



# Deep Learning

## Analytical Index

1. Introduction to Deep Learning
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)
  - a. Introduction to Sequence Modelling
  - b. Simple RNN
  - c. Gated RNN
  - d. Seq to Seq & Attention
4. Natural Language Processing with Deep Learning
5. Deep Generative Modelling



# 3

## Recurrent Neural Networks (RNN)

# 3.1

## Introduction to Sequence Modelling

# Sequences

- Most of machine learning algorithms are designed for independent, unordered data.
- Many real problems uses sequential data:  
 $\mathbf{x}^{(t)} : \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(T)}\}$ 
  - Time series, biomedical signals, audio signals, stock prices, text...
  - Does not have to be time, can be spatial measure (images), or any order measure (Recommender systems)
- The sequences are a natural way of representing reality: vision, hearing, action-reaction, words, sentences, etc.
- Don't forget order matters!!



# Sequential Processing

Speech recognition:



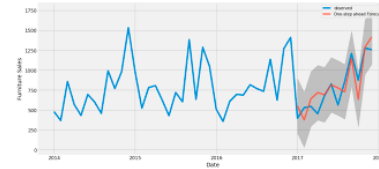
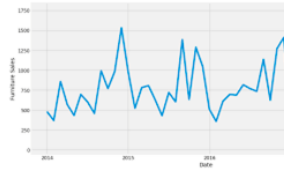
*Deep learning course D  
e e p l e a....*

Sentiment analysis:

*“Really good book...”*



Time series forecasting:



Machine translation

*“Really good book...”*

*“Muy buen libro...”*

Music generation:



# Problems With Dense Networks

- Inputs and outputs can be of different lengths:
  - Can try windowing.

*"I went to Greece on an end-of-term trip, ..., in 2020"*

- No parameter sharing:
  - Try convNets.

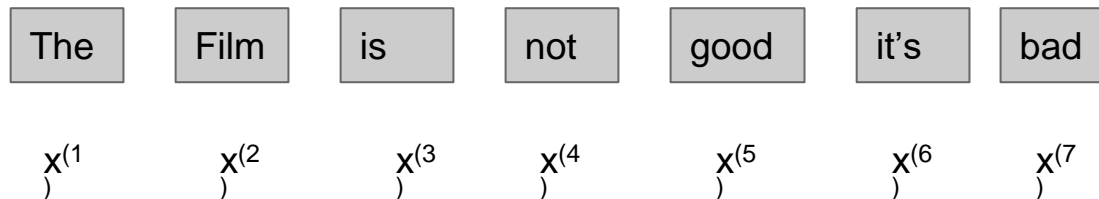
*"I went to Greece on an end-of-term trip, ..., in 2020"*  
*"In 2020, I went to Greece on an end-of-term trip, ..., "*

- Long Term dependencies (memory):

*"I went to Greece on an end-of-term trip, ..., in 2020"*  
*"In 2020, I went to Greece on an end-of-term trip, ..., "*

# Sequence Notation

*“The film is not good it’s bad”*



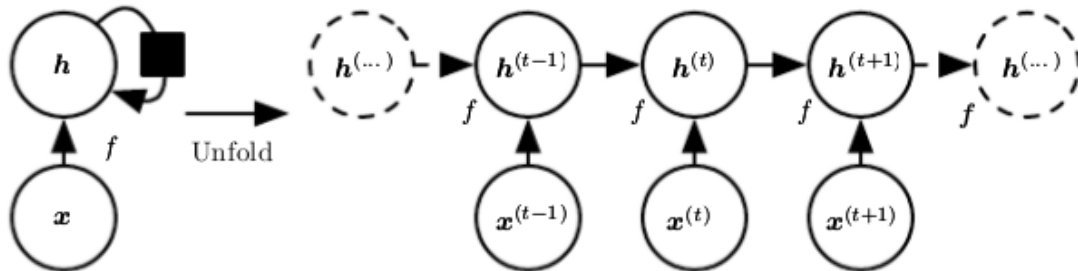
$$Sentiment = f(x^{(1)}, x^{(2)}, \dots, x^{(7)})$$

Order is important: *“The film is good it’s not bad”* (same words different order)



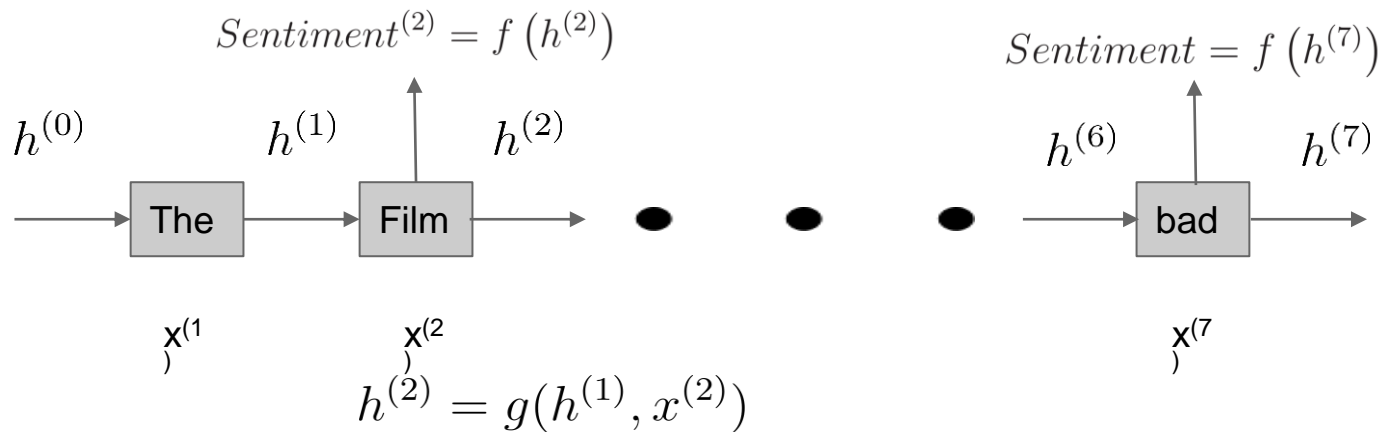
# Recurrent relations

- Recurrent function:  $x^{(t)} = f\left(x^{(t-1)}\right) = f\left(f\left(x^{(t-2)}\right)\right) = \dots = g\left(x^{(t-1)}, \dots, x^{(0)}\right)$
- If we want to predict future from past:  
$$\mathbf{h}^{(t)} = g^{(t)}(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)})$$
$$= f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta})$$
  - Repetitive structure
  - Don't need to learn g.
  - Same transition function f with the same parameters at every time step



# Recurrence Intuition

*“The film is not good it’s bad”*



- The state (memory) of every step is updated with the previous state and the actual input.

$$h^{(t)} = g(h^{(t-1)}, x^{(t)})$$

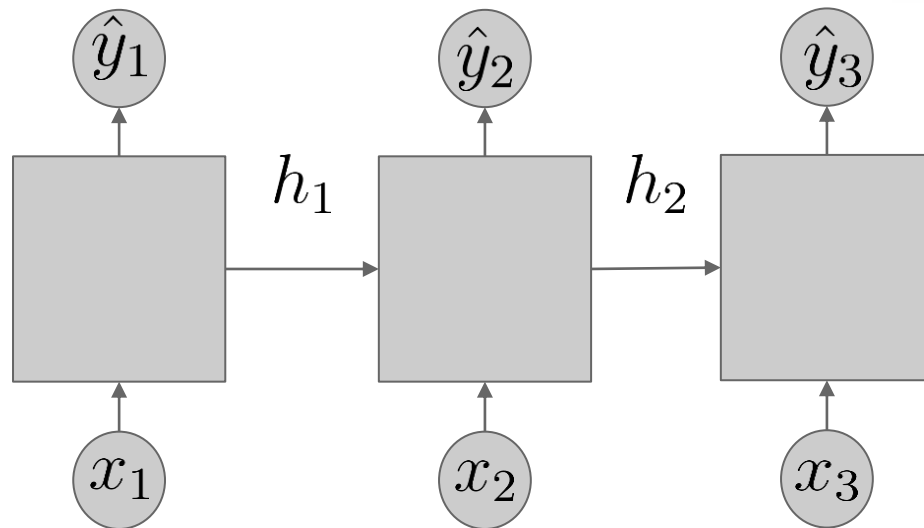
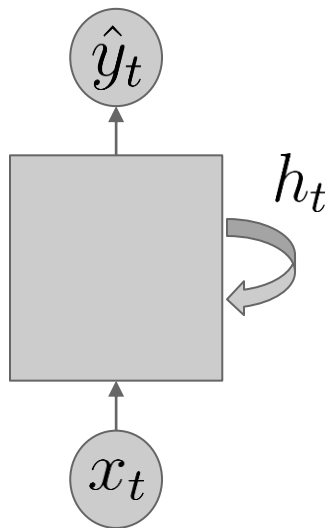
- The sentiment is a function of the state.

