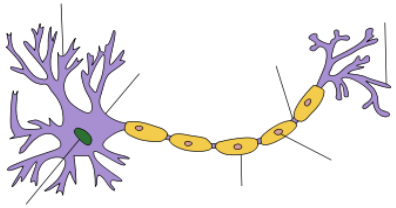
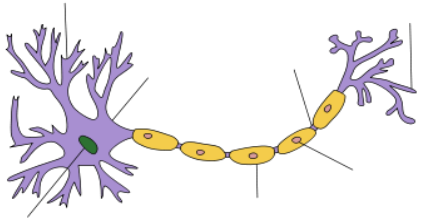
UMONS
Université de Mons


# ADVANCED DEEP LEARNING (I-ILIA-202)

## CHAPTER 03 : RECURRENT NEURAL NETWORKS (RNN)





Sidi Ahmed Mahmoudi

POLYTECH MONS

sidi.mahmoudi@umons.ac.be

10 December 2024

# PLAN

Introduction

I. Recurrent Neural Networks (RNNs)

II. Vanish Gradient Problem

III. Long Short-Term Memory networks (LSTM)

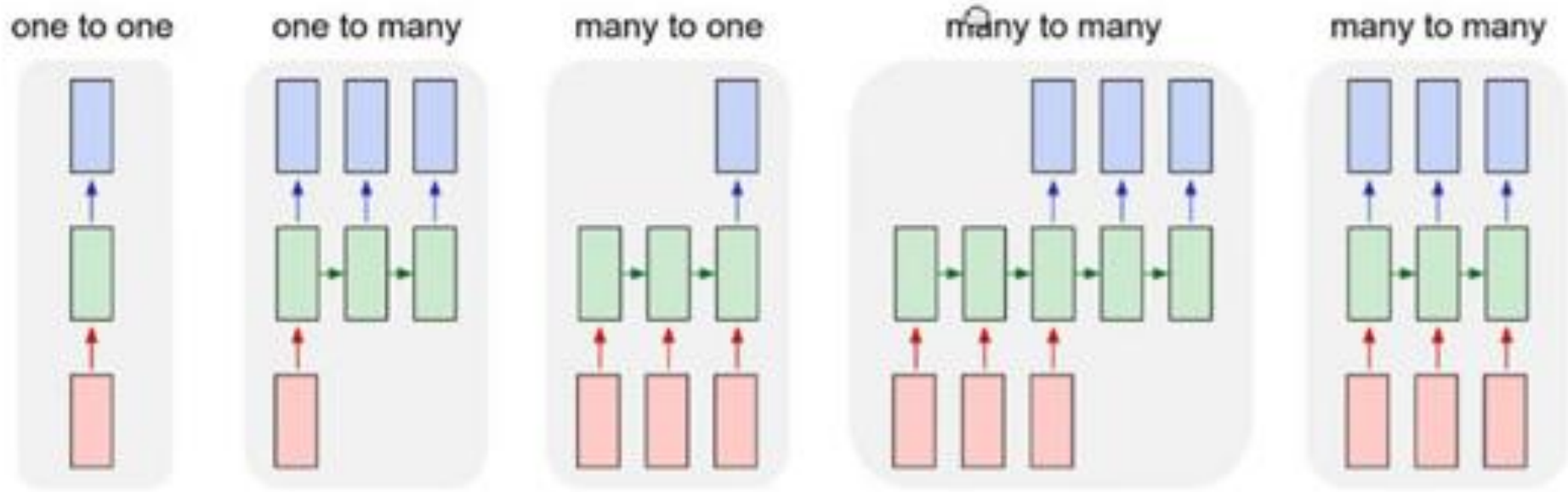
Conclusion

# PLAN

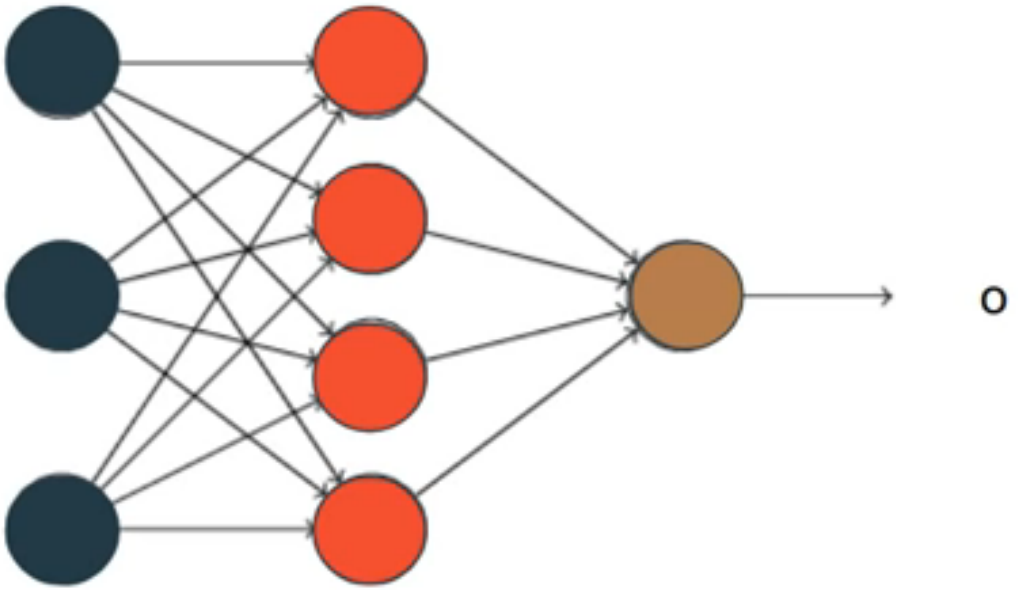
# Introduction

- **MLP :** solve several problems (regression, classification, etc.)
- **Convolutional Neural Networks (CNN) :** improve the results by considering the **spatial information of pixels**
- Traditional networks: input examples fed to the network → output
- Some problems such as speech recognition require a system that need to store and use the context information
- Example : If I say, " How are". The prediction should consider the **two ordered** inputs "**how**" and "**are**" to predict the value of "**You**"

