



# Natural Language Processing with Deep Learning

## Bài 8: RECURRENT NEURAL NETWORK



[https://csc.edu.vn/data-science-machine-learning/natural-language-processing-with-deep-learning\\_293](https://csc.edu.vn/data-science-machine-learning/natural-language-processing-with-deep-learning_293)



# RECURRENT NEURAL NETWORK



## I. Tổng quan Recurrent Neural Network

## II. Hạn chế của RNN

## III. Long Short Term Memory - LSTM

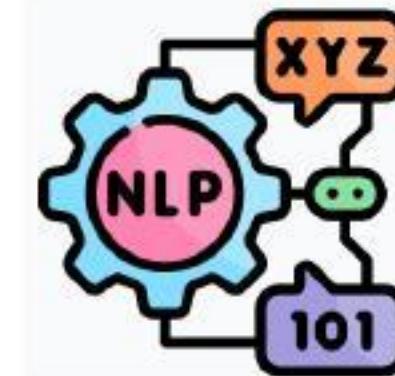




# Tổng quan Recurrent Neural Network

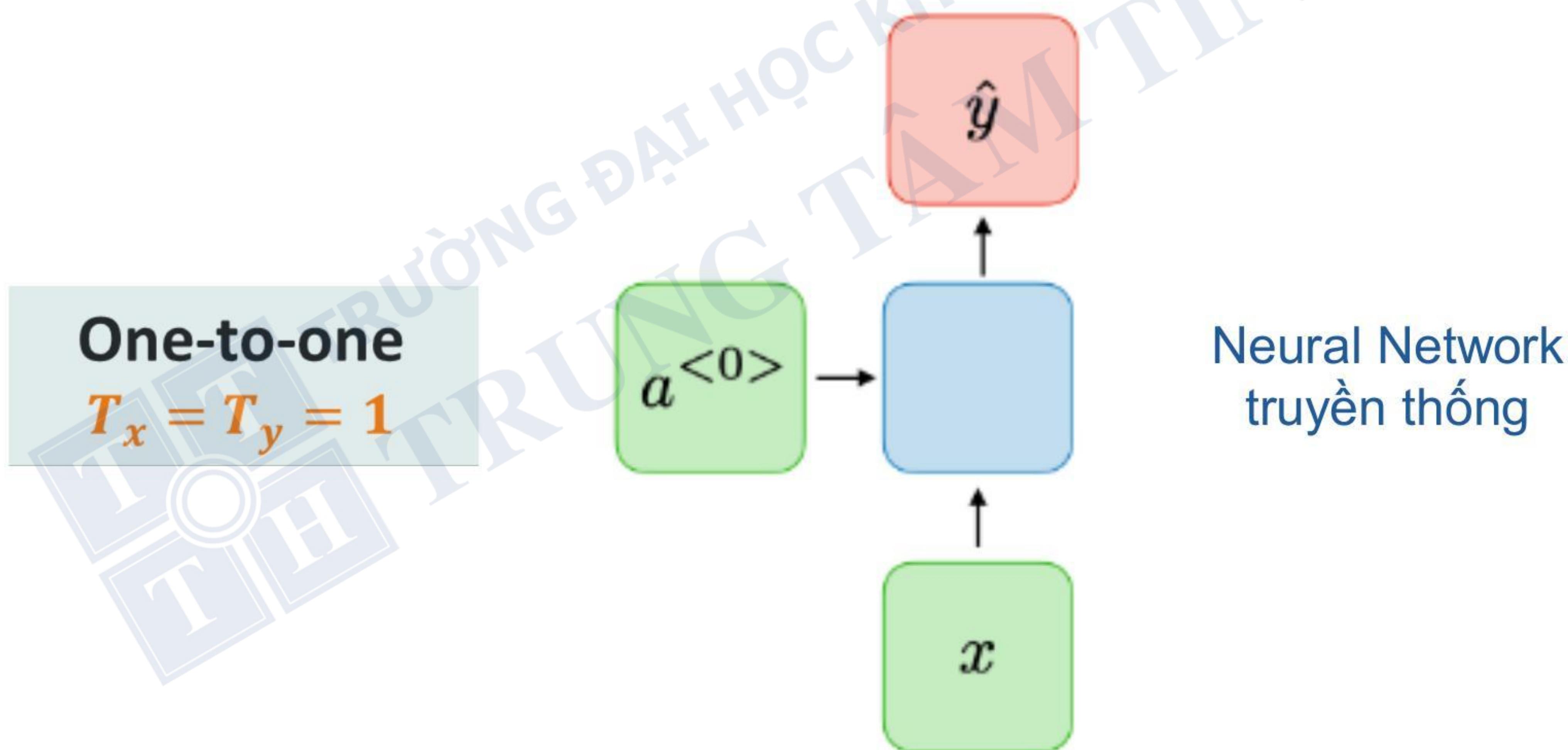
- RNN là mạng neural network có xét đến trật tự của các vector trong ma trận input => **Sequential input.**





# Các loại mạng RNN

- RNN được sử dụng chủ yếu cho các tác vụ xử lý sequential data như **Natural Language Processing** và **Speech Recognition**.

