
Epoch 65/100
5/5 - 0s - loss: 0.0068 - val_loss: 0.0113
```

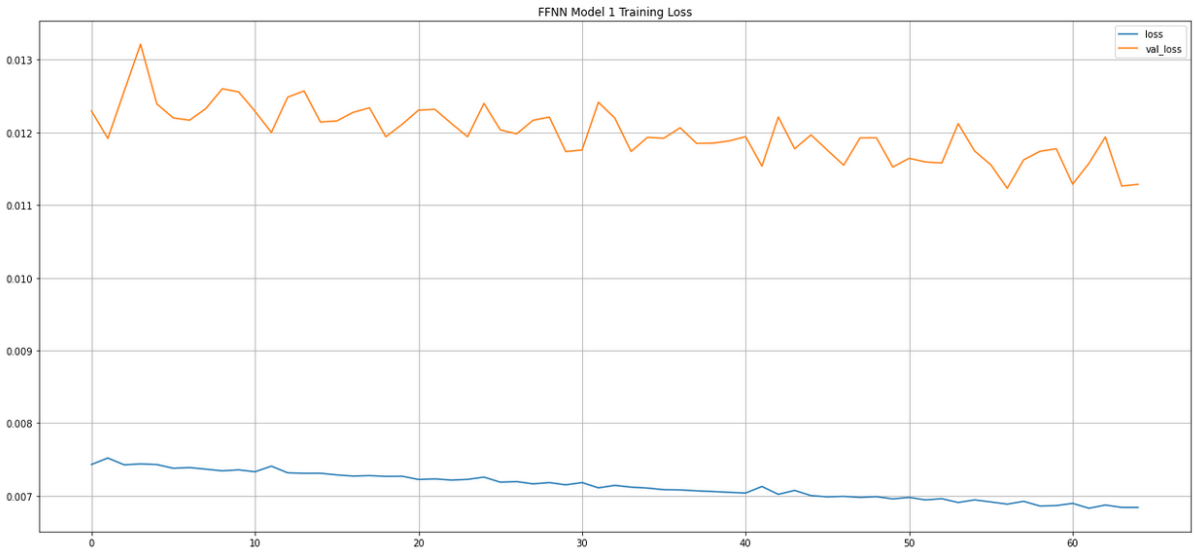
2.3.3 Plot Loss trong Model 1

- Tạo 1 hàm để vẽ đồ thị thể hiện loss của train và loss của validation để tiện dụng cho các model sau.

PLOT

```
def plotModelLoss(loss, val_loss, title):
    plt.figure(figsize=(22, 10))
    plt.plot(loss, label="loss")
    plt.plot(val_loss, label="val_loss")
    plt.grid()
    plt.legend()
    plt.title(title)
```

- Đồ thị thể hiện loss của train và loss của validation trong model 1.



2.3.3 Predict với Model 1

- Tạo 1 hàm predictionAndPlot để sử dụng model predict cho tập train và test và vẽ lên đồ thị thể hiện predict train và test so với dataset.

- Đầu tiên, chúng ta sẽ dùng model predict cho tập X_train, X_test.

PREDICT