$$Sentiment^{(t)} = f(h^{(t)})$$

## 3.2

### Simple RNN

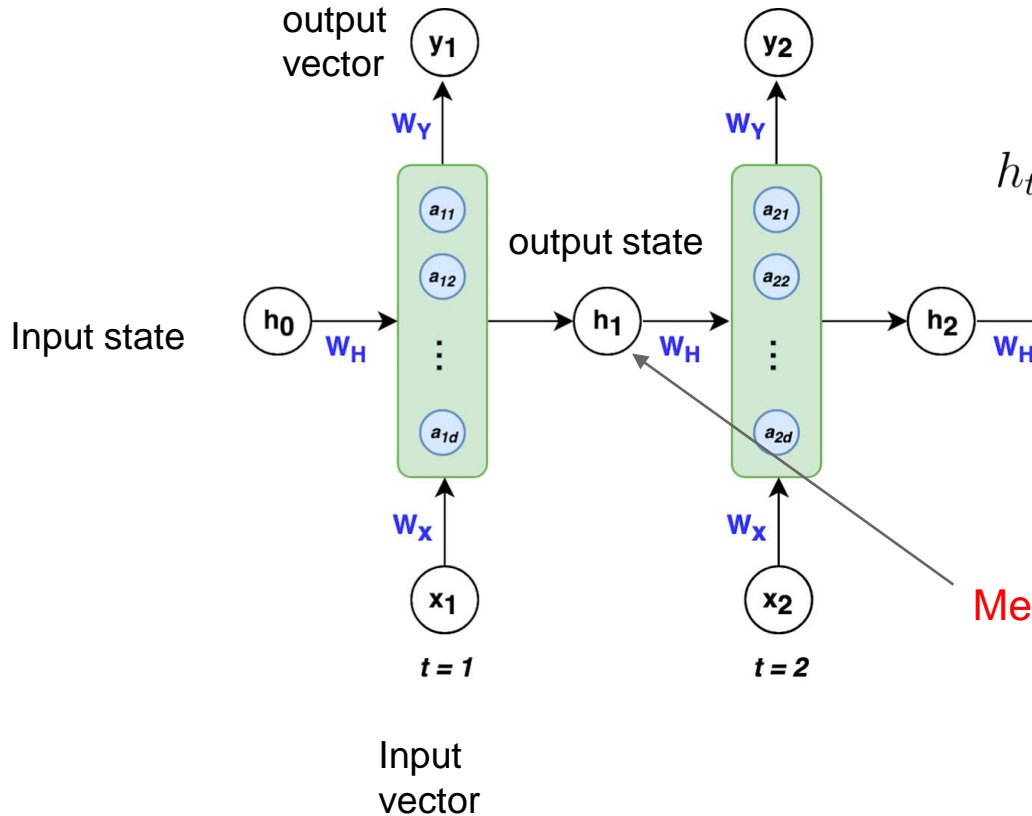
# RNN Intuition



$$\hat{y}_t = f(h_{t-1}, x_t)$$

Output      Previous state      input

# Simple RNN



Update hidden state:

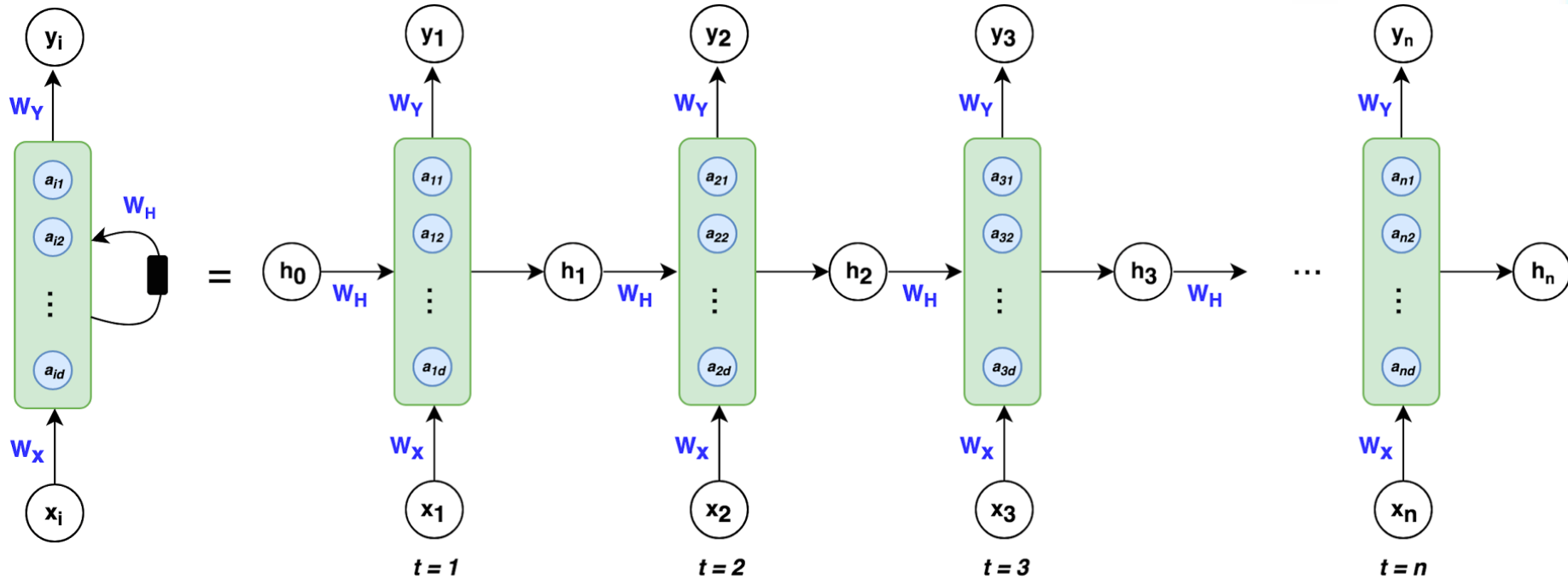
$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$

Output:

$$y_t = g(W_{yh}h_t + b_y)$$

# Simple RNN

$$y_t = g(W_{yh}h_t + b_y)$$



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

## Example: Language Model

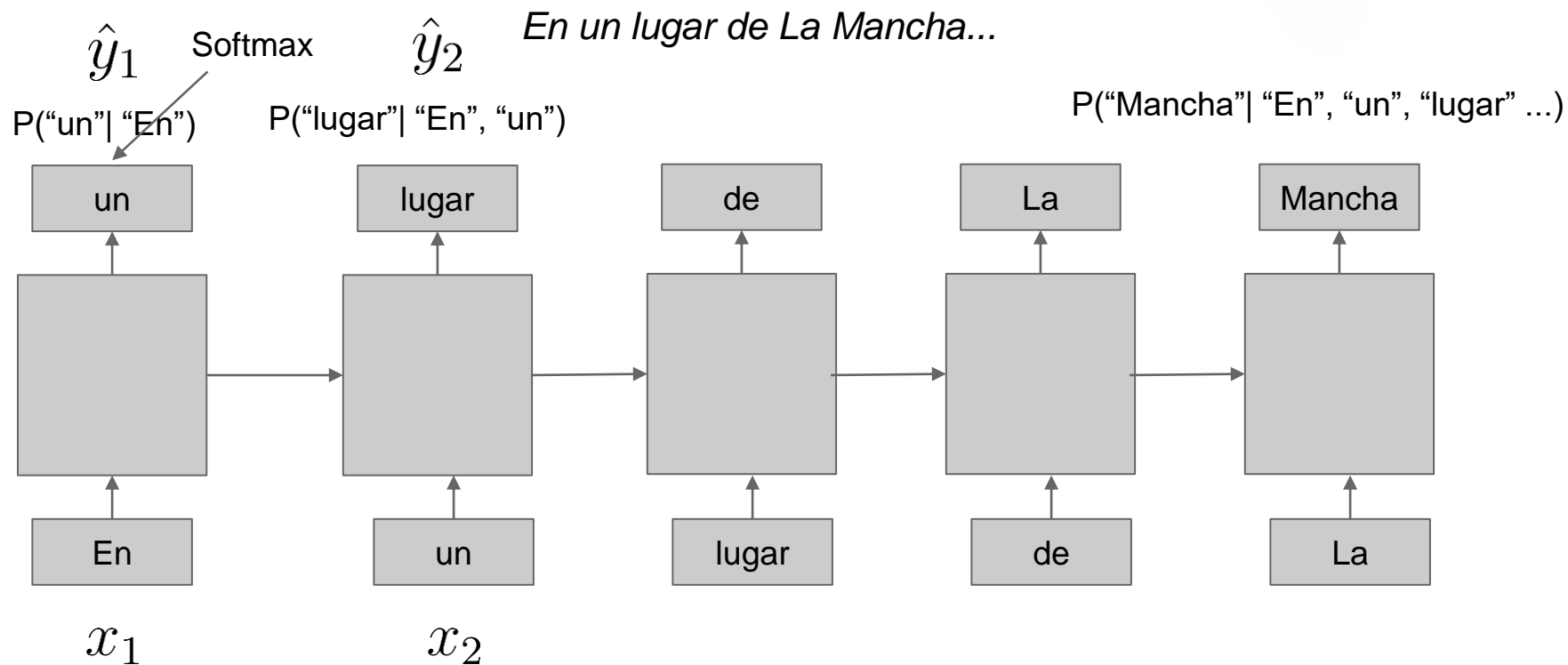
Sentence 1: “La noche larga”  $\Rightarrow P(\text{“La noche larga”}) = 0.001$

Sentence 2: “La coche larga”  $\Rightarrow P(\text{“La coche larga”}) = 0.0000000001$

Model:  $P(\text{“La noche larga”}) = P(\text{“La”})P(\text{“noche”} \mid \text{“La”})P(\text{“larga”} \mid \text{“La”, “noche”})$