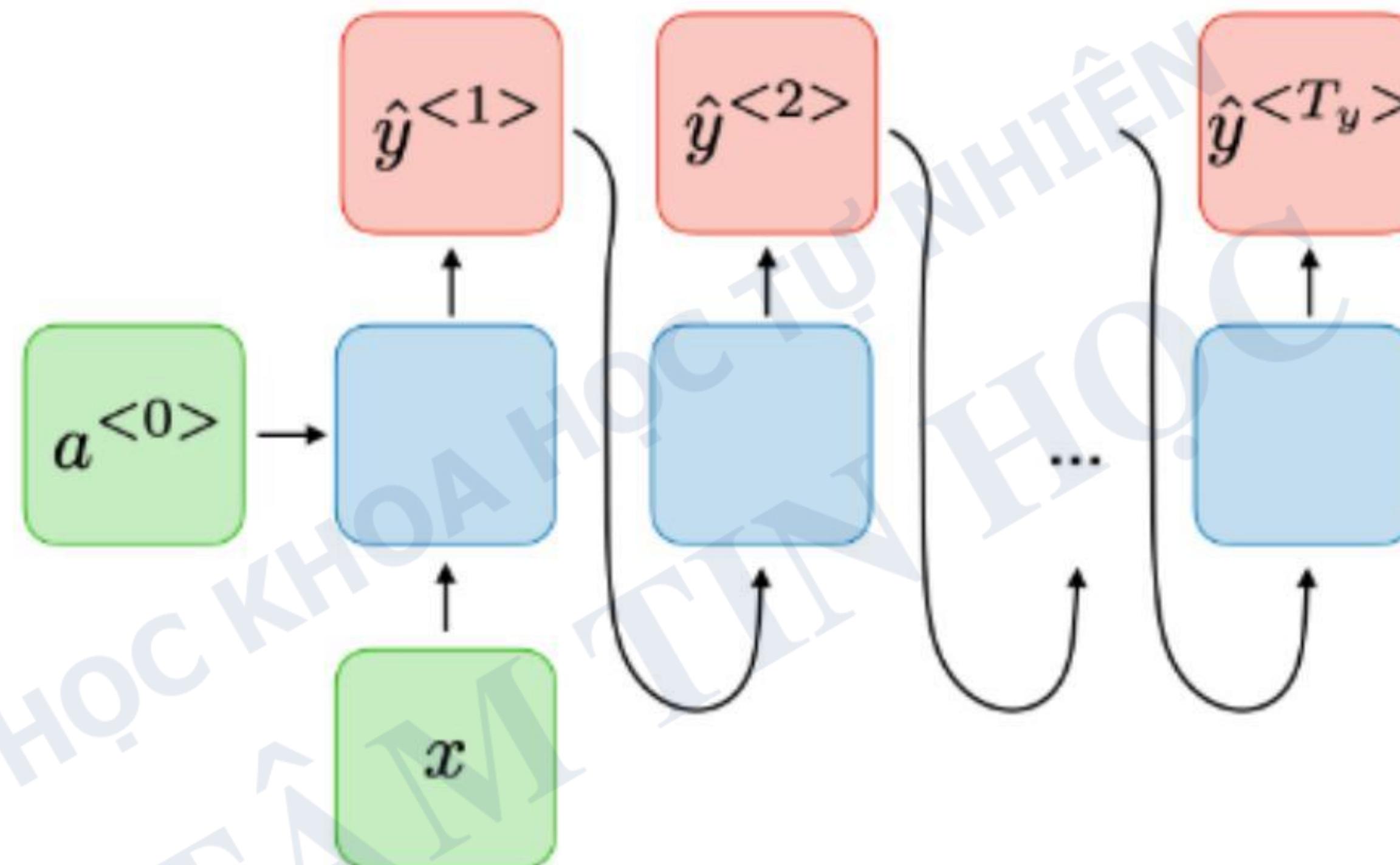


# Các loại mạng RNN

**One-to-many**

$$T_x = 1, T_y > 1$$

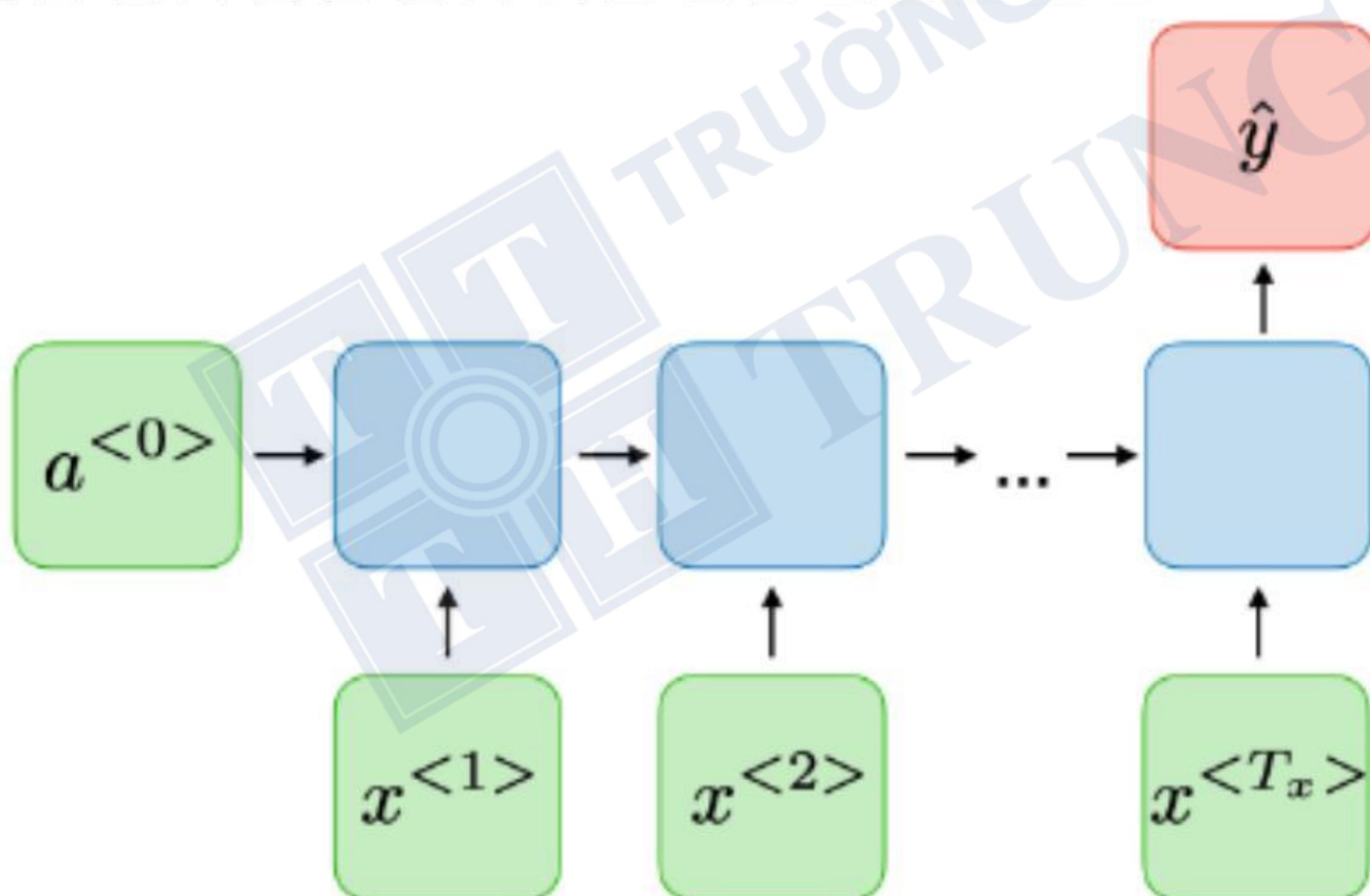
*Music generation*



**Many-to-one**

$$T_x > 1, T_y = 1$$

*Sentiment classification*

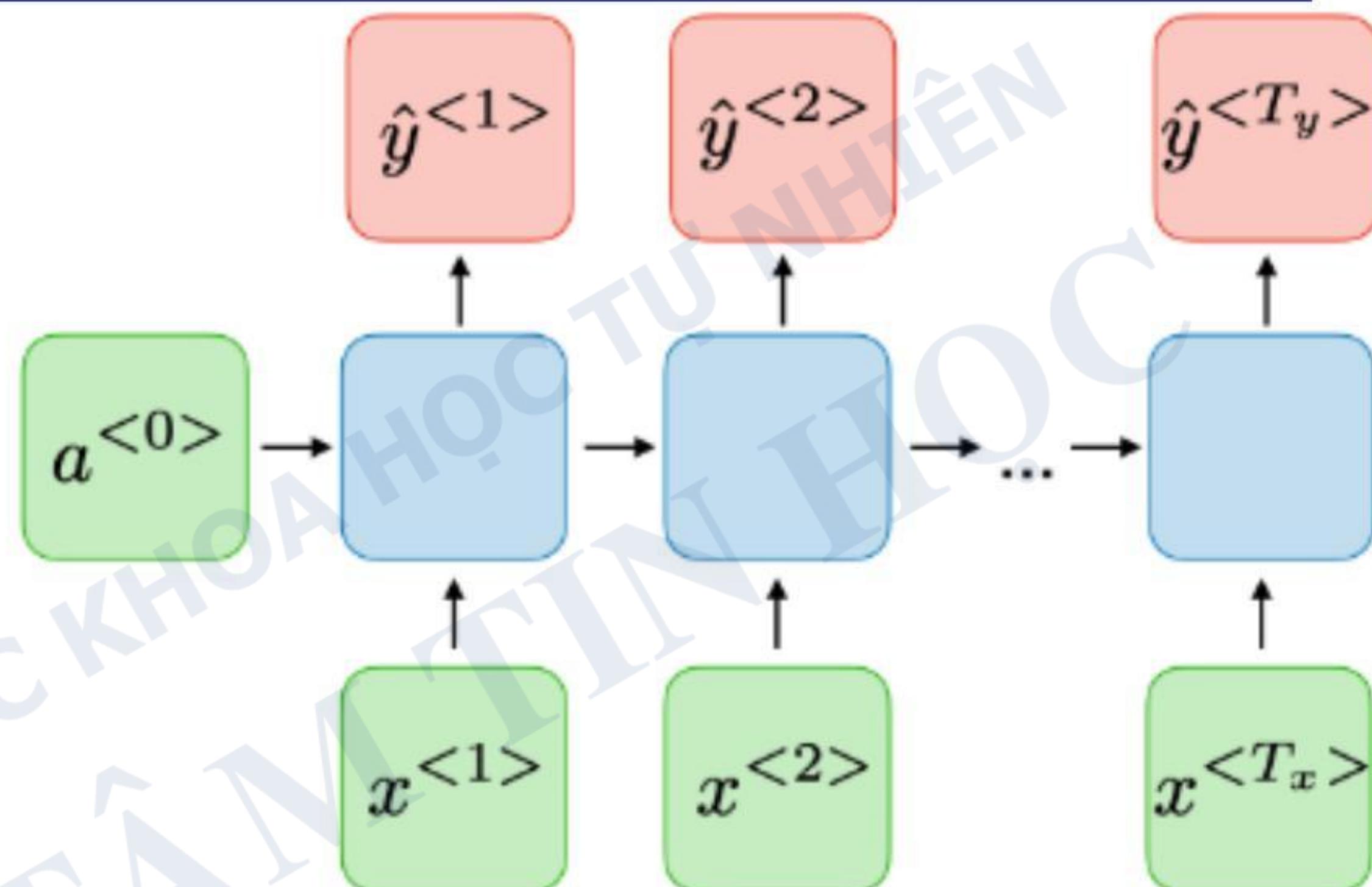


# Các loại mạng RNN

**Many-to-many**

$$T_x = T_y$$

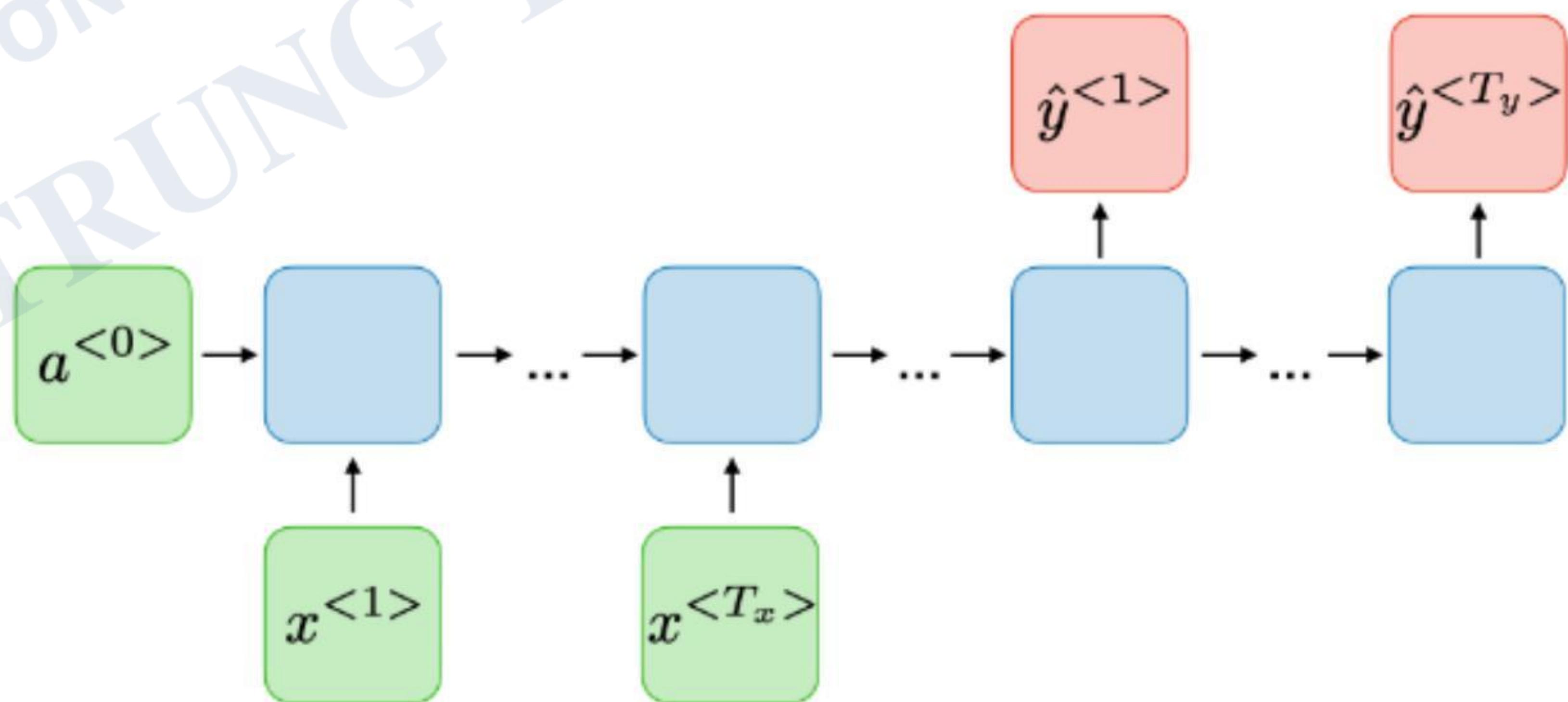
*Name entity recognition*



**Many-to-many**

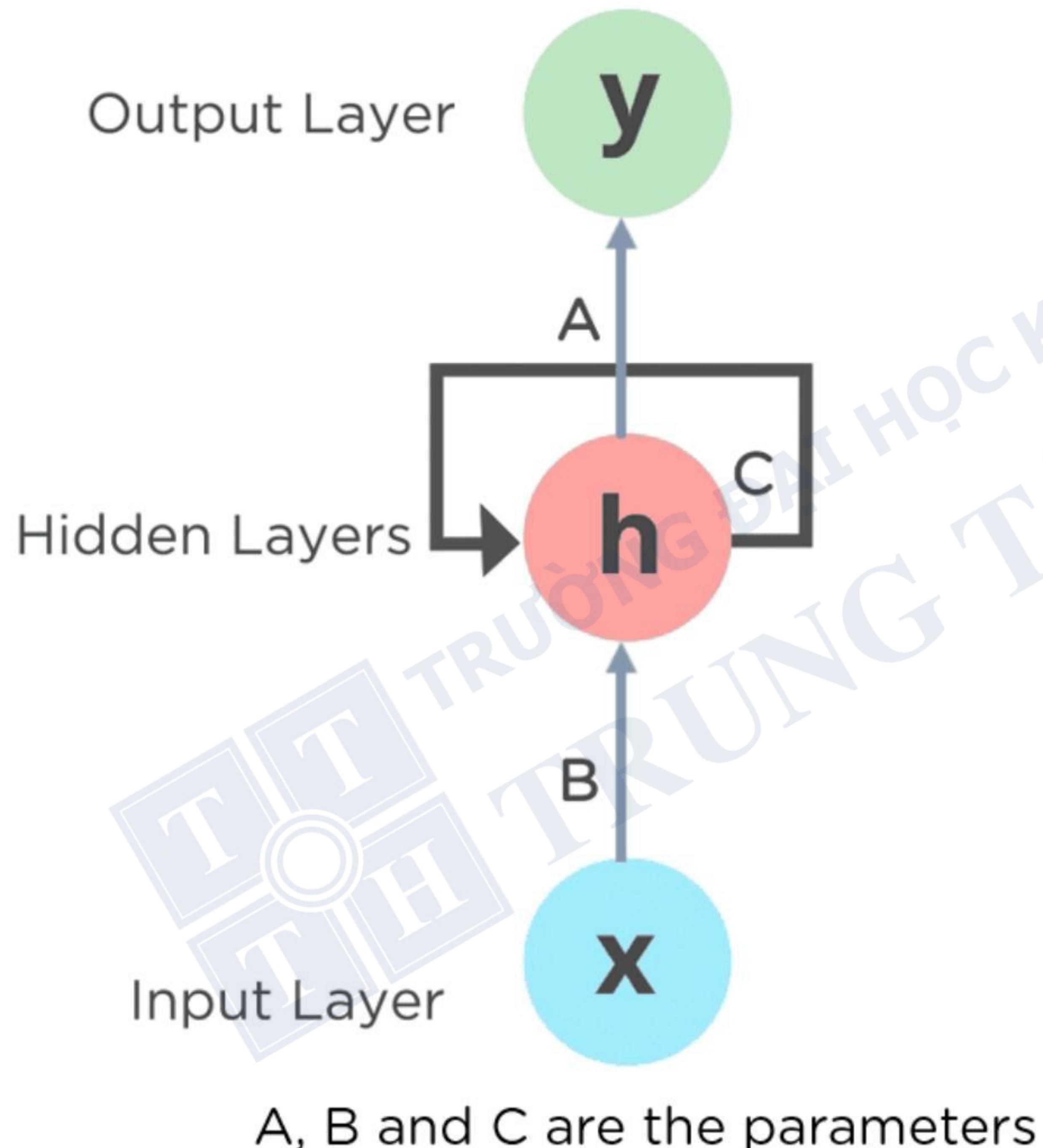
$$T_x \neq T_y$$

*Machine translation*





# Cơ chế hoạt động của RNN



RNN có cấu trúc là một mạng neural network.