```
def predictionAndPlot(model, X_train, X_test, dfTrainPredict, dfTestPredict):
    # make predictions
    trainPredict = model.predict(X_train)
    testPredict = model.predict(X_test)
```

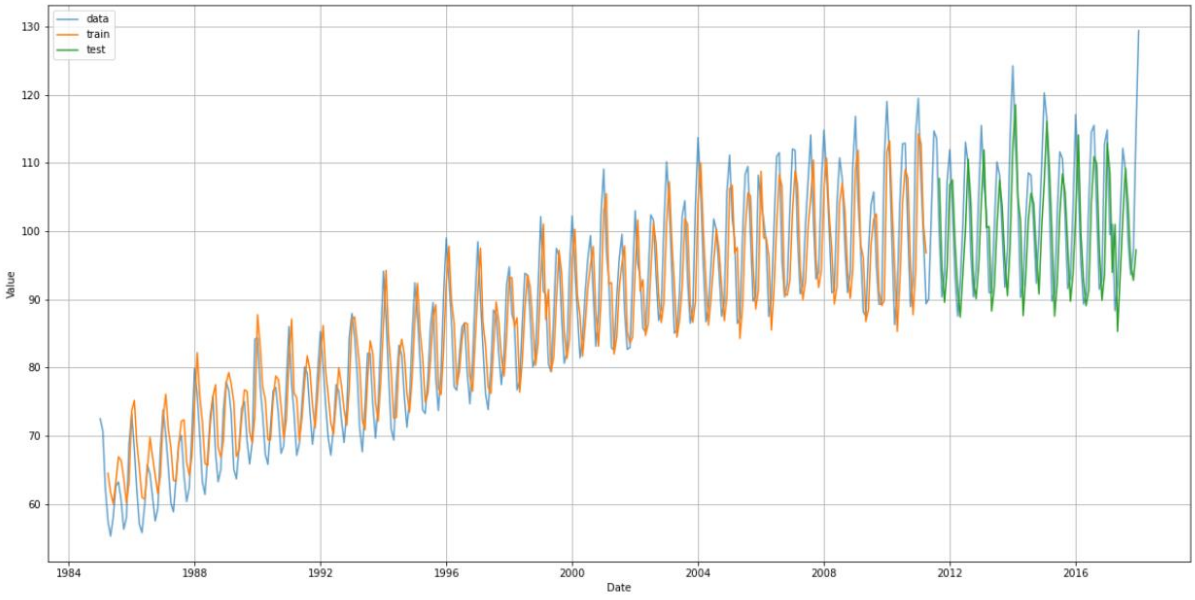
- Sau đó, chúng ta sẽ chuẩn hóa ngược lại với scaler.inverse_transform để cùng đơn vị với dataset ban đầu.

```
# Trước đó đã dùng minmaxscaler nên giờ đảo ngược nó lại
trainPredict = scaler.inverse_transform(trainPredict)
testPredict = scaler.inverse_transform(testPredict)
```

- Tiếp theo, chúng ta sẽ căn chỉnh trainPredict, testPredict lại cho khớp giá trị với dataset ban đầu để có thể vẽ ra để so sánh.

```
# Căn chỉnh lại để có thể so sánh dataset
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[seq_size:len(trainPredict) + seq_size, :] = trainPredict

testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(seq_size*2)+1:len(dataset)-1, :] = testPredict
```

- Sau đó, đưa trainPredictPlot và testPredictPlot vào df để vẽ ra vào so sanh với dataset theo năm.



2.3.4 Define Model 2

- Khởi tạo model bằng Sequential() của Keras.

- Sau đó, xác định Dense Layer với output_dim=64, input_dim=seq_size, activation=relu.

- Thêm một Dense Layer với output_dim=32, activation=relu.

- Lớp Dense cuối là kết quả đầu ra, chúng ta chỉ có 1 đầu ra là giá trị y - giá trị tiếp theo.

- Model compile với loss là mean_squared _error, optimizer là adam.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 64)                256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,369
Trainable params: 2,369
Non-trainable params: 0
_________________________________________________________________
```

2.3.5 Training Model 2

- EPOCHS = 30, BATCH_SIZE = 32

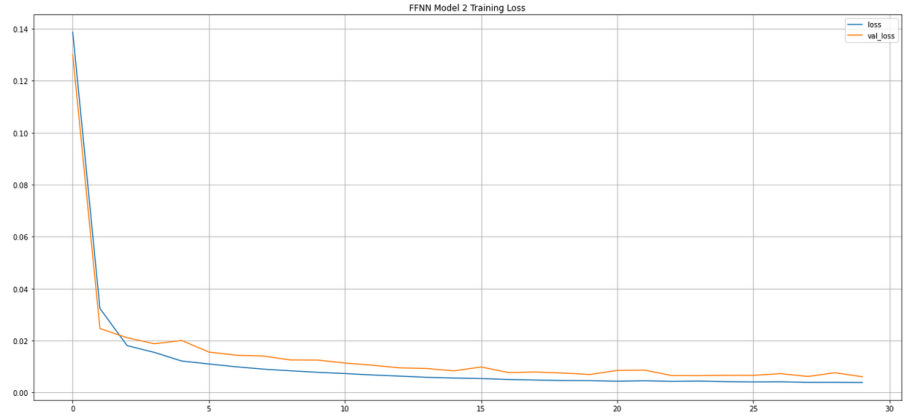
```
EPOCHS = 30
BATCH_SIZE = 32

history_FFNN_2 = model_FFNN_2.fit(X_train, y_train,
                                  epochs=EPOCHS,
                                  batch_size=BATCH_SIZE,
                                  callbacks=[earlyStopping],
                                  validation_data=(X_test, y_test),
                                  verbose=2)
```

```
10/10 - 0s - loss: 0.0043 - val_loss: 0.0085
Epoch 22/30
10/10 - 0s - loss: 0.0045 - val_loss: 0.0086
Epoch 23/30
10/10 - 0s - loss: 0.0043 - val_loss: 0.0065
Epoch 24/30
10/10 - 0s - loss: 0.0044 - val_loss: 0.0065
Epoch 25/30
10/10 - 0s - loss: 0.0042 - val_loss: 0.0066
Epoch 26/30
10/10 - 0s - loss: 0.0040 - val_loss: 0.0066
Epoch 27/30
10/10 - 0s - loss: 0.0041 - val_loss: 0.0073
Epoch 28/30
10/10 - 0s - loss: 0.0039 - val_loss: 0.0062
Epoch 29/30
10/10 - 0s - loss: 0.0039 - val_loss: 0.0076
Epoch 30/30
10/10 - 0s - loss: 0.0039 - val_loss: 0.0061
```

