

# **Bài 9:** **(Recurrent neural network)**



# Mục tiêu

## ■ Các mạng neuron truyền thẳng

- ◆ Đầu ra của mạng chỉ phụ thuộc vào đầu vào của mạng
- ◆ Hạn chế trong một số trường hợp

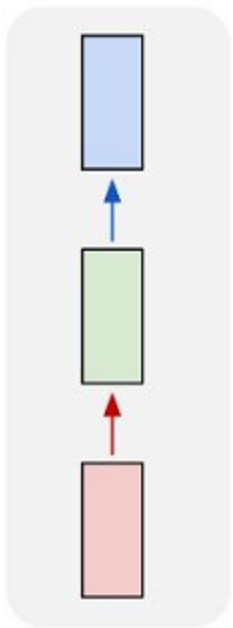
## ■ Mạng hồi quy

- ◆ Đầu ra của mạng không những phụ thuộc vào đầu vào mà còn phụ thuộc vào đầu ra trước đó
- ◆ Nhiều ứng dụng trong thực tế
  - ★ Các hệ thống điều khiển
  - ★ Tiếng nói
  - ★ Chuỗi ký tự
  - ★ Việc hiểu của người (dựa trên các tri thức đã có từ trước)

# Giới thiệu

## ■ Vanilla neural network

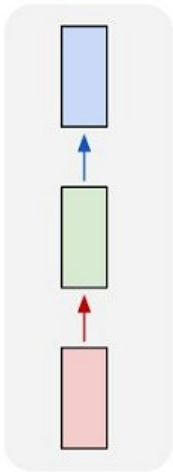
one to one



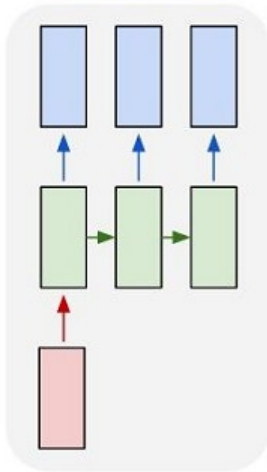
**Vanilla Neural Networks**

# Giới thiệu

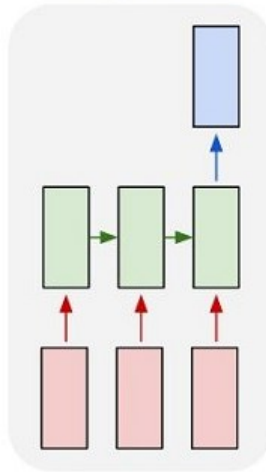
one to one



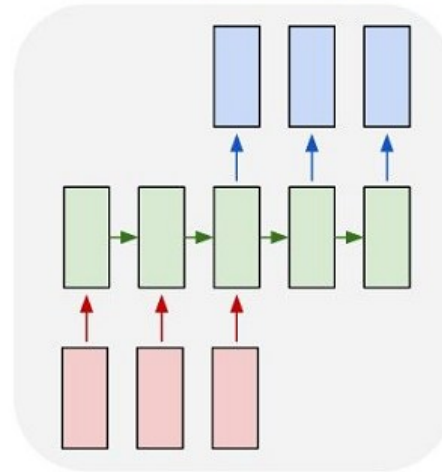
one to many



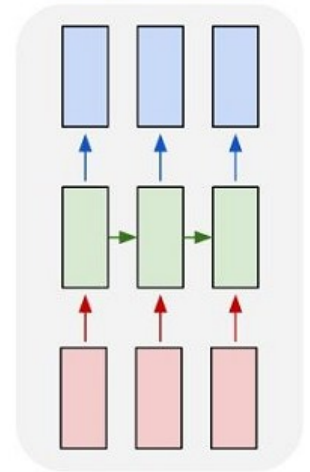
many to one



many to many



many to many



↖ e.g. **Image Captioning**  
image -> sequence of words

# Image Captioning



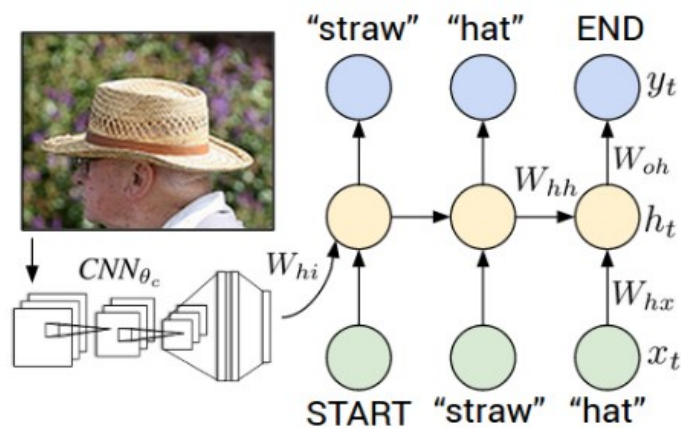
"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."

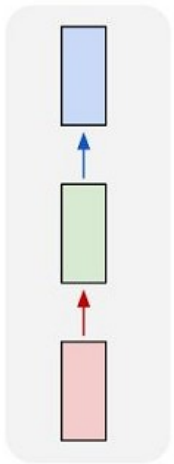


"two young girls are playing with lego toy."