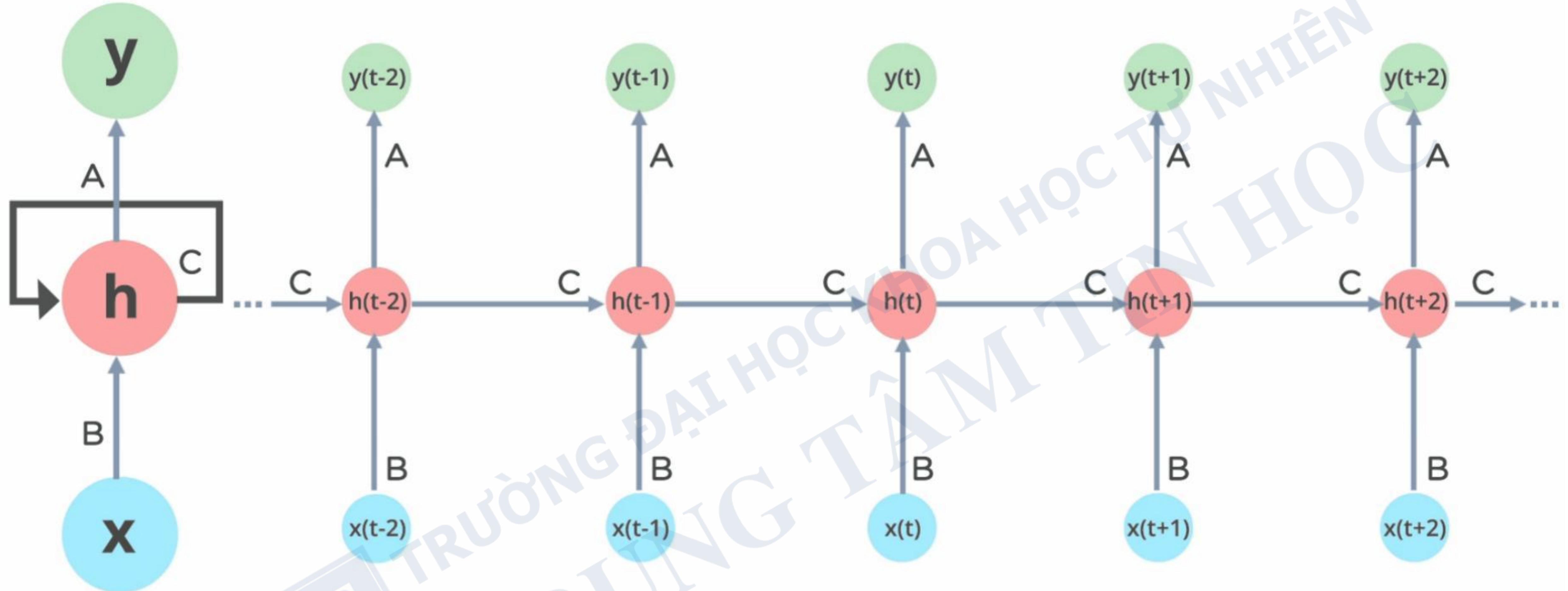
RNN lưu giữ state ( $h_t$ ) của layer trước đó để giữ các trạng thái phụ thuộc của input

- Tính trình tự của chuỗi
- **Recurrent cell**

$$\widehat{y}_t = f(\underline{x}_t, \underline{h}_{t-1})$$

**output**      **input**      **past memory**

# Cơ chế hoạt động của RNN



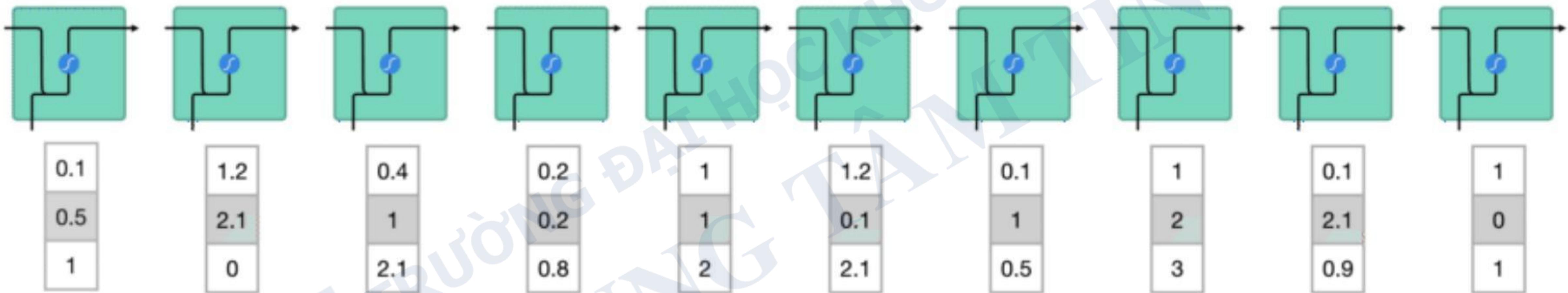
Áp dụng recurrence relation ở mỗi time step khi xử lý chuỗi:

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

cell state    weights W    input    old state

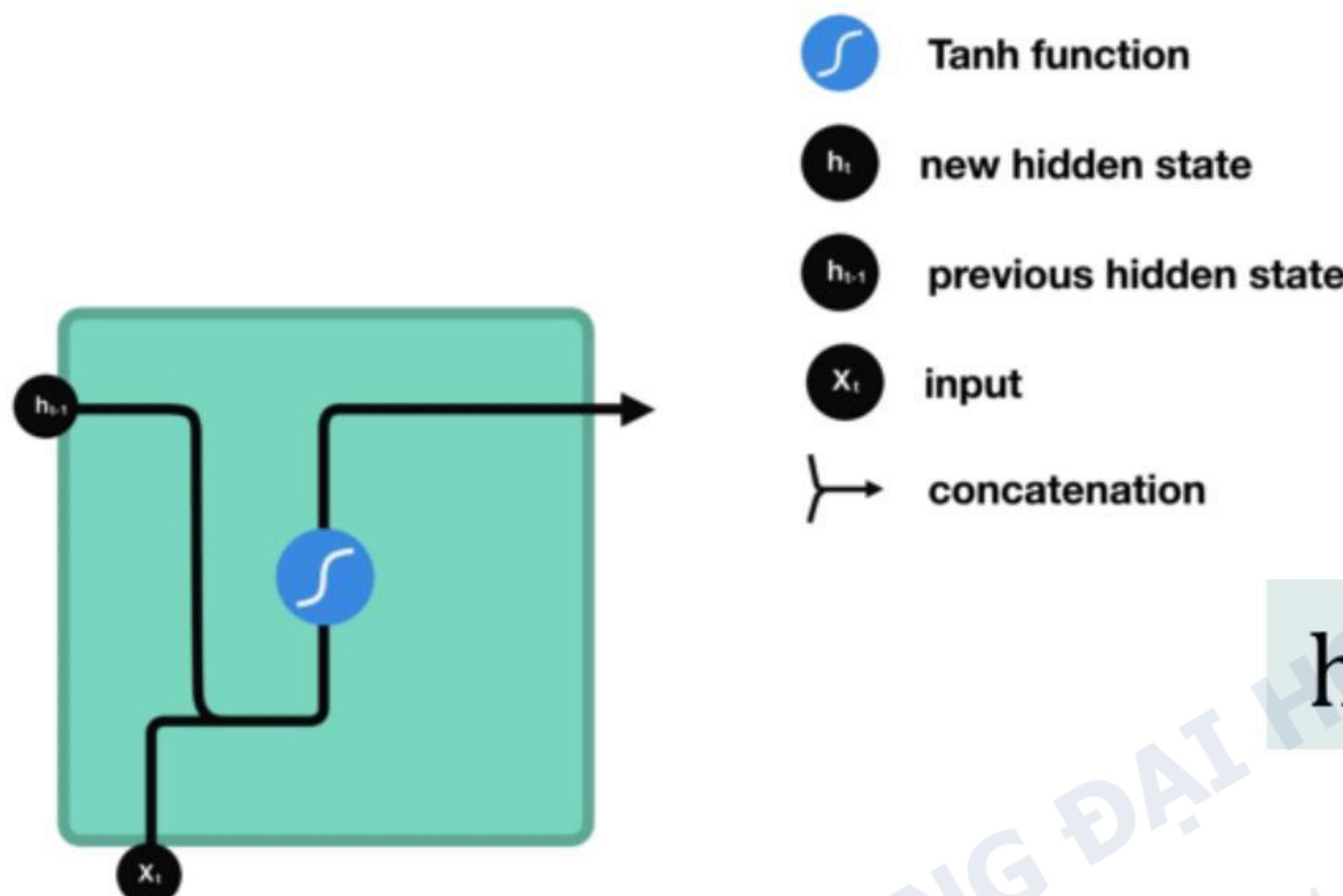
# Forward Propagation RNN

Ở mỗi time step, các **hàm và bộ tham số giống nhau** được sử dụng để tính toán.



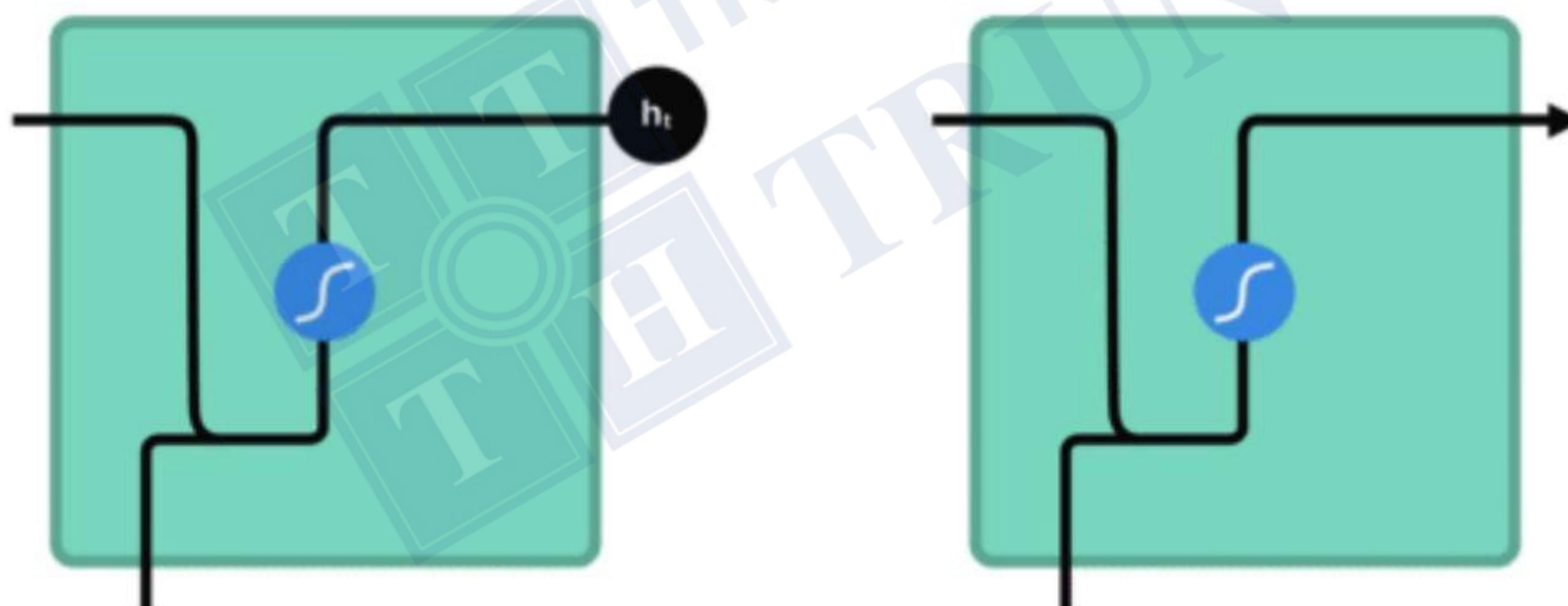
*RNNs cập nhật **state**  $h_{t-1}$  với state layer hiện tại  $h_t$  ở mỗi time step khi chuỗi được xử lý xong.*