# Introduction



# Feedforward neural network




Erreur : $(o - t)^2$

Gradient

# PLAN

Introduction

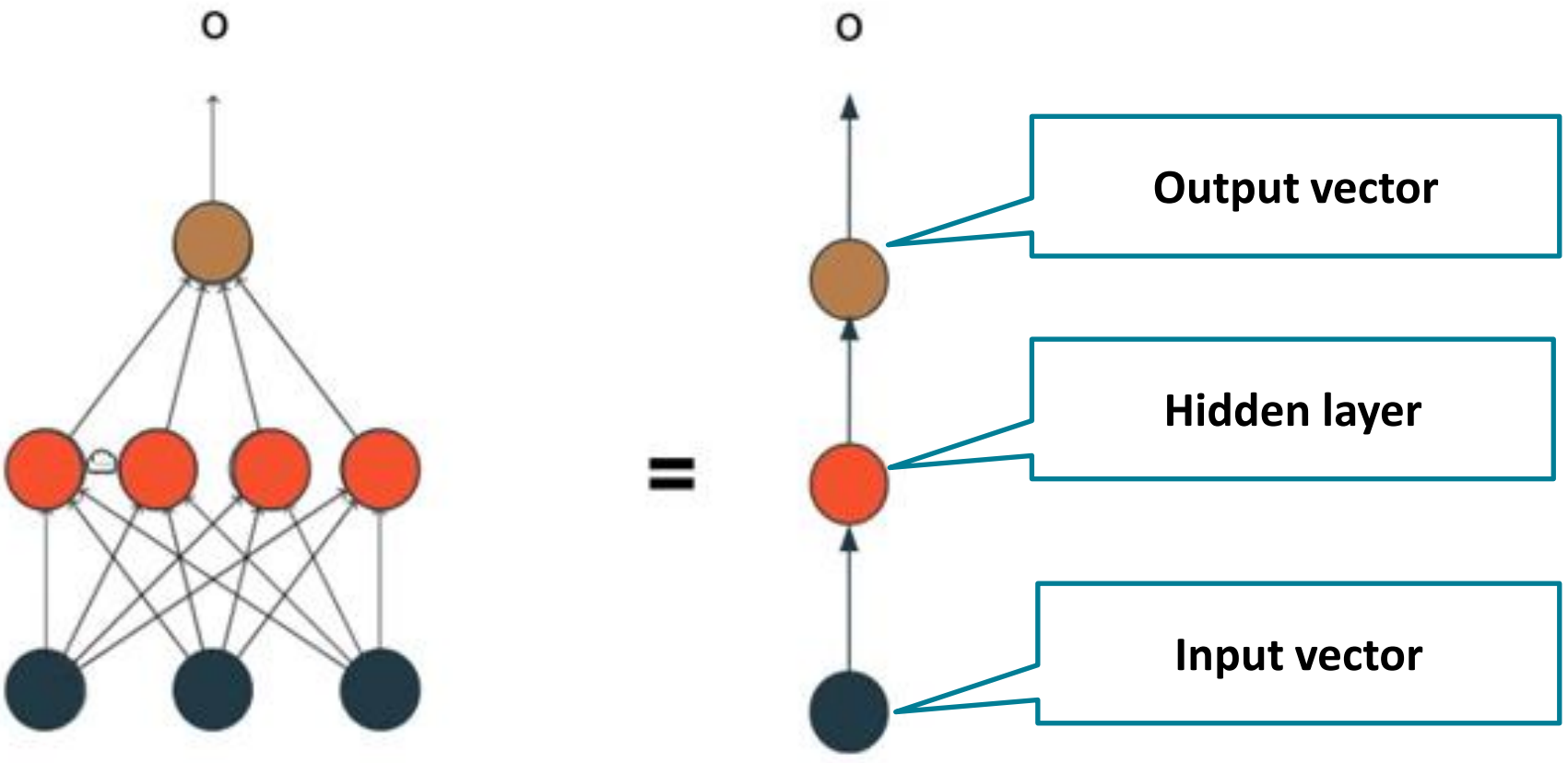
I. **Recurrent Neural Networks (RNNs)**

II. Vanish Gradient Problem

III. Long Short-Term Memory networks (LSTM)

Conclusion

# Feedforward neural network

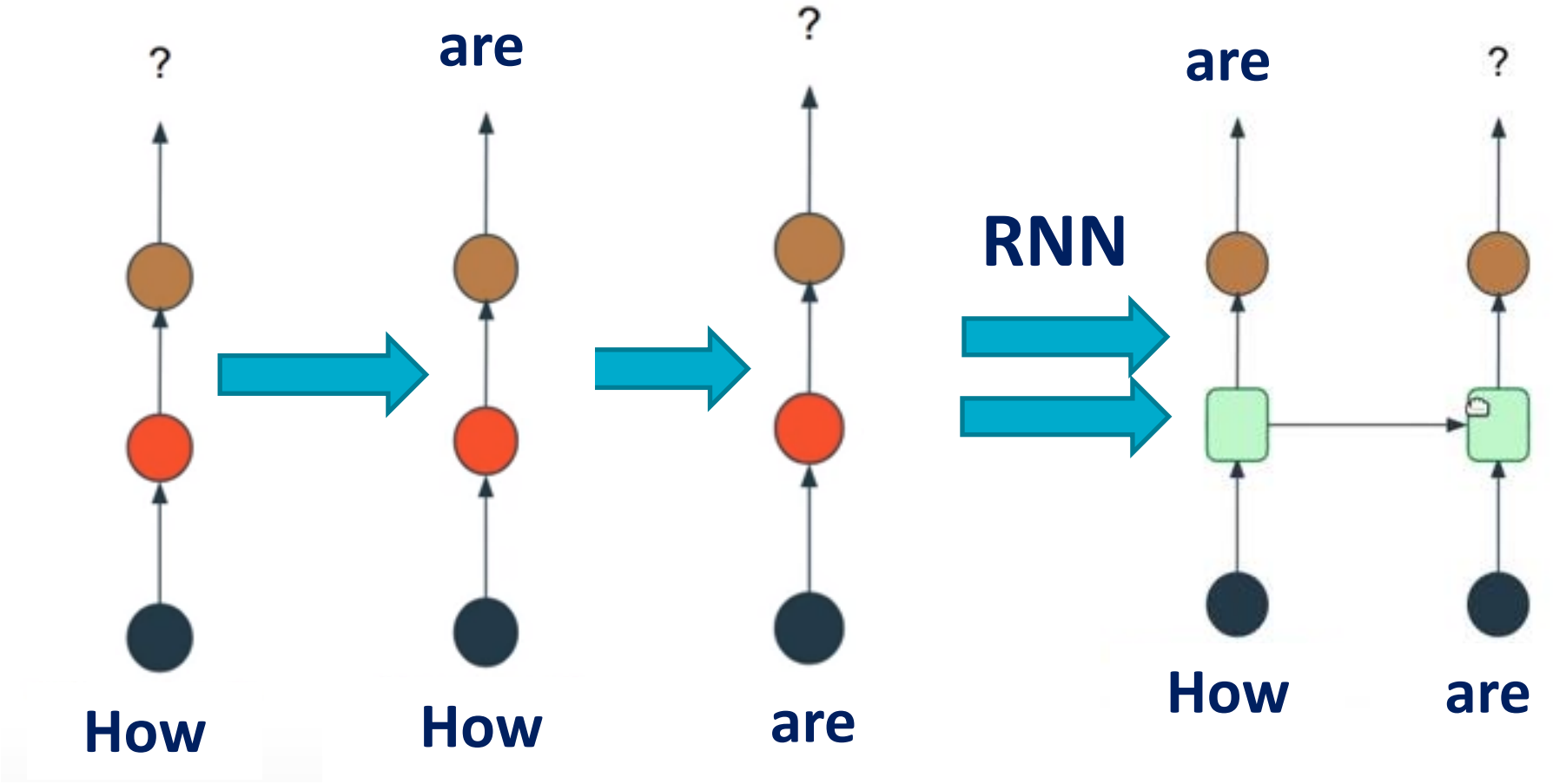


# Example : neural network for next word prediction

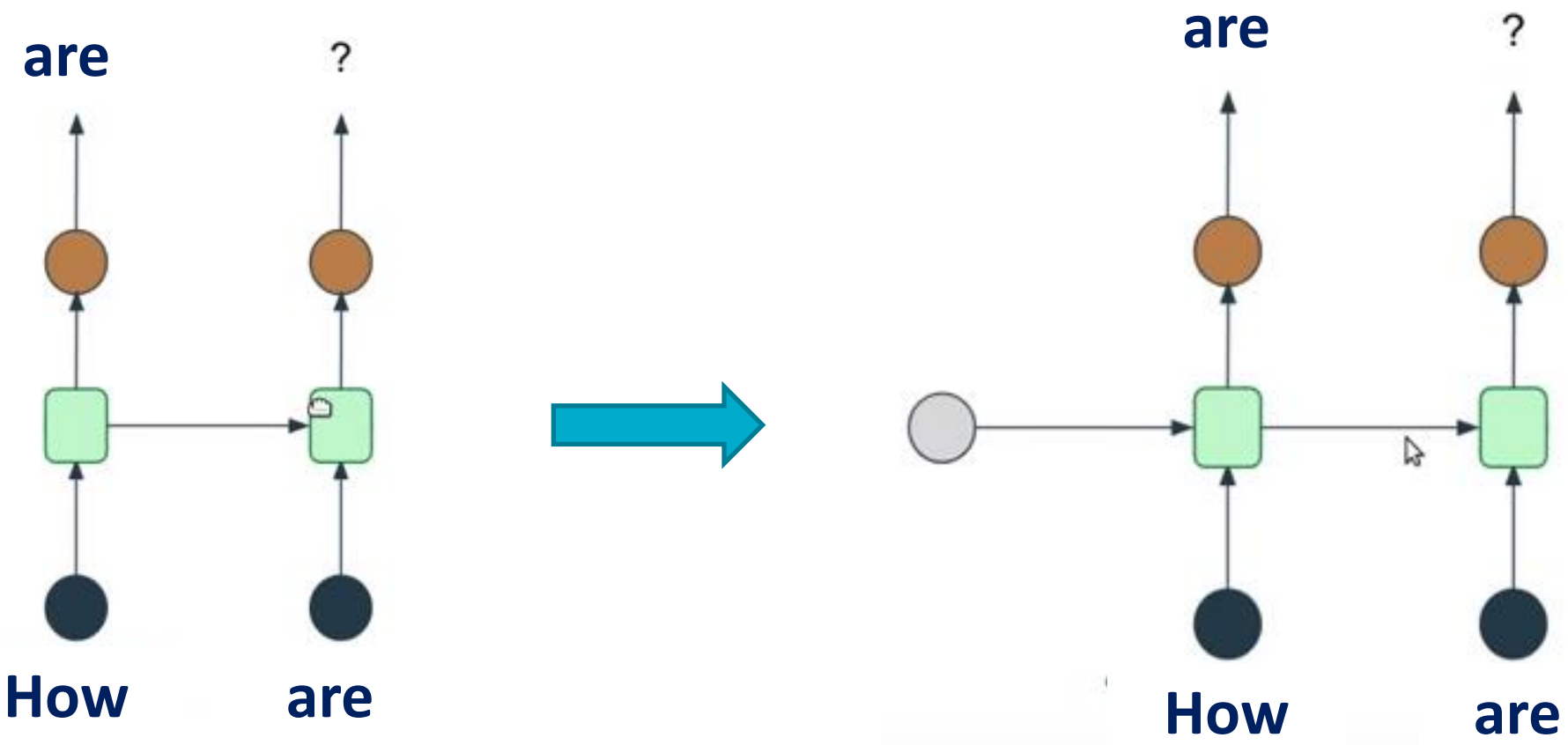
- **Problem :** predict the next word in a sentence ?
- Input : **How**
- Real output : **are you**

**Question :** what kind of deep neural network can you propose to solve this problem ?

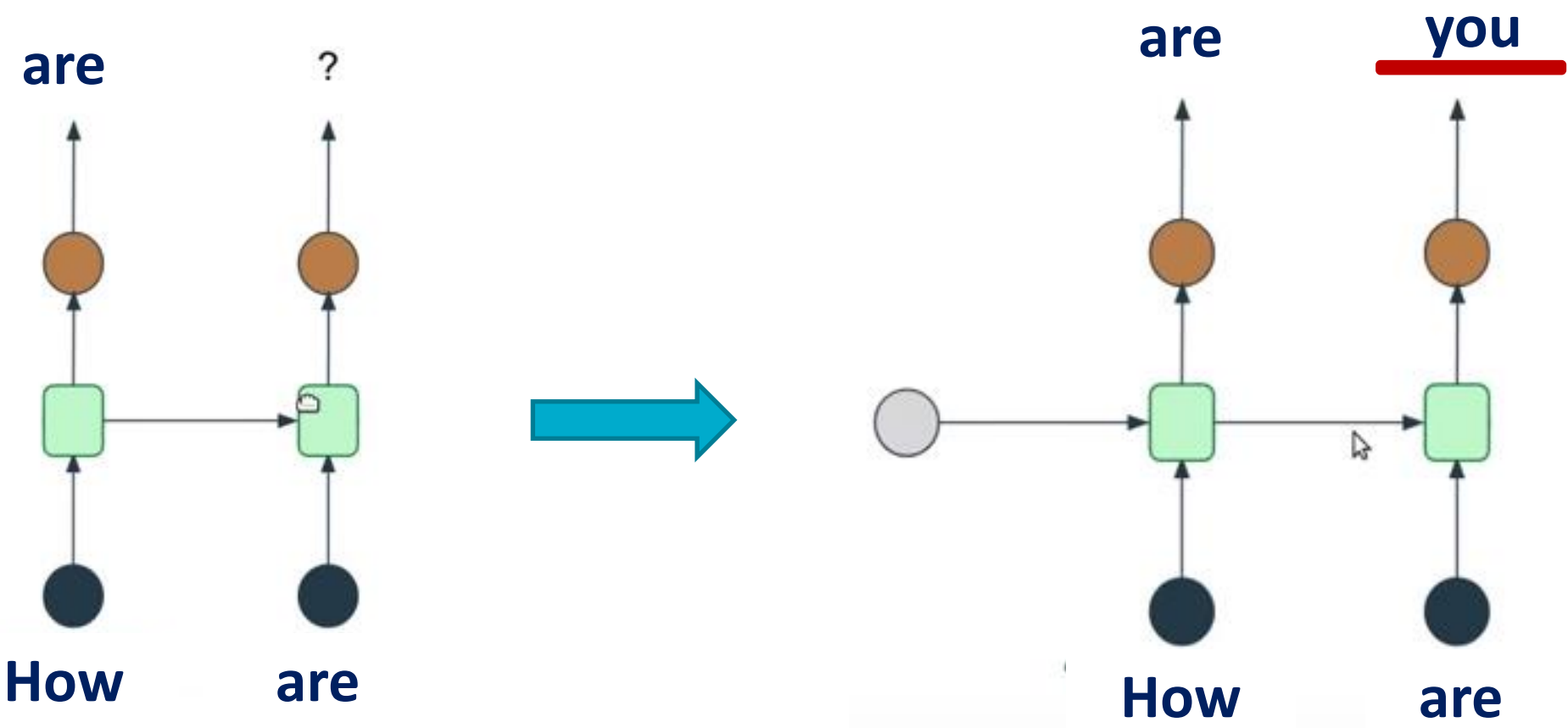
# Example : neural network for next word prediction