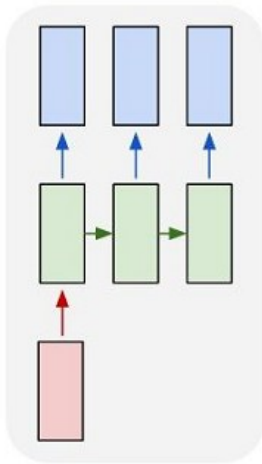


# Giới thiệu

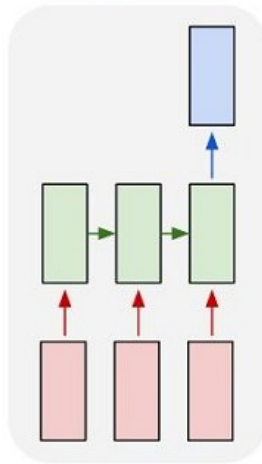
one to one



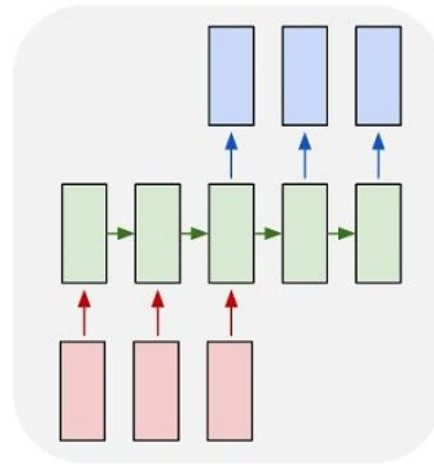
one to many



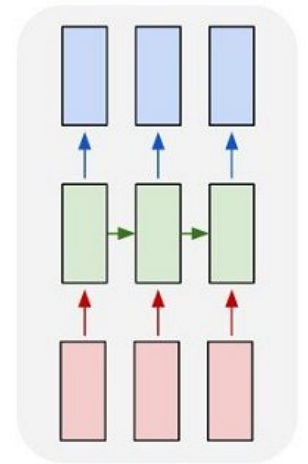
many to one



many to many

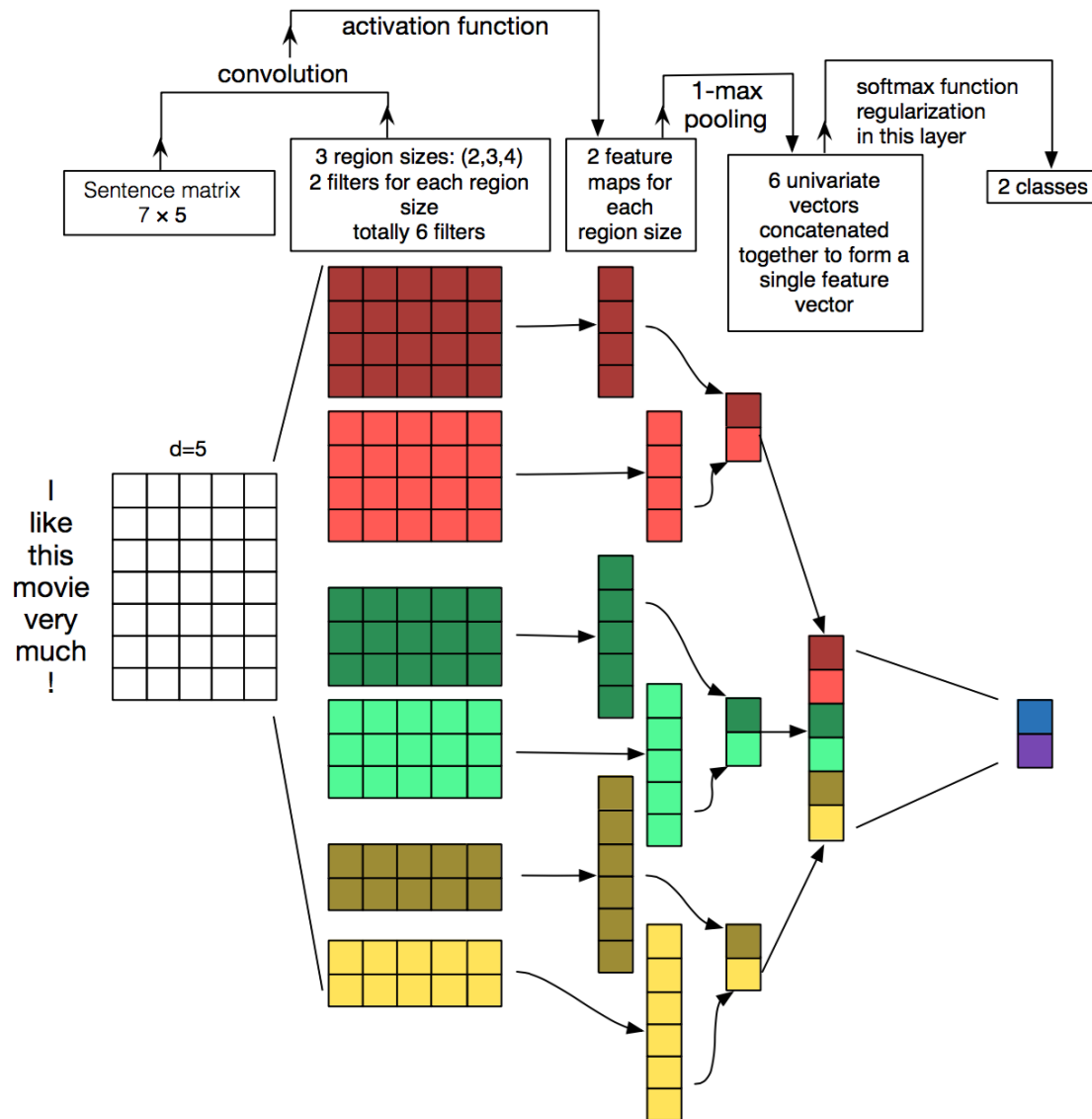


many to many



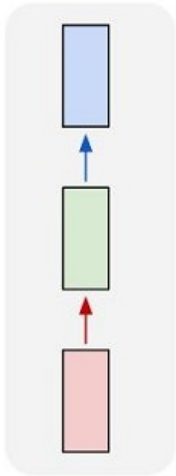
e.g. **Sentiment Classification**  
sequence of words -> sentiment