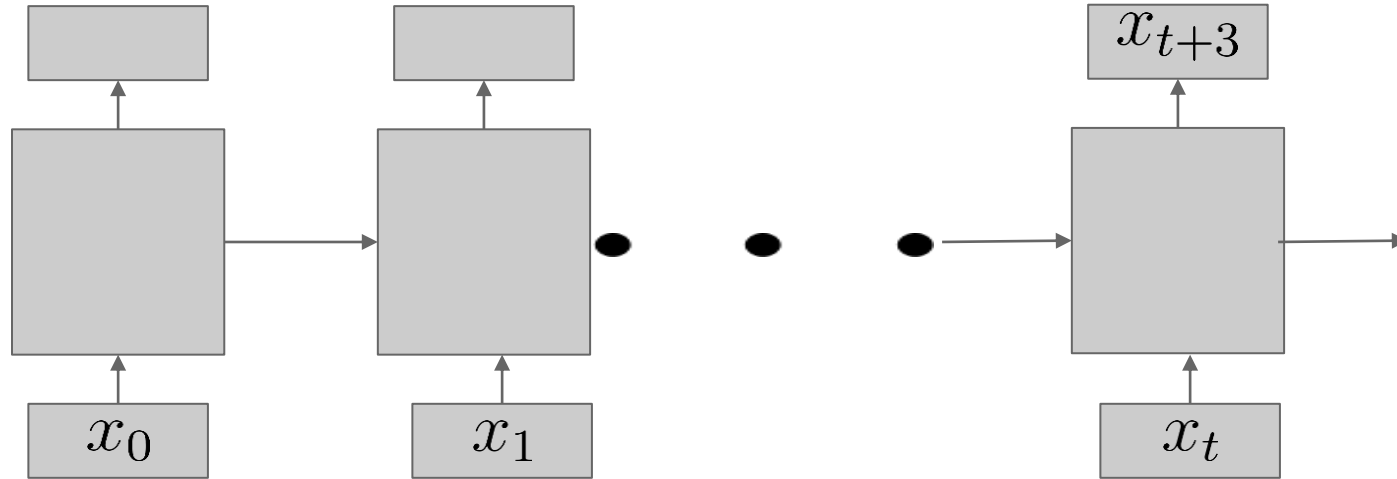
$$p(w_0, w_1, \dots, w_n) = p(w_0)p(w_1|w_0)p(w_2|w_0, w_1) \cdots p(w_n|w_0, w_1, \dots, w_{n-1})$$

## Example: Predicting Next Word



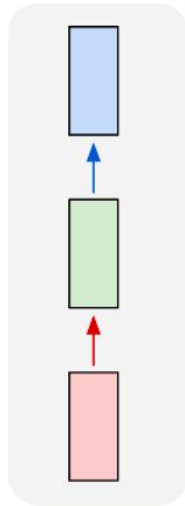


## Example: Time Series Forecasting



# Vanilla RNN

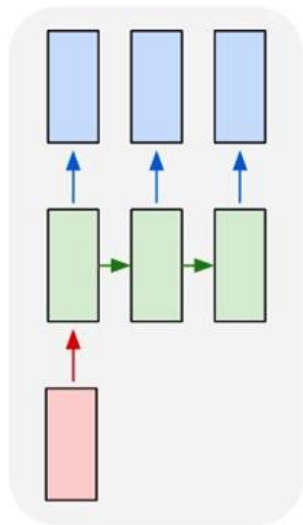
one to one



- Simple dense net

# Simple RNN

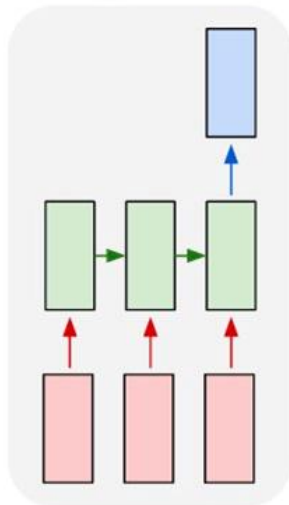
one to many



- Music generation
- Text generation
- Image captioning

# Simple RNN

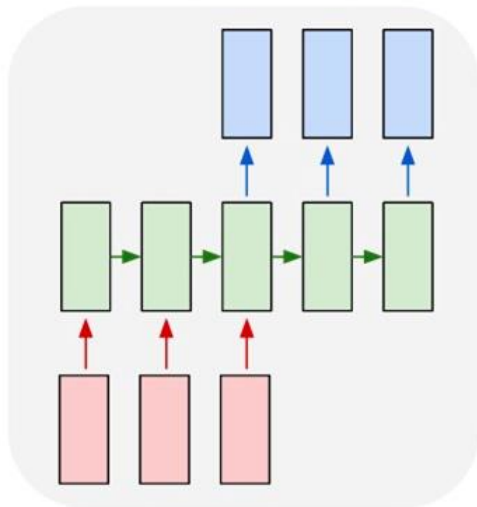
many to one



- Sentiment analysis.

# Simple RNN

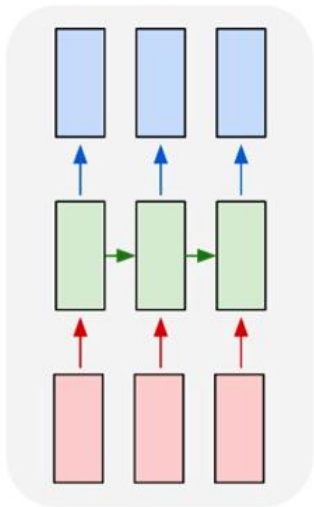
many to many



- Machine translation

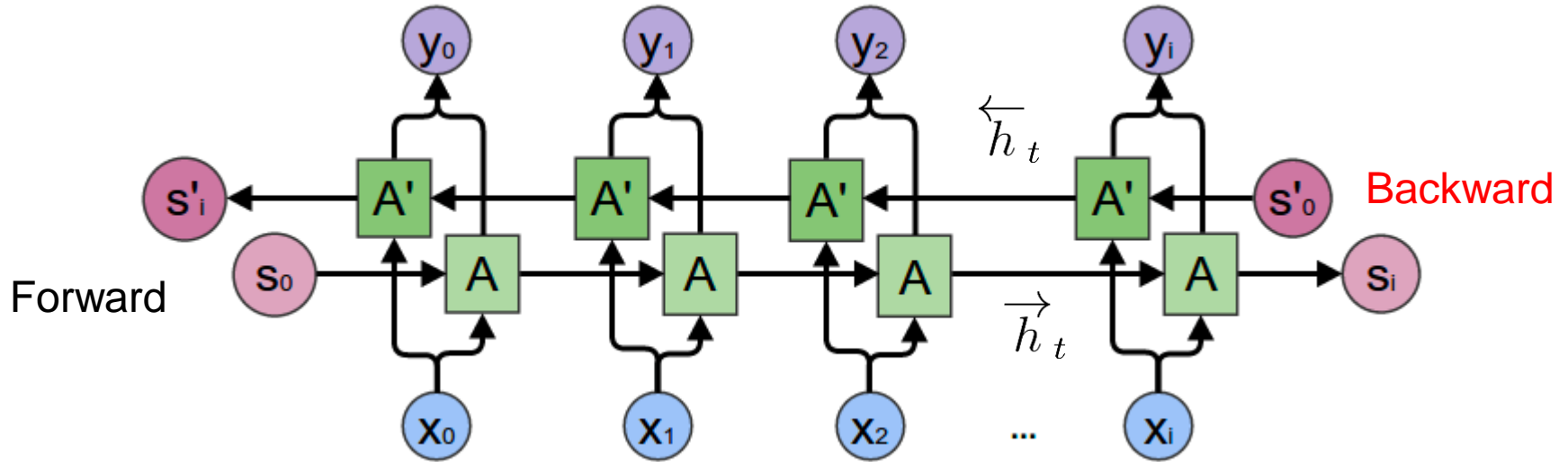
# Simple RNN

many to many



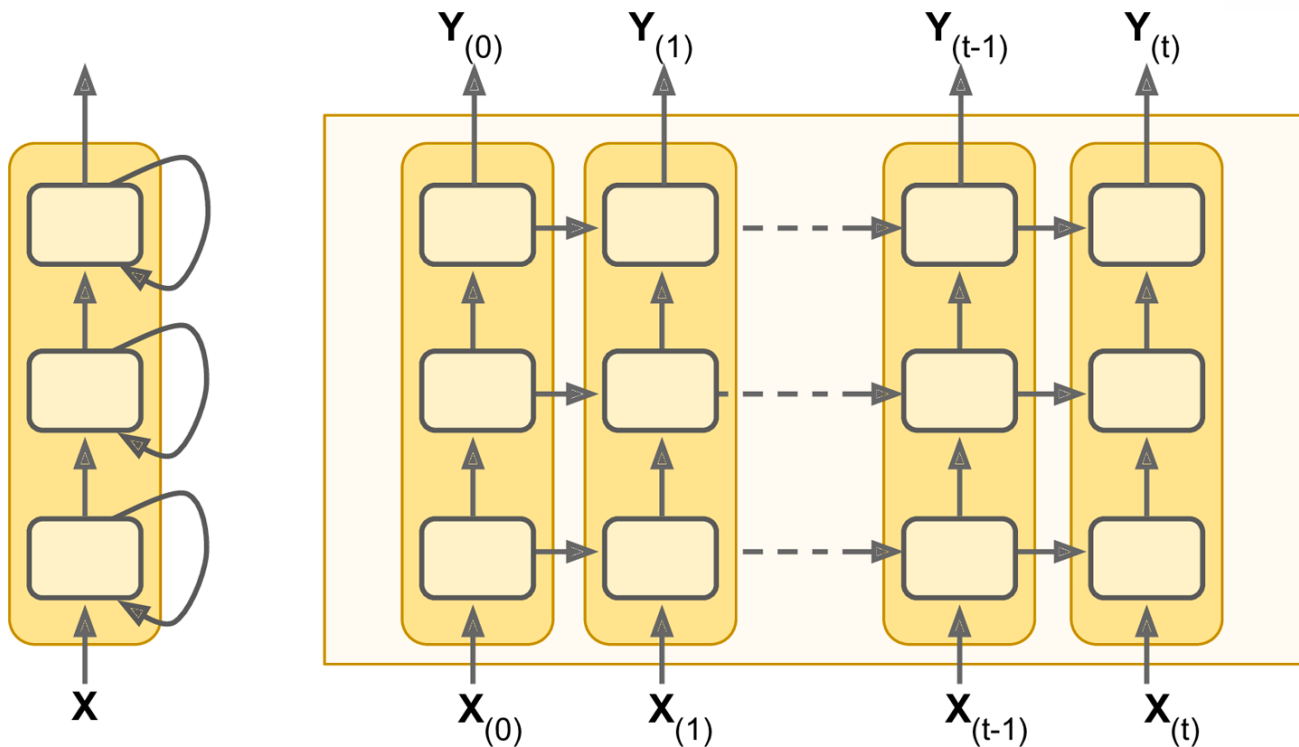
- Video classification

## Bidirectional RNN



I wanted \_\_\_\_\_ to call me. Mario never called. => Him or Her ??