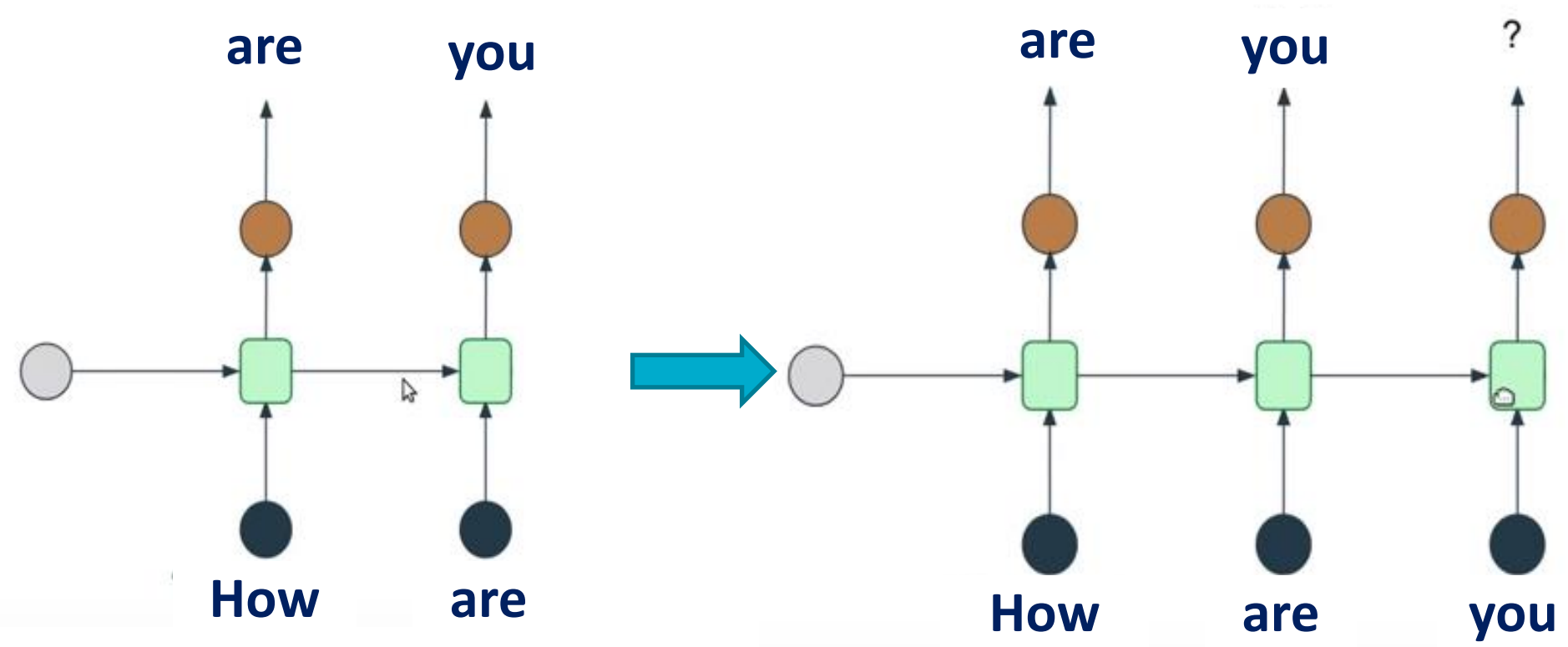
# Example : neural network for next word prediction



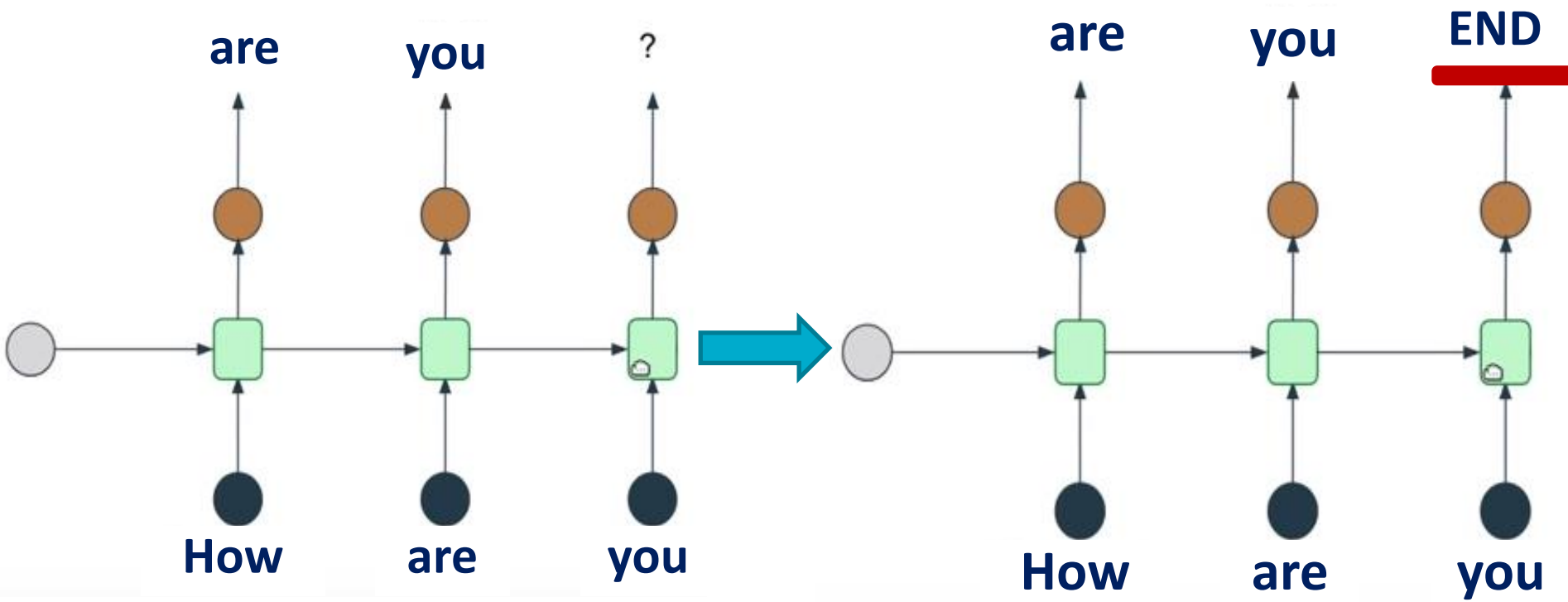
# Example : neural network for next word prediction



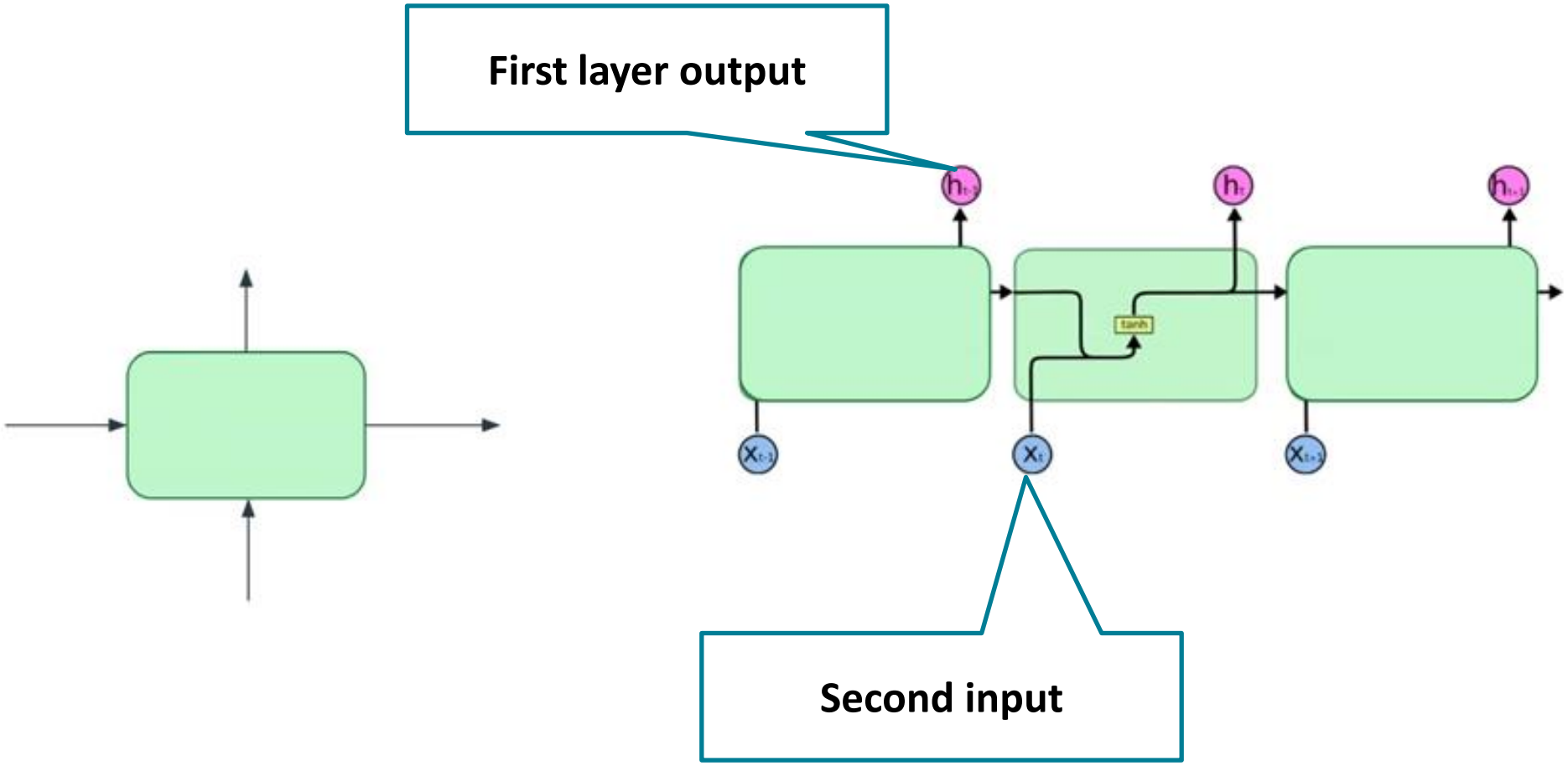
# Example : neural network for next word prediction