# Sentiment classification

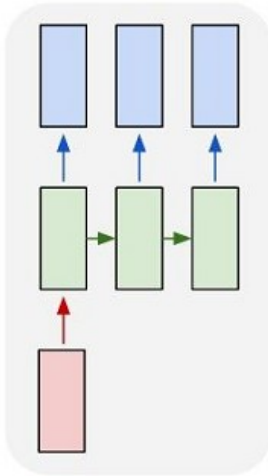


# Giới thiệu

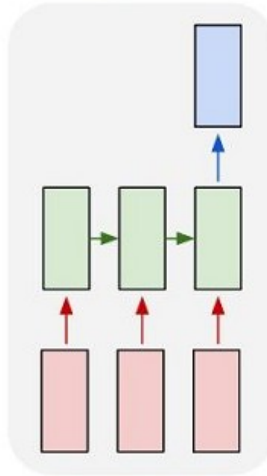
one to one



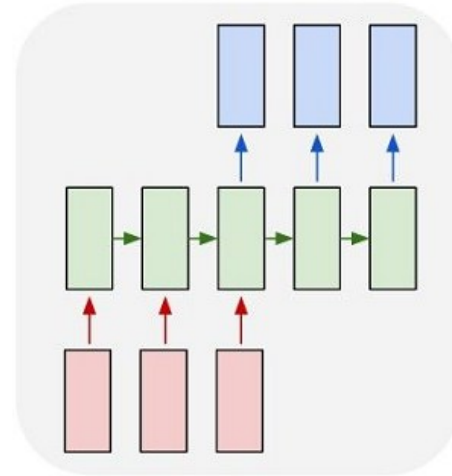
one to many



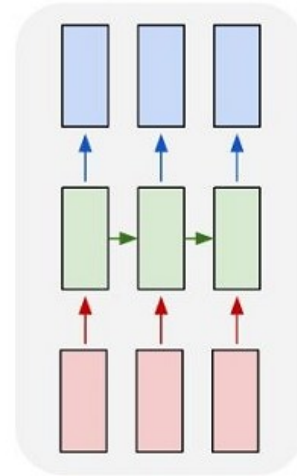
many to one



many to many



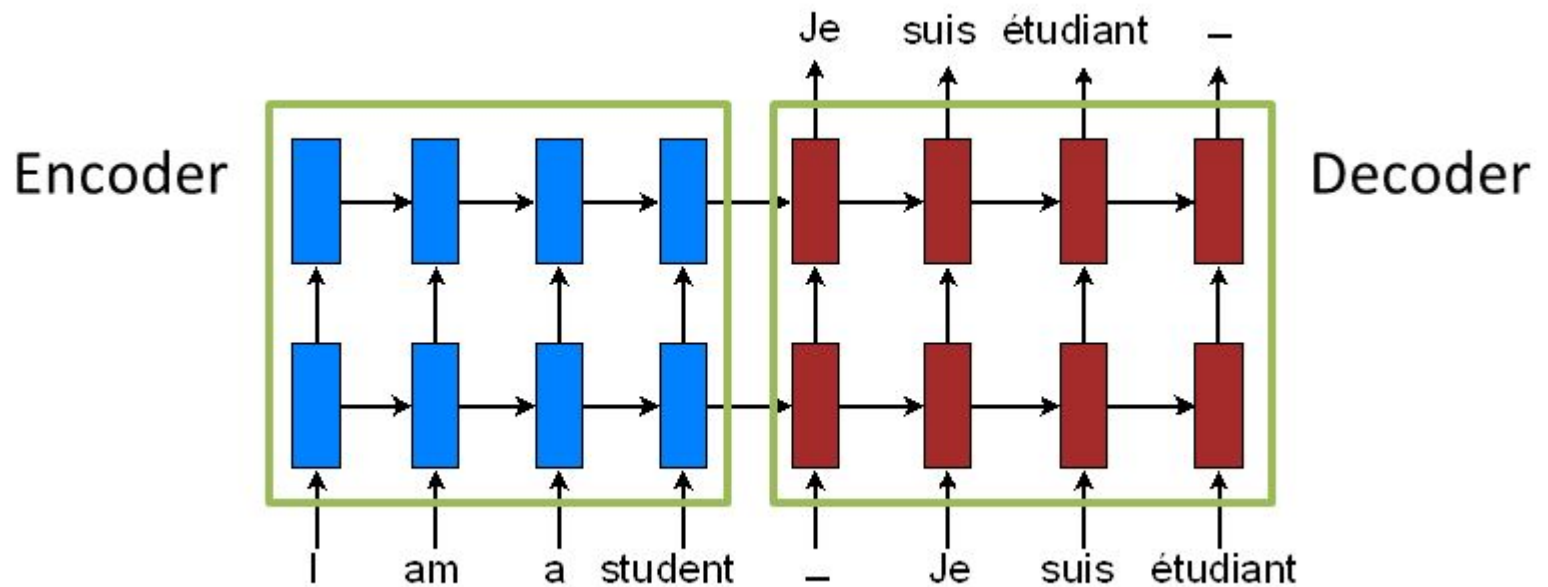
many to many



e.g. **Machine Translation**  
seq of words -> seq of words

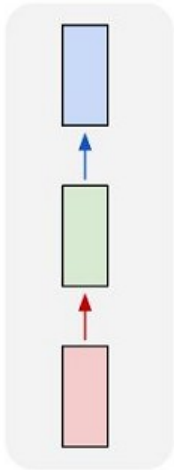


# Neural machine translation

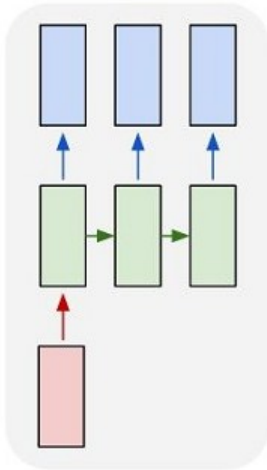


# Giới thiệu

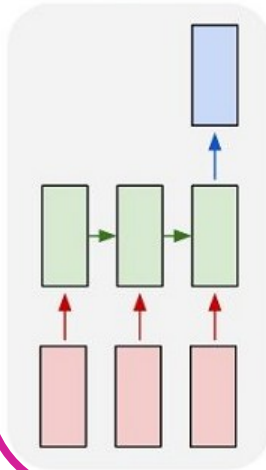
one to one



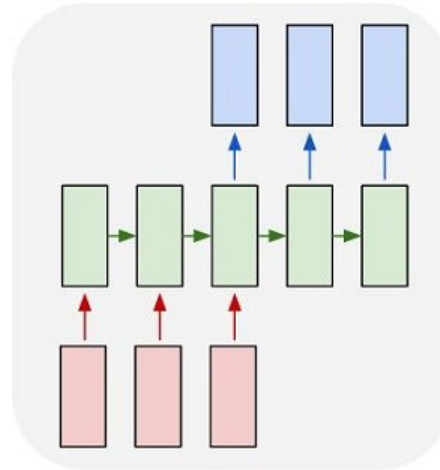
one to many



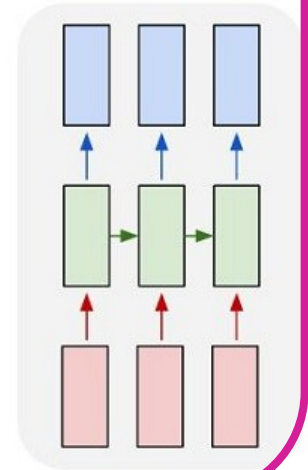
many to one



many to many

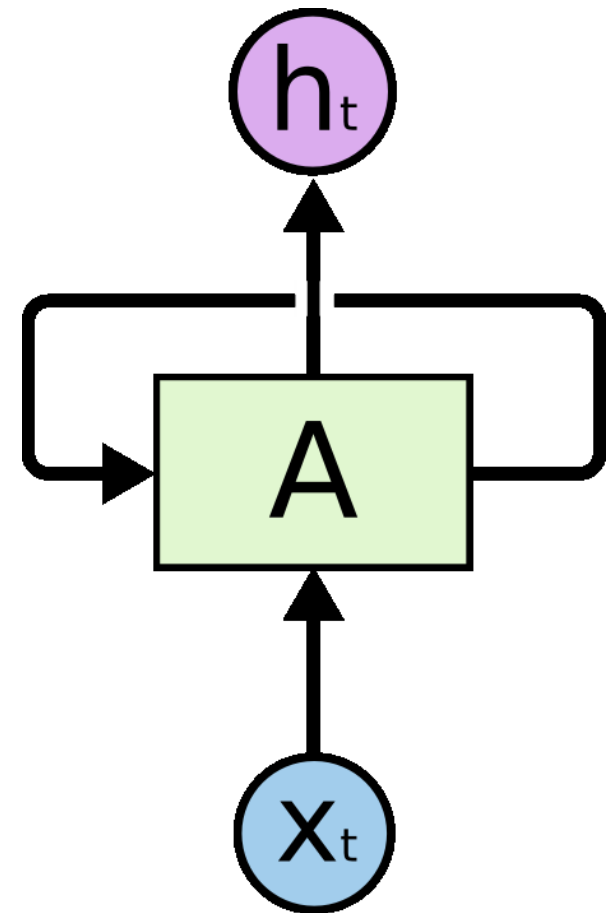
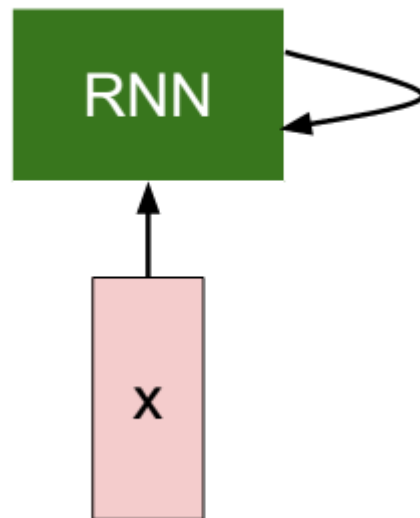