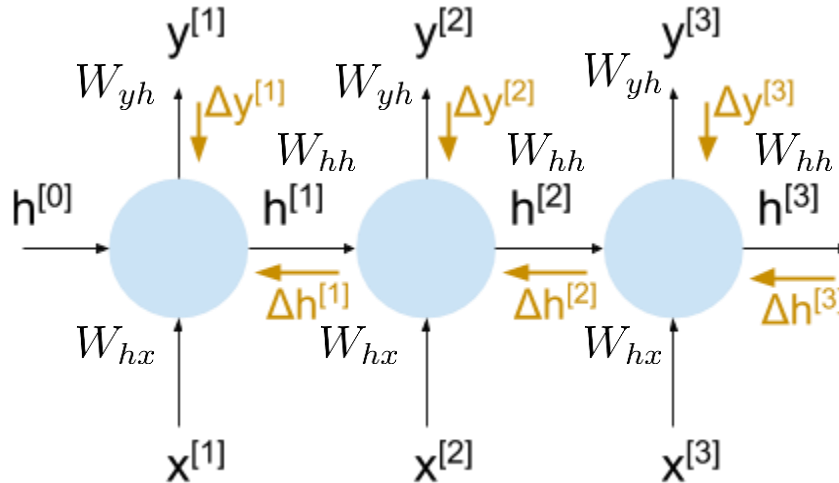
# Deep RNN



Stack multiple recurrent layers for learning more complex functions



# Backpropagation Through Time BTT

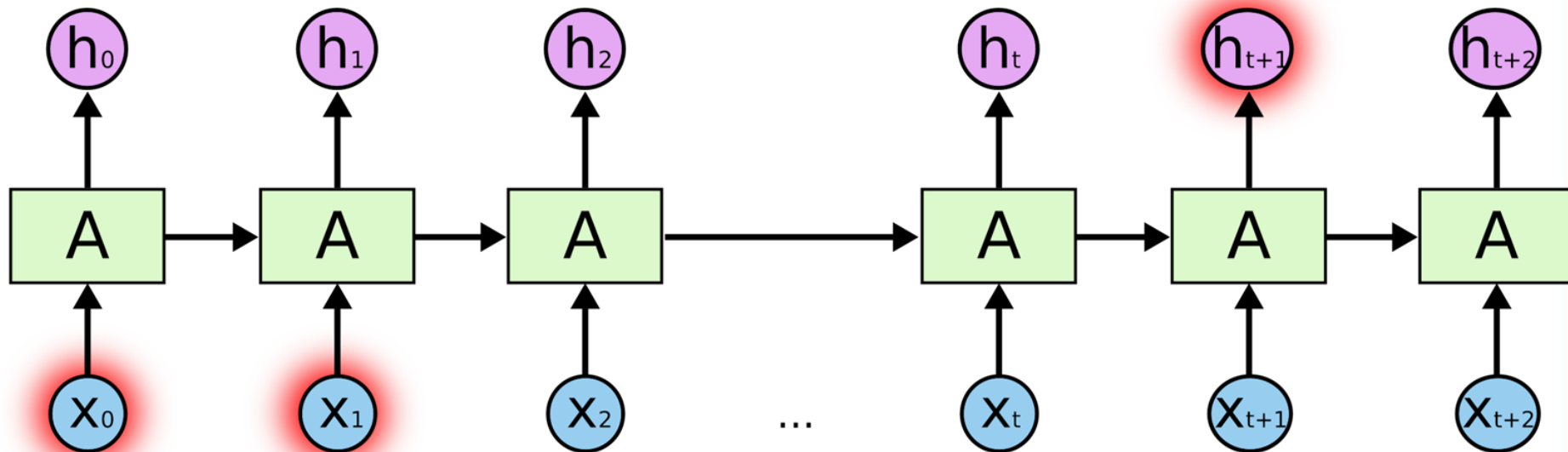


- Backpropagating the error in time involves as many recurrent derivation terms as timesteps on the net.
- Problems with gradients:
  - Vanishing
  - Exploding

# 3.3

## Gated RNN

## Simple RNN Problems: Long-Term Dependencies



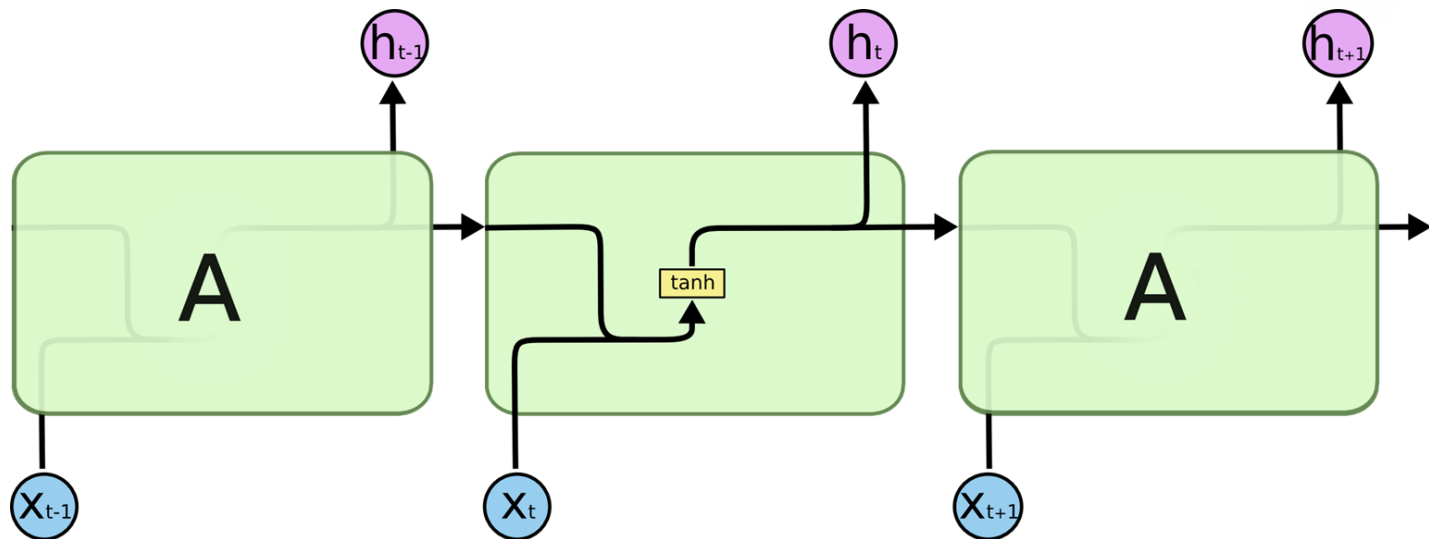
# RNN Advances

**Long-Term Dependencies:** Use more complex recurrent cells for control information flow.

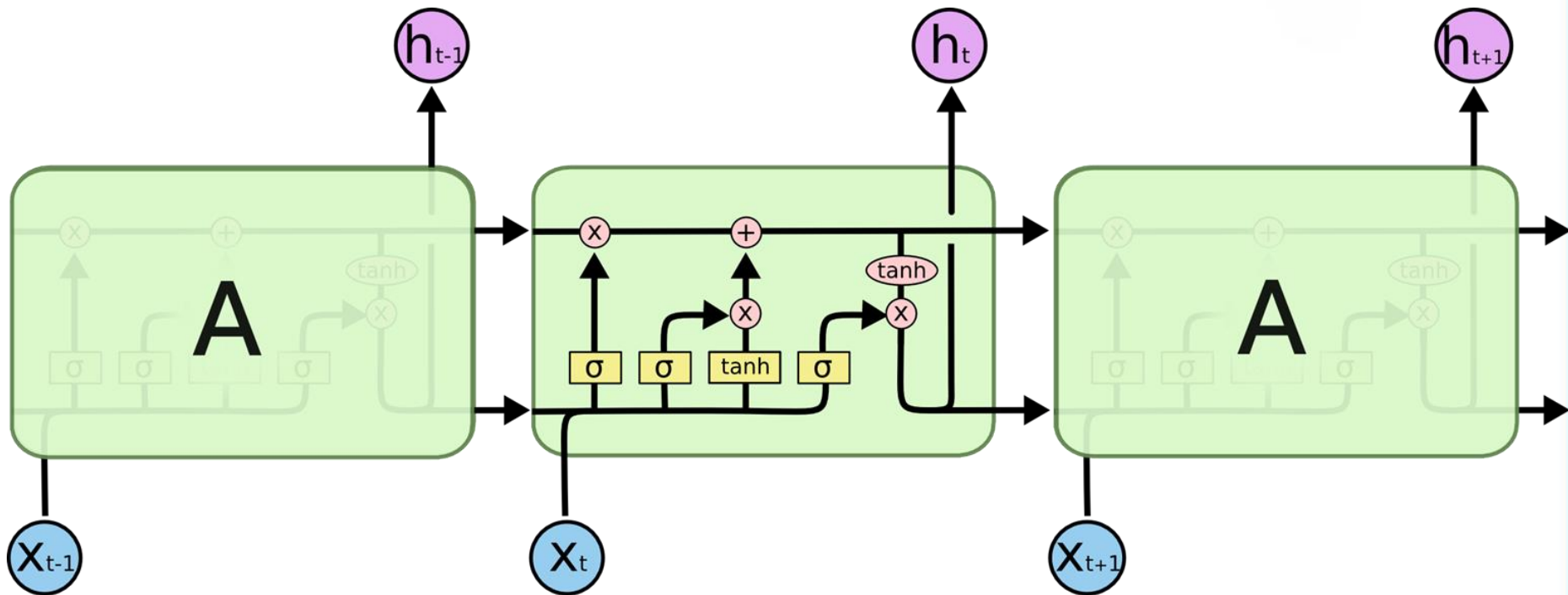
1. GRU: Gated recurrent unit
1. LSTM: Long short-term memory

```
model1.add(layers.SimpleRNN(64))  
model2.add(layers.GRU(64))  
model3.add(layers.LSTM(64))
```