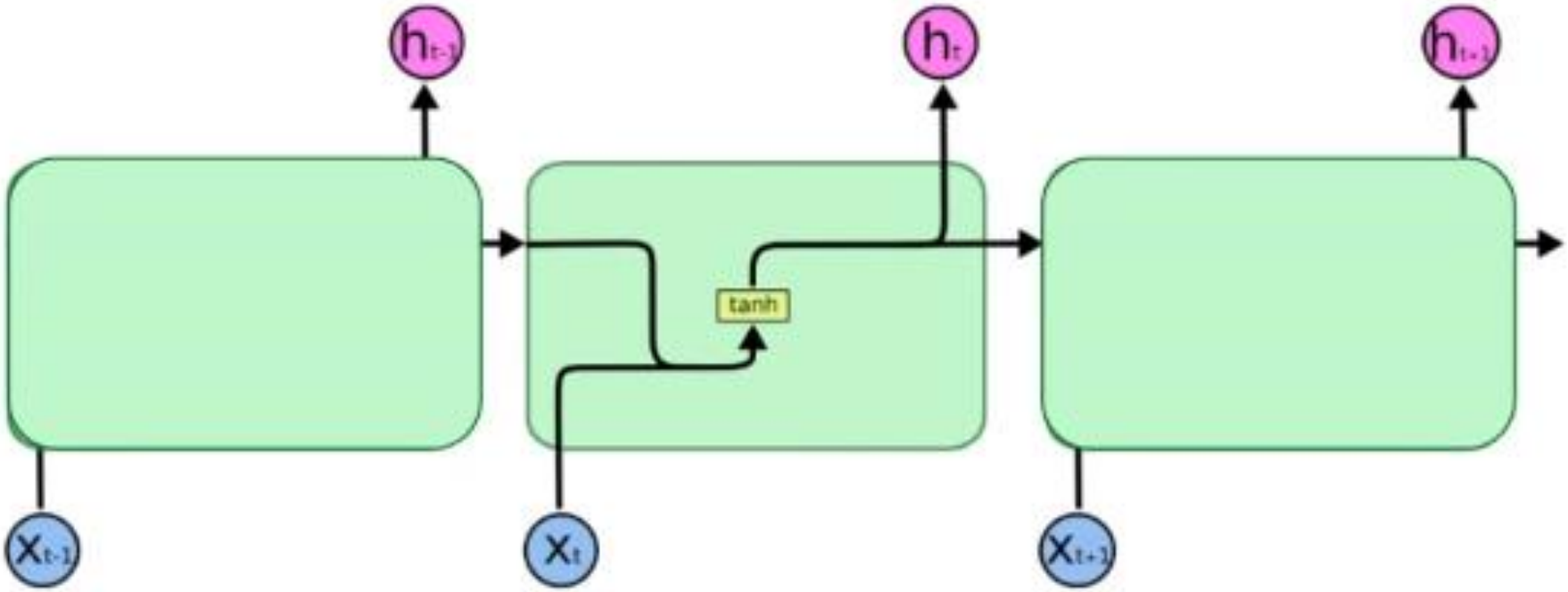
# Example : neural network for next word prediction



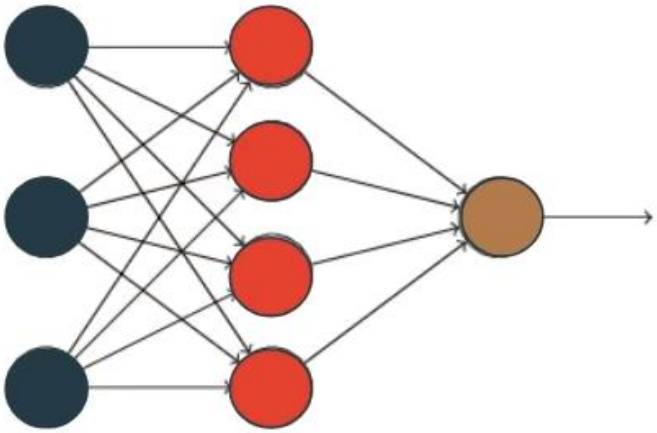
# Vanilla recurrent neural network



# Vanilla recurrent neural network



# Deep neural networks : reminder



o $W_{xh}x_t$

**Weights matrix**

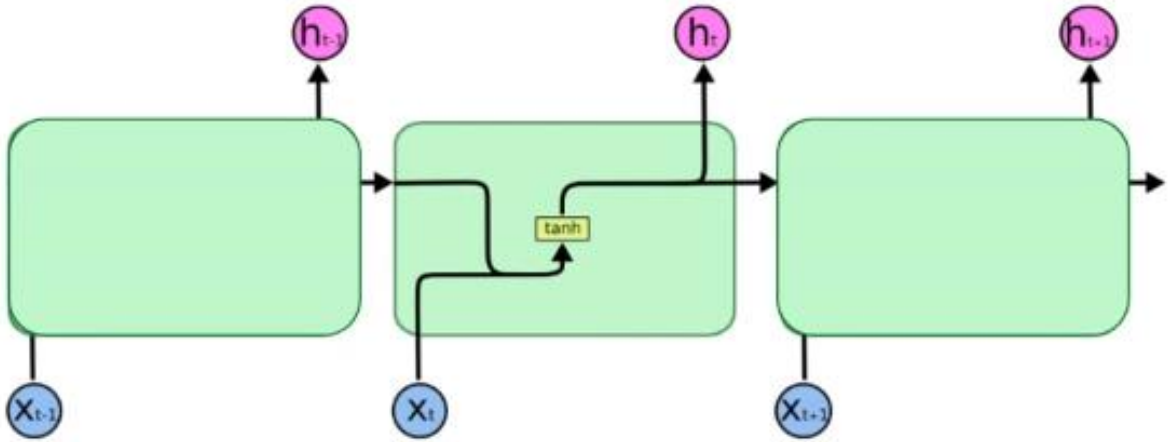
**Pré-activation**

**Activation 1**

**Activation 2**

```
>>> W = np.random.randn(3, 4)
>>> b = np.random.randn(4)
>>> x = np.random.randn(3)
>>> np.dot(x, W) + b
array([1.87339572, 2.07677249, 1.23722445, 3.25528786])
>>> sigmoid(np.dot(x, W) + b)
array([0.8668507 , 0.88862501, 0.77508052, 0.96286266])
>>> np.tanh(np.dot(x, W) + b)
array([0.95390098, 0.96906863, 0.84466214, 0.99702917])
```