2.3.6 Plot Loss trong Model 2

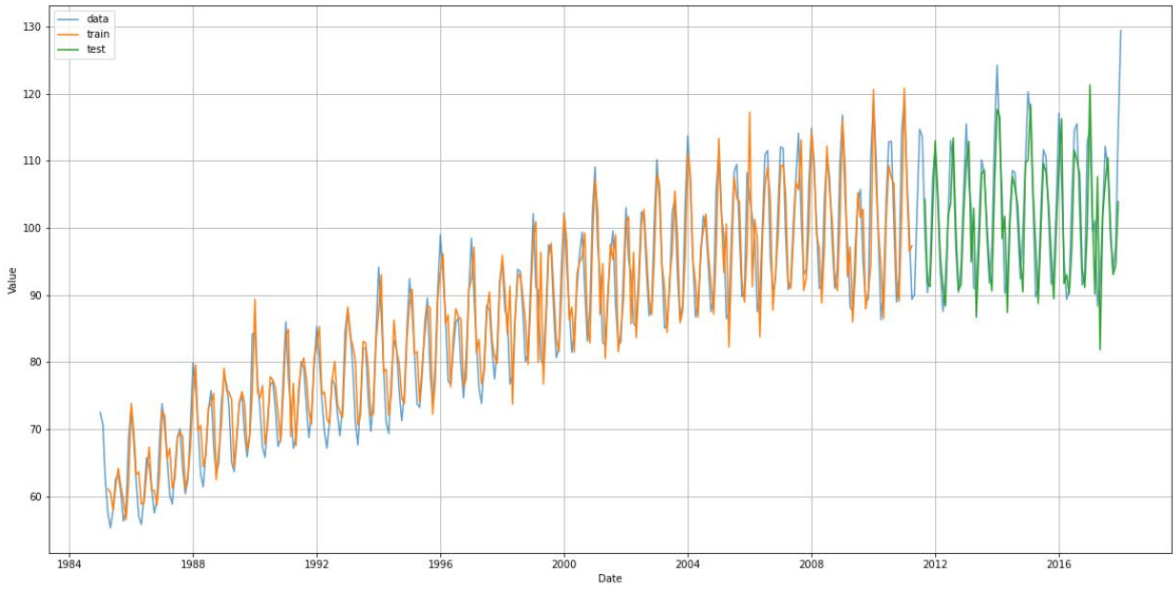
- Đồ thị thể hiện loss của train và loss của validation trong model 2.

```
plotModelLoss(history_FFNN_2.history['loss'], history_FFNN_2.history['val_loss'], "FFNN Model 2 Training Loss")
```



2.3.7 Predict với Model 2

```
predictionAndPlot(model_FFNN_2, X_train, X_test, "trainPredict_FFNN_2", "testPredict_FFNN_2")
```



2.4 Long short term memory – LSTM

LSTM model cần input data X với cấu trúc mảng ở dạng [samples, timesteps, features]. Nên chúng ta sẽ reshape lại data train và test.

```
# reshape input to be [samples, time steps, features]
X_train_LSTM = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test_LSTM = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

```
print(X_train_LSTM.shape, X_test_LSTM.shape)
```

```
(313, 1, 3) (76, 1, 3)
```

2.4.1 Define Model 1

- Khởi tạo model bằng Sequential() của Keras.

- Sau đó, thêm 1 LSTM Layer với unit= 64 và input_shape=(None, seq_size) cho khớp với X đầu vào.

- Thêm một Dense Layer với output_dim=32.

- Lớp Dense cuối là kết quả đầu ra, chúng ta chỉ có 1 đầu ra là giá trị y - giá trị tiếp theo.

- Model compile với loss là mean_squared _error, optimizer là adam.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 64)                17408
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 19,521
Trainable params: 19,521
Non-trainable params: 0
_________________________________________________________________
```