many to many



e.g. **Video classification on frame level**

# Mạng neuron hồi quy



# Hàm truyền đạt

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

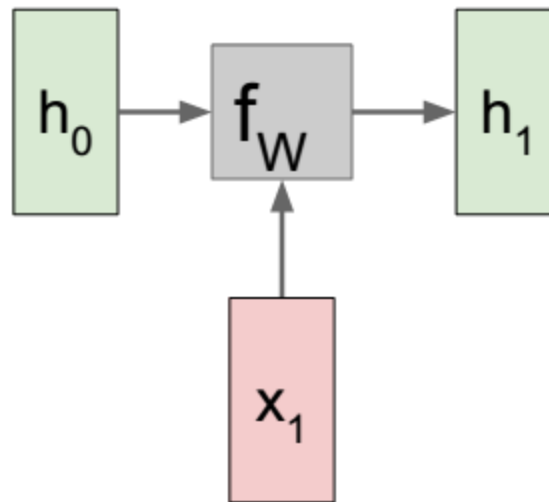
old state

input vector at  
some time step

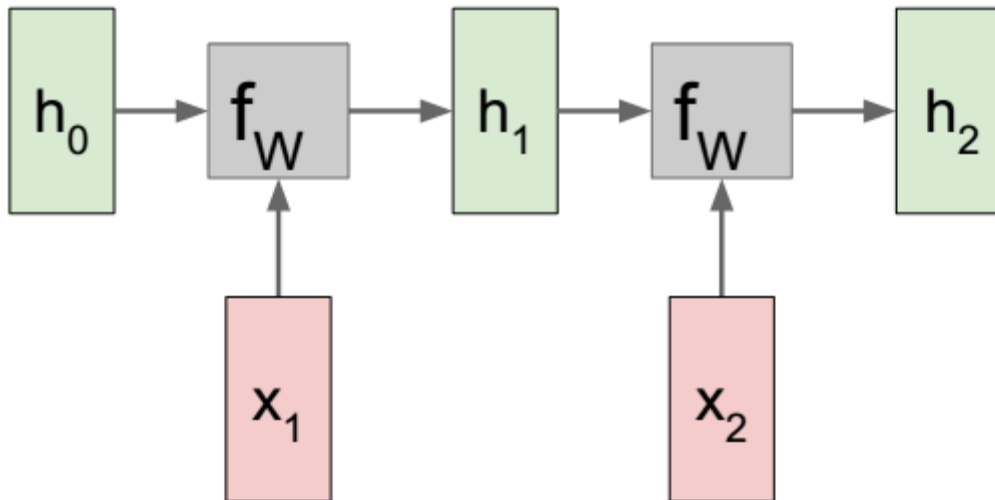
some function  
with parameters  $W$



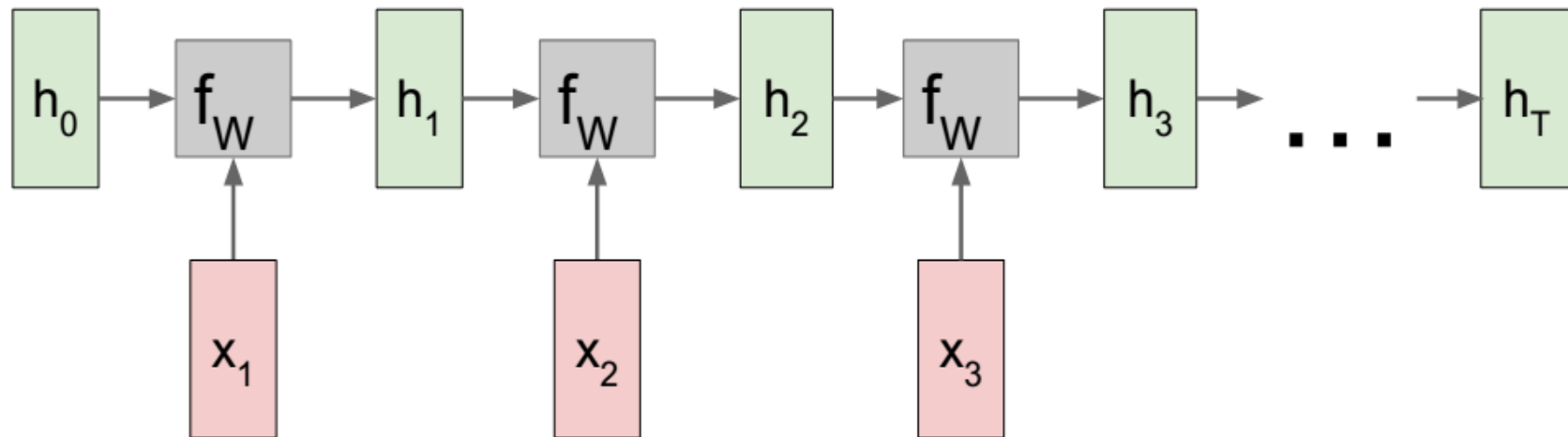
# Mạng neuron hồi quy



# Mạng neuron hồi quy

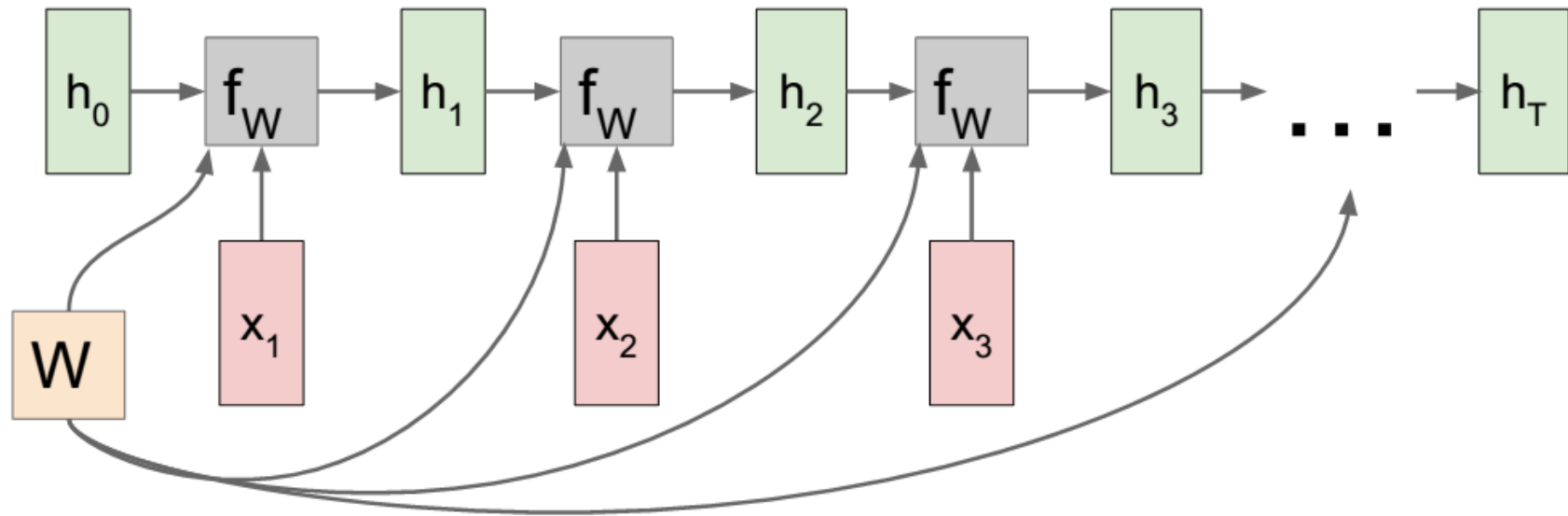


# Mạng neuron hồi quy



# Mạng neuron hồi quy

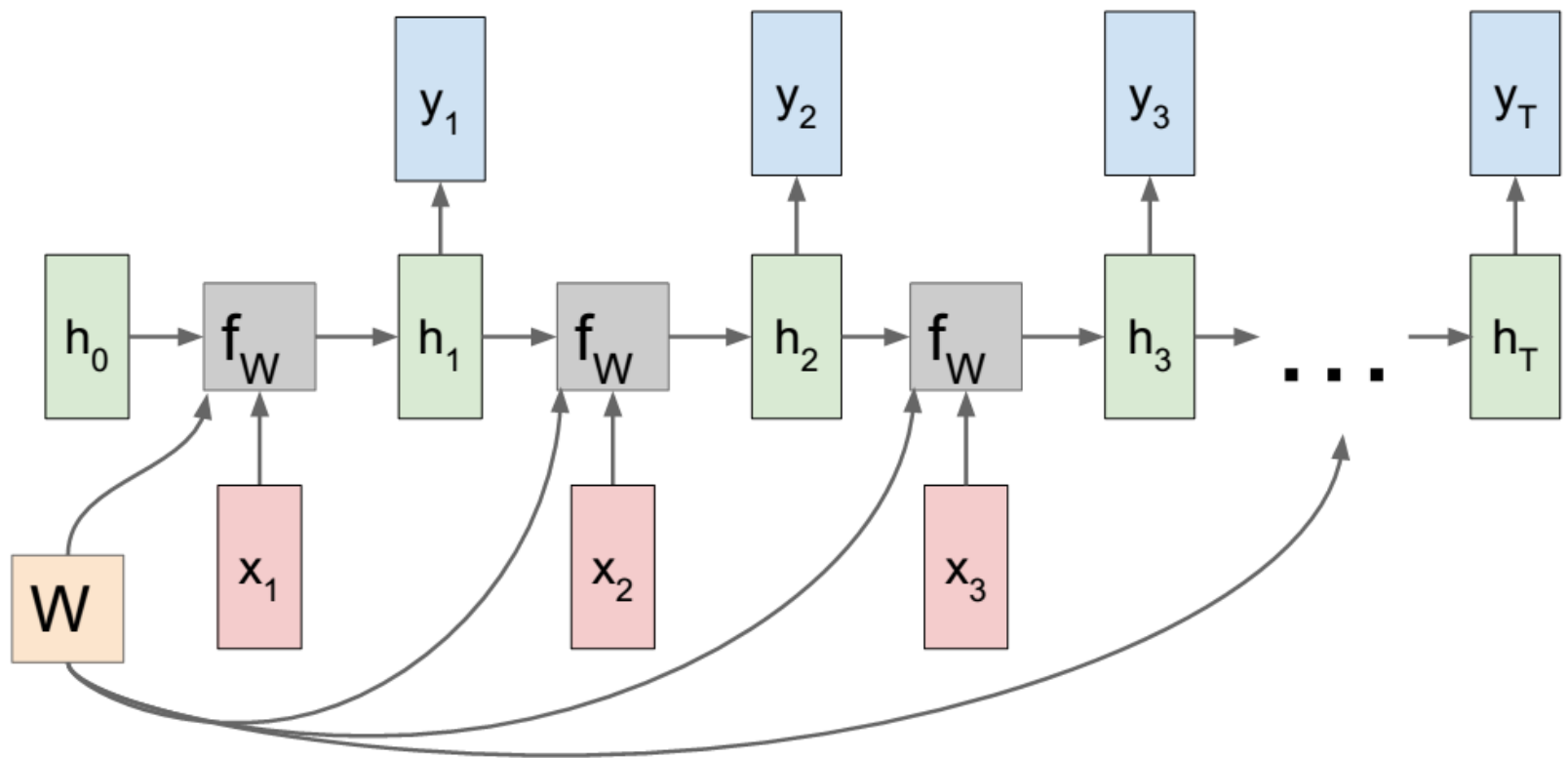
Re-use the same weight matrix at every time-step