# Simple RNN

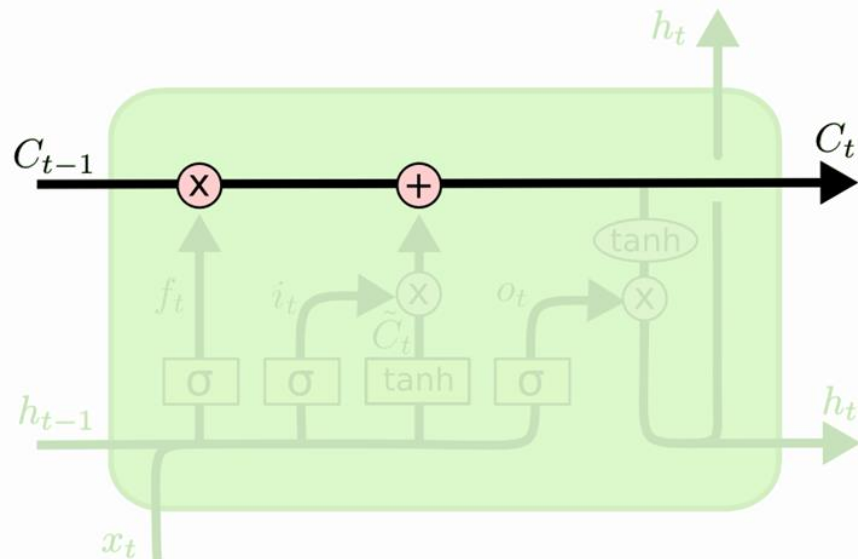


# LSTM: Long Short-Term Memory



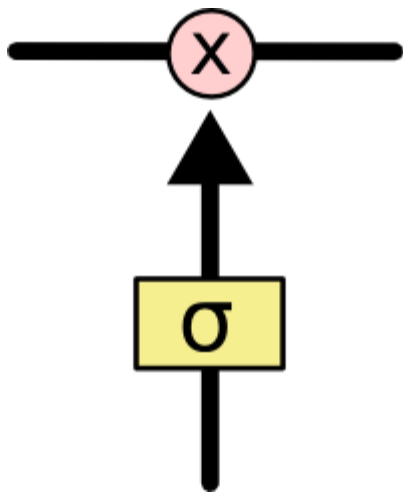
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior

# LSTM: Long Short-Term Memory



The LSTM can remove or add information to the cell state

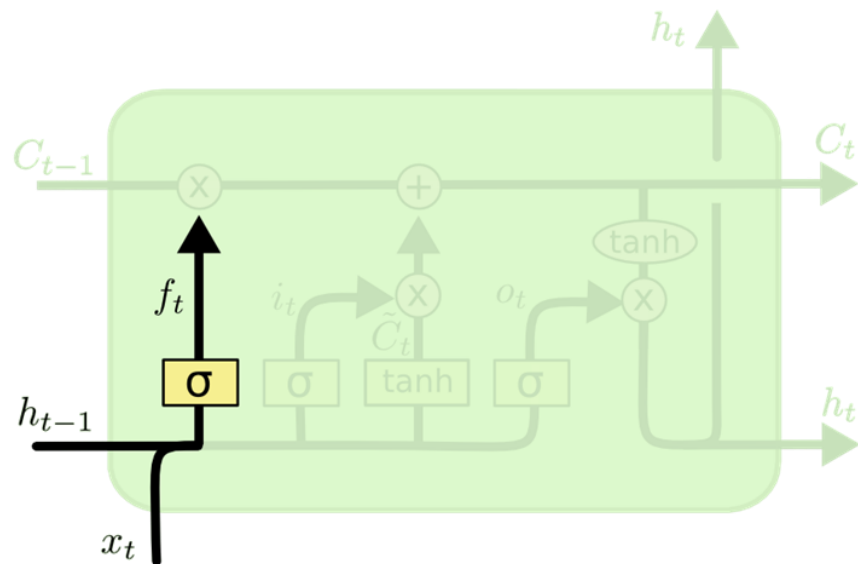
# LSTM: Information Flow



- Output of a sigmoid  $[0,1]$
- Information is added or removed with pointwise multiplication => **Gates**
- Steps:
  - 1. Forget
  - 2. Store
  - 3. Update
  - 4. Output