2.4.2 Training Model 1

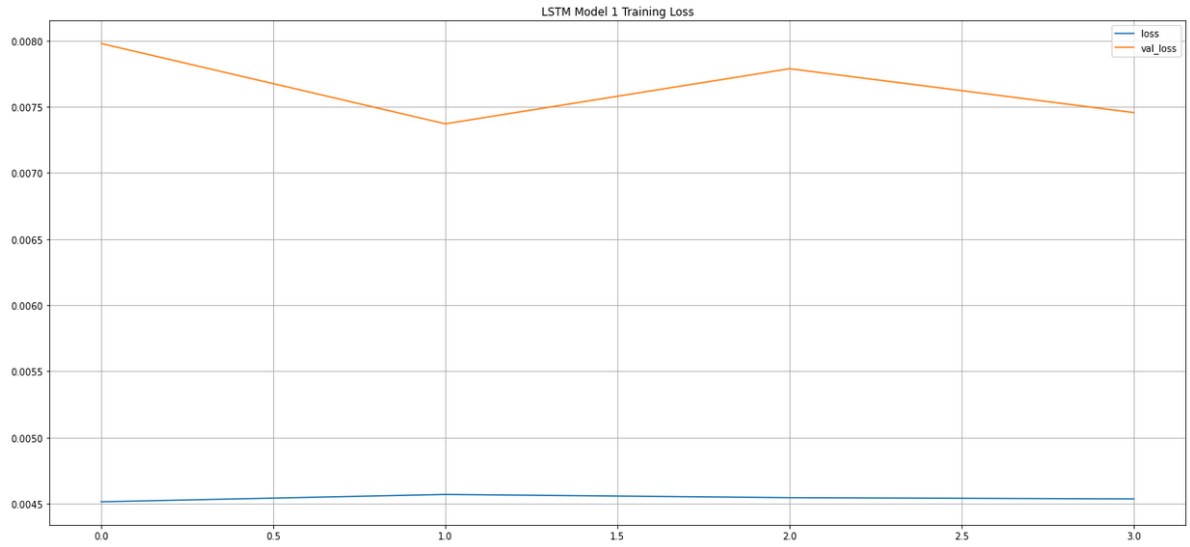
- EPOCHS = 30, BATCH_SIZE = 64

```
1  EPOCHS = 30
2  BATCH_SIZE = 64
3
4
5  history_LSTM_1 = model_LSTM_1.fit(X_train_LSTM, y_train,
6                       epochs=EPOCHS,
7                       batch_size=BATCH_SIZE,
8                       validation_data=(X_test_LSTM, y_test),
9                       callbacks=[earlyStopping],
10                      verbose=2)
11
```

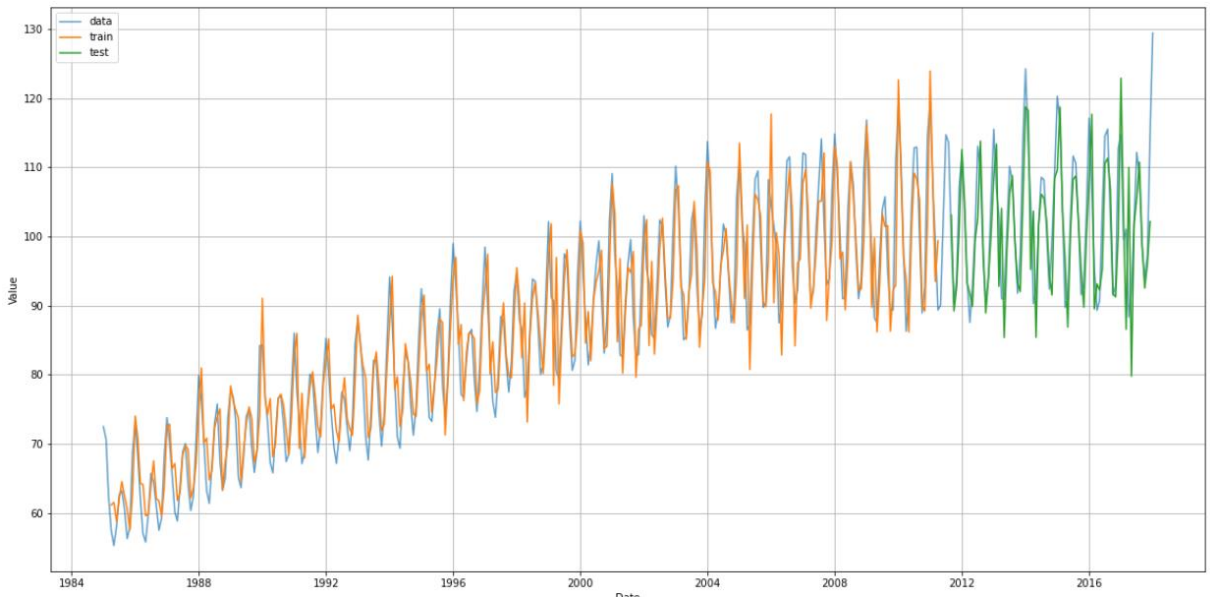
```
Epoch 1/30
5/5 - 0s - loss: 0.0045 - val_loss: 0.0080
Epoch 2/30
5/5 - 0s - loss: 0.0046 - val_loss: 0.0074
Epoch 3/30
5/5 - 0s - loss: 0.0045 - val_loss: 0.0078
Epoch 4/30
5/5 - 0s - loss: 0.0045 - val_loss: 0.0075
```