# Vanilla recurrent neural network

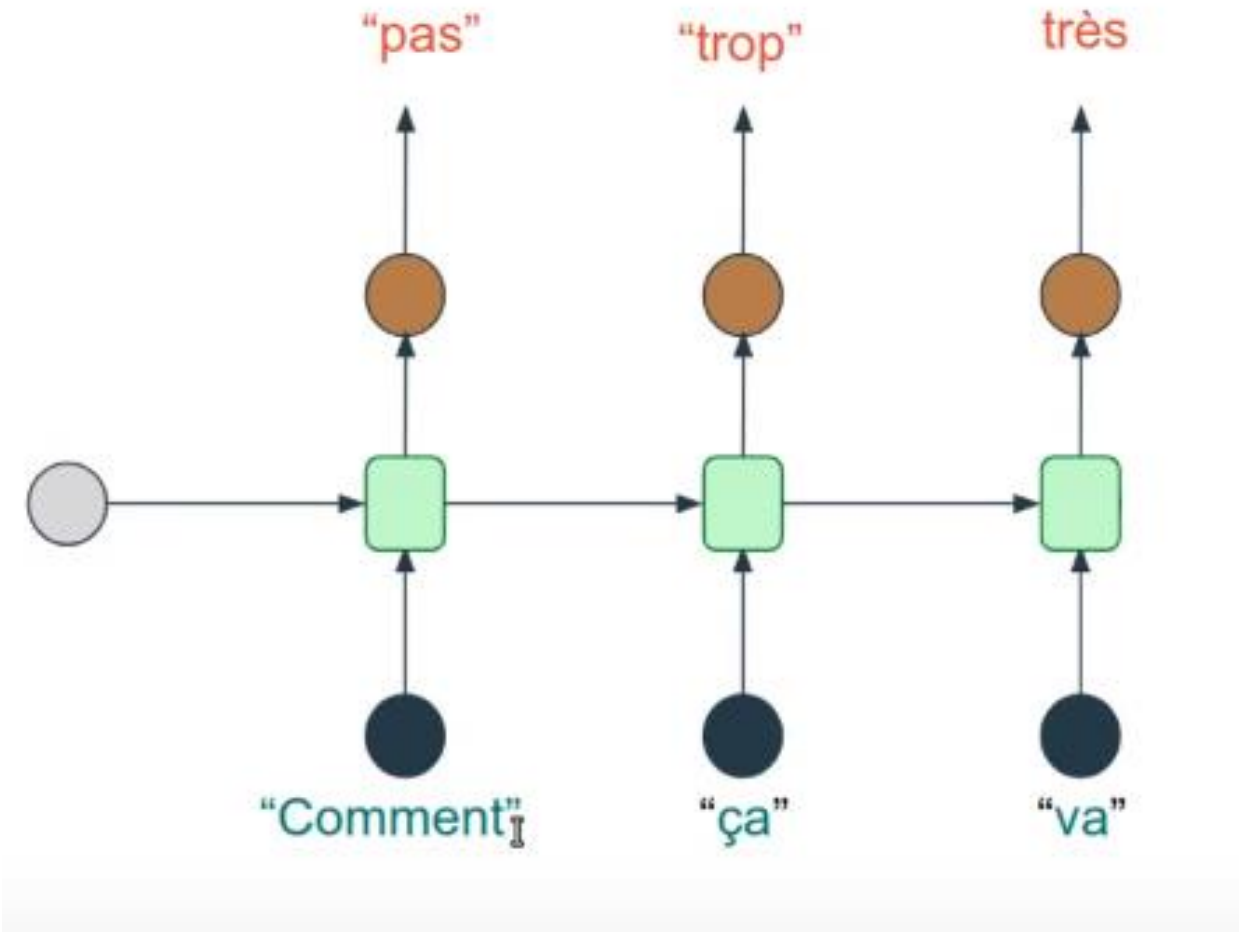



$$h_t = f_W(h_{t-1}, x_t)$$

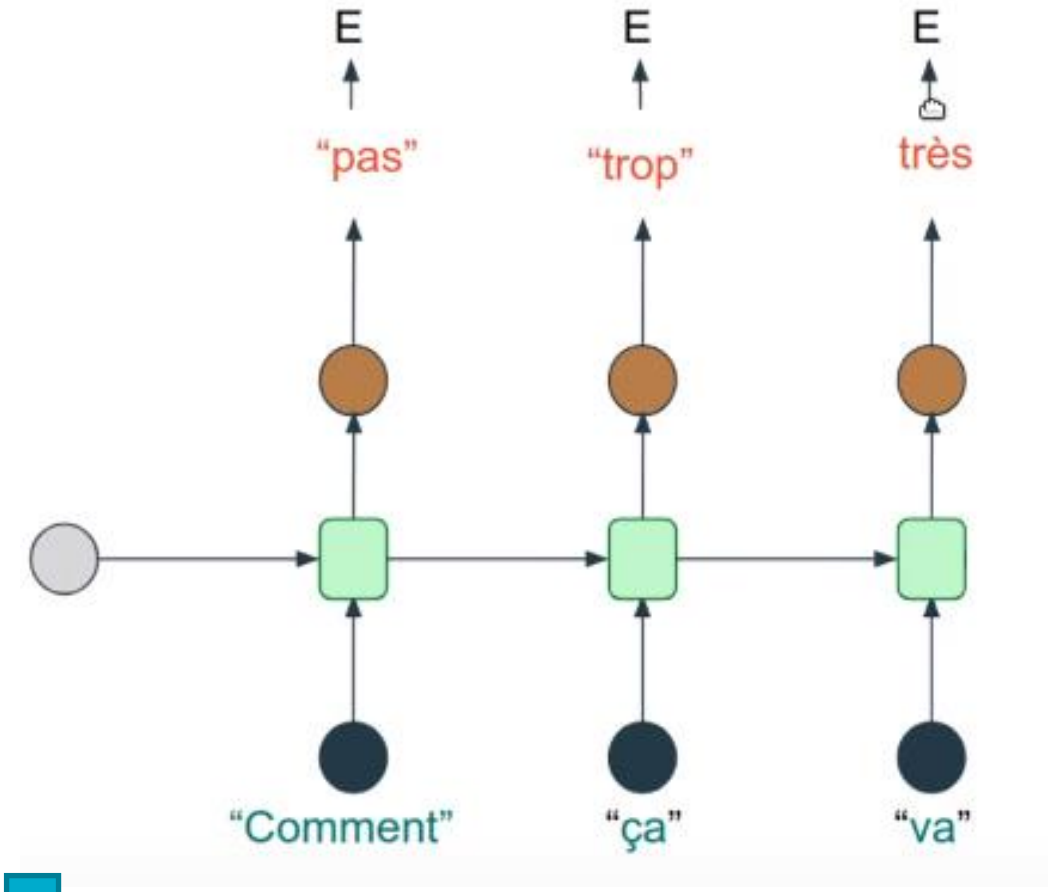
$$\downarrow$$

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

# RNN : training



# RNN : training




**Vanish gradient problem**

# PLAN

Introduction

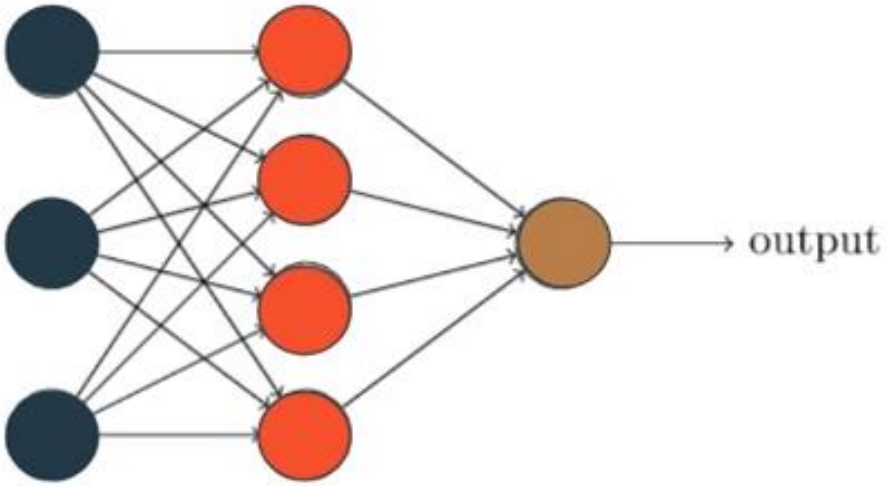
I. Recurrent Neural Networks (RNNs)

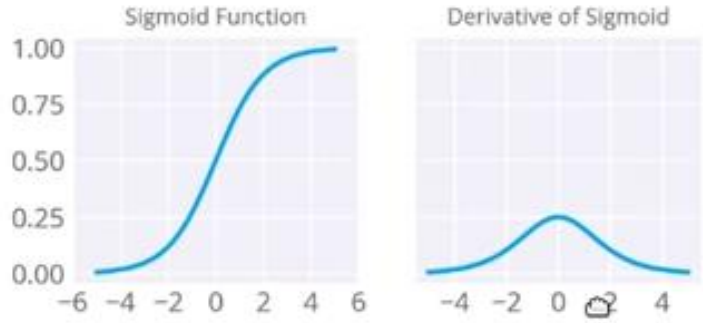
II. Vanish Gradient Problem

III. Long Short-Term Memory networks (LSTM)

Conclusion

# The Vanish Gradient Problem

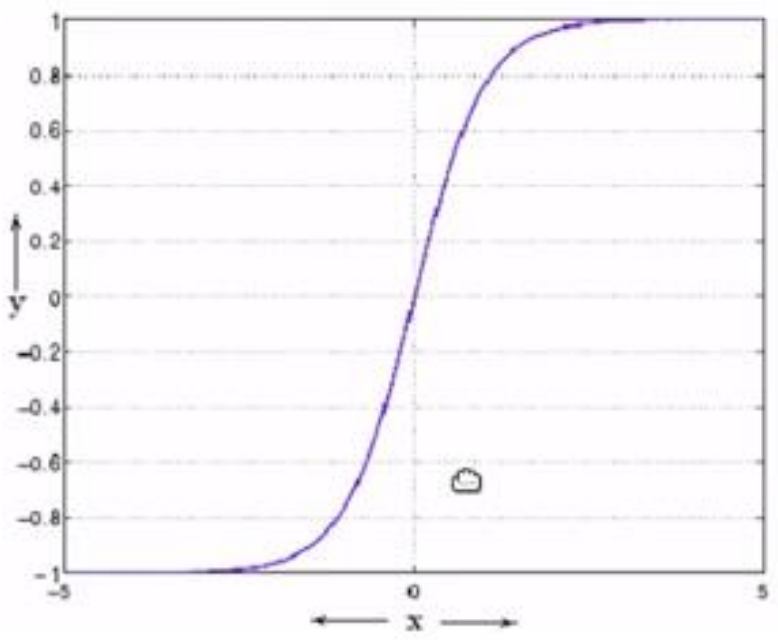


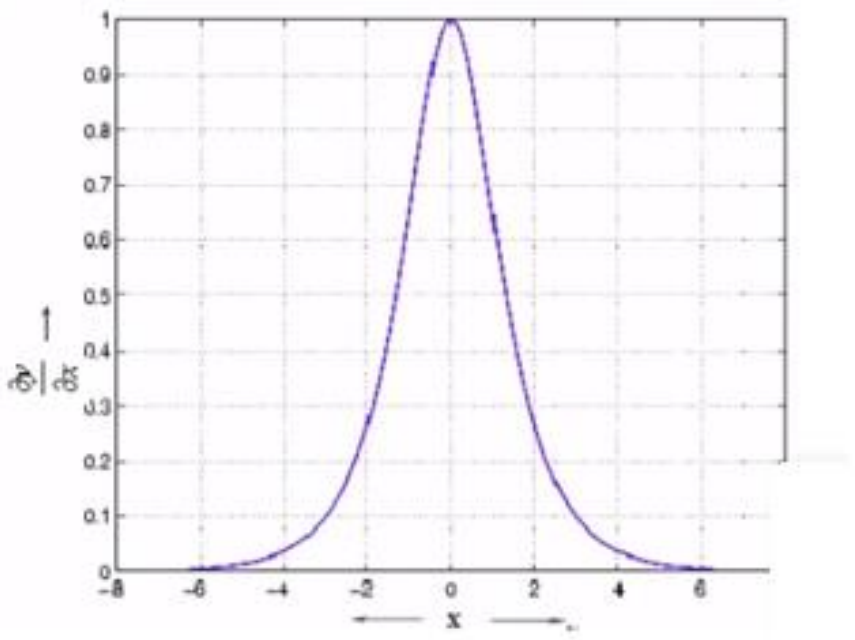




**Small Derivative** → **Small Gradient** → **Low modification of weights**

RNN : gradient multiplicated by several small gradients : **small gradient values**