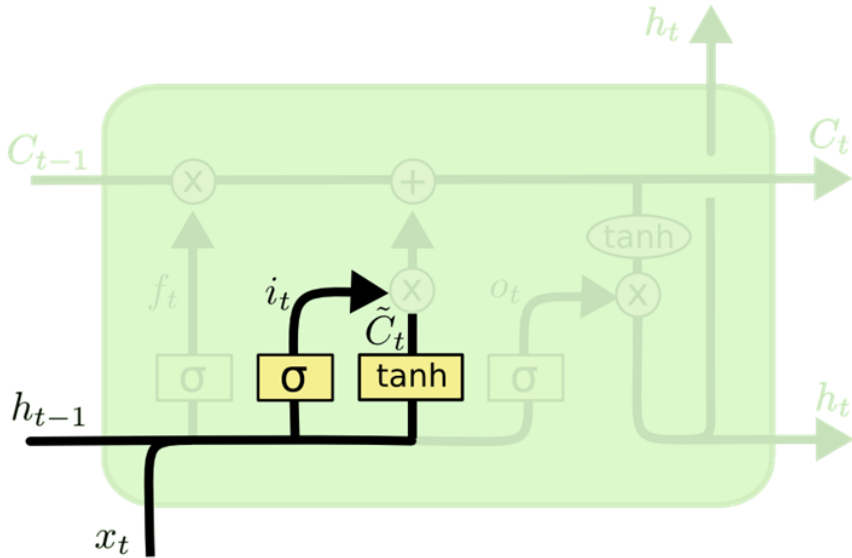


# LSTM: 1. Forget Gate Layer



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

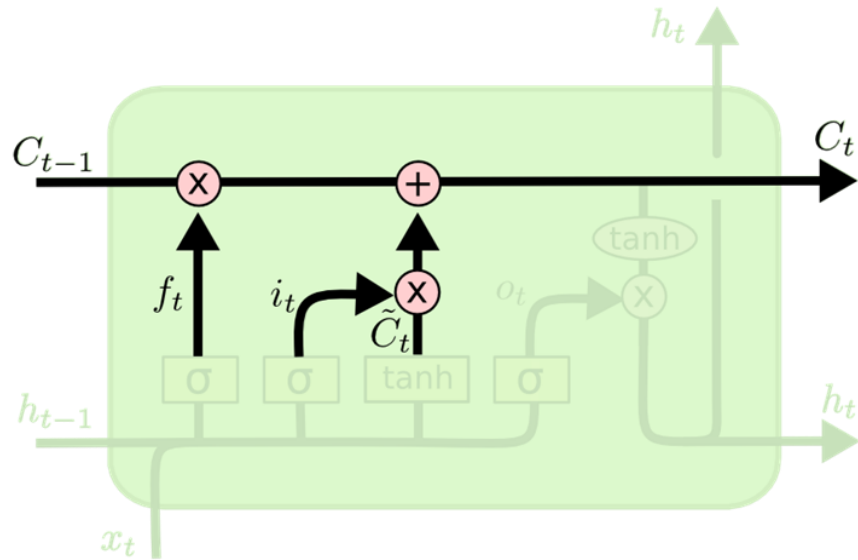
## LSTM: 2. Store



Store relevant information

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

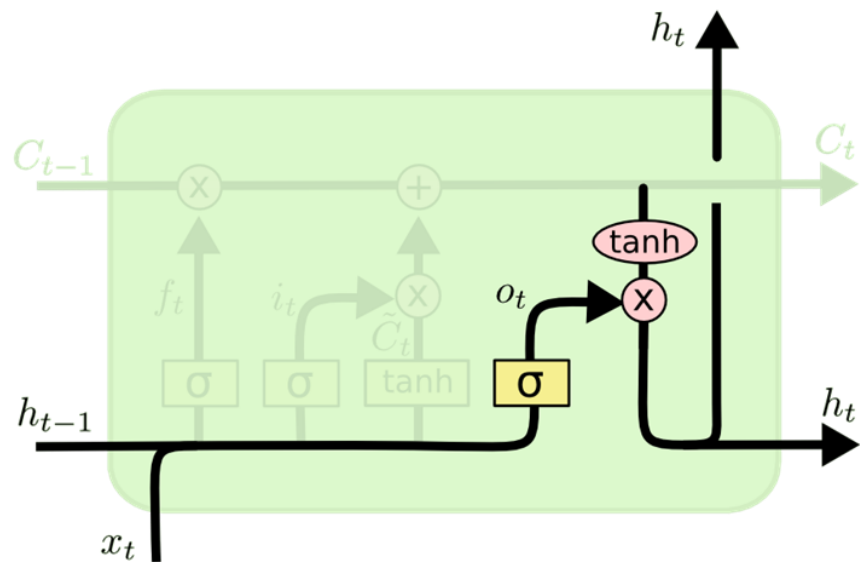
## LSTM: 3. Update



Update cell state

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

## LSTM: 4. Output

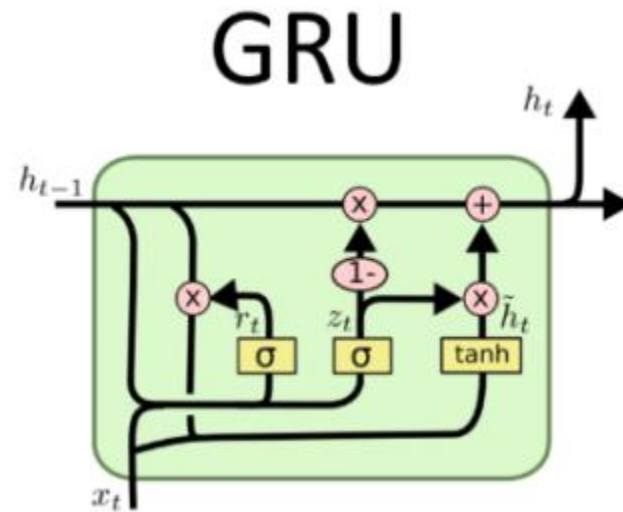
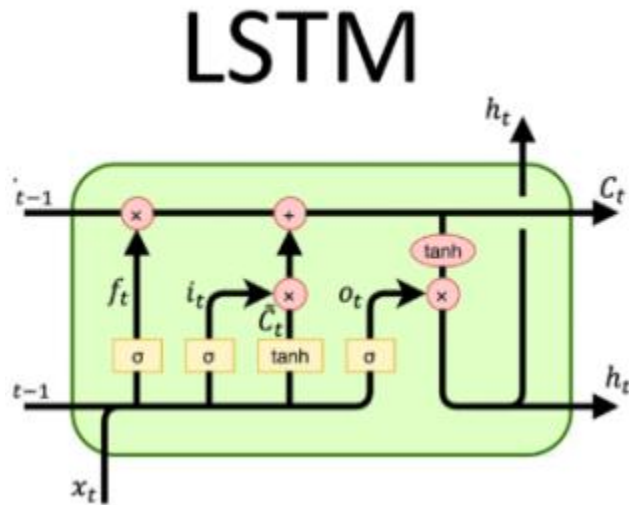


Control the relevant information

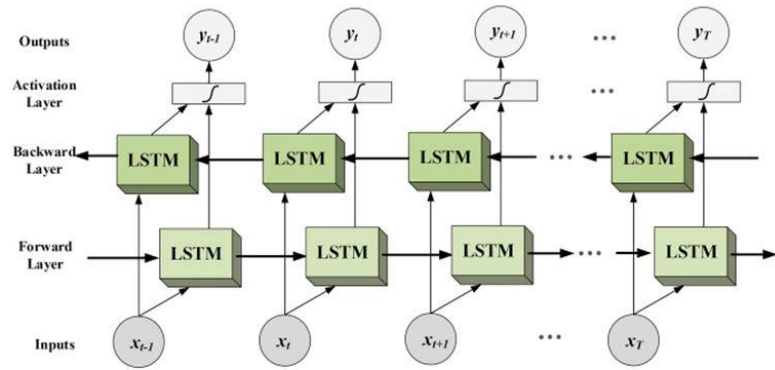
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

# GRU: Gated Recurrent Unit

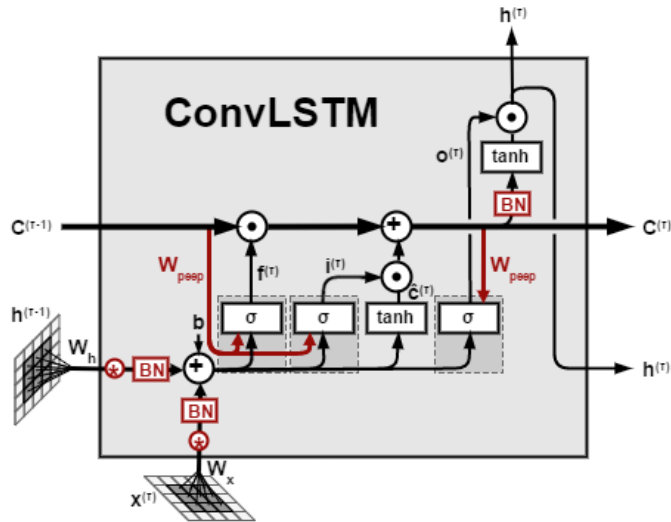


# Bi-LSTM



- Video classification

# Convolutions + LSTM



- Videos can be treated as a sequence of images as essentially, this is what they are. So, they are candidates to be treated sequentially.
- One possible approach is using **ConvLSTM** layers. It is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations. As a result, the data that flows through the ConvLSTM cells keeps the input dimension (3D in our case) instead of being just a 1D vector with features.
- It is very common to misunderstand ConvLSTM layers/networks with **Convolutional-LSTM** models, in which the image passes through the convolutional layers and its result is a set flattened to a 1D array with the obtained features. When repeating this process to all images in the time set, the result is a set of features over time, and this is the LSTM layer input.

# 3.4

## Use Cases

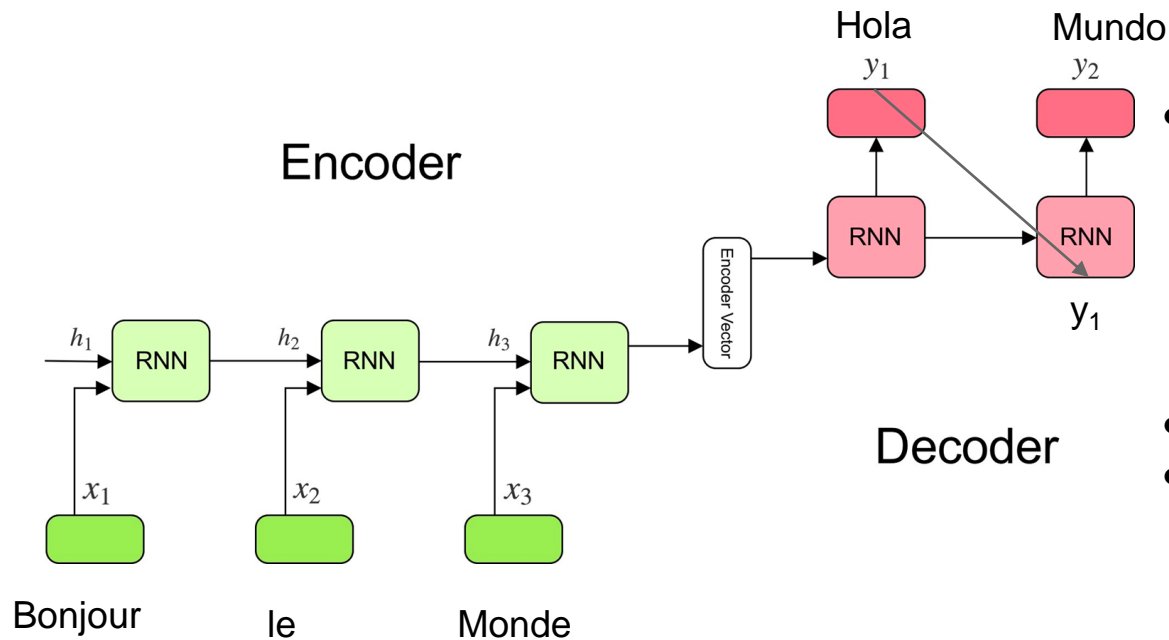




## 3.5

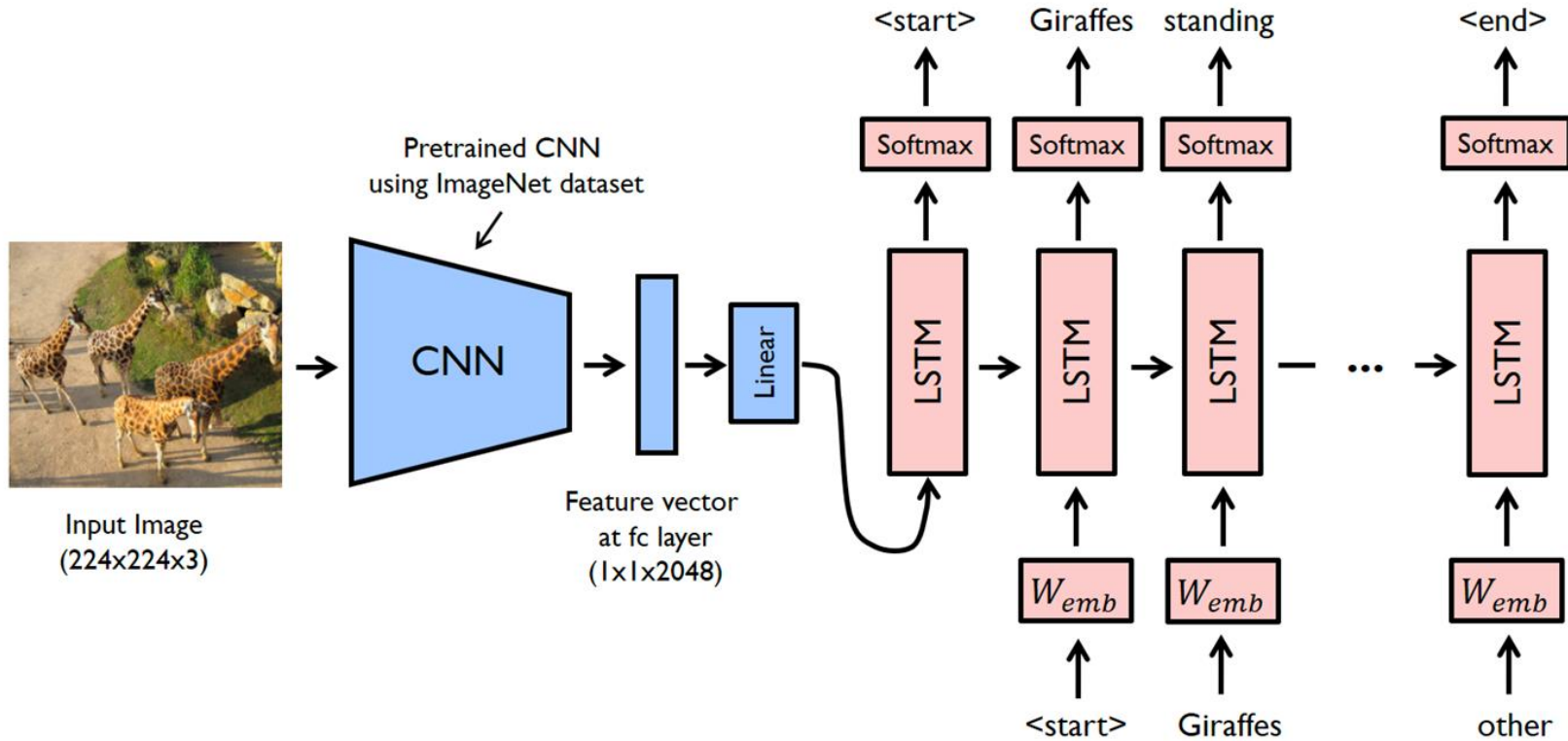
# Sequence to Sequence Model & Attention Mechanism