# Forward Propagation RNN



Cập nhật Hidden State

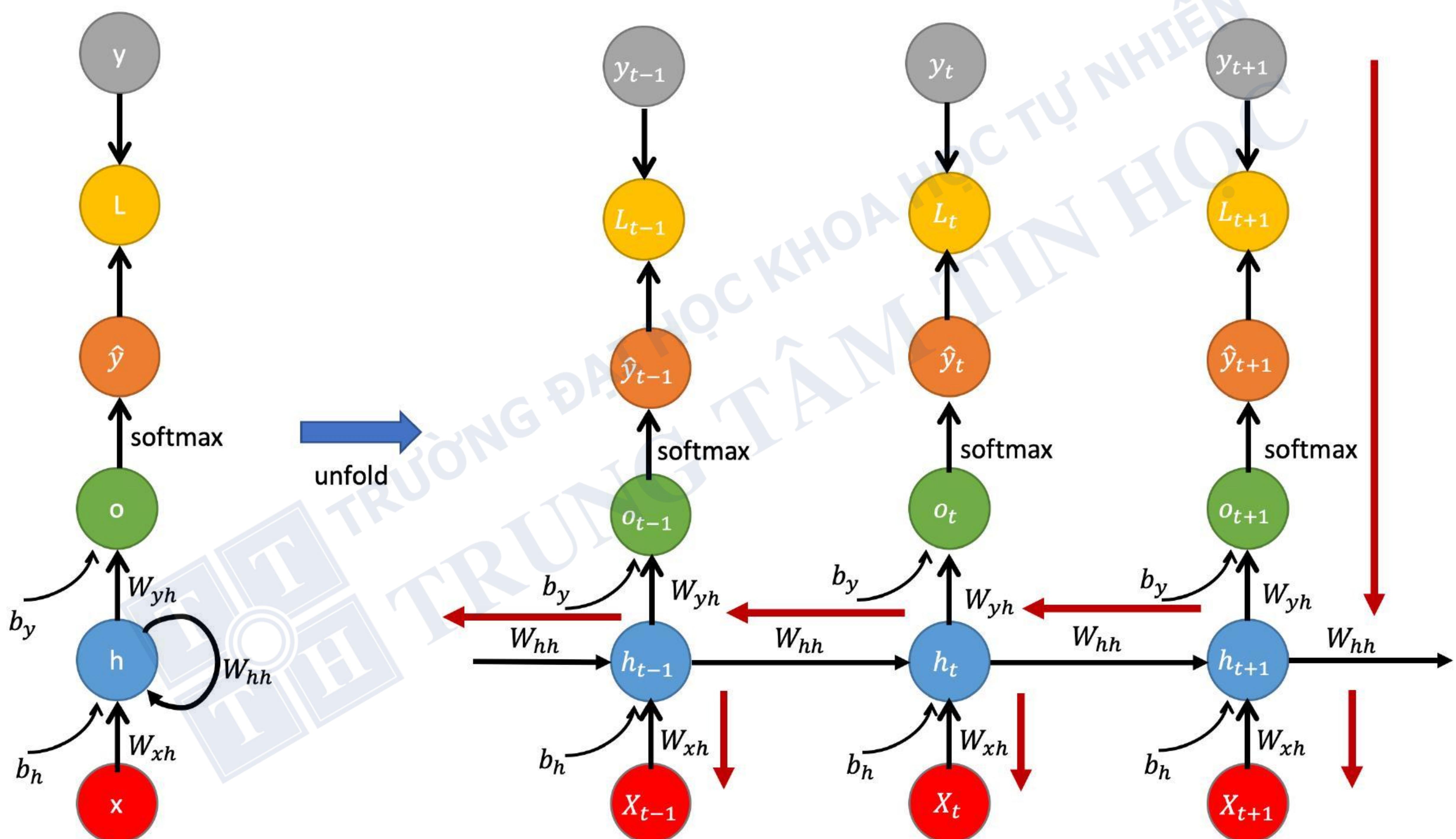
$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$



Output Vector

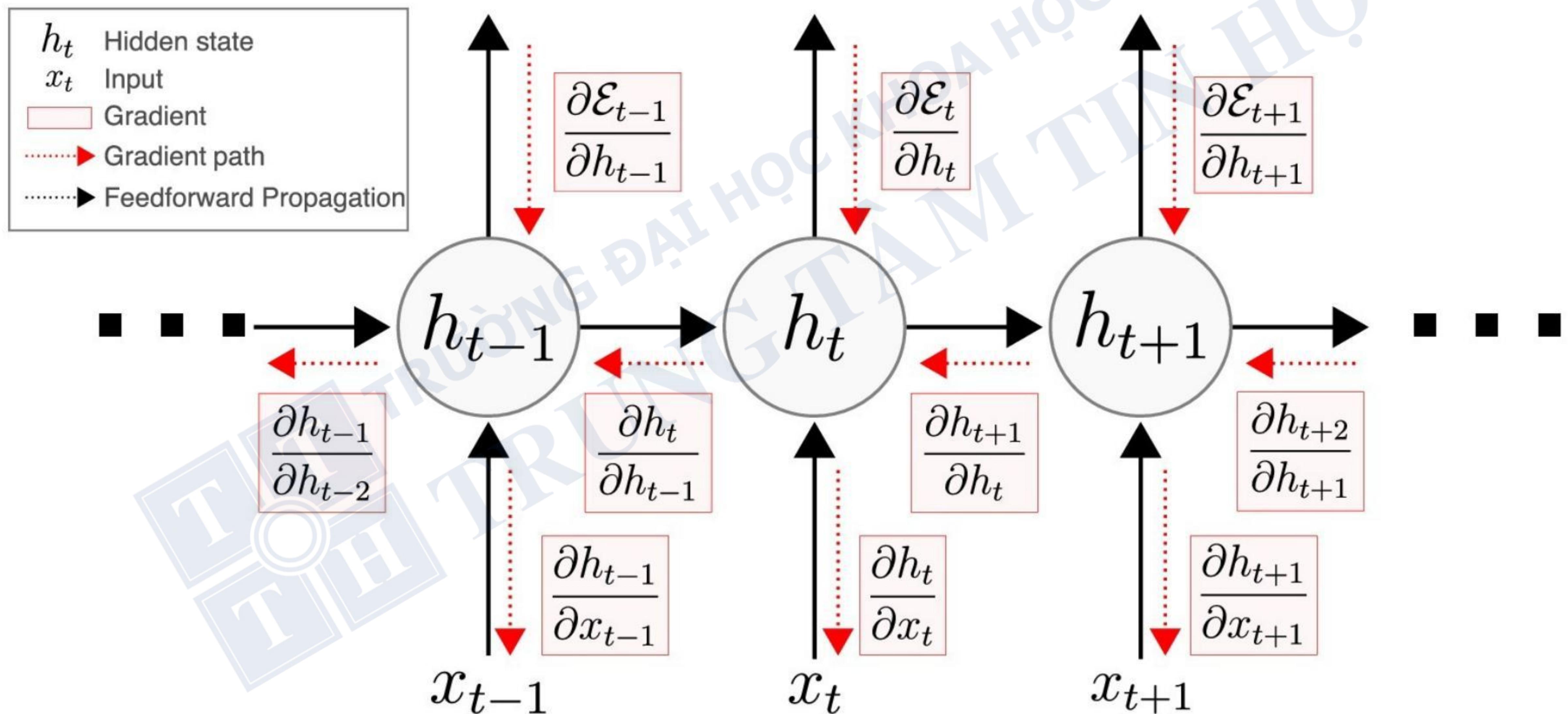
$$\hat{y}_t = W_{hy}^T h_t$$

# Backward Propagation RNN

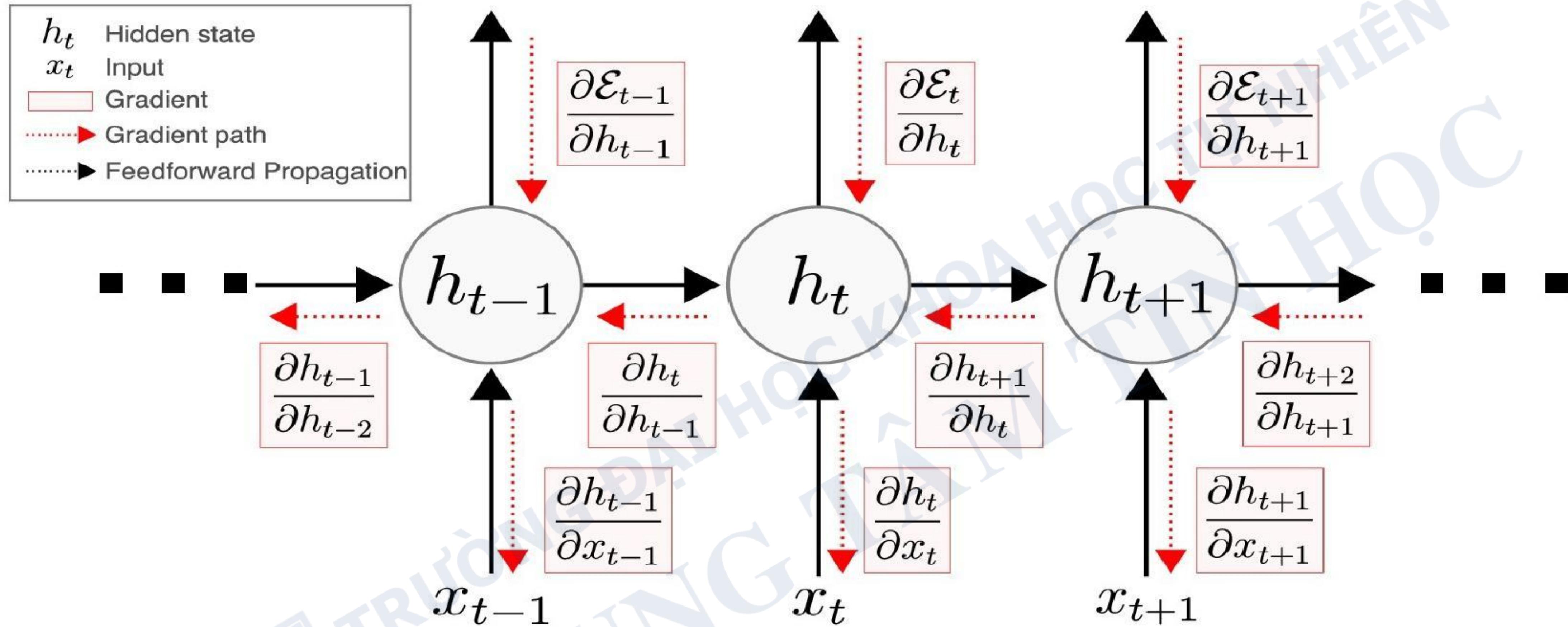


# Backward Propagation RNN

Thực hiện backpropagation để tìm ra bộ trọng số W  
tối ưu nhất → Loss nhỏ nhất.