# Vanish Gradient Problem

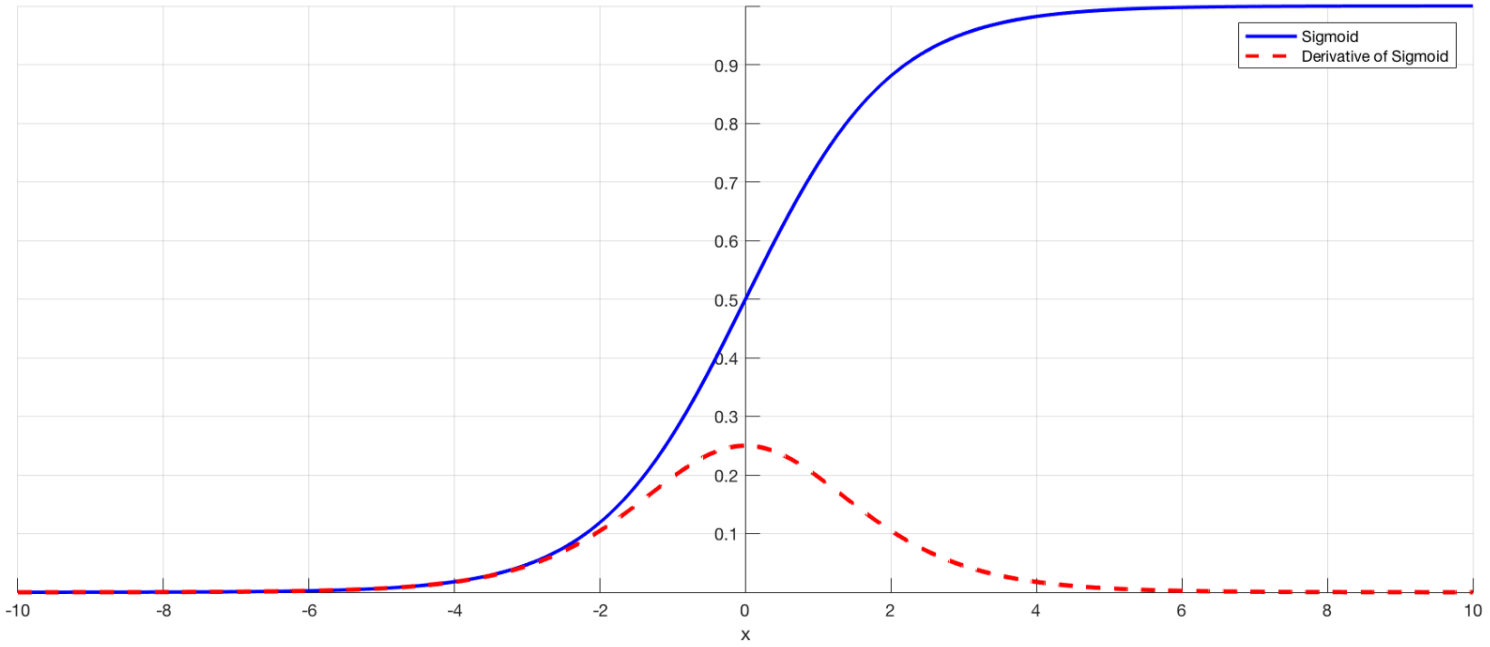



Vanish gradient : problem when using big neural network

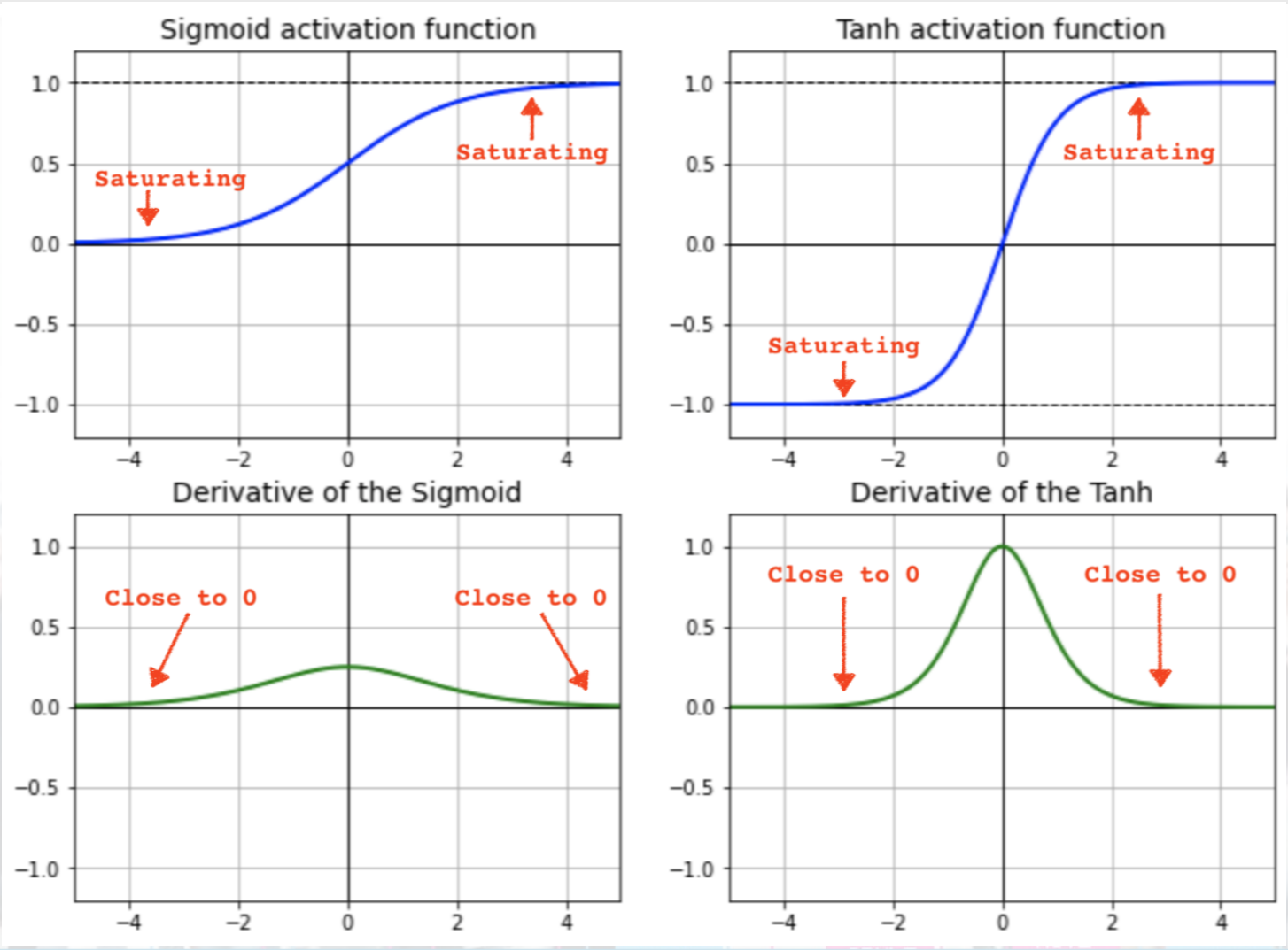
# The Vanish Gradient Problem

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$




- What do you remark ?

# Activations functions

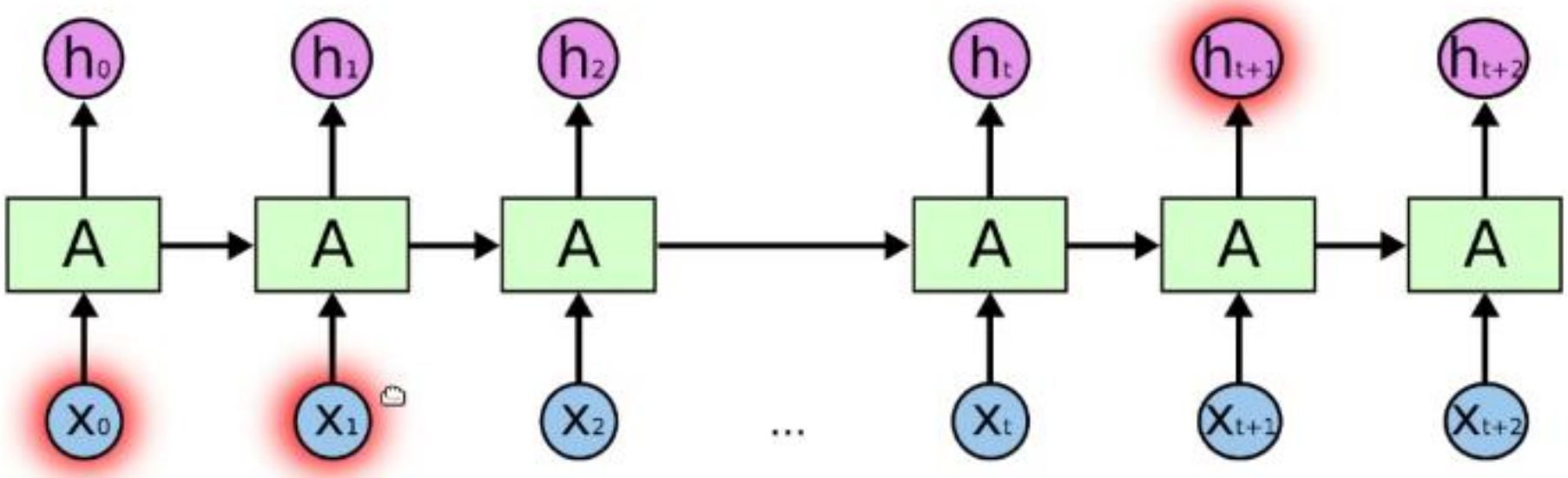



- What do you remark ?

# The Vanish Gradient Problem

The solution : LSTM

# Vanish Gradient Problem



# PLAN

Introduction

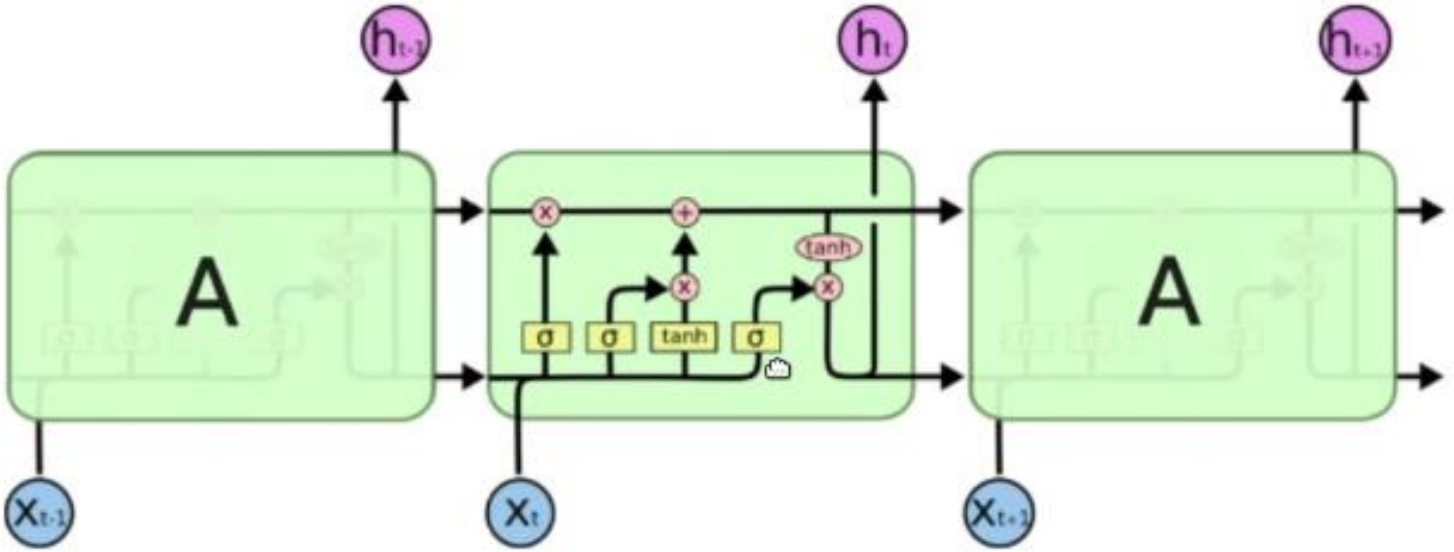
I. Recurrent Neural Networks (RNNs)

II. Vanish Gradient Problem

III. **Long Short-Term Memory networks (LSTM)**

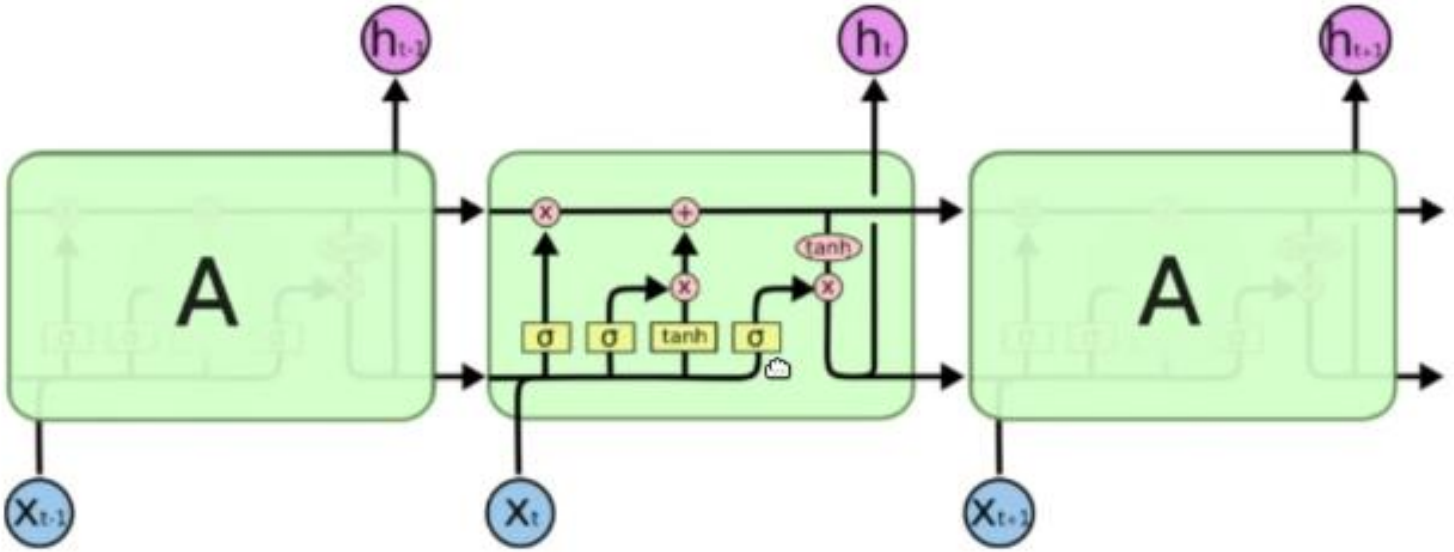
Conclusion

# LSTM




- Partial solution of vanish gradient problem
- Solution based on memory
- More complex neural network