# Sequence to Sequence

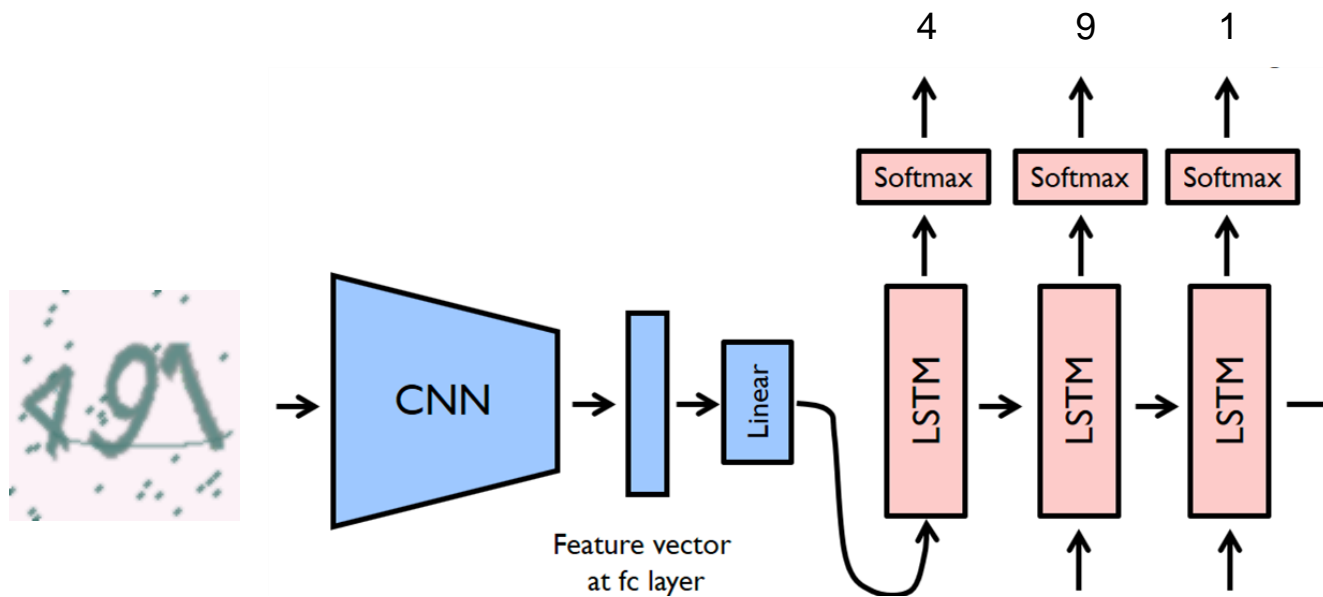


- Two sub-models:
  - **Encoder:** Encoding the entire sequence into a fixed length vector called a context vector.
  - **Decoder:** Reading from the context vector.
- Introduced by Google in **2014**.
- Maps variable-lengths sequences to a **fixed length** encoded vector.

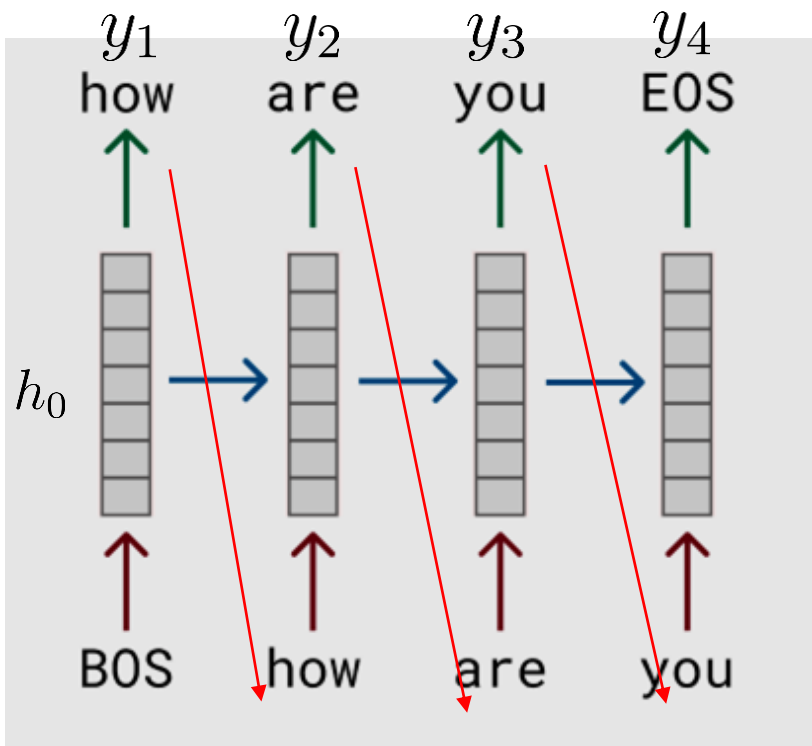
# Sequence to Sequence: Image Captioning



# Sequence to Sequence: Captcha Hacking



## Choose Output Sequence



Input:  $h_0 = \text{encoded}(\text{"Qué tal estás"})$

First output: Sample from softmax  
 $\hat{y}_1 = P(y_1|h_0) \rightarrow P(\text{"How"}|h_0)$

Second output: Sample from softmax  
 $\hat{y}_2 = P(y_2|h_0, y_1) \rightarrow P(\text{"are"}|\text{"How"}, h_0)$

Third output: Sample from softmax  
 $\hat{y}_3 = P(y_3|h_0, y_1, y_2) \rightarrow P(\text{"you"}|\text{"How"}, \text{"are"}, h_0)$

Fourth output: Sample from softmax  
 $\hat{y}_4 = P(y_4|h_0, y_1, y_2, y_3) \rightarrow P(\text{"EOS"}|\text{"How"}, \text{"are"}, \text{"you"}, h_0)$

# Choose Output Sequence

$$y^* = \operatorname{argmin} P(y_1, y_2, \dots, y_n | h_0) = P(y_1 | h_0) P(y_2 | y_1, h_0) \cdots P(y_n | h_0, y_1, y_2, \dots, y_{n-1})$$

***“El año pasado volví a ir a Roma con mis amigos”***

*“Last year I went back to Rome with my friends”*

*“Last year I went to Rome again with my friends”*

## Search Strategies

- Sampling from output softmax not always return the best output sentence.
- Greedy (choose the max probability from the softmax) doesn't work well.
- Need other search algorithms: **Beam search**

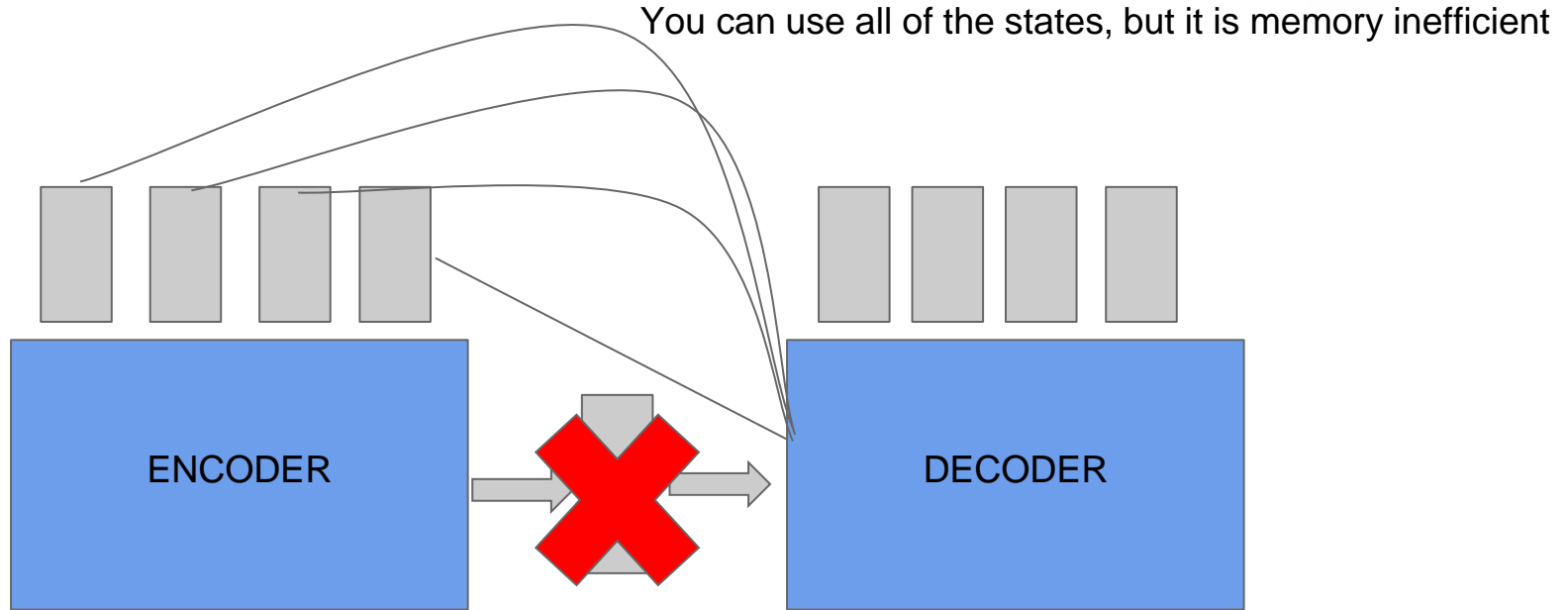
# Attention Mechanism

- The encoder processes the entire input sentence and encode it into a context vector. The decoder produces the words in a sentence one after another. Long-term dependency problems.
- Performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases.
- **Attention Mechanism:** Selectively concentrating on a few relevant things, while ignoring others. (Also provides some **interpretability**)
- *“Neural Machine Translation by Jointly Learning to Align and Translate” (2015)*