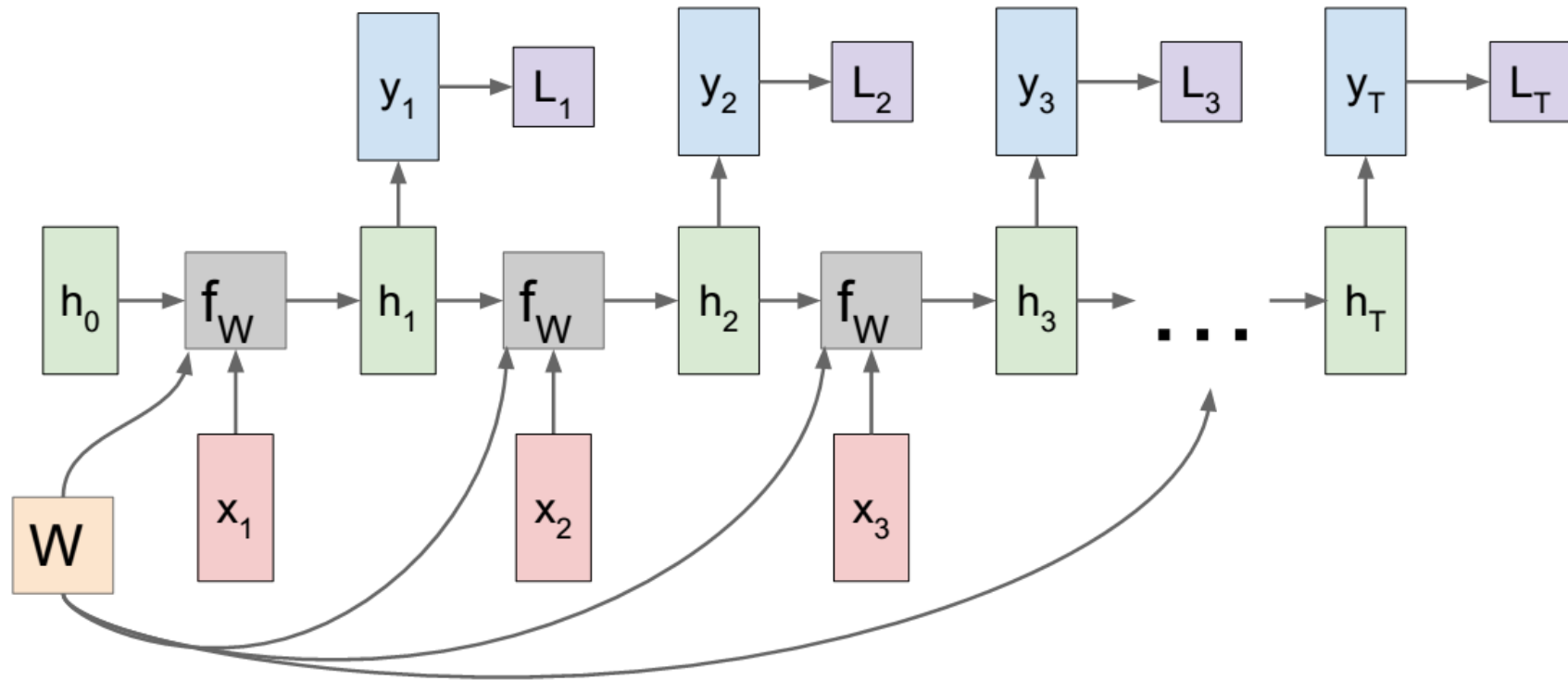


# Mạng neuron hồi quy

RNN: Computational Graph: Many to Many

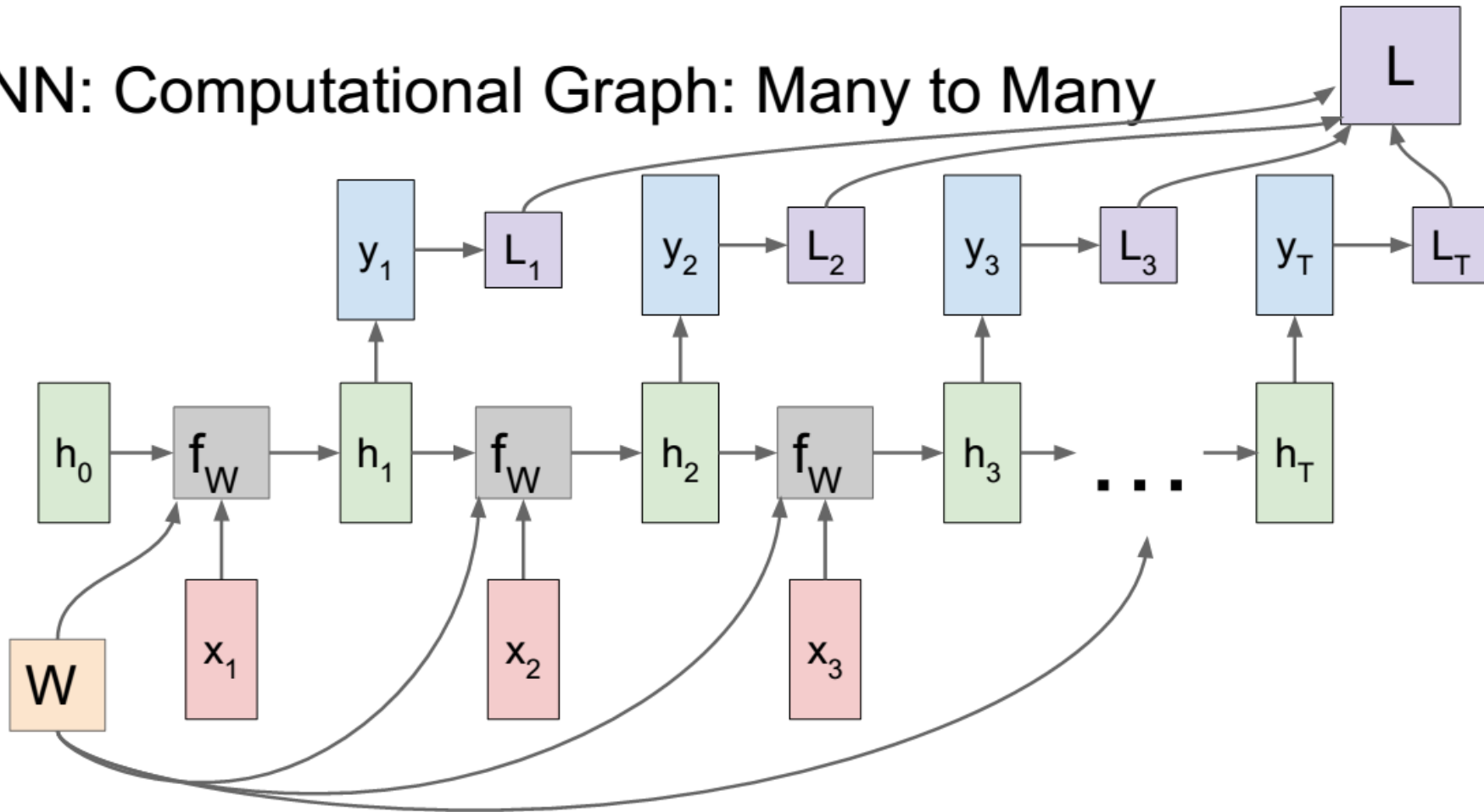


# Mạng neuron hồi quy



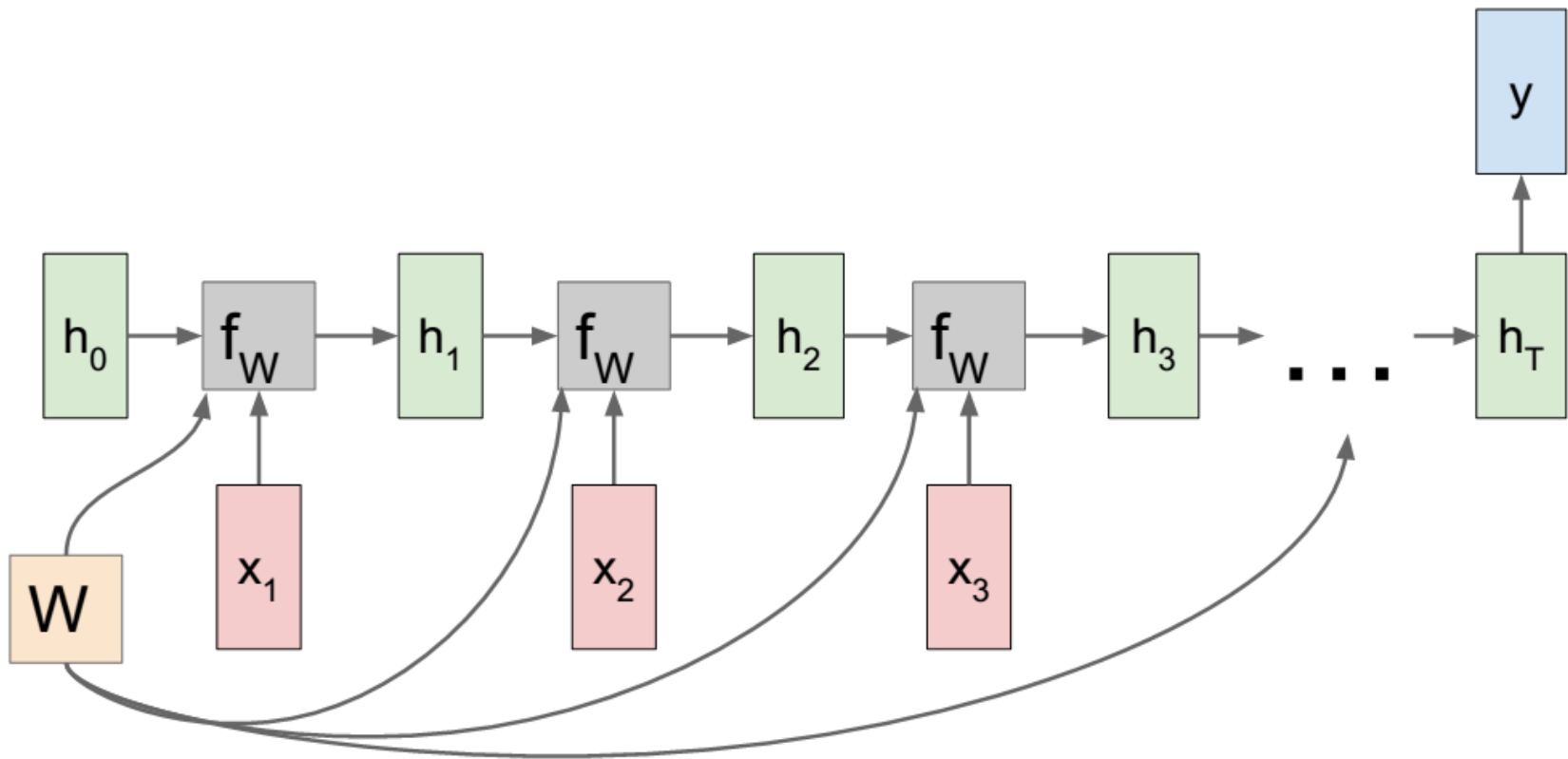
# Mạng neuron hồi quy

RNN: Computational Graph: Many to Many



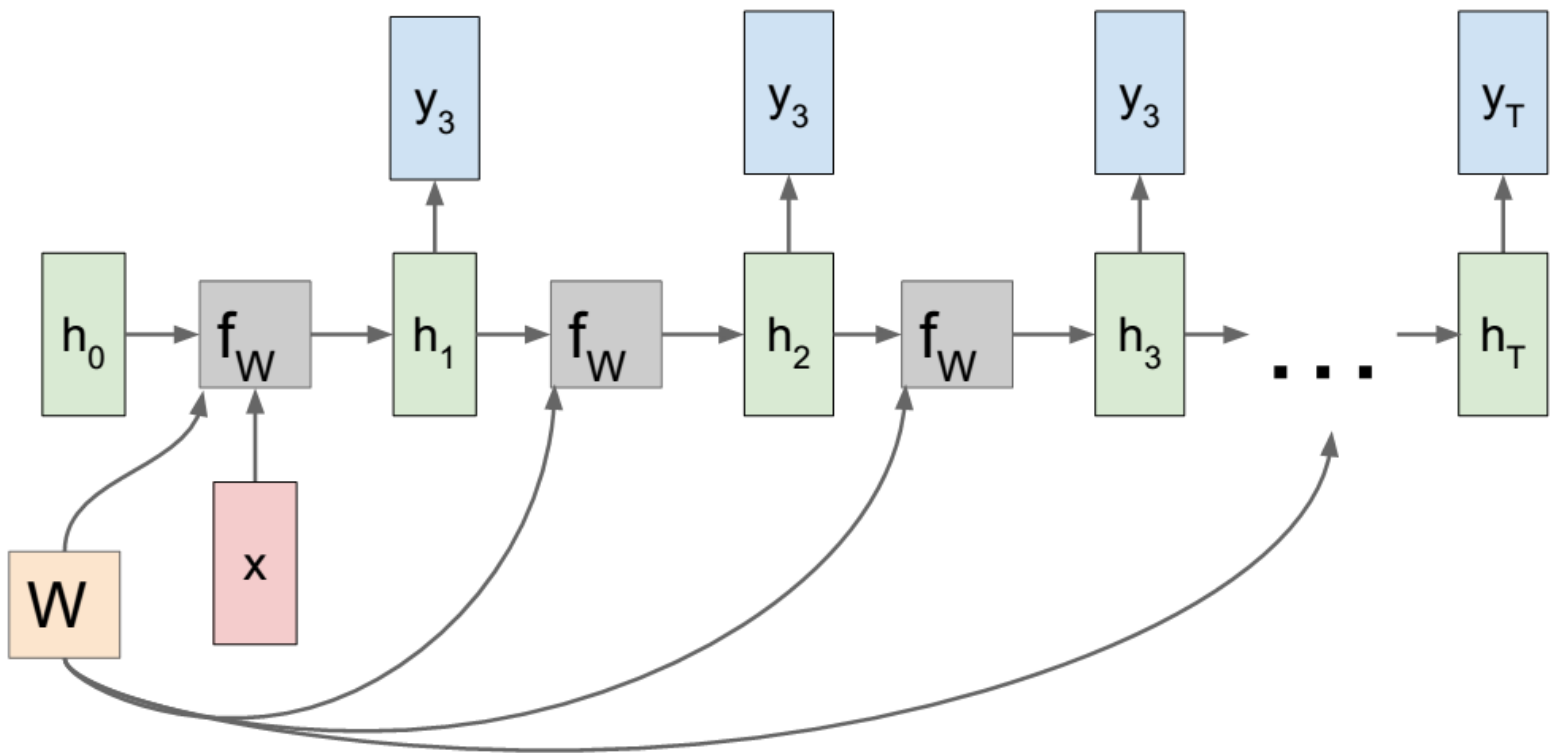
# Mạng neuron hồi quy

RNN: Computational Graph: Many to One



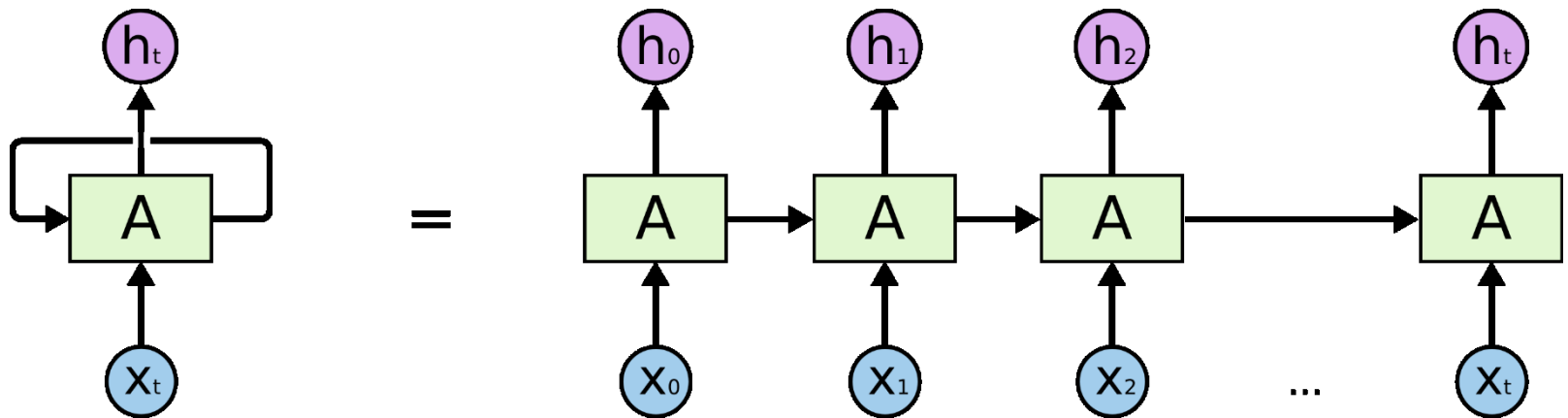
# Mạng neuron hồi quy

## RNN: Computational Graph: One to Many



# Mạng neuron hồi quy

- Mạng neuron hồi quy có thể coi như là copie liên tiếp một mạng



- Rất nhiều ứng dụng
  - ◆ Nhận dạng tiếng nói
  - ◆ Mô hình hóa ngôn ngữ
  - ◆ Dịch
  - ◆ Nhận dạng ảnh

# Long-term dependencies

- **Yêu cầu của mạng RNN:** các thông tin trước đó góp phần vào đáp ứng hiện tại của mạng