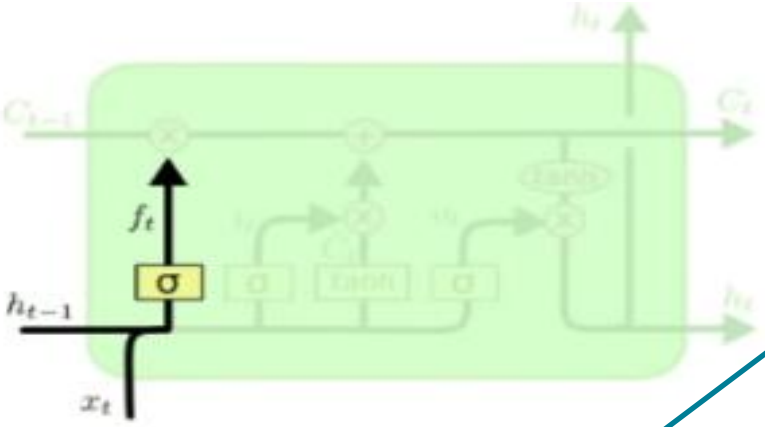
# LSTM




Three main operations :

- **Forget Gate :** ability of forget an information
- **Input Gate :** ability to add input information
- **Output Gate :** output considering Forget Gate and Input Gate

# LSTM : Forget Gate



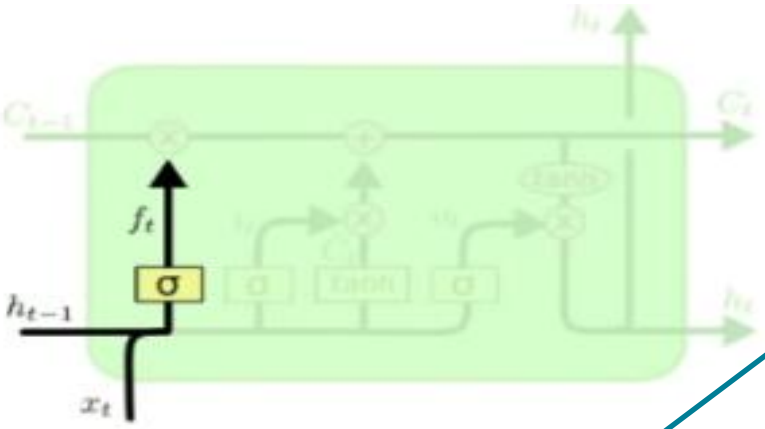

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

**Forget gate**

**Concatenation**

```
>>> c_prev.shape
(5,)
>>> h_prev.shape
(5,)
>>> x.shape
(4,)
```

# LSTM : Forget Gate




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

**Forget gate**

**Concatenation**

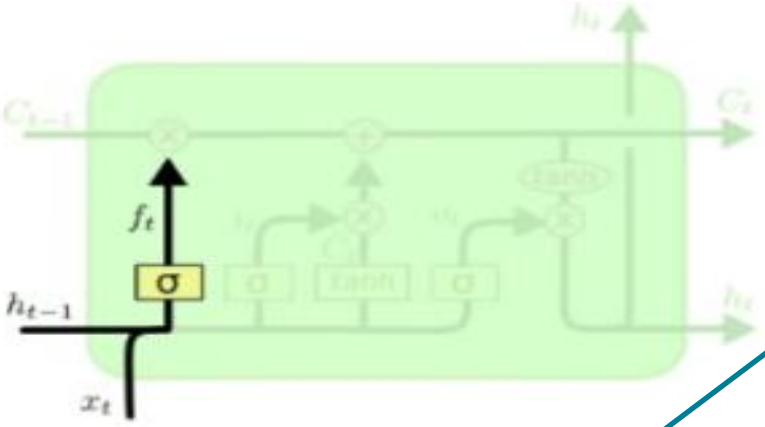
**Concatenation**

```
>>> x_h_prev = np.hstack((x, h_prev))
```

**Concatenated vector**

```
>>> x_h_prev.shape
(9,)
```

# LSTM : Forget Gate



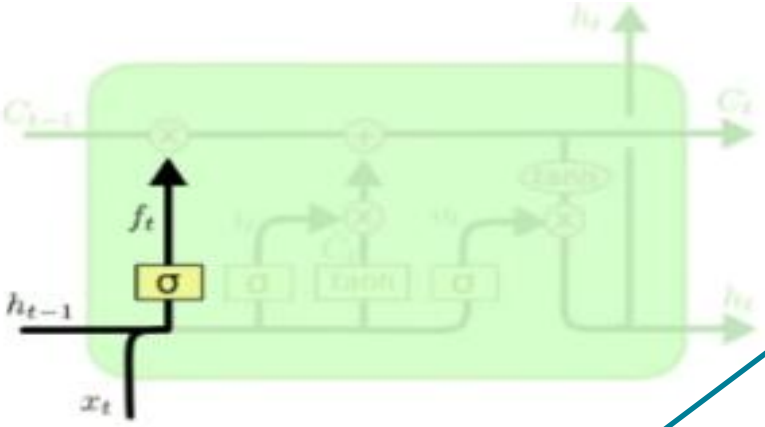

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

**Forget gate**

**Concatenation**

```
>>> Wf.shape
(9, 5)
>>> bf.shape
(5,)
>>> ft = sigmoid(np.dot(x_h_prev, Wf) + bf)
>>> ft.shape
(5,)
```

# LSTM : Forget Gate



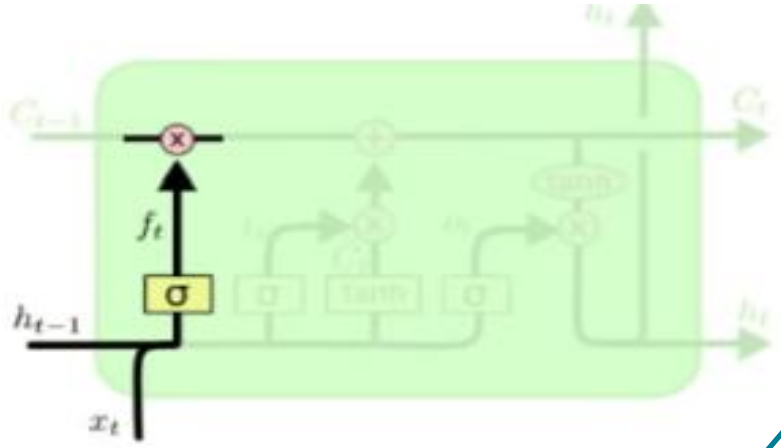

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

**Forget gate**

**Concatenation**

```
>>> ft
array([0.00605241, 0.02419927, 0.12958965, 0.83141943,
0.5440948 ])
>>> c_prev
array([ 0.38000574,  1.13691447,  1.57618308, -1.01247179,
1.02257568])
```

# LSTM : Forget Gate



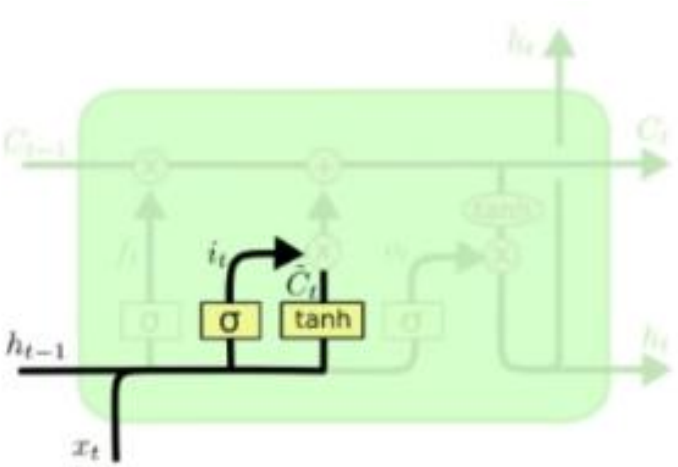

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

**Forget gate**

**Concatenation**

```
>>> c_prev_forgot = ft*c_prev
>>> c_prev_forgot.shape
(5,)
>>> c_prev_forgot.shape
(5,)
>>>
```

# LSTM : Input Gate

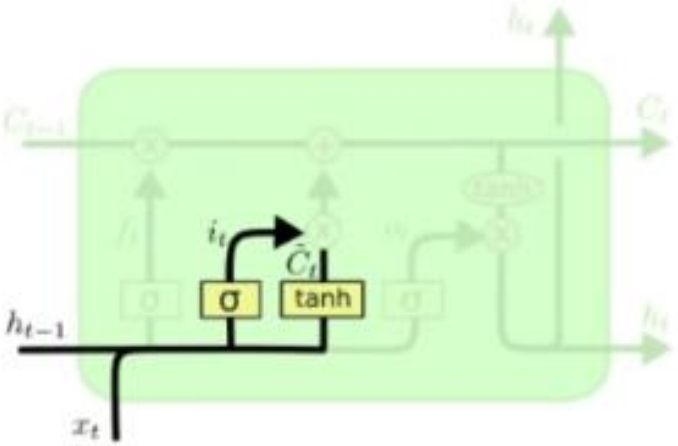



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

```
>>> Ĉt = np.tanh(np.dot(x_h_prev, Wc) + bc)
>>> Ĉt
array([-0.17806474, -0.99993564,  0.99164565,  0.92774236,
-0.99527522])
>>> Ĉt.shape
(5,)
```