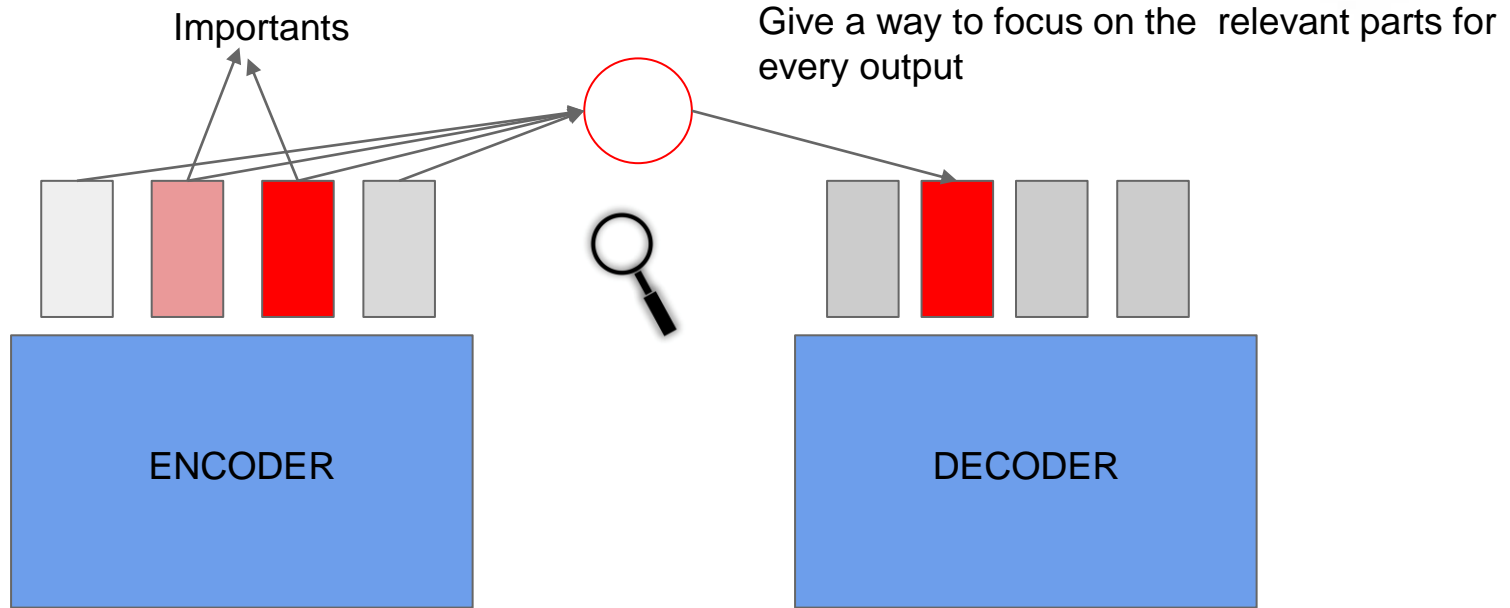


## Attention Mechanism: Intuition



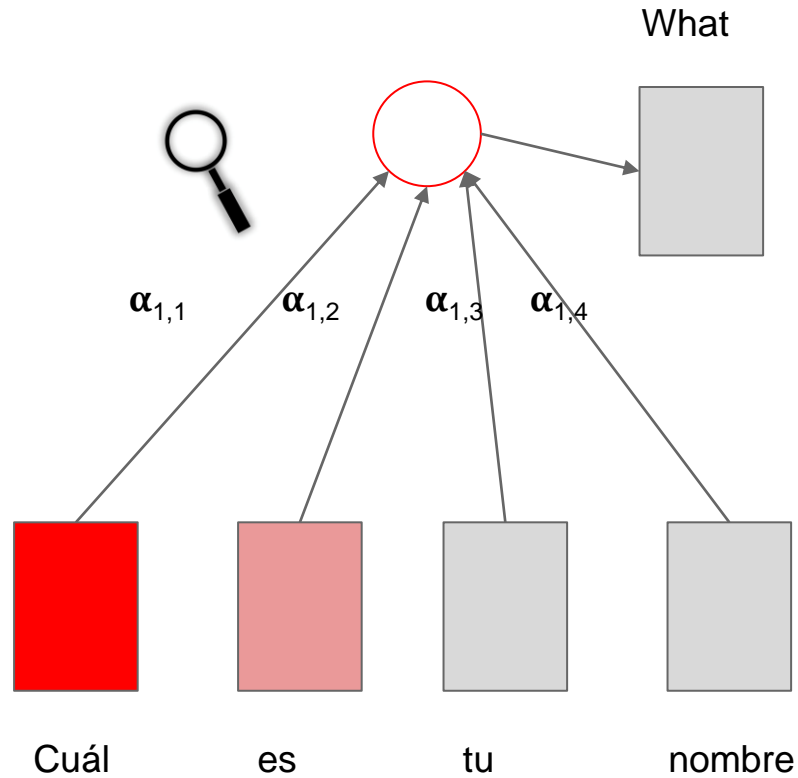
¿Cuál es tu nombre?

# Attention Mechanism: Intuition

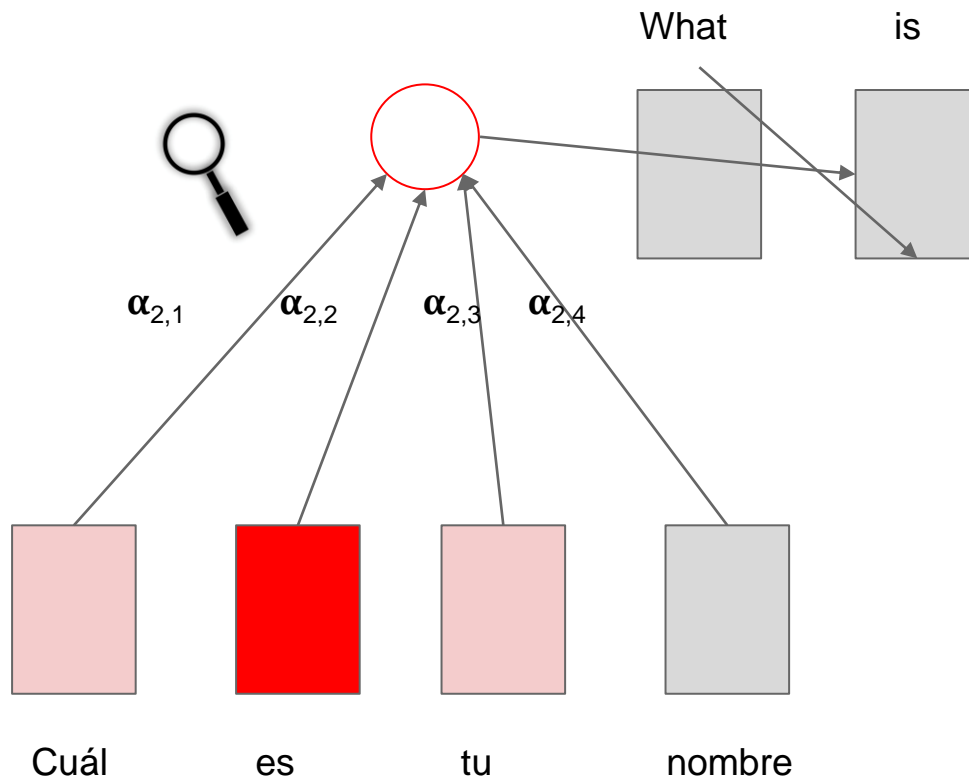


¿Cuál es tu nombre?

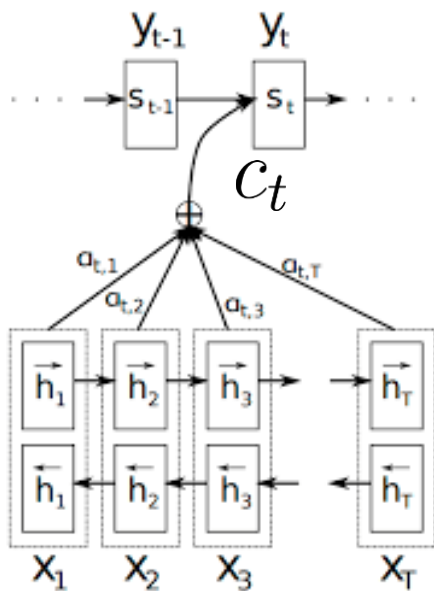
# Attention Mechanism: Intuition



# Attention Mechanism: Intuition



# Attention Mechanism



You don't encode all the input in the last state, you use all the hidden states for every output.

$$h_t = (\vec{h}_t, \overleftarrow{h}_t)$$

Focus in the relevant inputs with the context vector (input of  $y_{nn}$ ).  $a_{tt'}$  weights every state  $h_{t'}$

$$c_t = \sum_{t'} a_{tt'} h_{t'}$$

$a_{tt'}$  is the amount of attention  $y_t$  should pay to  $h_{t'}$ .

$$a_{tt'} = \frac{\exp(e_{tt'})}{\sum_{k=1} \exp(e_{tk})}$$

$$e_{tt'} = f(s_{t-1}, h_{t'})$$

Simple NN

# Attention Mechanism