# Backward Propagation RNN



Quá nhiều giá trị  $> 1$ :  
**Exploding gradients**

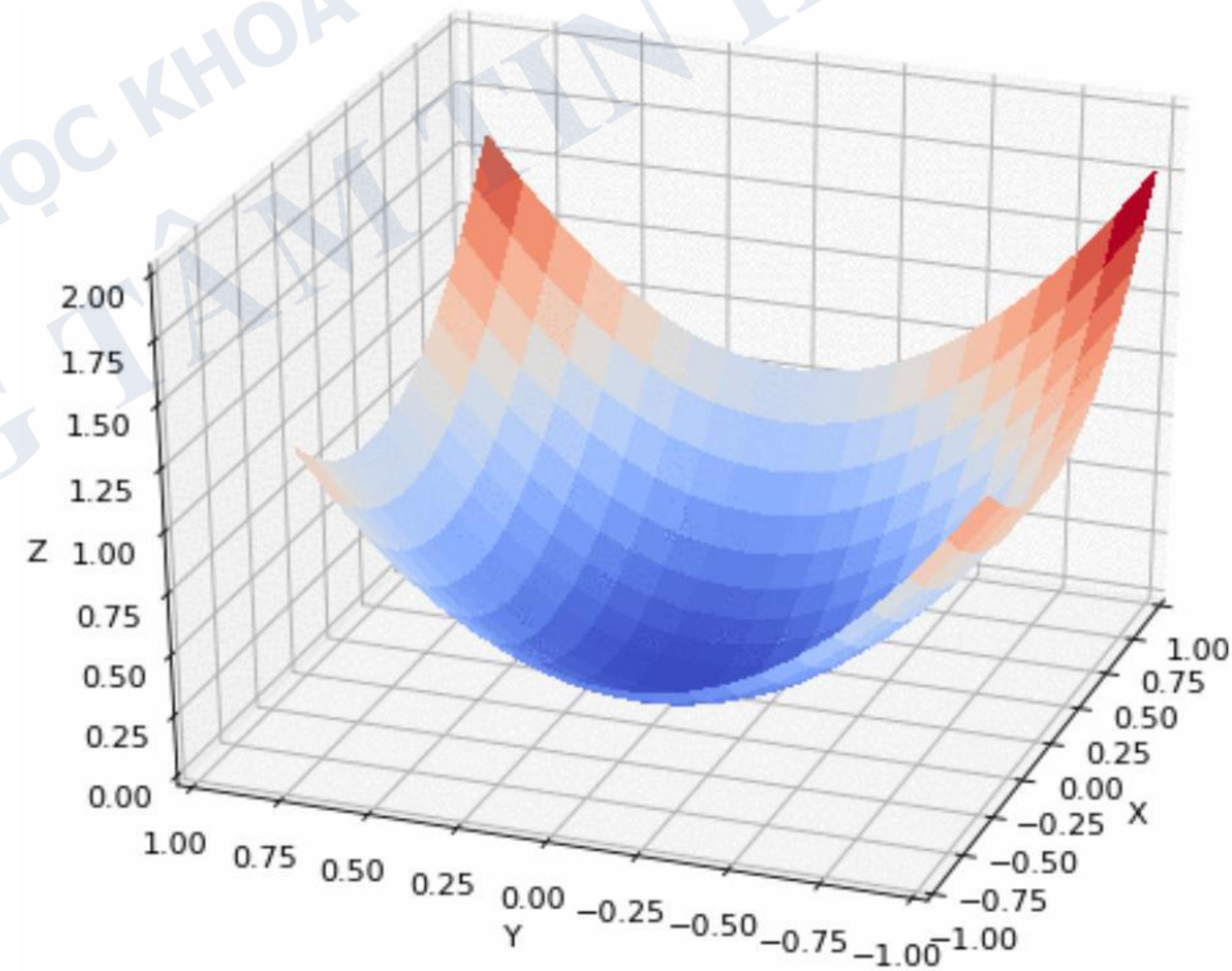
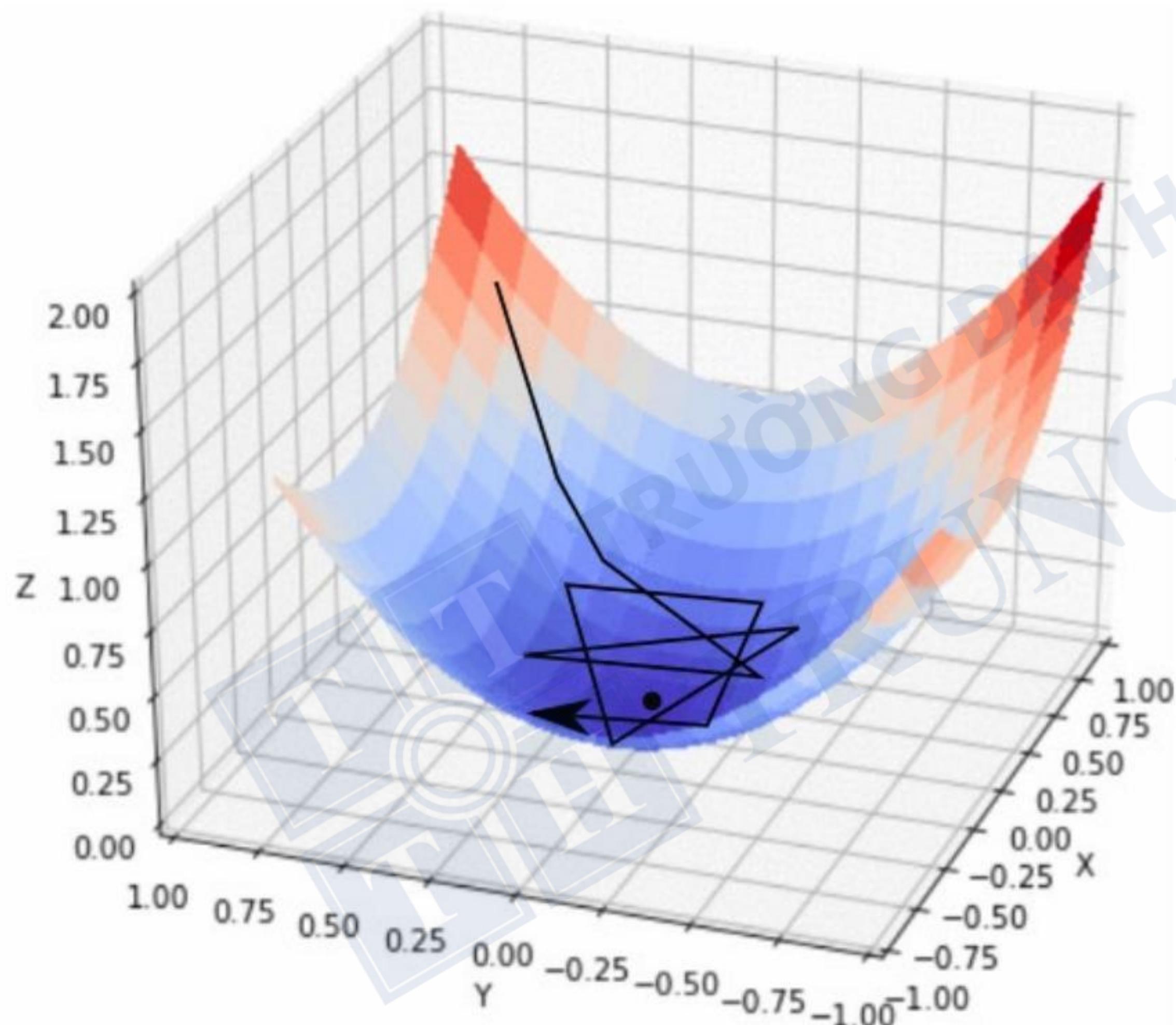
Dẫn đến con số quá lớn  
→ Loss không tối ưu.

Quá nhiều giá trị  $< 1$ :  
**Vanishing gradients**

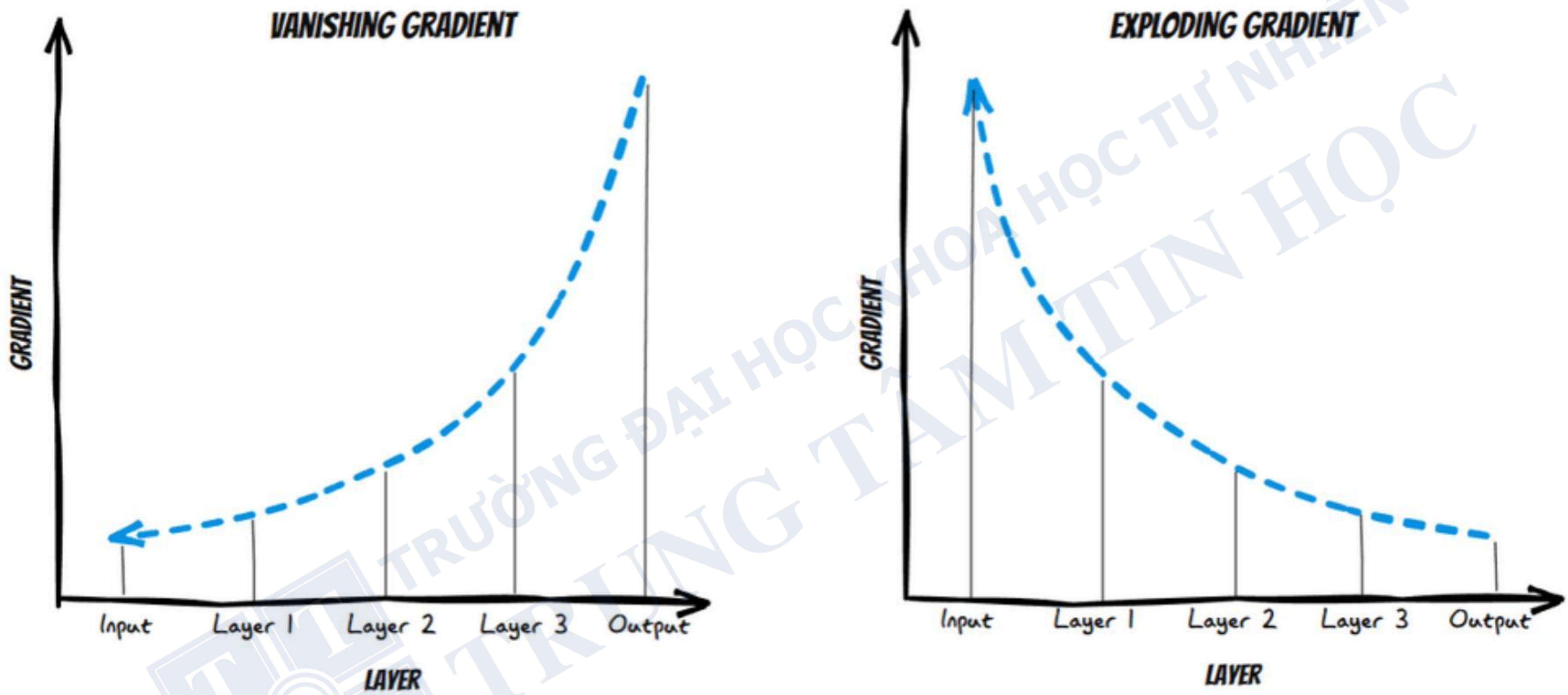
Dẫn đến giá trị gần về 0  
→ Bị triệt tiêu.

# Giải pháp cho Exploding Gradients

Sử dụng **Gradient Clipping**:  
giới hạn giá trị tối đa cho gradient

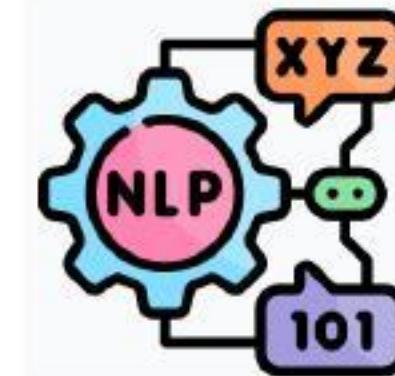


# Giải pháp cho Vanishing Gradients



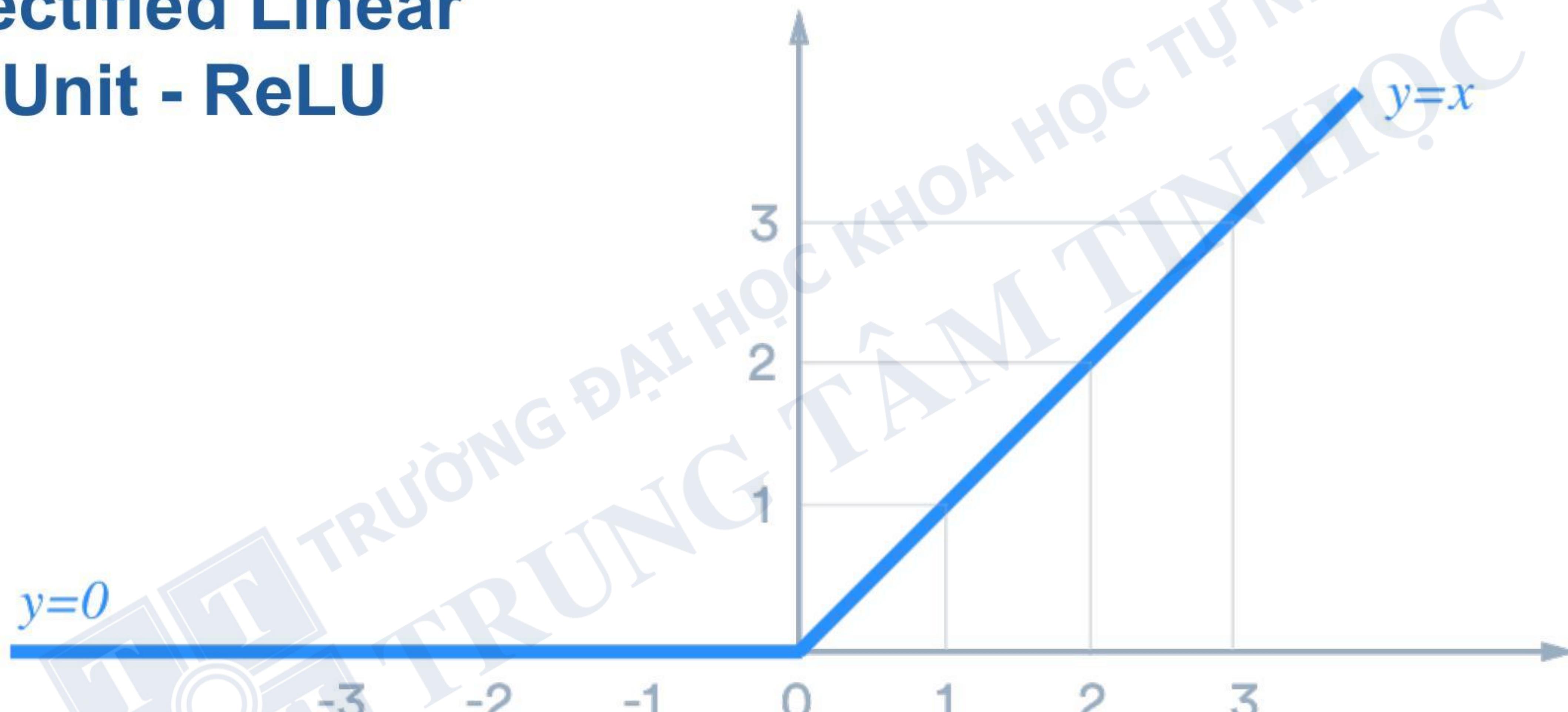
1. Activation functions
2. Weight initialization
3. Network architecture

## Gradient Clipping



# Giải pháp cho Vanishing Gradients

Activation function:  
**Rectified Linear  
Unit - ReLU**



**Hàm ReLU ngăn mô hình triệt tiêu gradient khi  $0 < x < 1$ .**

# RECURRENT NEURAL NETWORK



## I. Tổng quan Recurrent Neural Network

## II. Hạn chế của RNN

## III. Long Short Term Memory - LSTM



# Ưu và nhược điểm của RNN

Ở mỗi time step, các **hàm và bộ tham số giống nhau** được sử dụng để tính toán.