# LSTM : Input Gate

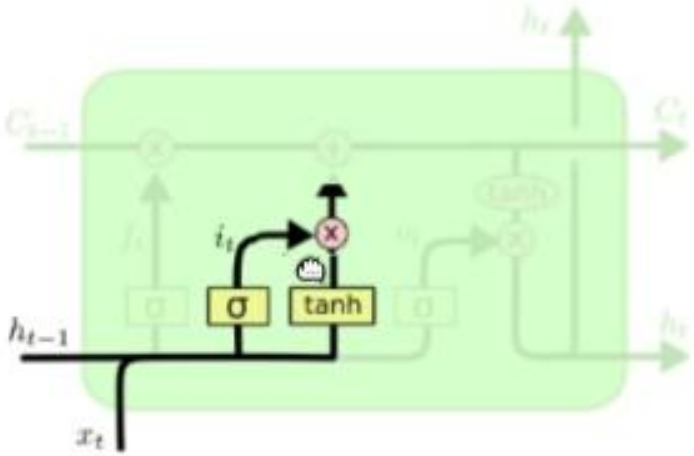



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

```
>>> it = sigmoid(np.dot(x_h_prev, Wi) + bi)
>>> it
array([0.00798643, 0.92300084, 0.22905397, 0.27818745,
0.96195338])
>>> it.shape
(5,)
```

# LSTM : Input Gate

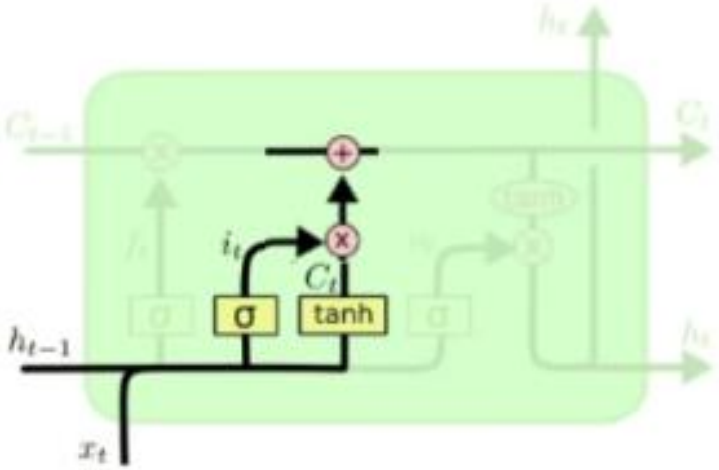



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

```
>>> new_c = it*Ĉt
>>> new_c
array([-0.0014221 , -0.92294144,  0.22714037,  0.25808628,
-0.95740836])
>>> new_c.shape
(5,)
```

# LSTM : Input Gate

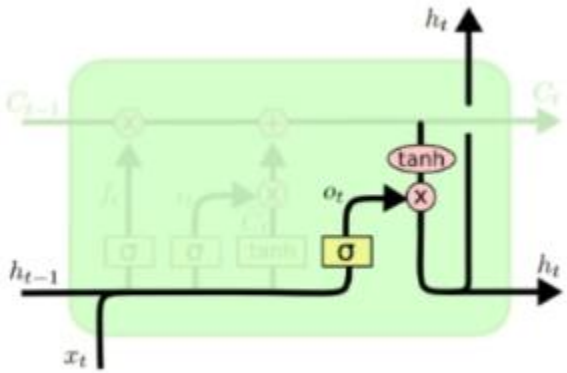



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

```
>>> c = c_prev*ft + it*Ĉt
>>> c.shape
(5,)
>>> c
```

# LSTM : Output Gate



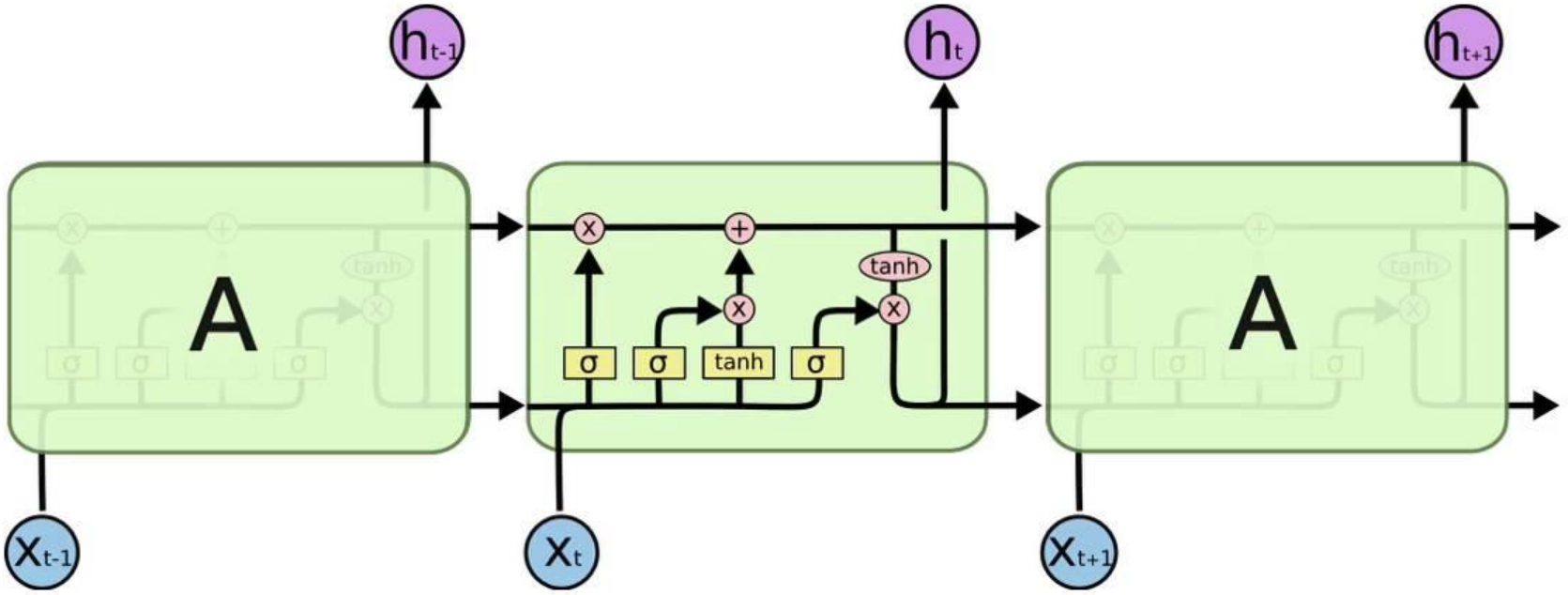

$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

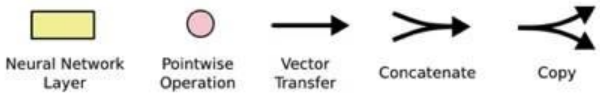
```
>>> ot = sigmoid(np.dot(x_h_prev, Wo) + bo)
>>> ot.shape
(5,)
>>> h = ot * np.tanh(c)
>>> h.shape
(5,)
```

# LSTM : Overview

**Long-Short Term Memory module: LSTM**



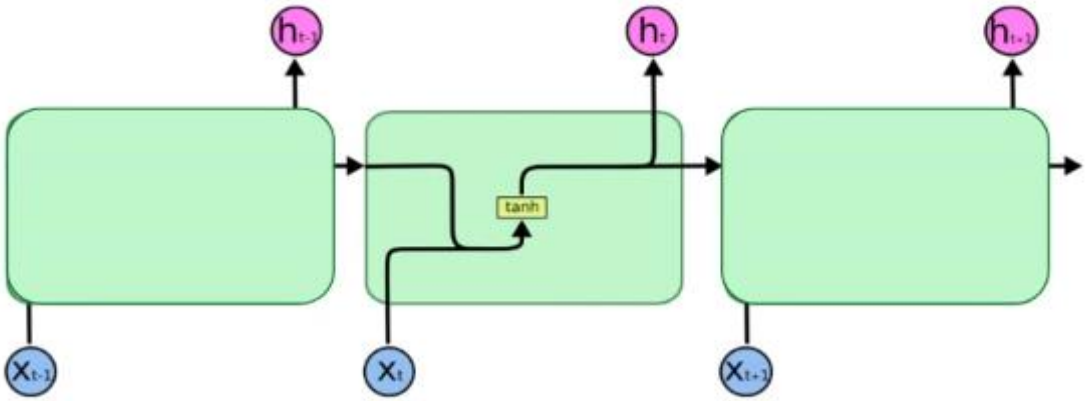

long-short term memory modules used in an RNN



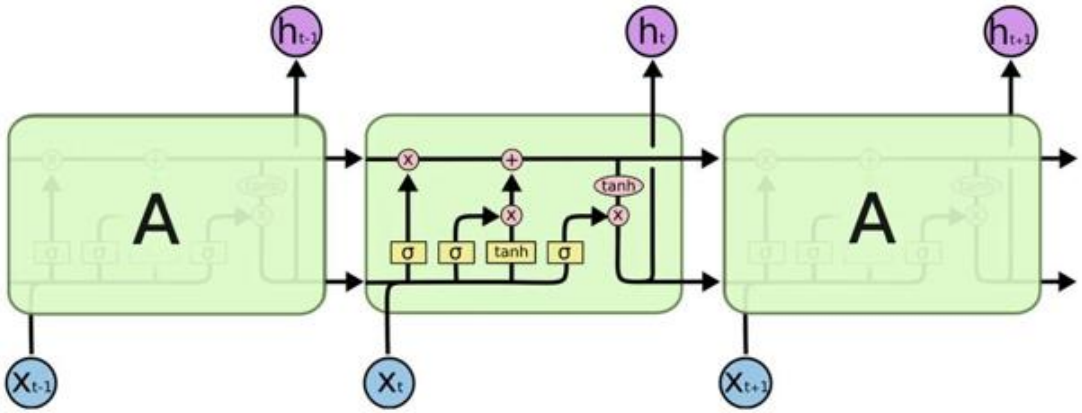

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

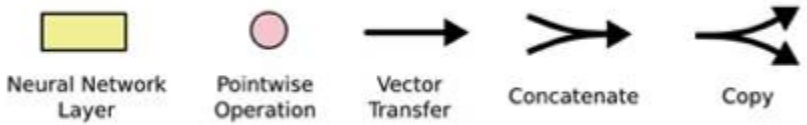
# RNN vs. LSTM

**RNN**




**LSTM**



# Conclusion

- **RNN :** consider temporal information
- Vanish gradient problem
- **LSTM :** select only pertinent information
- LSTM : partial solution of Vanish problem
- RNN networks : **high intensive** in computation & **Vanish Gradient**
- **Transformers :** new DNN architecture for **sequence** & **low intensive**

# RNN use cas applications

**Energy**

- Energy consumption prediction
- Electrical daily price prediction
- Energy generation prediction


**Industry 4.0 :**

- Predictive maintenance, process quality control, etc.

**Computer vision and video surveillance :**

- Actions recognition, security, etc.

**Text Analysis :**

- Text classification, text generation, etc.

**Etc.**

# Questions ?

# Thank you