- **Ví dụ:** video thì frames trước đó sẽ cho hiểu rõ hơn về frames hiện tại

**Nếu RNN làm được việc đó thì thực sự hữu ích, nhưng RNN có thể làm được hay không ?**



# Mạng neuron hồi quy

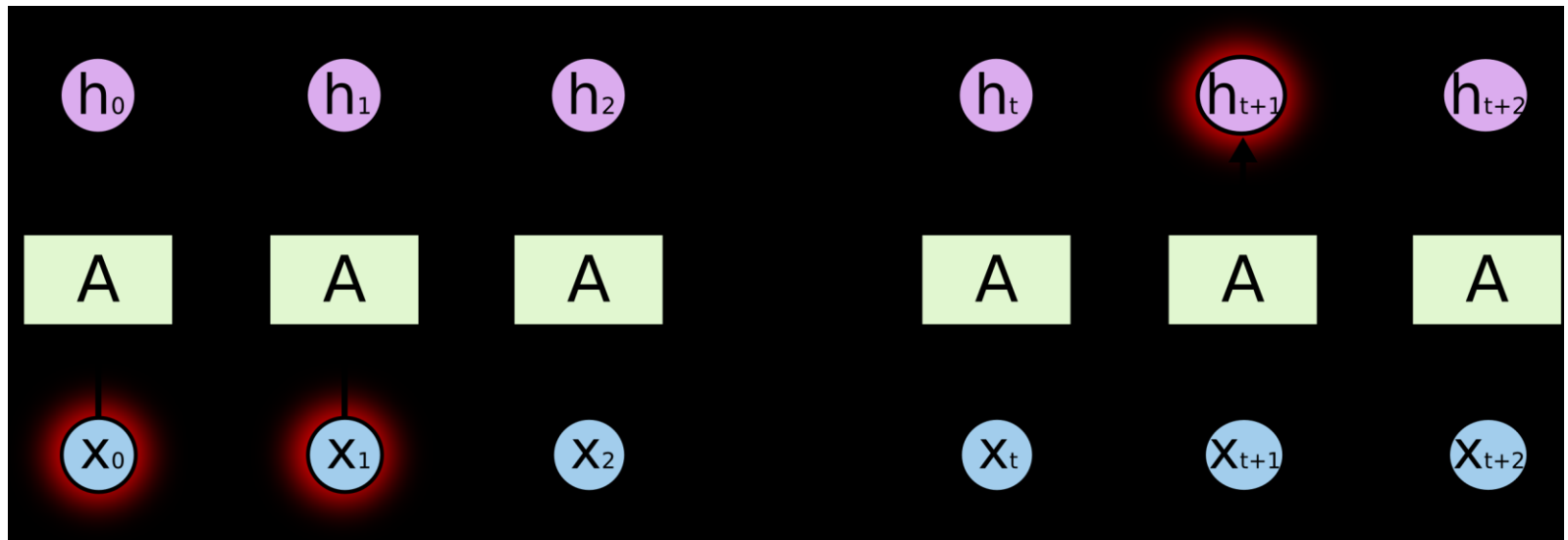
- Giả sử cần dự báo câu “**the clouds are in the sky**”, ta không cần thông tin ngữ cảnh
- Trong trường hợp này mạng RNN đòi hỏi thông tin trước đó là nhỏ
- Giả sử cần dự báo từ cuối của câu: “**I grew up in France... I speak fluent French**”
- Trong trường hợp này, các thông tin trước đó gợi ý rằng từ tiếp theo có thể liên quan đến tên của một ngôn ngữ
- Nếu ta lùi về phía trước ta có thể dự đoán đó là ngôn ngữ của **nước pháp**. Trong trường hợp này, khoảng cách là lớn