Ưu điểm	Nhược điểm
<span style="color: green;">✓</span> Khả năng xử lý đầu vào có độ dài bất kỳ.	<span style="color: red;">✗</span> Quá trình tính toán chậm.
<span style="color: green;">✓</span> Kích thước mô hình không tăng theo kích thước đầu vào.	<span style="color: red;">✗</span> Khó tiếp cận thông tin lịch sử đã xử lý quá lâu trước đó.
<span style="color: green;">✓</span> Bộ trọng số W được chia sẻ theo suốt các time step.	<span style="color: red;">✗</span> Không thể xem xét input nào khác trong tương lai cho state hiện tại.
<span style="color: green;">✓</span> Có xét đến thông tin lịch sử.	

# Bài 8: RECURRENT NEURAL NETWORK



## I. Tổng quan Recurrent Neural Network

## II. Hạn chế của RNN

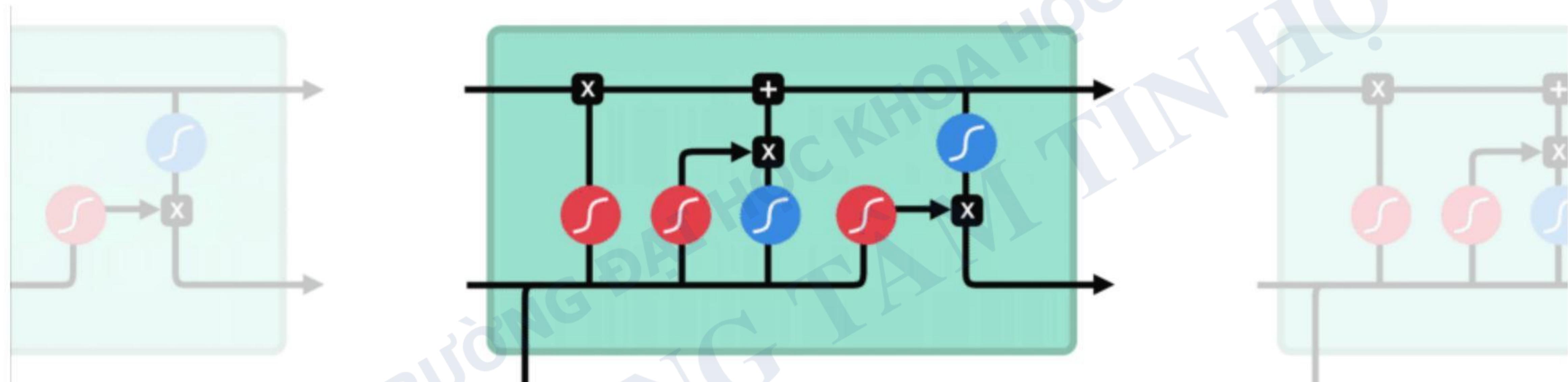
## III. Long Short Term Memory - LSTM





# Long Short Term Memory - LSTM

Để khắc phục vanishing gradient, **gates** được sử dụng để thêm hoặc loại bỏ thông tin ở trong mỗi **recurrent unit**.



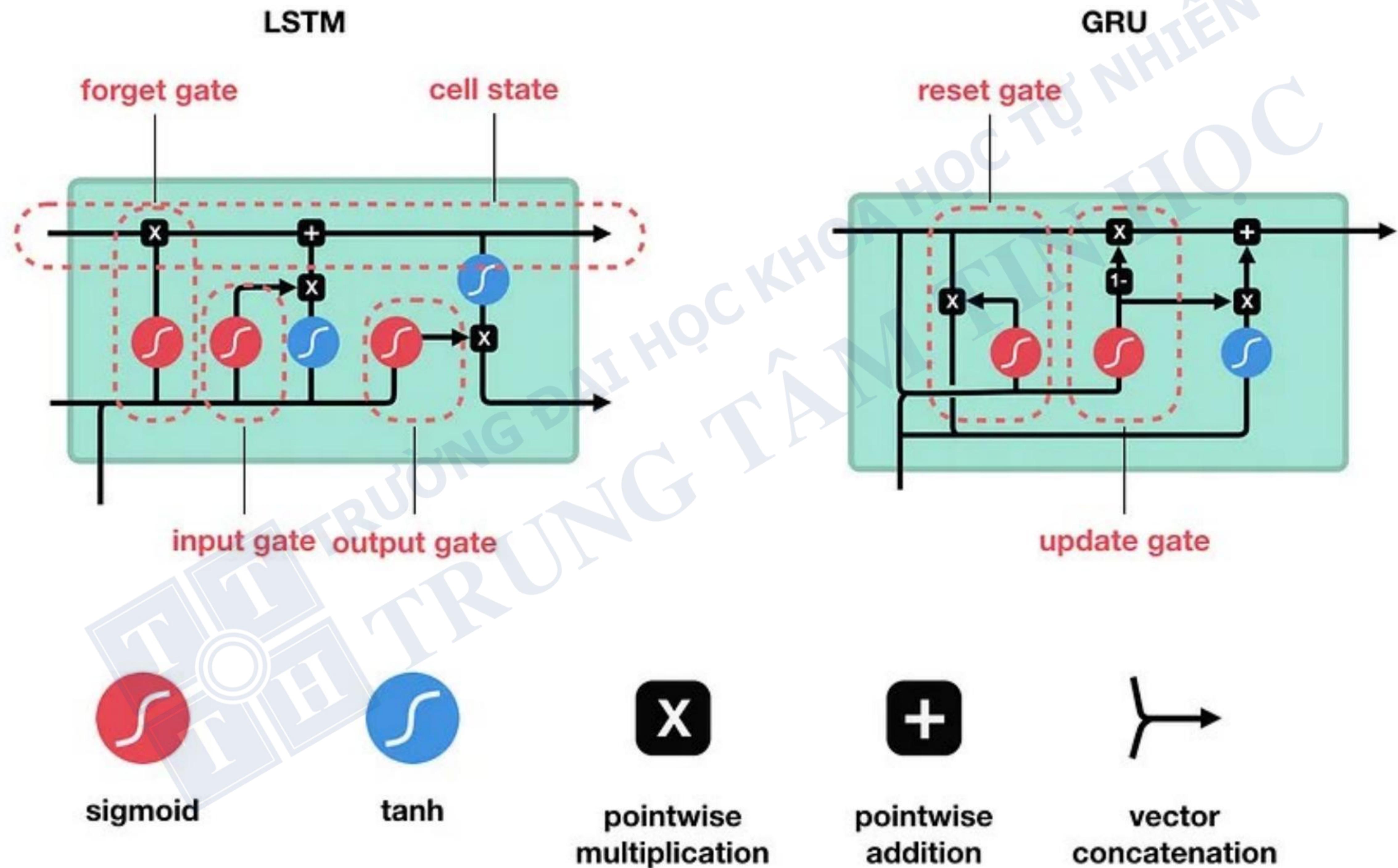


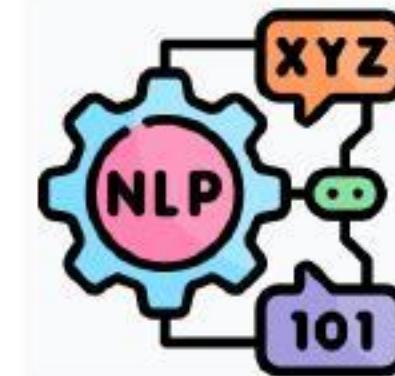
# Các loại Gates

**Gated Recurrent Unit (GRU) và Long Short Term Memory (LSTM)** dựa vào gated cell để theo dõi thông tin xuyên suốt các time step.

Tên	Chức năng	Sử dụng ở
Update gate $\Gamma_u$	Thêm/ cập nhật state cũ cho $h_t$	GRU, LSTM
Relevance gate $\Gamma_r$	Lược bỏ các thông tin tương đồng	GRU, LSTM
Forget gate $\Gamma_f$	Loại bỏ các thông tin không liên quan	LSTM
Output gate $\Gamma_o$	Truyền giá trị đã được cập nhật	LSTM

# LSTM và GRU





# Long Short Term Memory - LSTM

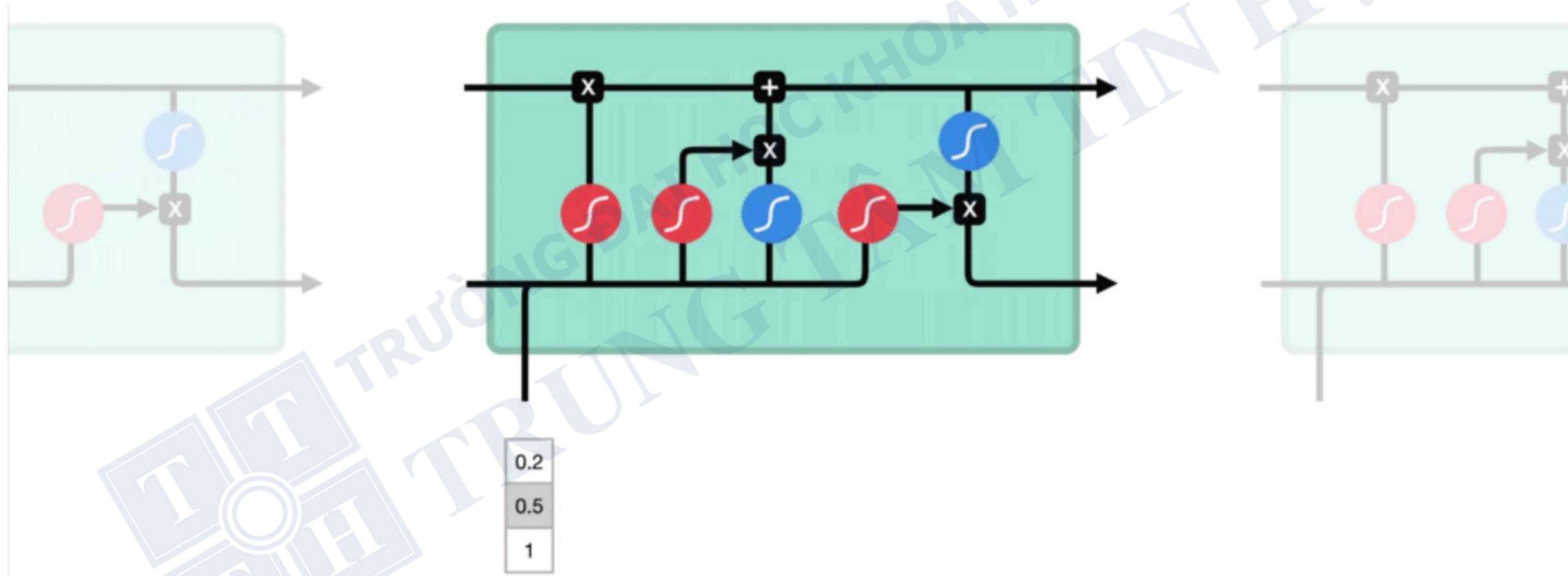
LSTM là mạng Recurrent Neural Network. Các **gated LSTM cells** quản lý luồng thông tin có:

1. Forget

2. Store

3. Update

4. Output



*LSTM có khả năng theo dõi thông tin trong suốt nhiều time step, tối ưu hơn RNN*



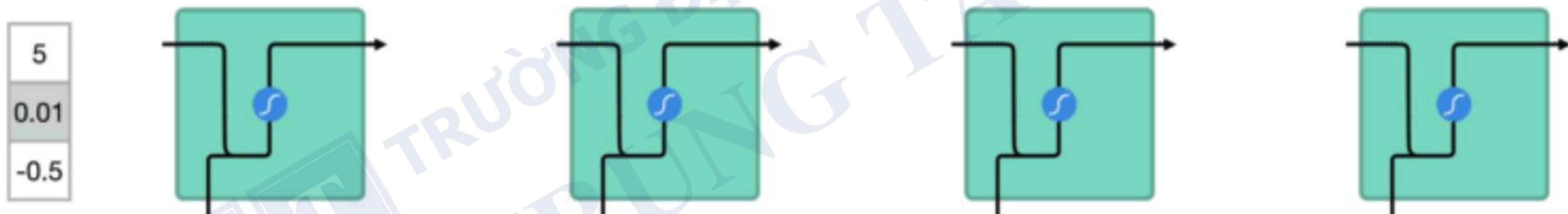
# Long Short Term Memory - LSTM

1. Thông tin từ các time step trước có thể truyền đến các time step sau.
2. Gate là một mạng NN quyết định thông tin nào được phép truyền qua cell state. Gate có thể học được thông tin quan trọng để thêm vào hoặc loại bỏ khỏi cell state.
  - **Forget gate** loại bỏ các thông tin không liên quan.
  - **Store**: lưu trữ thông tin liên quan từ input hiện tại.
  - **Update**: cập nhật cell state có chọn lọc.
  - **Output gate** trả về kết quả đã được lọc bớt.



