

# Aprendizaje Profundo

Facultad de Ingeniería  
Universidad de Buenos Aires



Profesores:

Marcos Maillot  
Antonio Zarauz  
Gerardo Vilcamiza

# Redes Recurrentes

## Recurrent Neural Network (RNN)

- . Introducción
- . Neurona recurrente básica
- . Implementación en pytorch (práctica 1)
- . Back propagation through time (BPTT) (básico)
- . Implementación (práctica - predecir serie temporal)
- . Arquitectura enconder-decoder (seq to seq)
- . Mecanismos de atención



Red neuronal **favorita** para el trabajo secuencias ( datos que en cuya naturaleza exista un **comportamiento secuencial**):

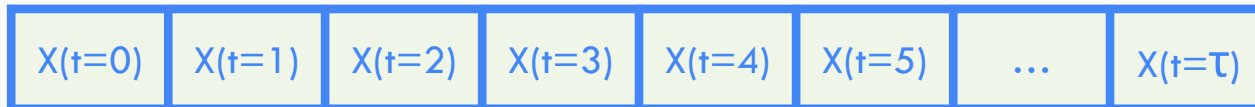
- señales temporales
- series temporales
- texto
- habla
- música
- etc

## Redes recurrentes - Introducción

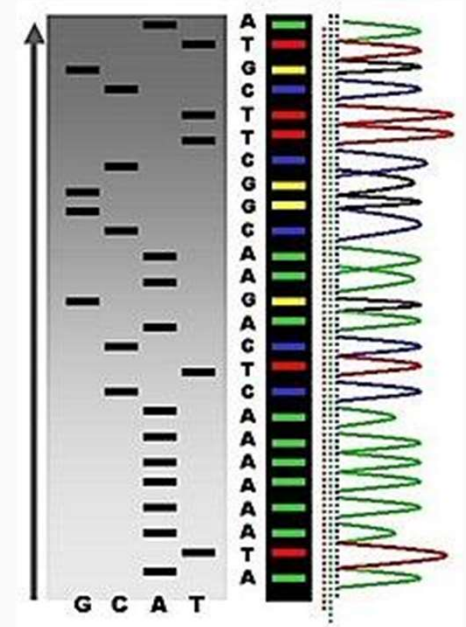
En **cada paso**, se repiten los **mismos cálculos**, empleando **datos del paso actual y datos del pasado**.

**Los pasos, no son necesariamente en unidad tiempo!!**

Temperatura  $f(t)$ :

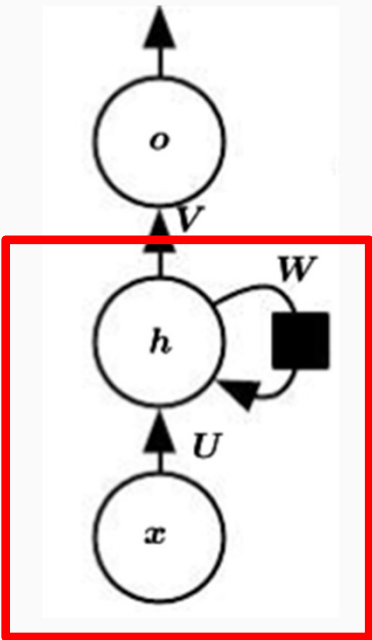


Mensaje:



wikipedia

## Redes recurrentes - Neurona recurrente básica



Ecuaciones

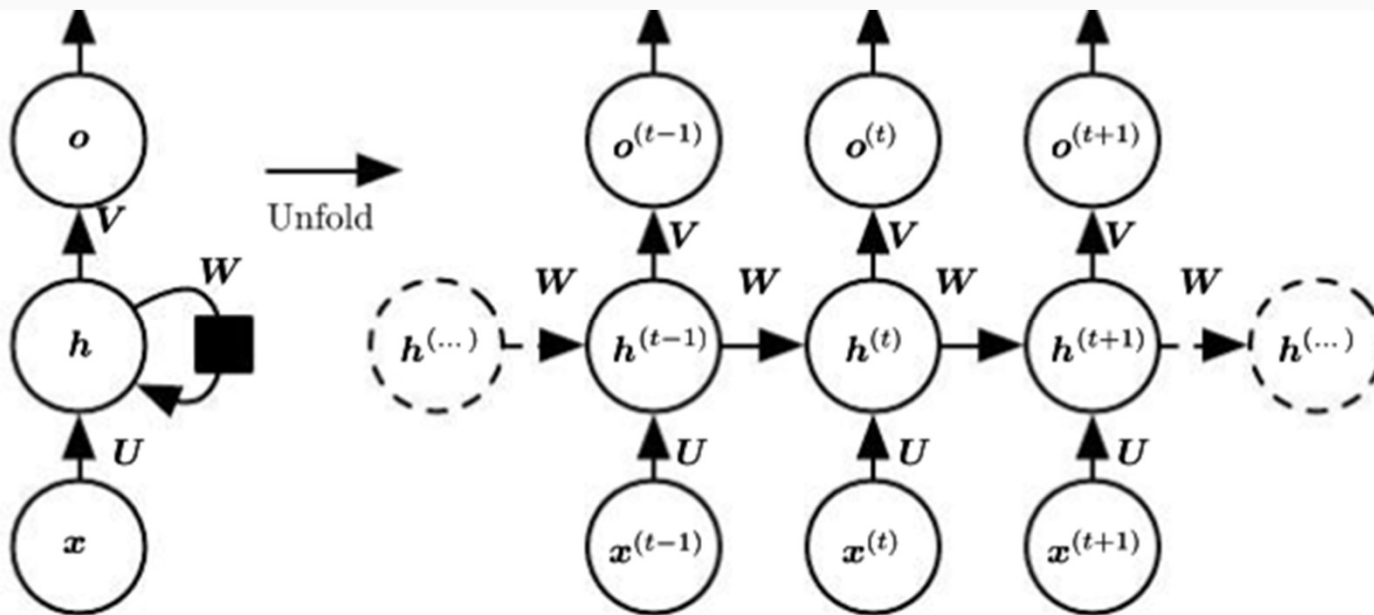
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

## Redes recurrentes - Neurona recurrente básica

$$\begin{aligned}a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)}, \\h^{(t)} &= \tanh(a^{(t)}), \\o^{(t)} &= c + Vh^{(t)},\end{aligned}$$



**$U, W$   
son los  
mismos!!**

Parameters sharing

## Redes recurrentes - Neurona recurrente básica

$$x[n] = [0,7; 0,9; 1,1]$$

$$\left\{ \begin{array}{l} U = 0,2 \\ W = 0,3 \end{array} ; b = -0,1 \right\} \text{ Parámetros inventados}$$

$$h[0] = \tanh[-0,3 + \underbrace{0,3 \cdot h[-1]}_{=0 \text{ para la 1ª muestra}} + 0,2 \cdot 0,7] = 0,0399$$

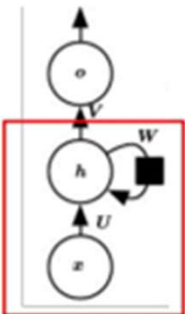
$$h[1] = \tanh[-0,3 + 0,3 \cdot 0,0399 + 0,2 \cdot 0,9] = 0,0917$$

$$h[2] = \tanh[-0,3 + 0,3 \cdot 0,0917 + 0,2 \cdot 1,1] = 0,1464$$

Valido para cada neurona hidden

Las hidden se conectan todas contra todas

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)}, \\ h^{(t)} &= \tanh(a^{(t)}), \\ o^{(t)} &= c + Vh^{(t)}, \end{aligned}$$



# Redes recurrentes - Implementación en pytorch

## RNN

CLASS `torch.nn.RNN(*args, **kwargs)` [\[SOURCE\]](#)

Applies a multi-layer Elman RNN with `tanh` or `ReLU` non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

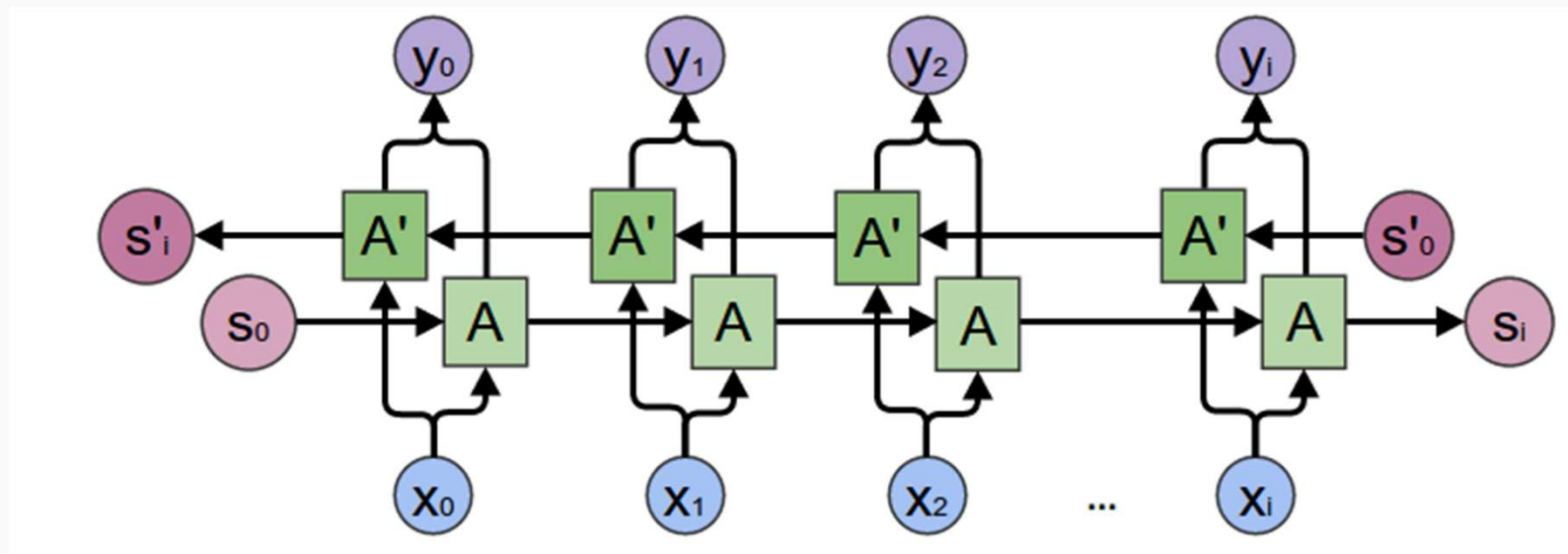
$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

where  $h_t$  is the hidden state at time  $t$ ,  $x_t$  is the input at time  $t$ , and  $h_{(t-1)}$  is the hidden state of the previous layer at time  $t-1$  or the initial hidden state at time 0. If `nonlinearity` is `'relu'`, then `ReLU` is used instead of `tanh`.

```
torch.nn.RNN(input_size, hidden_size, num_layers=1, nonlinearity='tanh', ...  
               bias=True, batch_first=False, dropout=0, bidirectional=False )
```



### Bi-directional



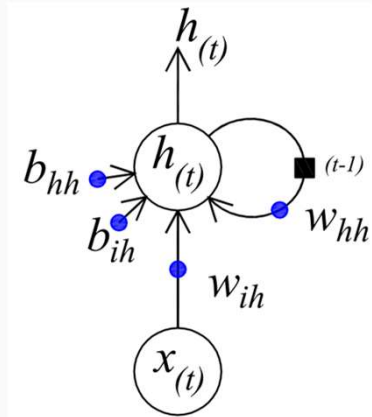
Para la traducción, suele ser útil tener la frase entera.