# Mạng neuron hồi quy

- Khi khoảng cách này tăng lên, rất khó có thể học để kết nối các thông tin



- Về mặt lý thuyết, RNN có khả năng lưu giữ sự phụ thuộc dài hạn, trên thực tế thì không
- Lý do: **việc tính toán gradient rất khó khi sự phụ thuộc là lớn**

# Mạng neuron hồi quy

---

- Một dạng đặc biệt của mạng neuron hồi quy:
  - ◆ LSTM (**L**ong **S**hort **T**erm **M**emory)



# LSTM

- Là một dạng đặc biệt của RNN
- LSTM: long short term memory [ [Hochreiter & Schmidhuber \(1997\)](#)]
- Đây là một dạng thiết kế RNN để tránh vấn đề phụ thuộc lâu dài



# Mô hình LTSM

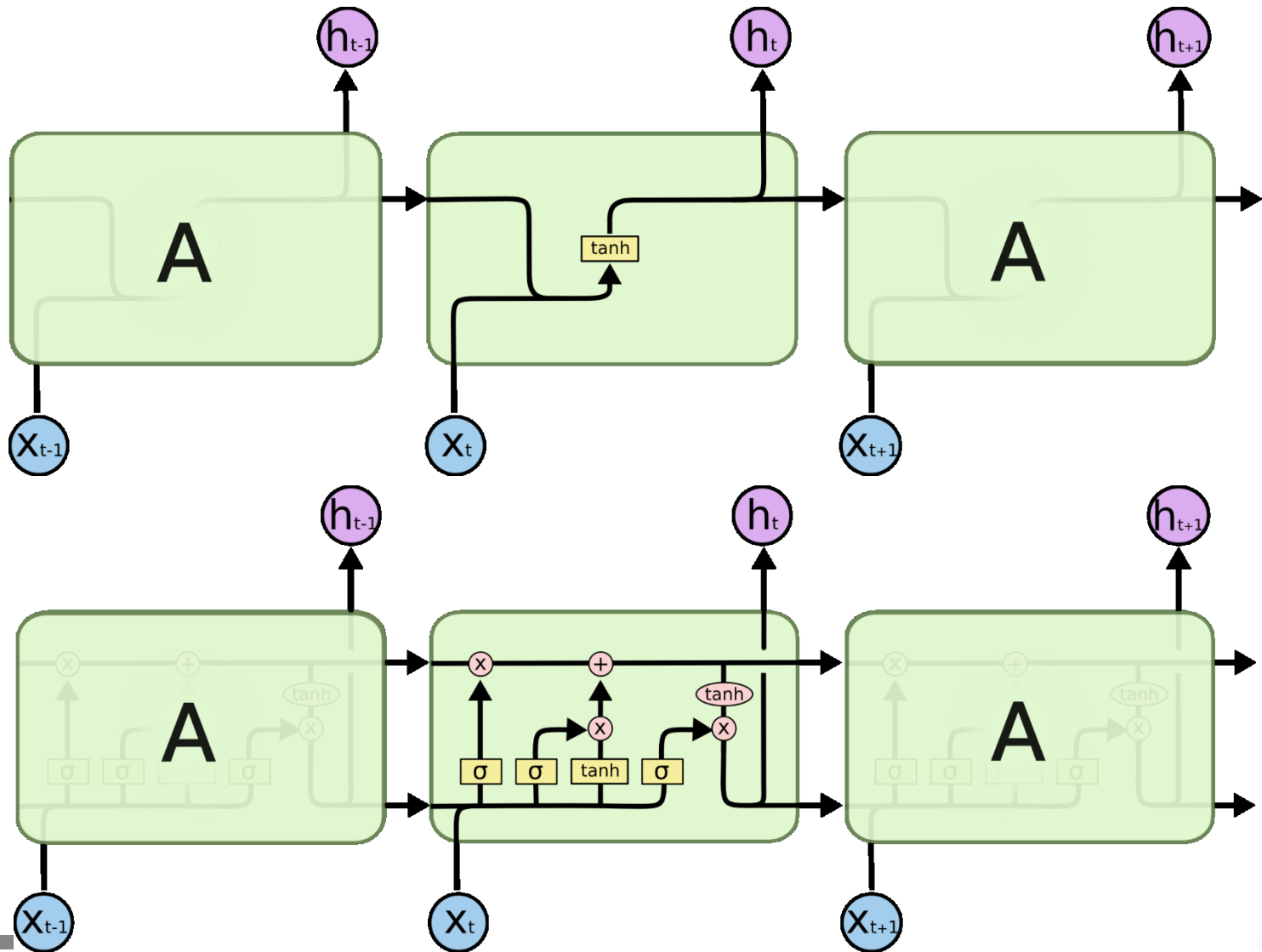
Neural Network Layer

Pointwise Operation

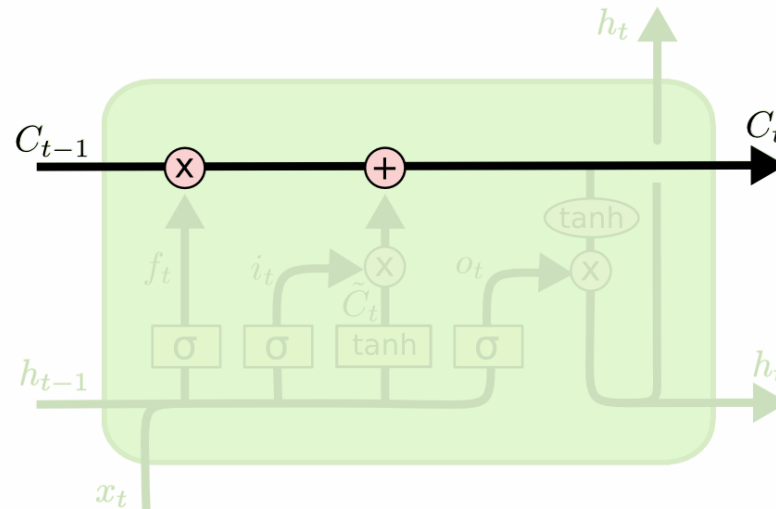
Vector Transfer

Concatenate

Copy



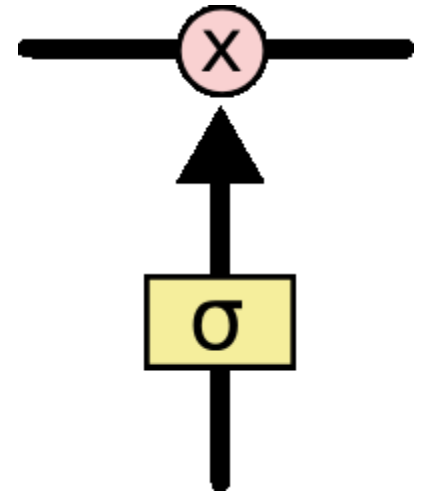
# Ý tưởng cơ bản của LSTM



- Thành phần chính của LSTM là cell state, đường nằm ngang  $C_{t-1}$  đến  $C_t$  nó như một dạng băng truyền.
- Nó đi thẳng từ đầu đến cuối mạng
- LSTM có khả năng bỏ bớt hoặc thêm vào các thông tin vào cell state thông qua cấu trúc cổng (gates)

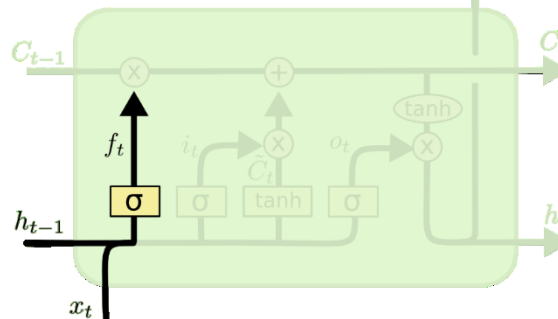
# Gates

- Cổng là một cách để cho thông tin đi qua
- Nó gồm 1 lớp mạng sigmoid và một toán tử nhân
- Sigmoid có đầu ra là 0 và 1, thể hiện bao nhiêu thông tin sẽ được đưa qua cổng
- Một LSTM có 3 cổng như vậy để bảo vệ và điều khiển cell state.



# Gates

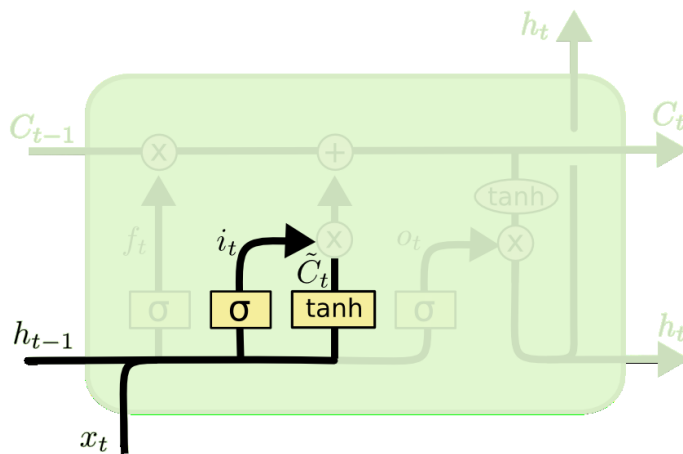
- Bước đầu tiên là quyết định thông tin nào sẽ được đưa đến cell state thông qua cổng
- Quyết định này được thực hiện bởi lớp sigmoid với hai đầu vào là  $h_{t-1}$  và  $x_t$  và cho đầu ra là 0 hoặc 1 cho mỗi đầu vào  $C_{t-1}$ 
  - ◆ 0: không tính đến thông tin trước đó  $C_{t-1}$
  - ◆ 1: có tính đến thông tin trước đó  $C_{t-1}$
- Lấy ví dụ trường hợp mô hình ngôn ngữ để dự báo từ tiếp theo dựa trên các từ trước đó
- Trong trường hợp này, cell state phải chứa thông tin về giới tính (gender) của chủ thể (subject) hiện tại để các đại từ sẽ được sử dụng một cách phù hợp
- Khi có một subject (chủ thể mới), gender của chủ thể cũ không cần phải ghi nhớ nữa.