2.4.3 Plot Loss trong Model 1

- Đồ thị thể hiện loss của train và loss của validation trong model 1.



2.4.4 Predict với Model 1



2.4.5 Define Model 2

- Khởi tạo model bằng Sequential() của Keras.

- Sau đó, thêm 1 LSTM Layer với unit=50, activation=relu, return_sequences=true, nếu không nó sẽ không trả về chuỗi và sẽ không có đầu vào cho LSTM Layer tiếp theo và input_shape=(None, seq_size) cho khớp với X đầu vào.

- Thêm 1 LSTM Layer với unit=50, activation=relu.

- Thêm một Dense Layer với output_dim=32.

- Lớp Dense cuối là kết quả đầu ra, chúng ta chỉ có 1 đầu ra là giá trị y - giá trị tiếp theo.

- Model compile với loss là mean_squared _error, optimizer là adam.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, None, 50)          10800
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_4 (Dense)              (None, 32)                1632
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 32,665
Trainable params: 32,665
Non-trainable params: 0
```

2.4.6 Training Model 2

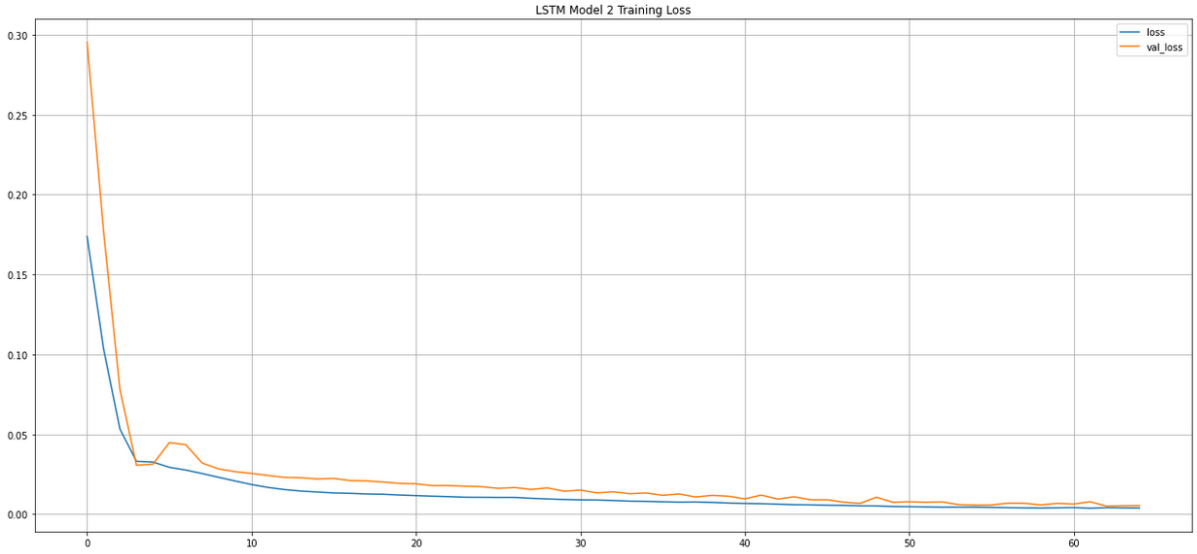
- EPOCHS = 100, BATCH_SIZE = 32