# Hàm Sigmoid - LSTM

- Mỗi gate có hàm sigmoid làm *activation function*.
- Hàm sigmoid tương tự như hàm tanh, song nén giá trị trong khoảng từ 0 đến 1 thay vì từ -1 đến 1 → **Thích hợp cho việc update hay forget dữ liệu:**

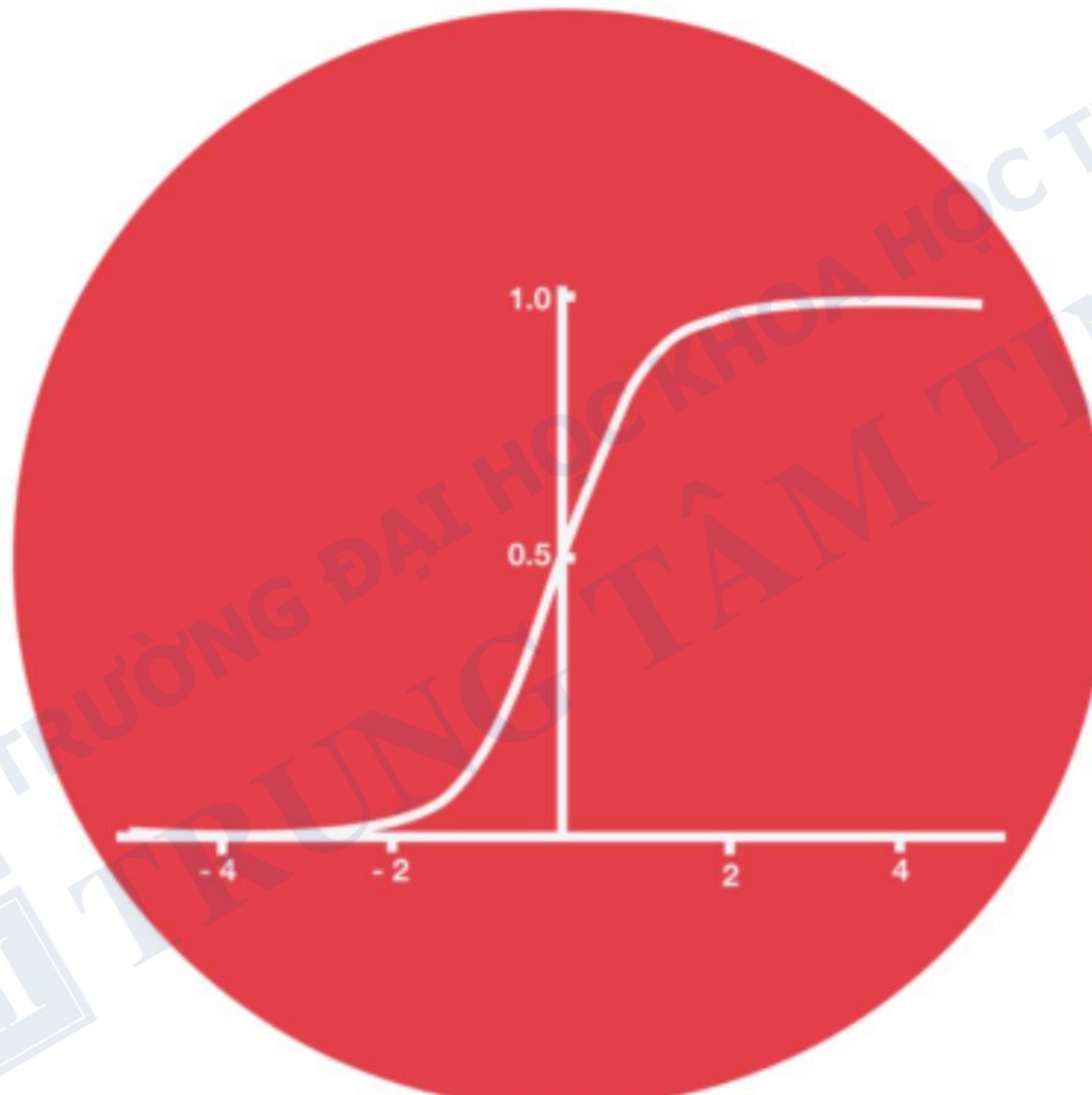


- Từ đó, mạng **LSTM** có thể học được dữ liệu nào không quan trọng để loại bỏ, hoặc dữ liệu nào quan trọng để giữ lại.



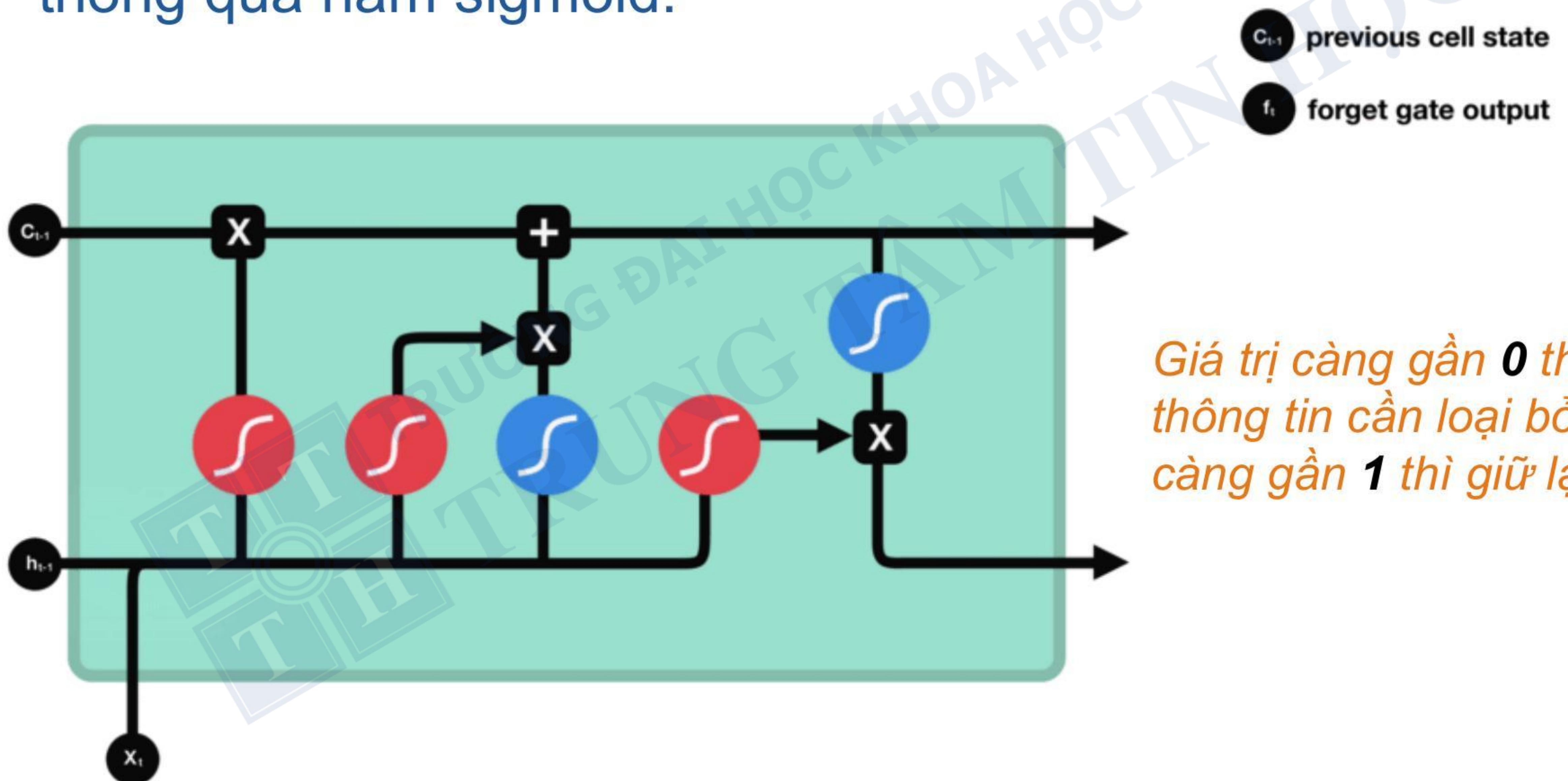
# Hàm Sigmoid - LSTM

5  
0.1  
-0.5



# Forget Gate - LSTM

- **Forget Gate** quyết định loại bỏ hay giữ lại thông tin bằng cách xử lý hidden state trước đó và input hiện tại thông qua hàm sigmoid.