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

# Gates

- Bước tiếp theo là xác định loại thông tin mới nào sẽ được lưu trữ trong cell state. Có hai thành phần
  - ◆ A **sigmoid** layer (gọi là input gate layer) quyết định giá trị nào cần update
  - ◆ A **tanh** layer tạo ra vector với các giá trị mới có thể đưa vào cell state
- Trong trường hợp của mô hình ngôn ngữ, gender của chủ thể mới sẽ được đưa vào để cập nhật cell state



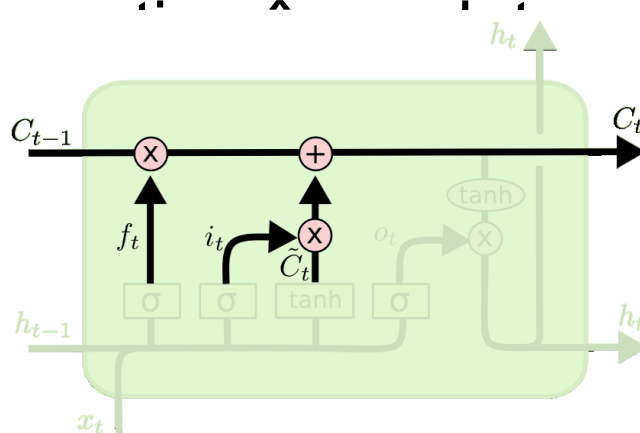
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



# Cập nhật

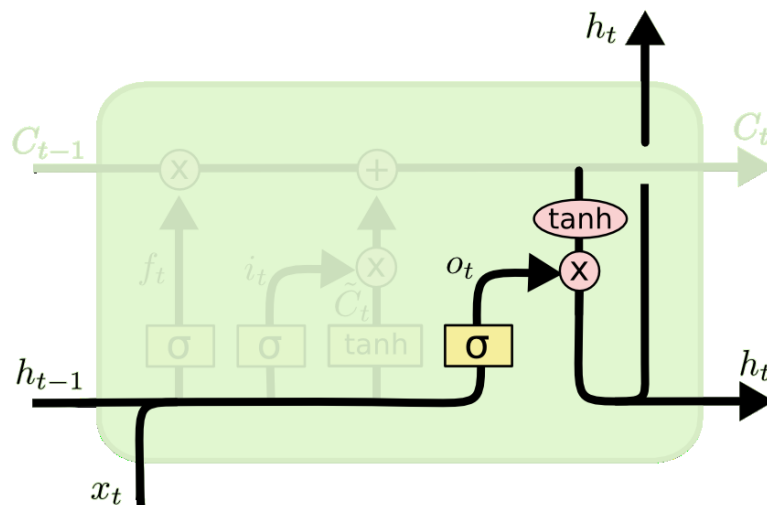
- Bước tiếp theo là cập nhật cellstate cũ  $C_{t-1}$  vào cell state mới  $C_t$
- Các bước trước đó đã quyết định phải làm gì nên bước này chỉ cần thực hiện nó
  - ◆ Nhân trạng thái cũ với  $f_t$  (để cần nhớ hoặc quên trạng thái cũ trước đó hay không)
  - ◆ Bổ sung  $i_t * \tilde{C}_t$ : scale bởi  $i_t$  để thể hiện bao nhiêu lượng



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

# Cập nhật

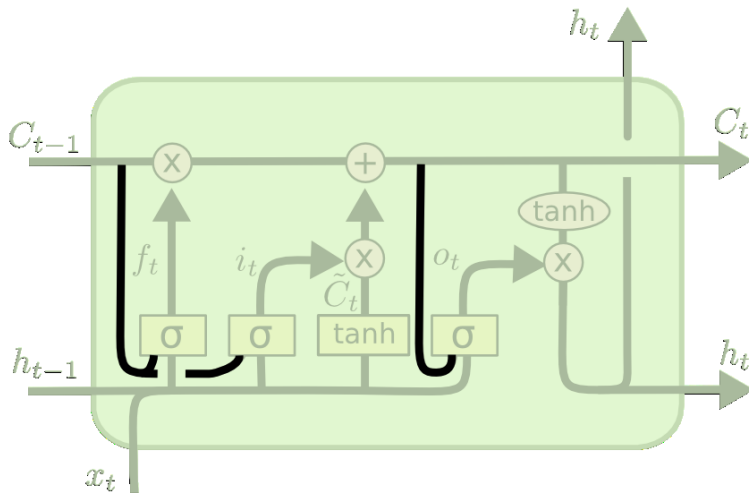
- Tiếp theo là quyết định nên đưa đầu ra là gì ?
  - ◆ Đầu ra sẽ dựa trên trạng thái của cell state nhưng sẽ là giá trị được lọc bỏ một số thông tin (filtered version)
  - ◆ Ta chạy một sigmoid layer để quyết định phần tử nào sẽ tác động đến đầu ra
  - ◆ Sau đó đưa cellstate đi qua một tanh function



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

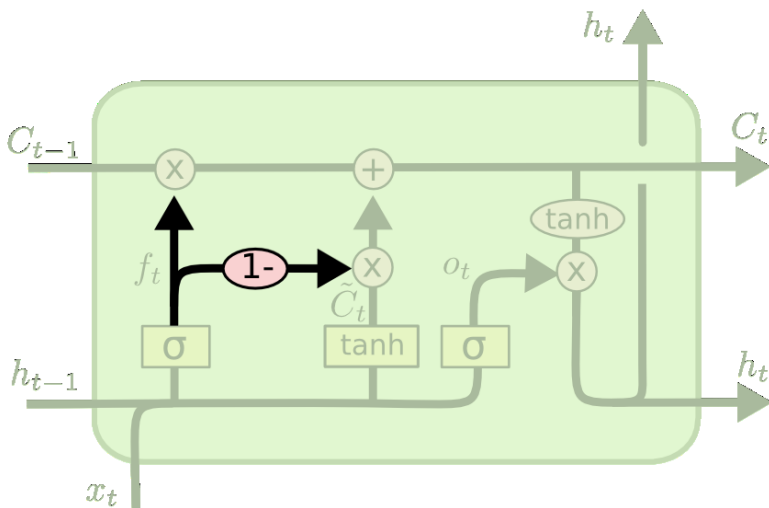
# Các biến thể của LSTM



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

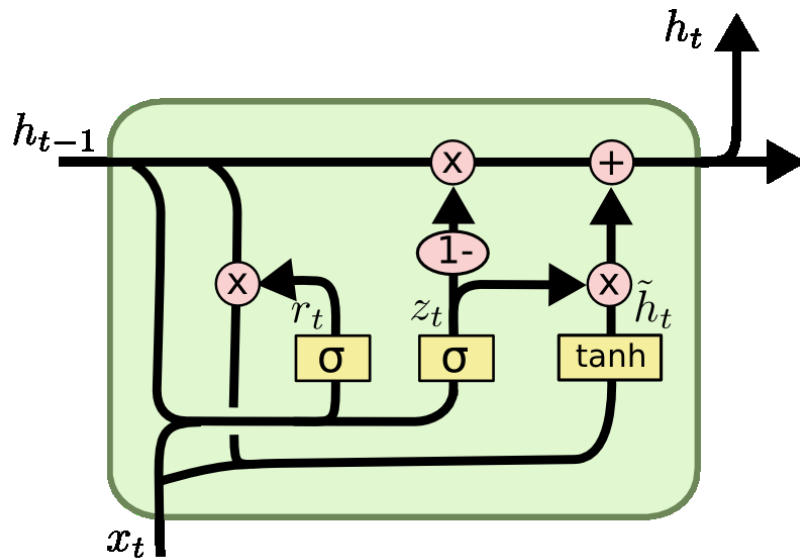
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$



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

# Các biến thể của LSTM



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Mô hình LSTM

- Có thể coi hidden state như là bộ nhớ của mạng.  $S_t$  lưu thông tin của xảy ra ở các bước trước đó. Tất nhiên  $S_t$  không thể lưu thông tin của rất nhiều trước đó
- Không giống mạng neuron truyền thống, các tham số ở các layer khác nhau thì khác nhau, ở RNN, các tham số là giống nhau vì thế mà số lượng parameters của mạng RNN giảm đi