# Redes recurrentes - Implementación en pytorch

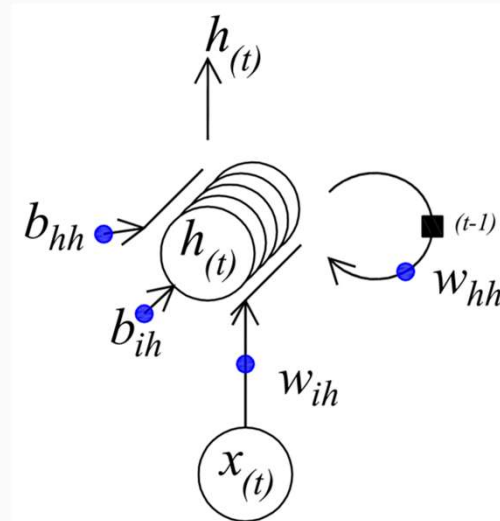
Pytorch

[Ver colab RNN\\_teoria.ipynb](#)

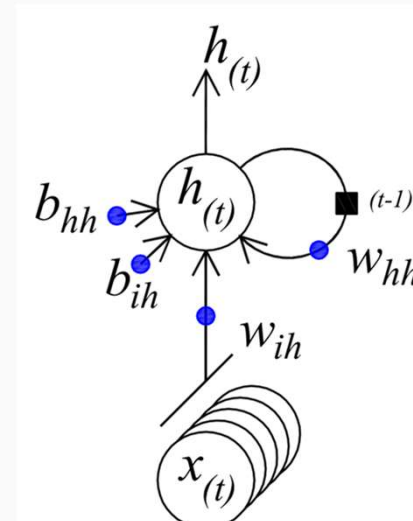
$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$



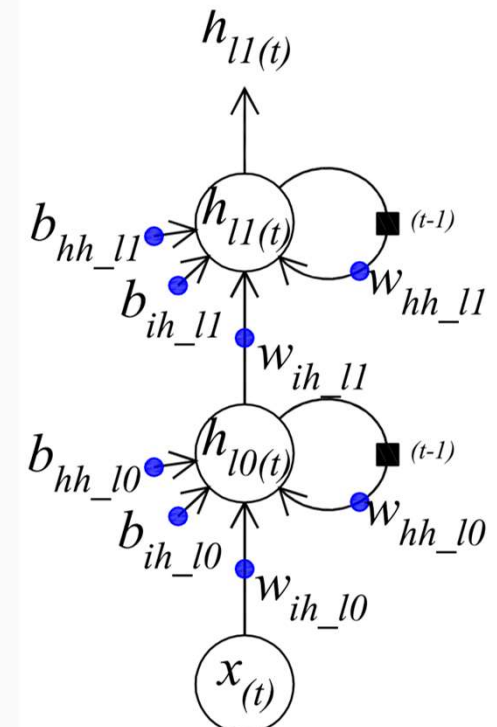
Básica



Varias hidden



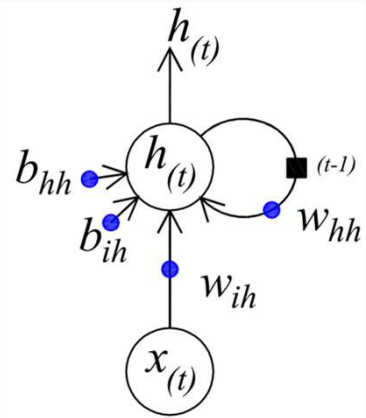
Input  
multivariable



2 layers

## Redes recurrentes - Implementación en pytorch

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

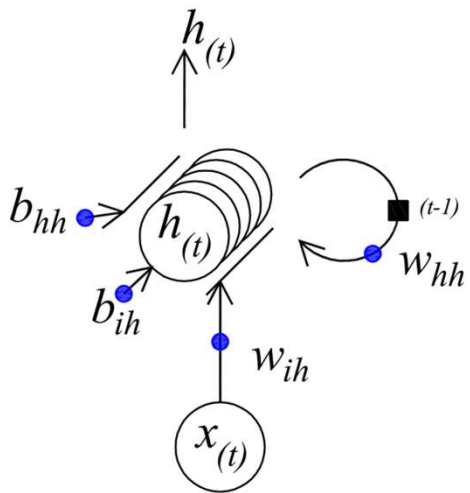


Variable	Tamaño	Parámetro	Tamaño
$x$		$W_{ih}$	
$h$		$B_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

Básica

## Redes recurrentes - Implementación en pytorch

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

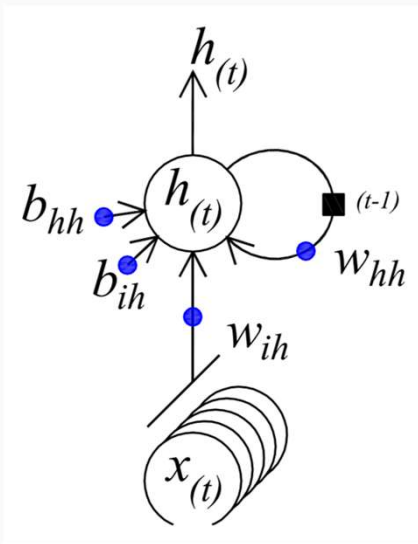


Varias hidden

Variable	Tamaño	Parámetro	Tamaño
$x$		$W_{ih}$	
$h$		$b_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

## Redes recurrentes - Implementación en pytorch

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

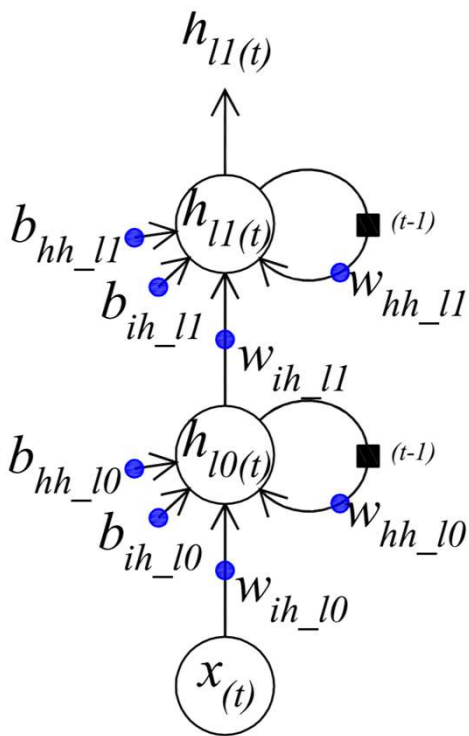


Input  
multivariable

Variable	Tamaño	Parámetro	Tamaño
$x$		$W_{ih}$	
$h$		$B_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

## Redes recurrentes - Implementación en pytorch

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

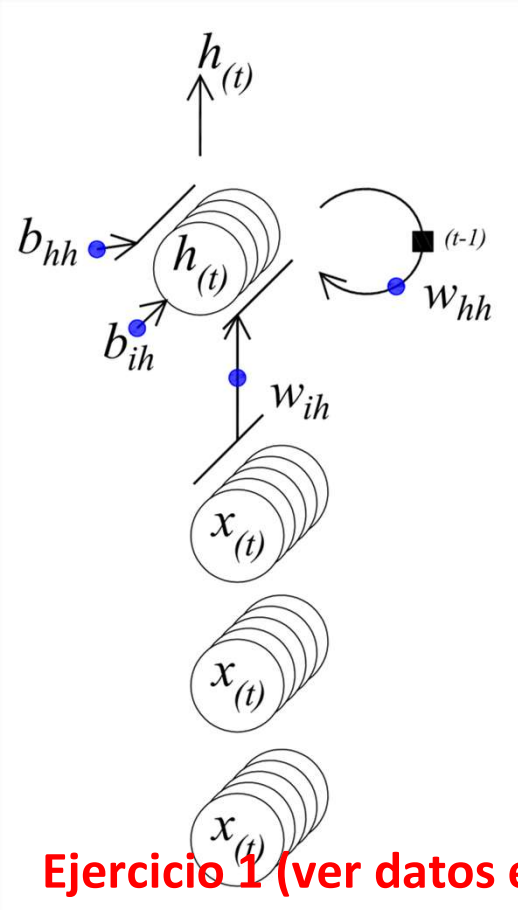


2 layers

Variable	Tamaño	Parámetro	Tamaño
$x$		$W_{ih}$	
$h$		$b_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

# Redes recurrentes - Implementación en pytorch

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



Ejemplo A

Ejemplo B

Ejemplo C

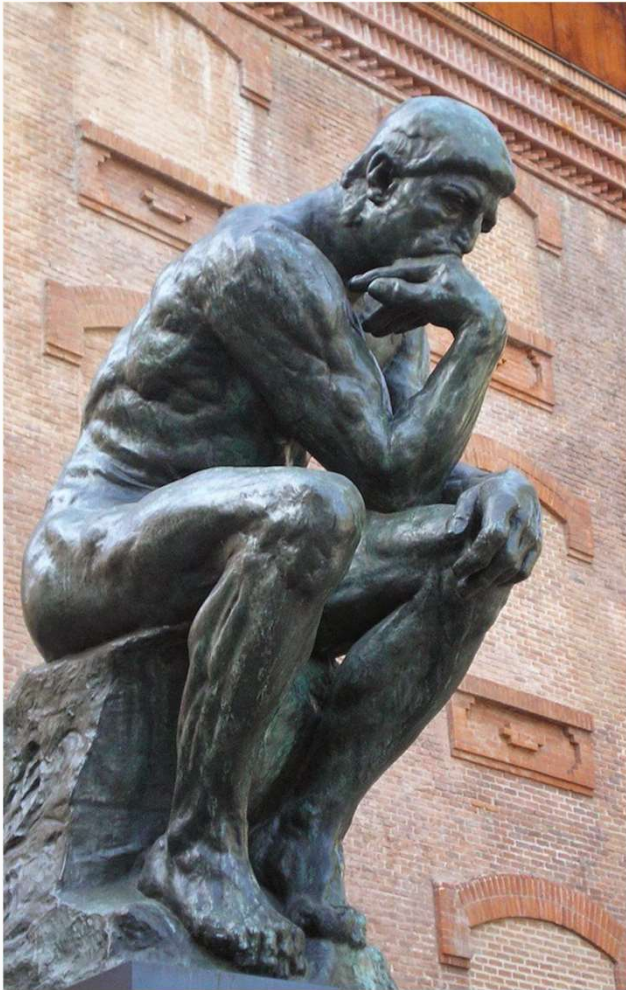
Variable	Tamaño	Parámetro	Tamaño
x		$W_{ih}$	
h		$b_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