```
EPOCHS1 = 100
BATCH_SIZE1 = 32

history_LSTM_2 = model_LSTM_2.fit(X_train_LSTM, y_train,
                    epochs=EPOCHS1,
                    batch_size=BATCH_SIZE1,
                    validation_data=(X_test_LSTM, y_test),
                    callbacks=[earlyStopping],
                    verbose=2
                   )
```

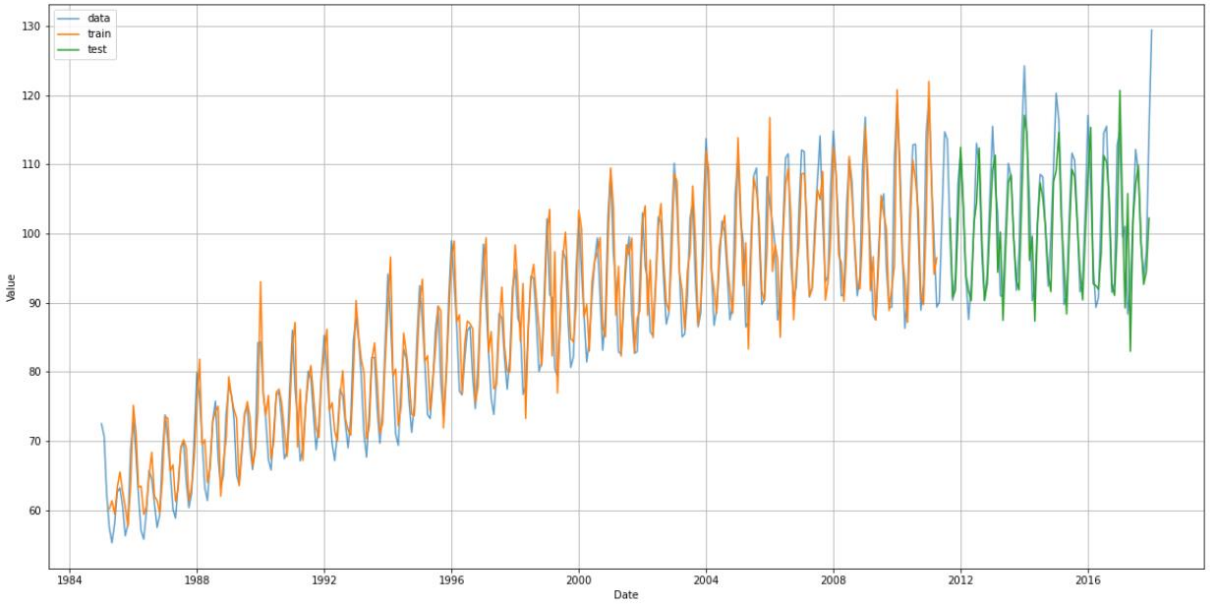
```
10/10 - 0s - loss: 0.0043 - val_loss: 0.0058
Epoch 57/100
10/10 - 0s - loss: 0.0041 - val_loss: 0.0069
Epoch 58/100
10/10 - 0s - loss: 0.0040 - val_loss: 0.0069
Epoch 59/100
10/10 - 0s - loss: 0.0039 - val_loss: 0.0058
Epoch 60/100
10/10 - 0s - loss: 0.0040 - val_loss: 0.0068
Epoch 61/100
10/10 - 0s - loss: 0.0042 - val_loss: 0.0064
Epoch 62/100
10/10 - 0s - loss: 0.0038 - val_loss: 0.0078
Epoch 63/100
10/10 - 0s - loss: 0.0041 - val_loss: 0.0051
Epoch 64/100
10/10 - 0s - loss: 0.0040 - val_loss: 0.0053
Epoch 65/100
10/10 - 0s - loss: 0.0039 - val_loss: 0.0054
```

2.4.7 Plot Loss trong Model 2

- Đồ thị thể hiện loss của train và loss của validation trong model 1.



2.4.8 Predict với Model 2



TÀI LIỆU THAM KHẢO

1. https://www.bogotobogo.com/python/python_Neural_Networks_Backpropagation_for_XOR_using_one_hidden_layer.phphttps://www.vuthao.com.vn/tintuc/business-process-management
2. https://machinelearningcoban.com/2017/02/24/mlp/#-backpropagation-cho-stochastic-gradient-descent
3. https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/14_mlp/Example%20.ipynb