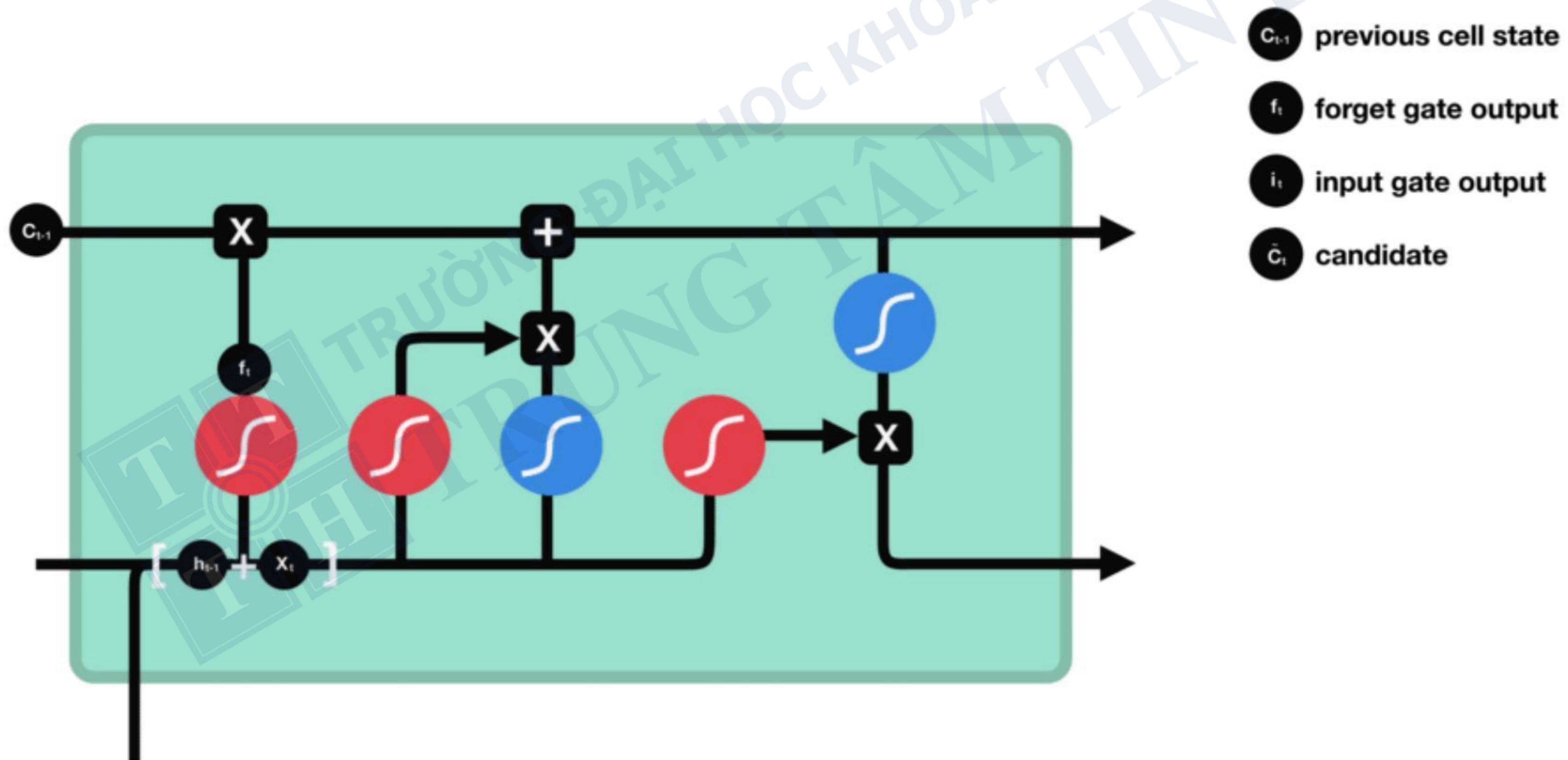




# Input Gate - LSTM

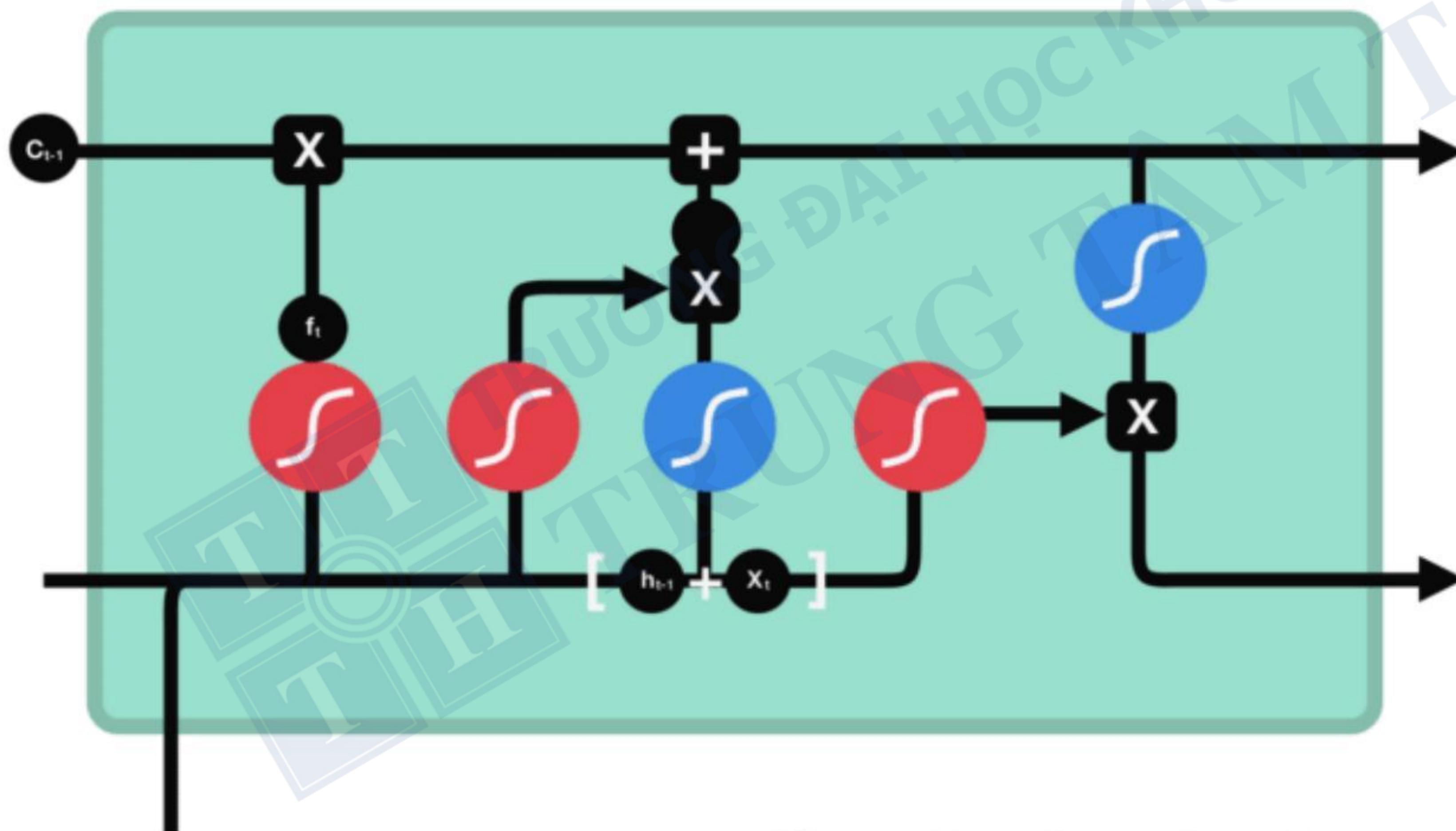
- Input Gate cập nhật cell state trong đó hidden state của cell hiện tại được xử lý bởi **Hàm Tanh**.

Output hàm Tanh sẽ được nhân với output **hàm sigmoid**.



# Cell State - LSTM

- Nhân **cell state cũ** với output **forget gate**:
- Thêm output **input gate** để cập nhật **cell state** với giá trị liên quan  
**→ Cell state mới.**



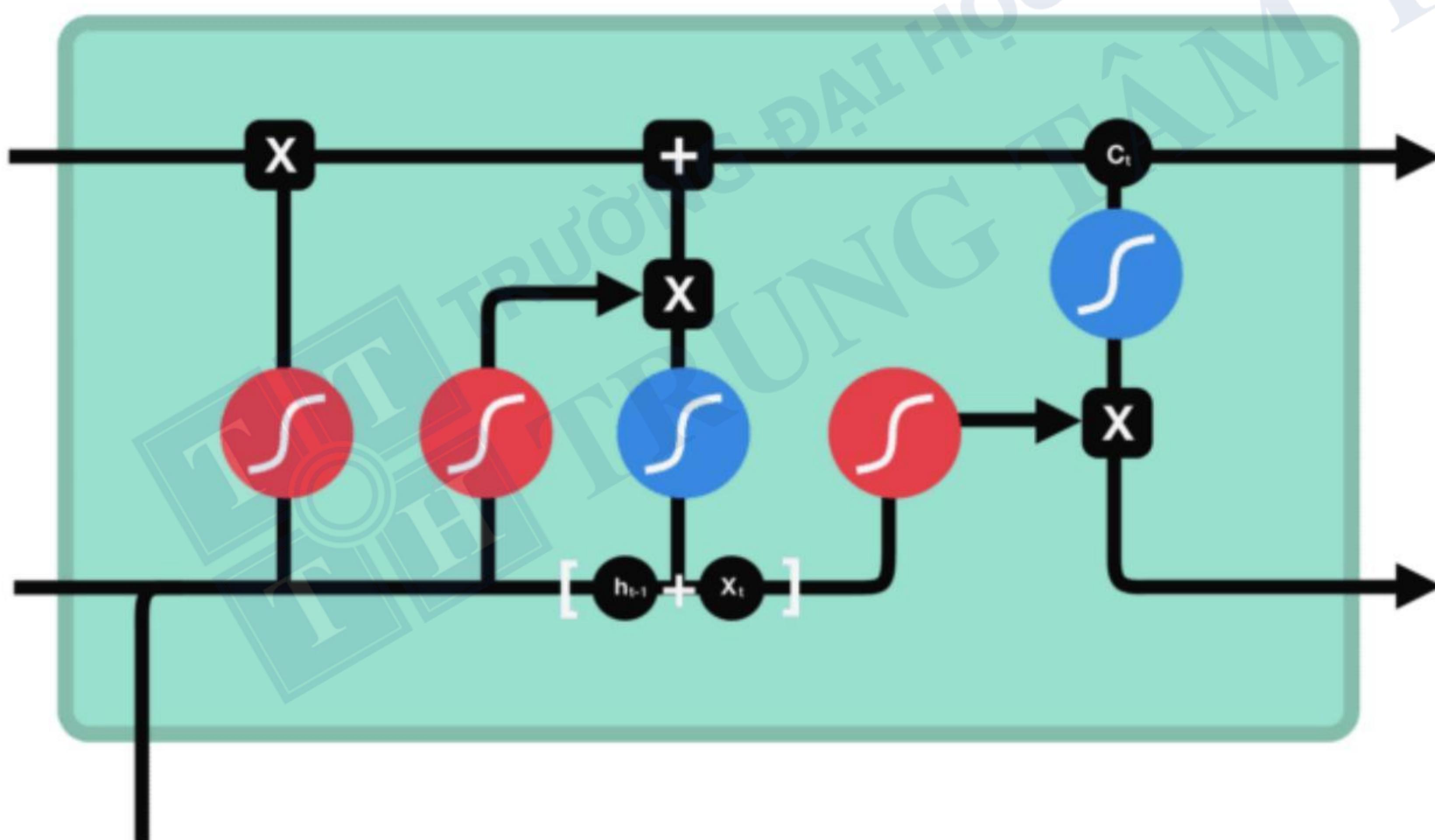
$c_{t-1}$	previous cell state
$f_t$	forget gate output
$i_t$	input gate output
$\tilde{c}_t$	candidate
$c_t$	new cell state

*Output forget gate gần 0 thì giá trị tương ứng trong cell state bị loại bỏ.*

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

# Output Gate - LSTM

- Đưa hidden state cũ và input hiện tại vào **hàm sigmoid**.  
Đưa cell state mới vào **hàm tanh**.
- Nhân output của sigmoid và tanh để quyết định thông tin nào nên được truyền tiếp.  
→ **Hidden state mới.**

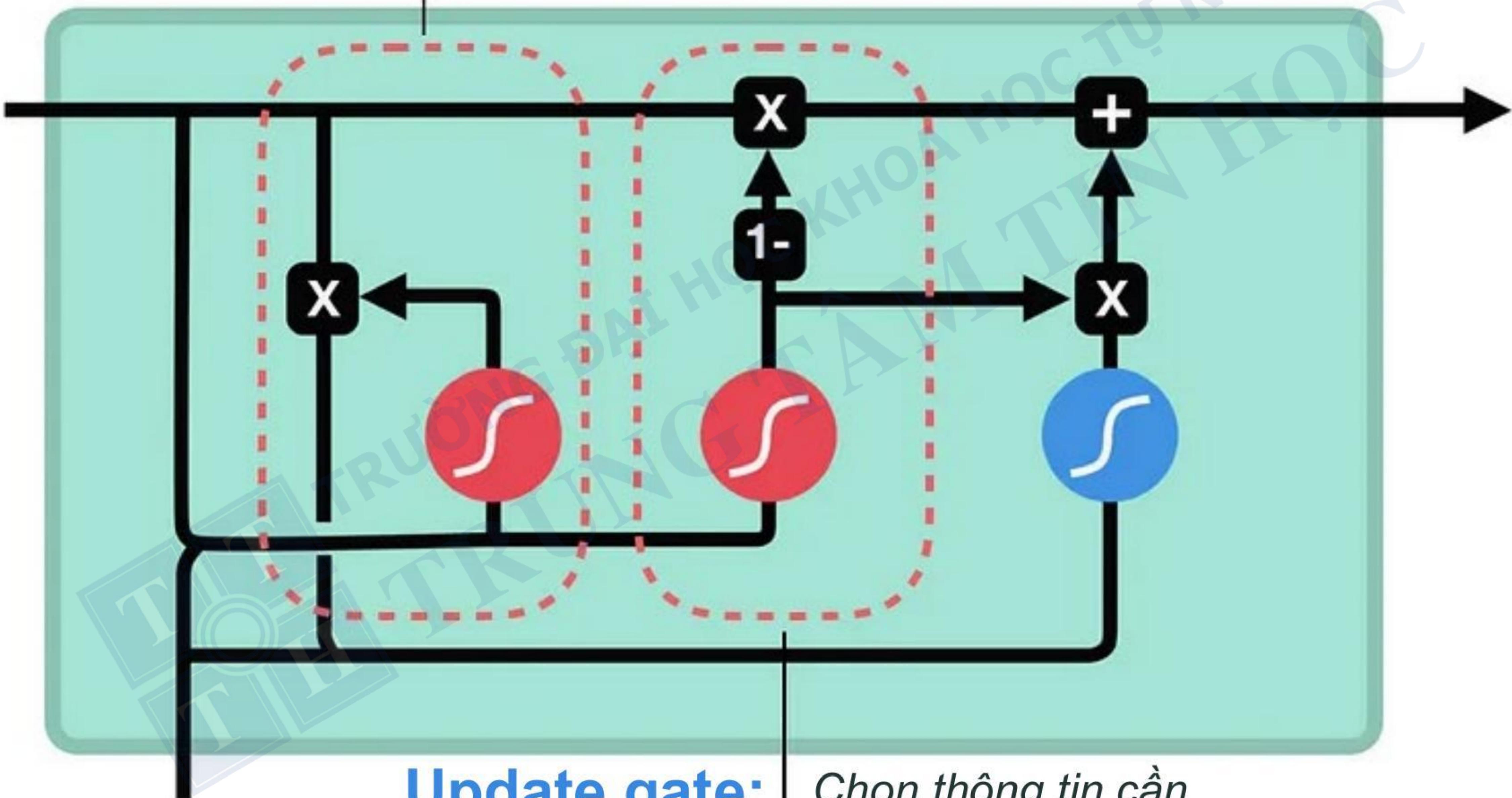


$c_{t-1}$  previous cell state  
 $f_t$  forget gate output  
 $i_t$  input gate output  
 $\tilde{c}_t$  candidate  
 $c_t$  new cell state  
 $o_t$  output gate output  
 $h_t$  hidden state

*Cell state và hidden state là input cho time step tiếp theo.*

# Gated Recurrent Unit - GRU

**Reset gate:** Xác định bao nhiêu thông tin cũ cần loại bỏ (*forget*).



**Update gate:** Chọn thông tin cần loại bỏ hay thêm vào.  
Forget gate, Input gate

# Code Demo



→ Truy cập vào:

[LINK](#)



# Q&A

---