Variable	Tamaño	Parámetro	Tamaño
x		$W_{ih}$	
h		$b_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

Variable	Tamaño	Parámetro	Tamaño
x		$W_{ih}$	
h		$b_{ih}$	
		$W_{hh}$	
		$b_{hh}$	

Ejercicio 1 (ver datos en colab)

## Redes recurrentes - Implementación en pytorch



**A pensar!**

“El pensador” de Rodin



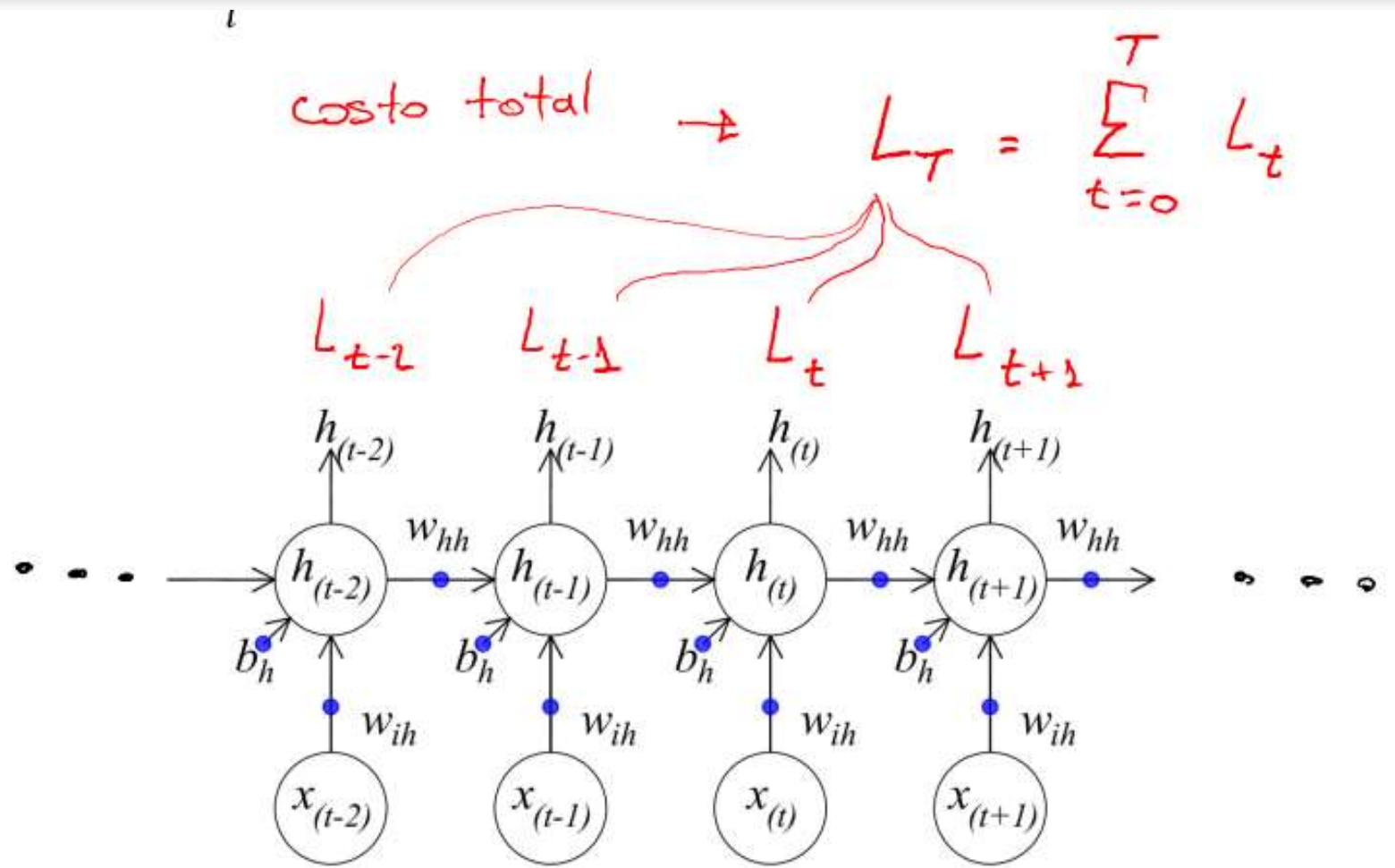
Redes recurrentes - Back propagation through time (BPTT)

# **BACK PROPAGATION THROUGH TIME [BPTT]**



Ver desarrollo teórico

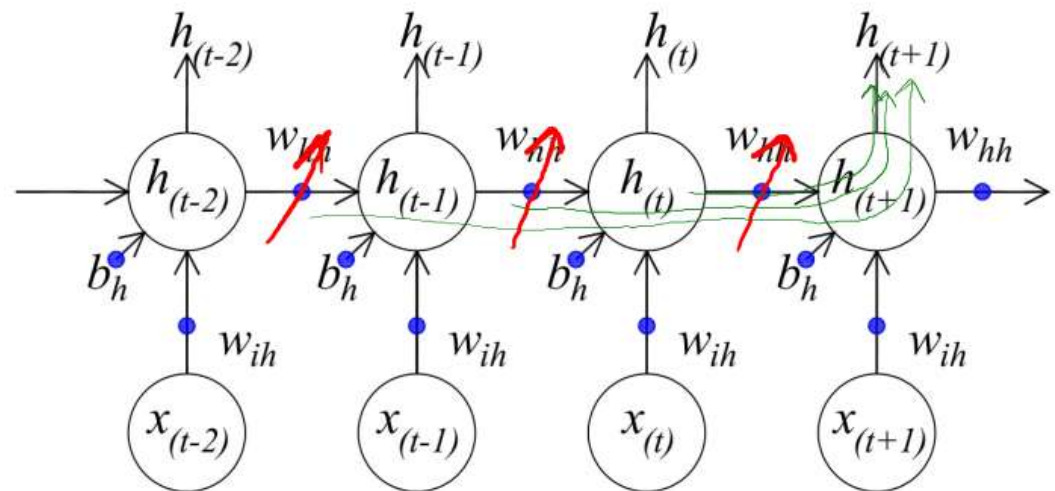
## Redes recurrentes - Back propagation through time (BPTT)



## Redes recurrentes - Back propagation through time (BPTT)

1º  $w_{hh} \rightarrow$  pesos hidden - hidden

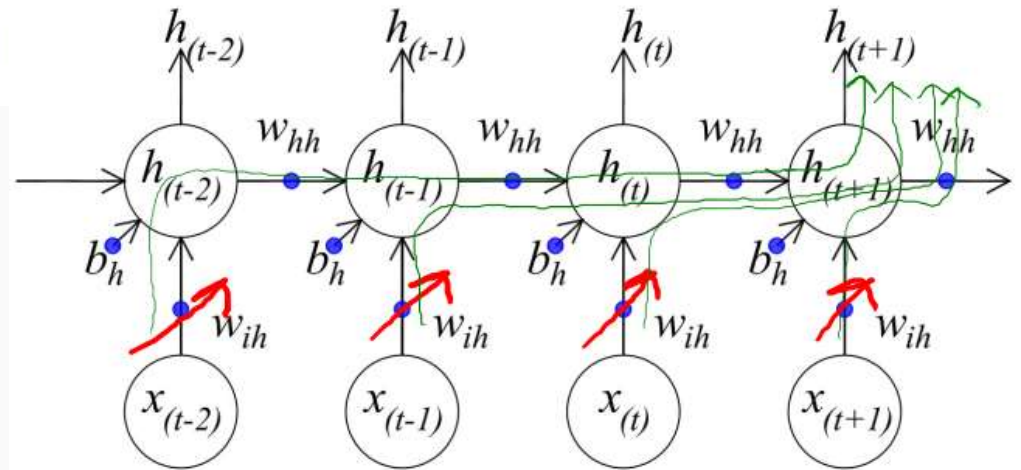
$$\frac{\partial L_T}{\partial w_{hh}} = ?$$



$$\frac{\partial L_T}{\partial w_{hh}} = \sum_{t=0}^T \frac{\partial L_t}{\partial L_F} \cdot \frac{\partial L_F}{\partial h_t} \sum_{k=0}^t \left[ \prod_{j=k}^t \left( \frac{\partial h_{j+1}}{\partial h_j} \right) \cdot \frac{\partial h_k}{\partial w_{hh}} \right]$$

## Redes recurrentes - Back propagation through time (BPTT)

2º  $w_{ih}$  → pesos input-hidden

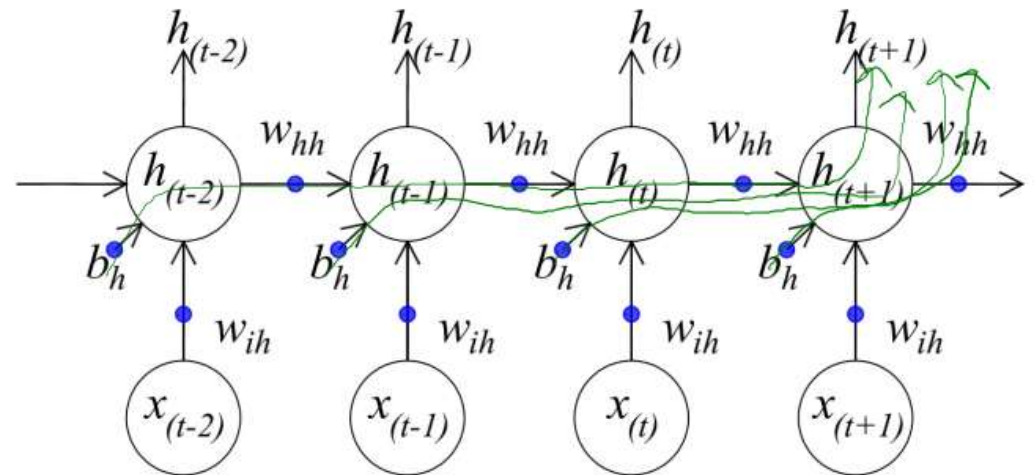


$$\frac{\partial L_T}{\partial w_{ih}} = \sum_{t=0}^T \frac{\partial L_t}{\partial L_F} \cdot \frac{\partial L_F}{\partial h_t} \sum_{k=0}^t \left[ \prod_{j=k}^t \left( \frac{\partial h_{j+1}}{\partial h_j} \right) \cdot \frac{\partial h_k}{\partial w_{ih}} \right]$$

a cá cambio

## Redes recurrentes - Back propagation through time (BPTT)

3º  $b_h \rightarrow$  bias hidden



$$\frac{\partial L_T}{\partial b_h} = \sum_{t=0}^T \frac{\partial L_t}{\partial L_F} \cdot \frac{\partial L_F}{\partial h_t} \sum_{k=0}^t \left[ \prod_{j=k}^t \left( \frac{\partial h_{j+1}}{\partial h_j} \right) \cdot \frac{\partial h_k}{\partial b_h} \right]$$

← acá cambió

### Problemas de la RNN básica con el BPTT

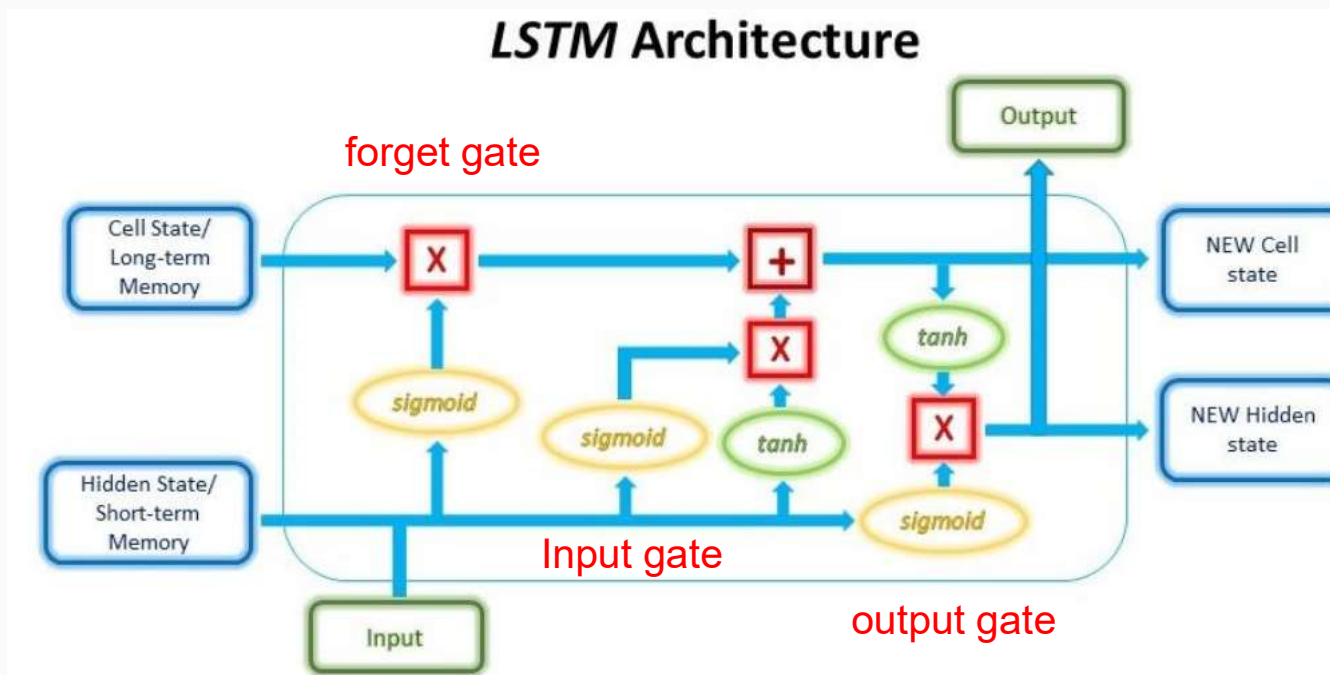
**Vanishing gradient → pérdida de aportes de long-term states  
(gradientes próximos a cero)**

**Exploding gradient → se soluciona con clipping gradient  
(gradientes mayores a 1)**

**Solución con otras RNN mas avanzadas (LSTM y GRU)**

## Redes recurrentes - Back propagation through time (BPTT)

### Solución con otras RNN mas avanzadas (LSTM y GRU)

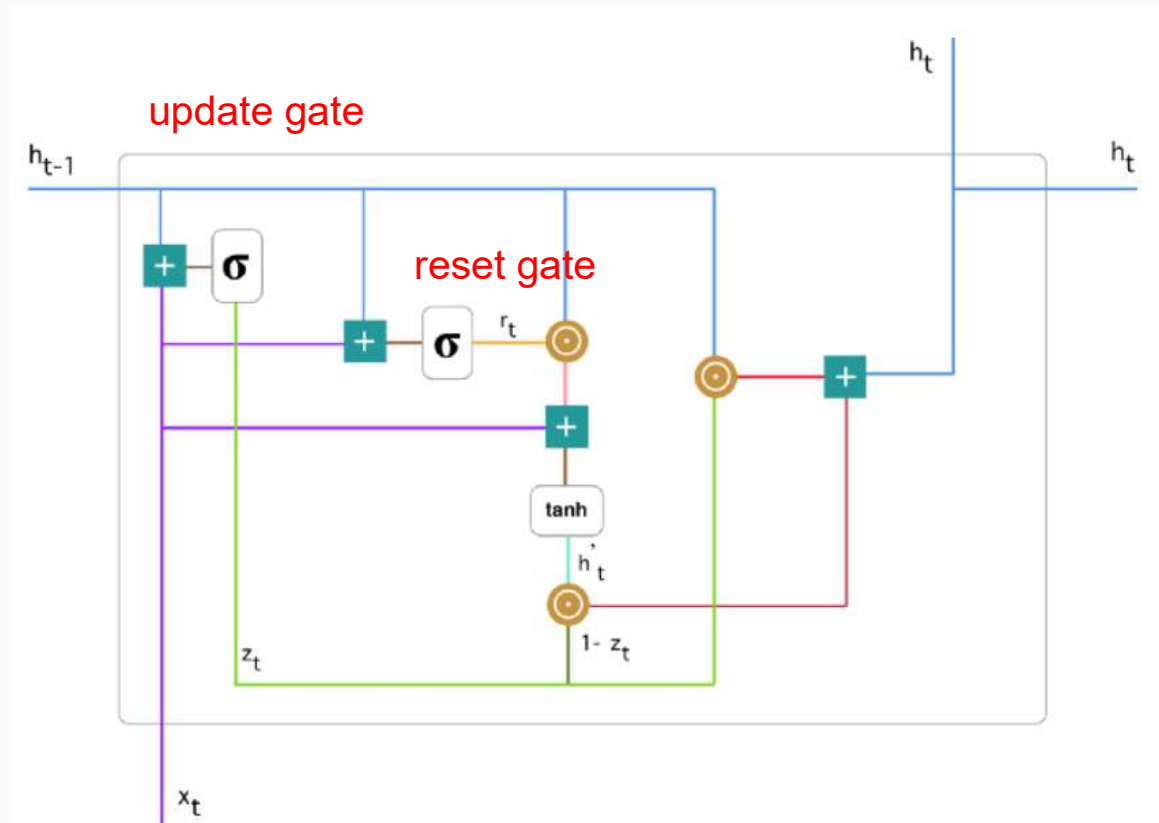


<https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/>



## Redes recurrentes - Back propagation through time (BPTT)

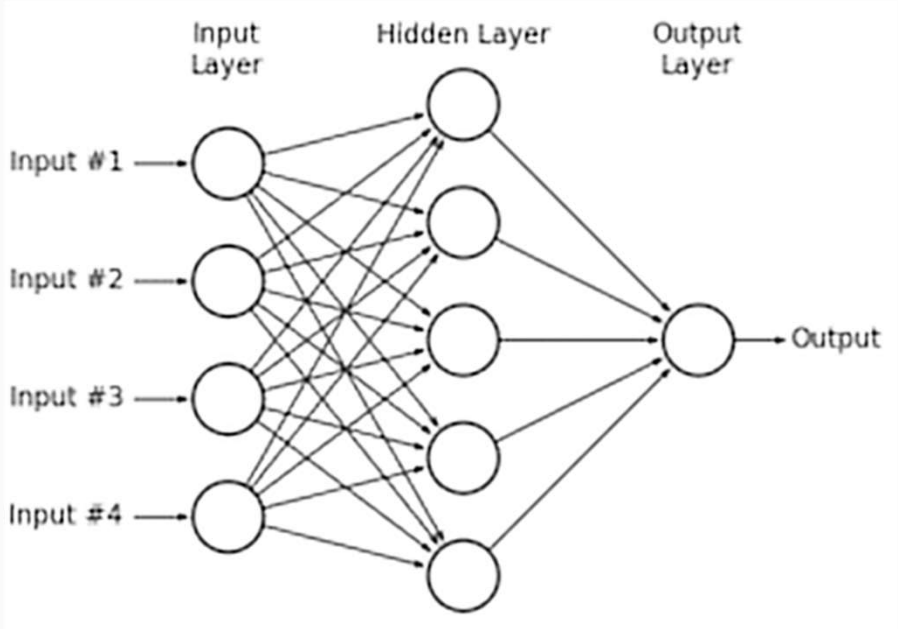
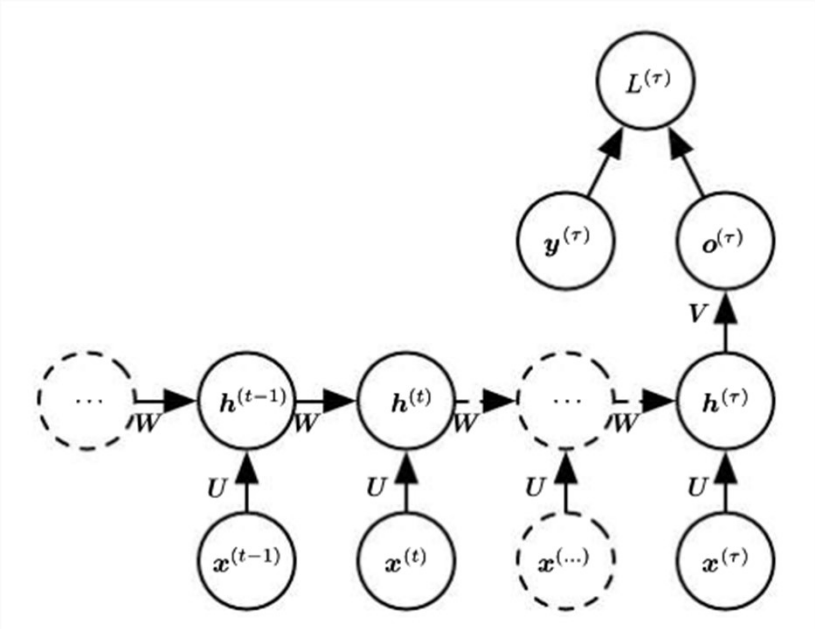
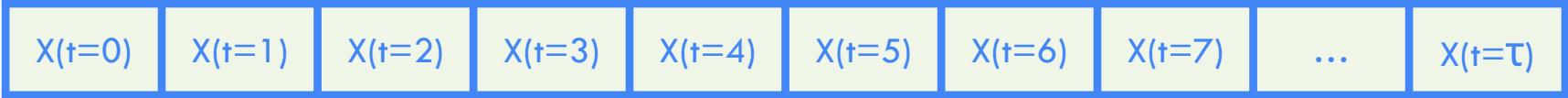
### Solución con otras RNN mas avanzadas (LSTM y GRU)



<https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>



# Redes recurrentes - Implementación de modelos

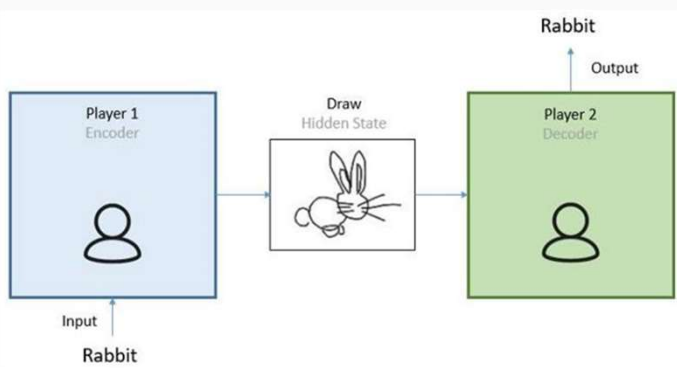
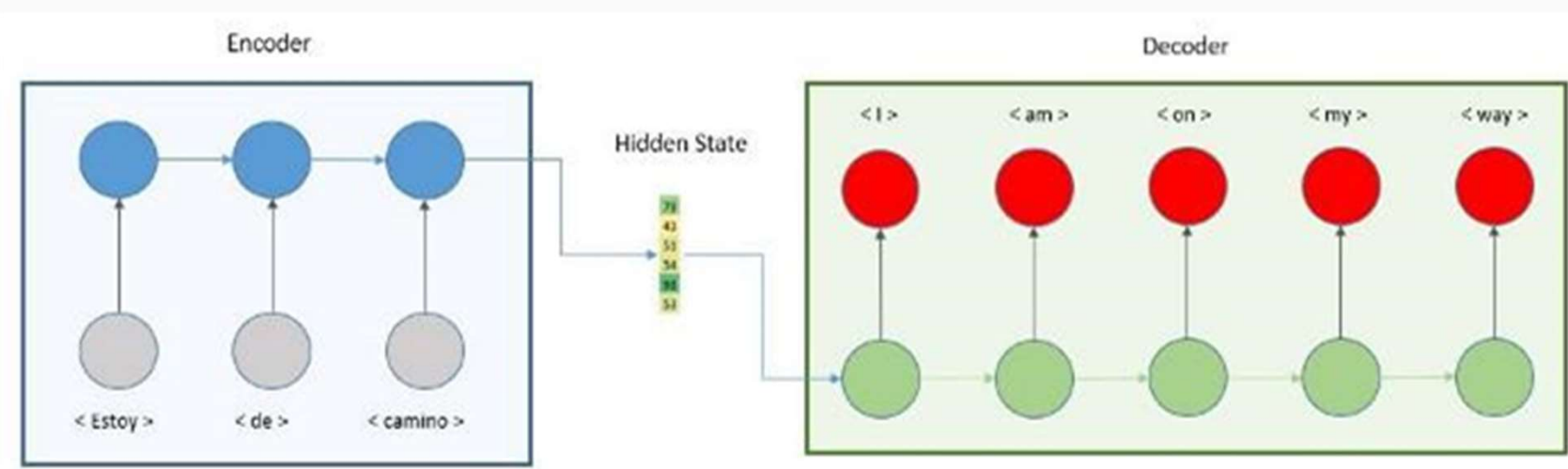


Ver colab [RNN\\_signal\\_TP.ipynb](#)

**¡Un merecido descanso!**

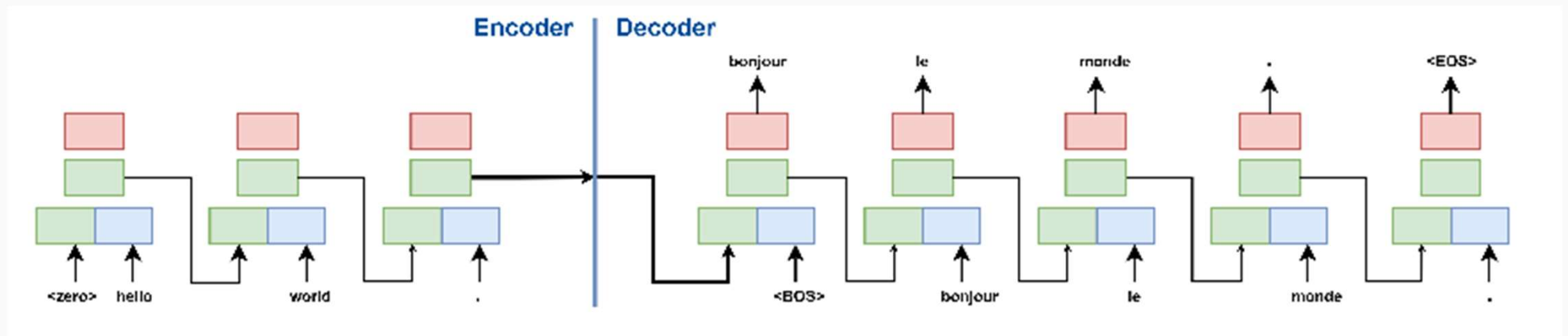


# Redes recurrentes - Arquitectura enconder-decoder (seq to seq)



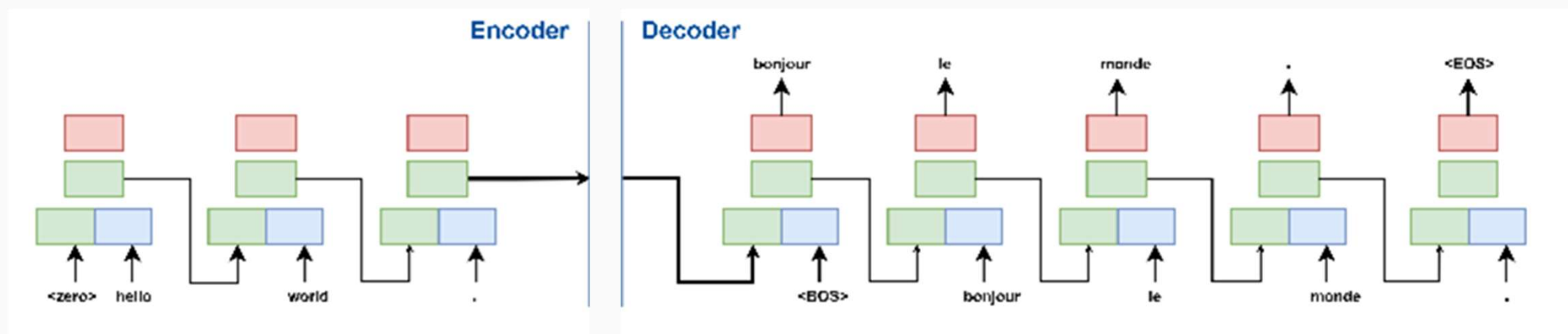
- 2 RNN de distinto tamaño
- 1 Hidden state que “resume” toda la información de la input.
- Flexibilidad máxima para inputs/outputs de distinta longitud

## Redes recurrentes - Arquitectura enconder-decoder (seq to seq)



Unfolded!!  
!!

# Redes recurrentes - Arquitectura enconder-decoder (seq to seq) entrenamiento



## ENCODER

Siempre leen la secuencia entera

Emiten un hidden state final

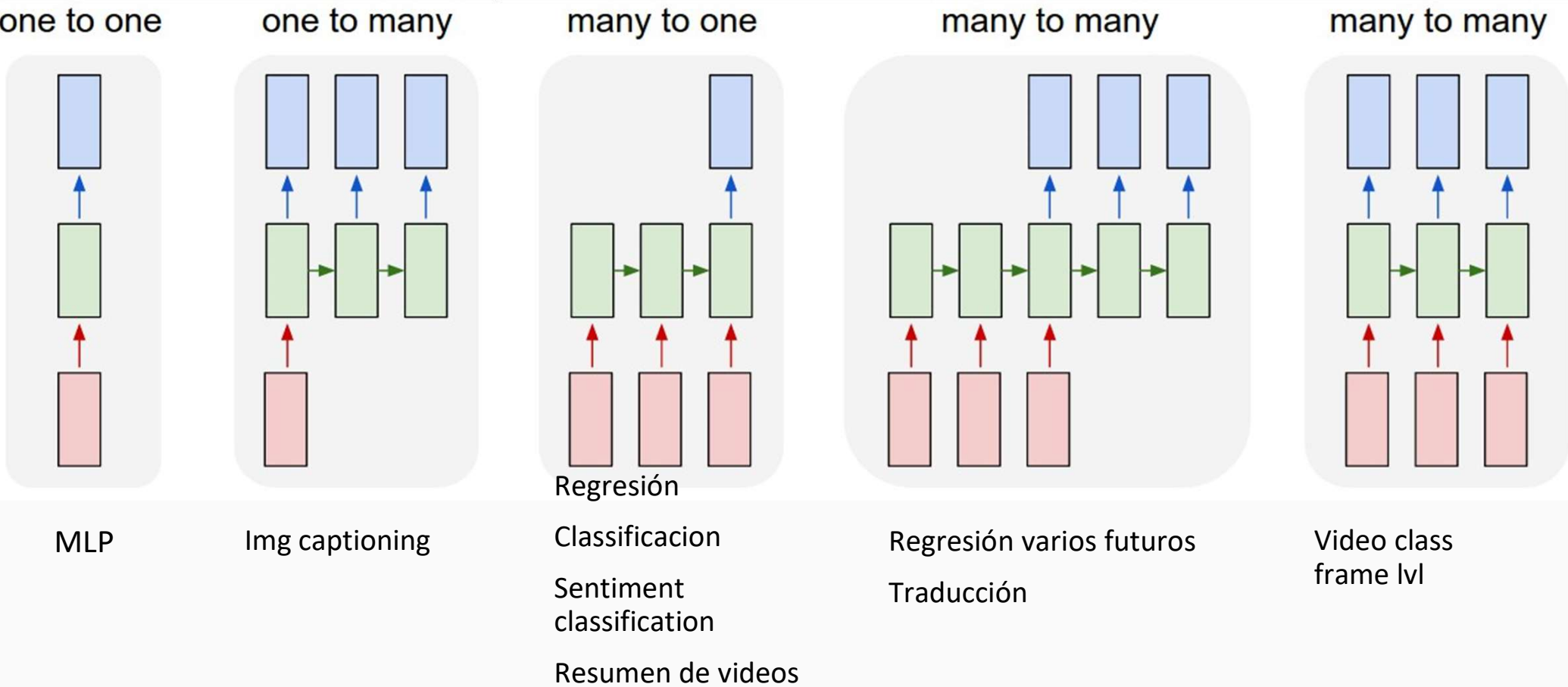
## DECODER

Entrenamiento → `for i in range(len(y_deseado):`  
`genero_token`

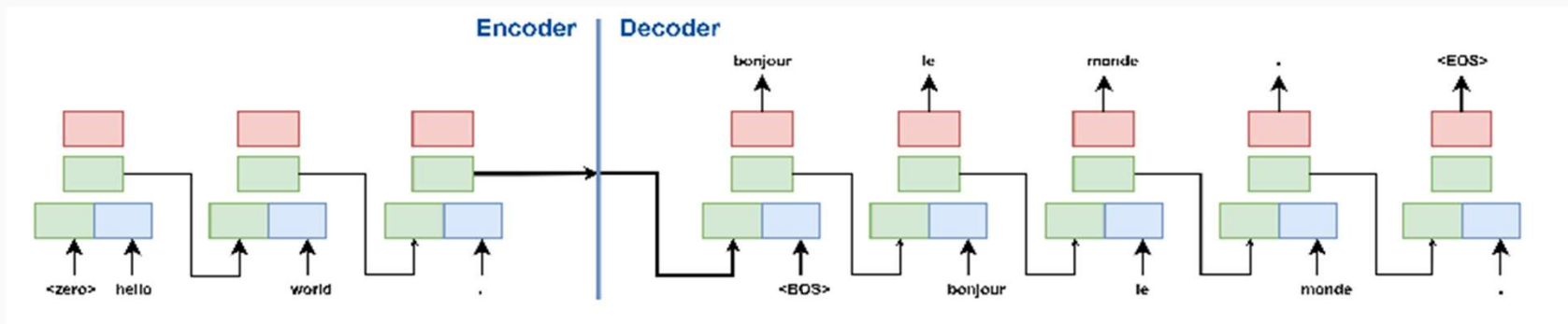
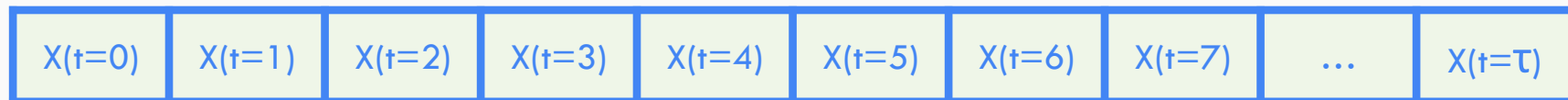
Uso → `while last_token != <EOS>:`  
`genero_token`

Redes recurrentes - Arquitectura en coder-decoder (seq to seq)

Arquitecturas flexibles IN/OUT

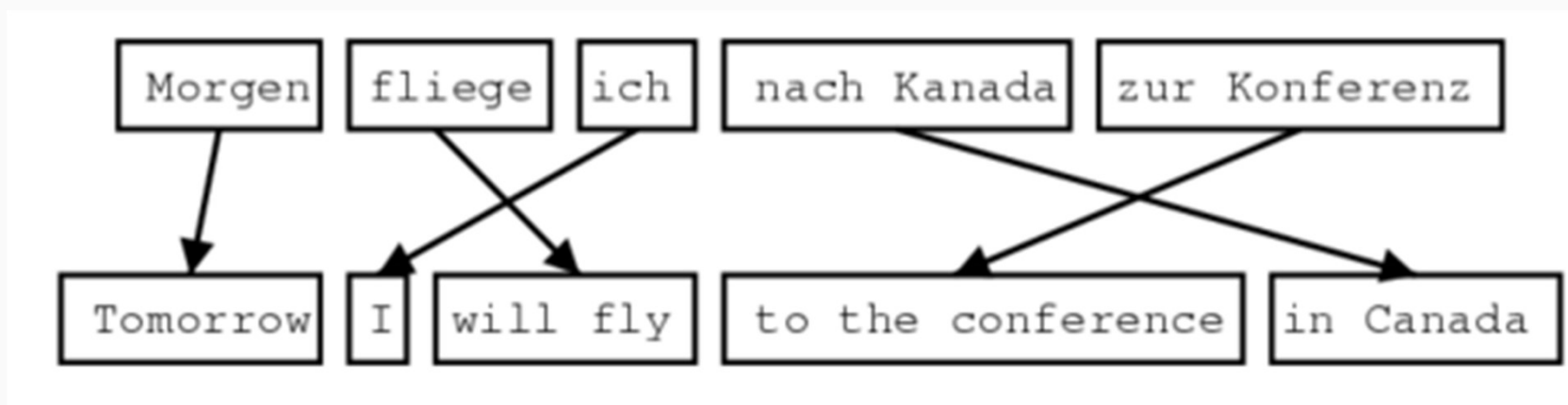


# Redes recurrentes - Implementación de modelos encoder-decoder



**Ver colab [RNN\\_enc\\_dec.ipynb](#)**

## Redes recurrentes - Mecanismos de atención



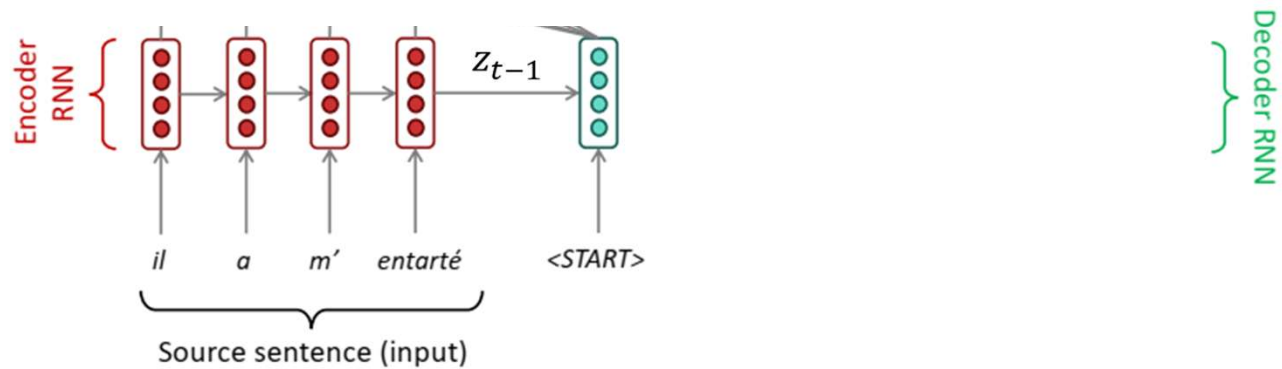
*El mecanismo de atención permite al decoder **utilizar las partes más relevantes** de la entrada **como una suma ponderada** del vector de entrada codificados para predecir la siguiente palabra.*

*Una **palabra relevante** tendrá un **mayor peso** que una palabra no relevante*



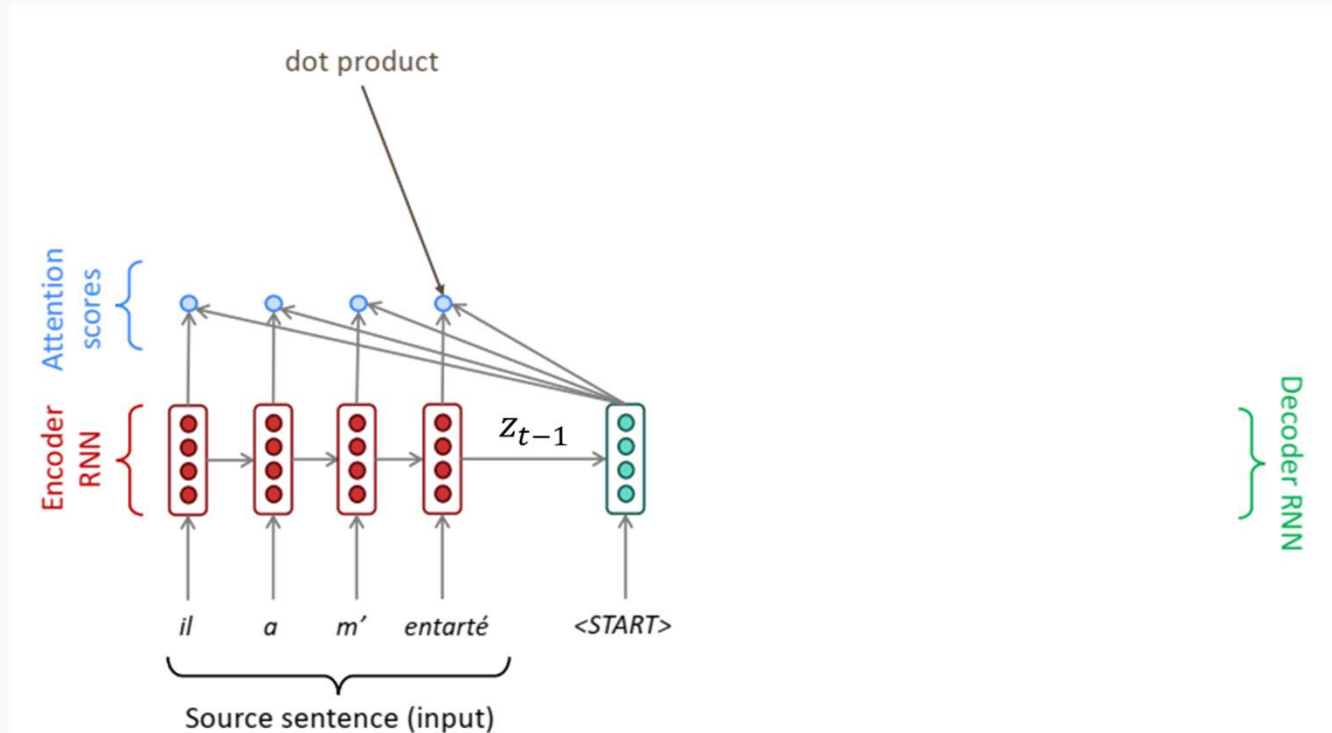
## Redes recurrentes - Mecanismos de atención

Ver en bibliografía: [cs224n-2021-lecture07-nmt.pdf](#)



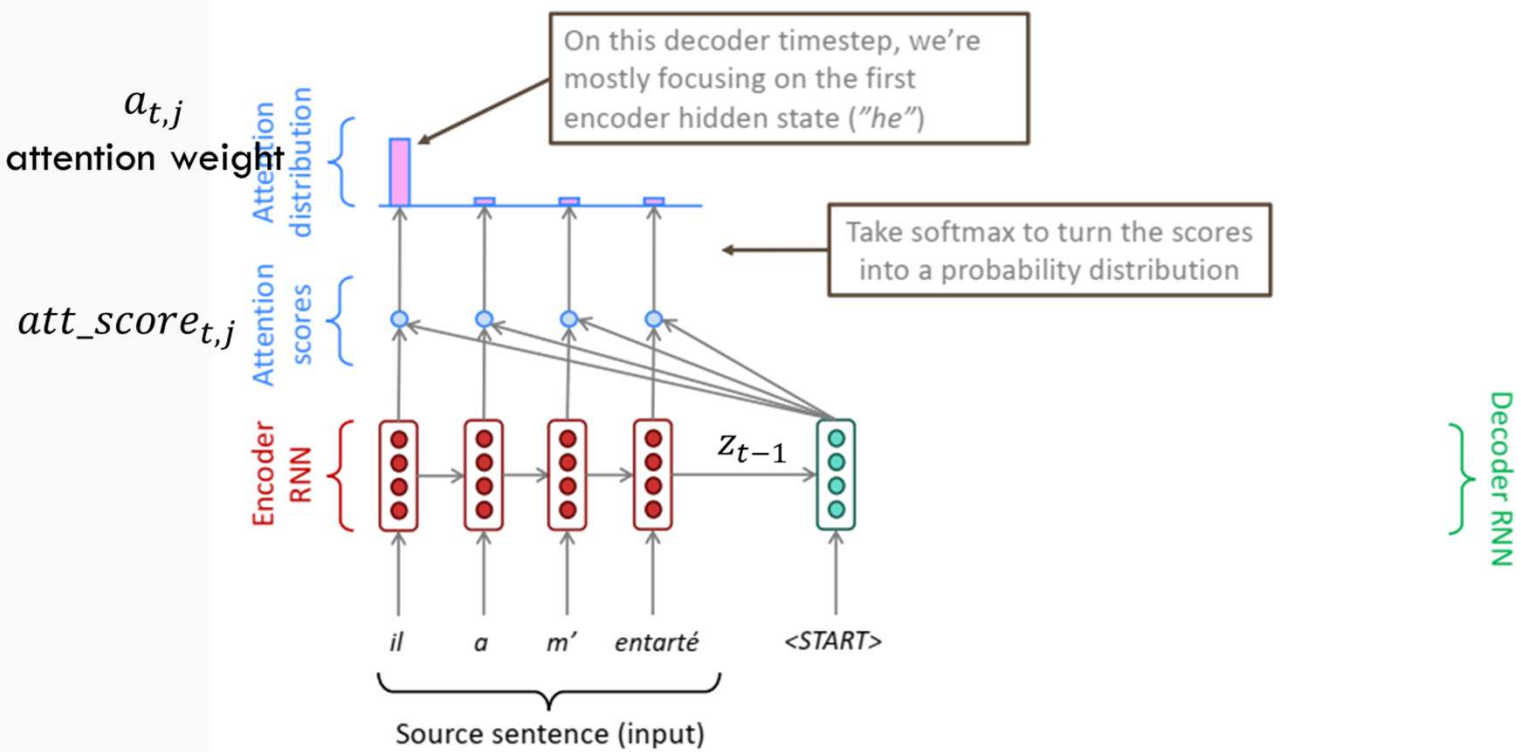
## Redes recurrentes - Mecanismos de atención

$att\_score_{t,j}$



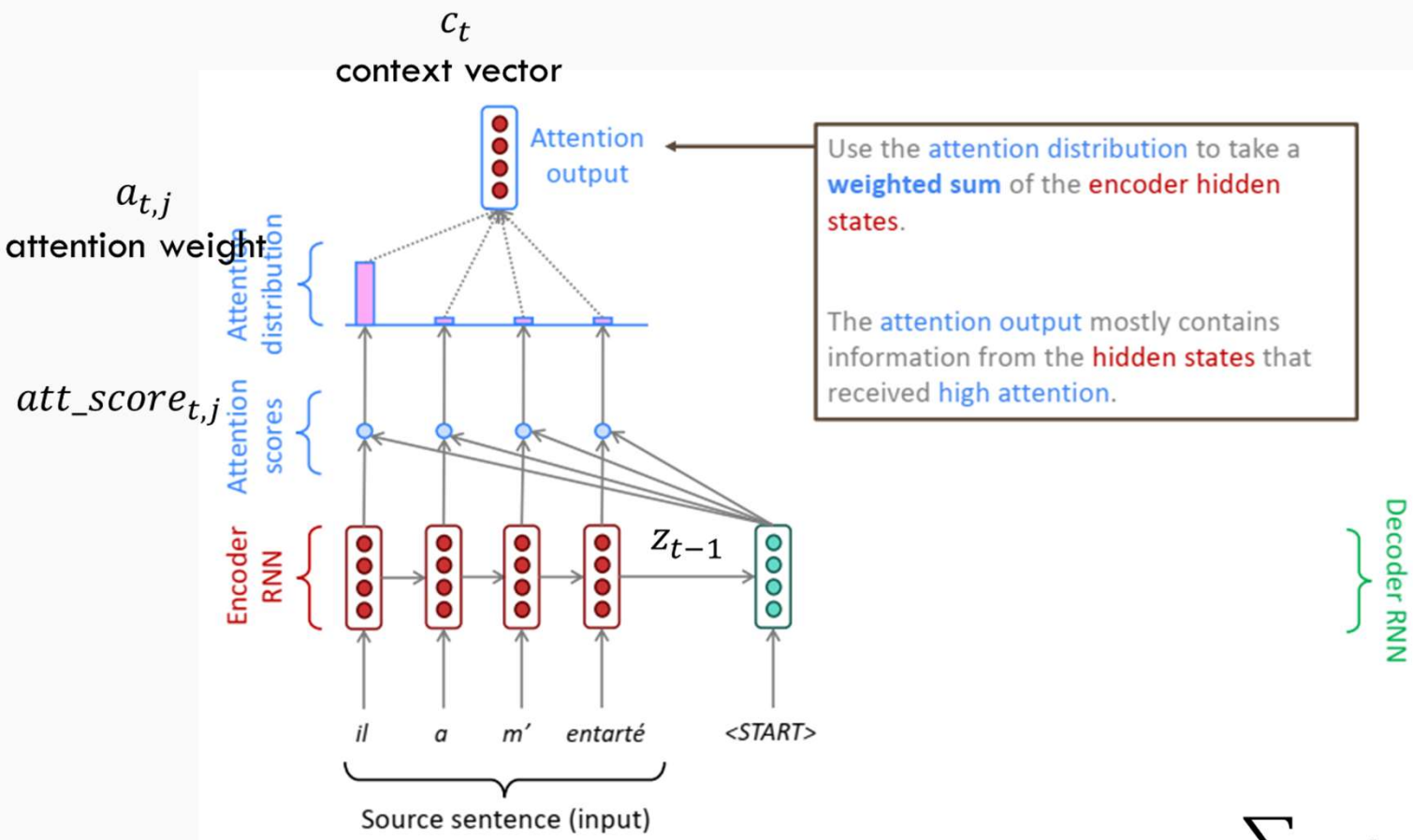
$$att\_score_{t,j} = \text{dot}(z_t, h_j)$$

# Redes recurrentes - Mecanismos de atención



$$att\_score_{t,j} = \text{dot}(z_t, h_j) \qquad a_{t,j} = \text{softmax}(att\_score_{t,j})$$

# Redes recurrentes - Mecanismos de atención

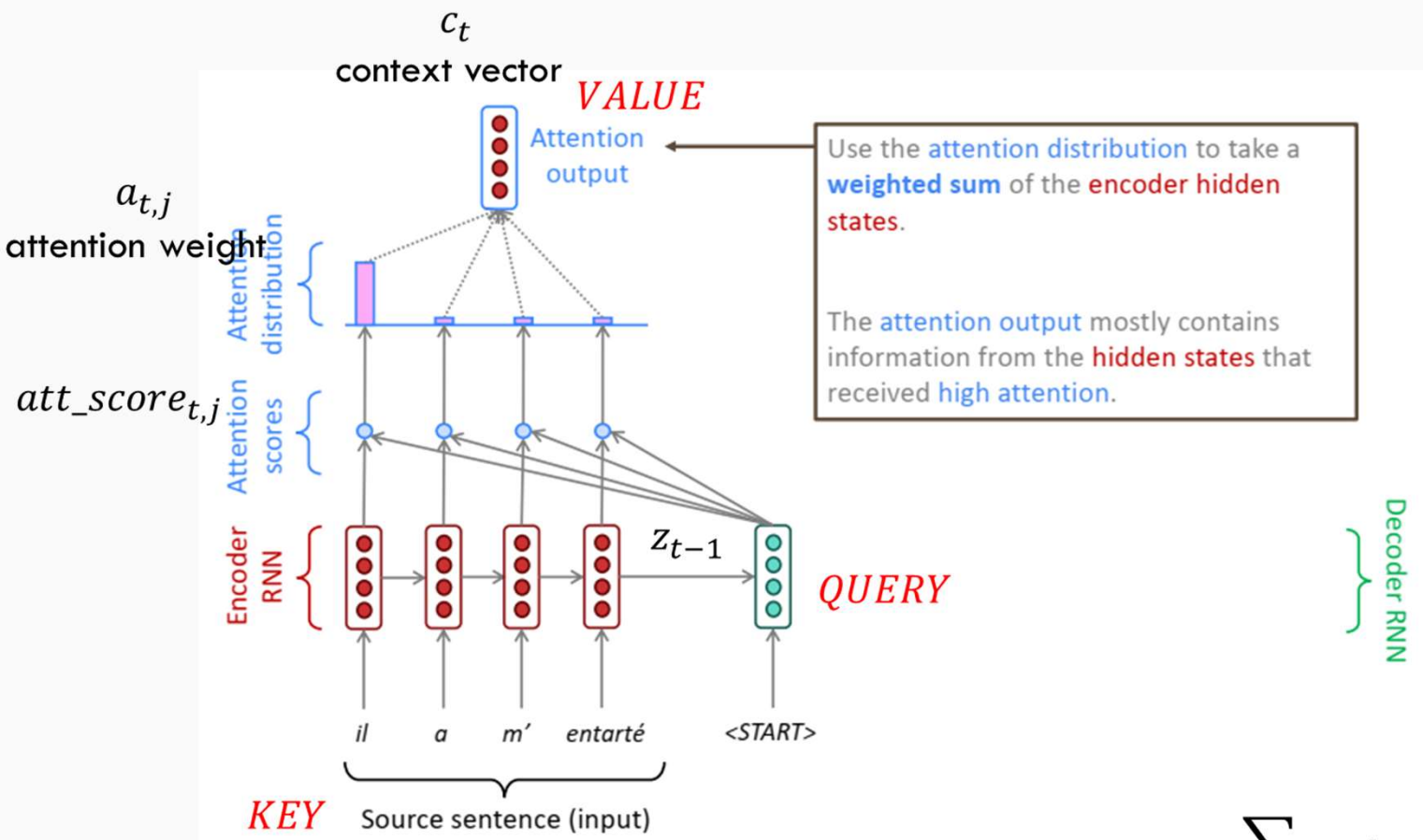


$$att\_score_{t,j} = \text{dot}(z_t, h_j)$$

$$a_{t,j} = \text{softmax}(att\_score_{t,j})$$

$$c_t = \sum_j a_{t,j} h_j$$

# Redes recurrentes - Mecanismos de atención

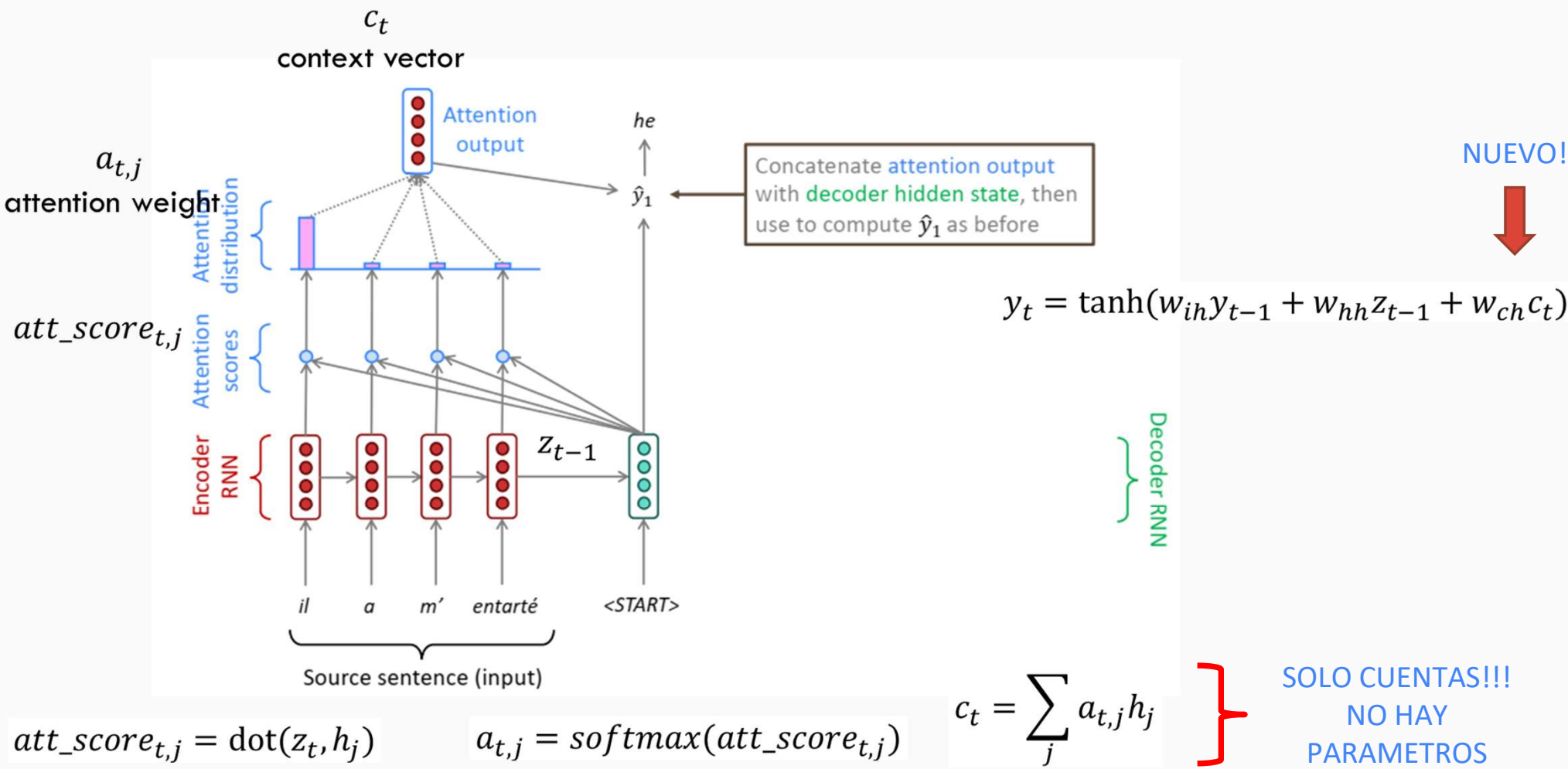


$$att\_score_{t,j} = \text{dot}(z_t, h_j)$$

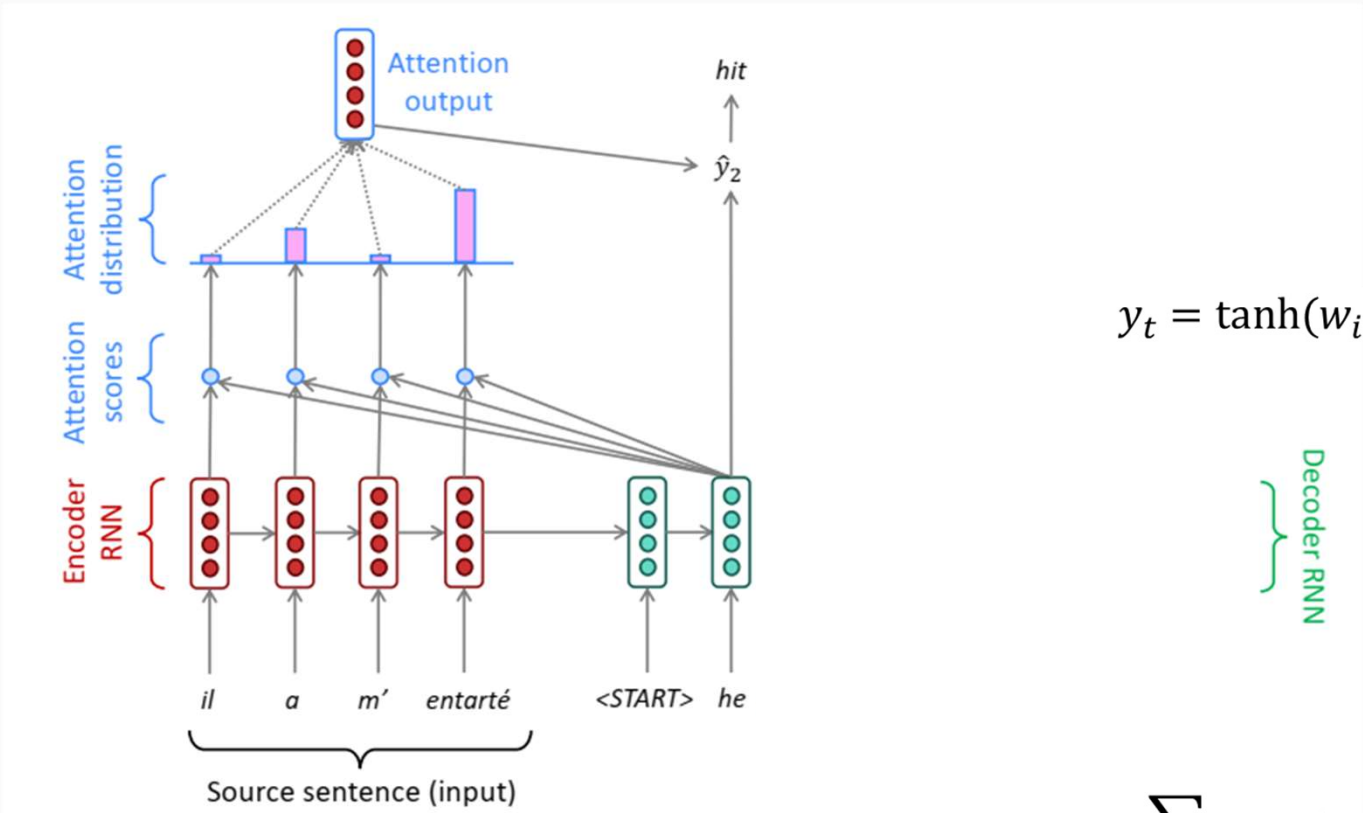
$$a_{t,j} = \text{softmax}(att\_score_{t,j})$$

$$c_t = \sum_j a_{t,j} h_j$$

# Redes recurrentes - Mecanismos de atención



# Redes recurrentes - Mecanismos de atención



$$y_t = \tanh(w_{ih}y_{t-1} + w_{hh}z_{t-1} + w_{ch}c_t)$$

Decoder RNN

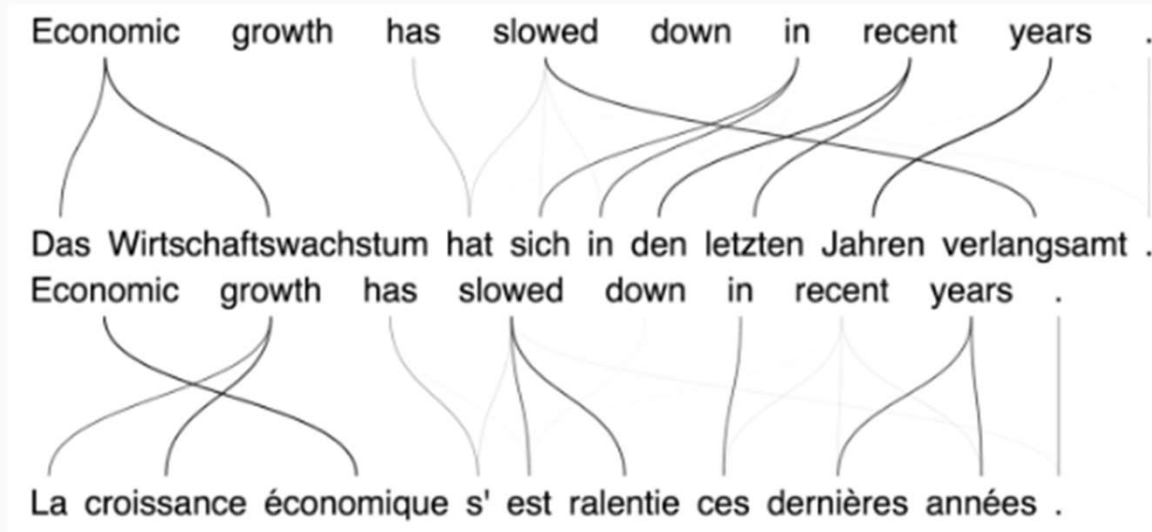
$$att\_score_{t,j} = \text{dot}(z_t, h_j) \quad a_{t,j} = \text{softmax}(att\_score_{t,j})$$

$$c_t = \sum_j a_{t,j} h_j$$

## Redes recurrentes - Mecanismos de atención

Desde un punto de vista probabilístico...

el **attention weight**  $a_j$  puede ser visto como la probabilidad de que el decoder use esa palabra (representación) para realizar la decodificación del contexto.

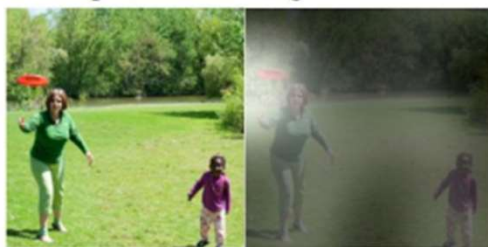


Una definición más general:

Dado un **conjunto de valores** y una **consulta**; el mecanismo de atención devuelve una **suma ponderada** (resumen selectivo) de los valores, **dependiente de la consulta**.



# Image Captioning with Attention



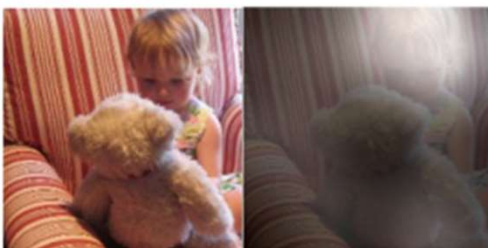
A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.