

# Redes Neuronales Recurrentes

- Surgen de la necesidad de tener entradas con diferentes dimensiones.
- Suelen utilizarse para predicción de series temporales.
- La arquitectura del modelo posee funciones de activación, bias o sesgos, parámetros: pero además tienen Loops. Igual que las NN
- Revisar este tipo de modelos permitirá construir las siguientes ideas:

RNN → GRU / LSTM → Transformers

## Ejemplos y Motivación

- Supongamos tenemos una serie temporal
- Queremos que el valor del pasado y el actual nos permitan prever el valor futuro.

- A veces se denominan "Sequential Learning" a esta área de estudio.
- Los problemas más interesantes se encuentran en el área del lenguaje. Por ejemplo:

i) Traducción :

I'm under the weather today



Estoy deceido hoy

Observe :

- Secuencias de diferente tamaño.
- Con un significado común.
- infinitas combinaciones de entrada - salida con diferentes tamaños.

## Possibles arquitecturas :

- i) Un input → Muchas salidas
- ii) Muchos input → Una salida
- iii) Muchos inputs → Muchas Salidas

## Ejemplos de Aplicaciones :

- i) Imagen → Descripción de la imagen
- ii) Imágenes y descripciones → Sentimiento
- iii) Una frase incompleta → Una frase completa

\* Una red neuronal común no tiene incorporada la noción de "orden" de las secuencias,  
(Redes vistas en clases anteriores)

O un sentido de "dirección" para el caso de problemas de lenguaje.

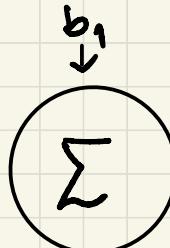
## Ventajas de las RNN:

- i) Almacenan mucha info del pasado en las capas ocultas.
  - ii) El uso de funciones no lineales i.e. permite aprender patrones complejos de salida.
  - iii) Los pesos son compartidos entre los diferentes estados.
- Ahora revisaremos un ejemplo de una red recurrente Vanilla (que podremos entender como la versión más simple de una RNN).

Ejemplo:

Valor de  
Ayer

$$w_1 \rightarrow$$



Ref U

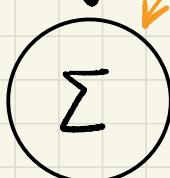
$$w_3 \rightarrow$$

$$b_2 \downarrow$$

Predicció!  
Valor  
de  
hoy

Valor de  
Hoy

$$w_1 \rightarrow$$



$$w_3 \rightarrow$$

$$b_2 \downarrow$$

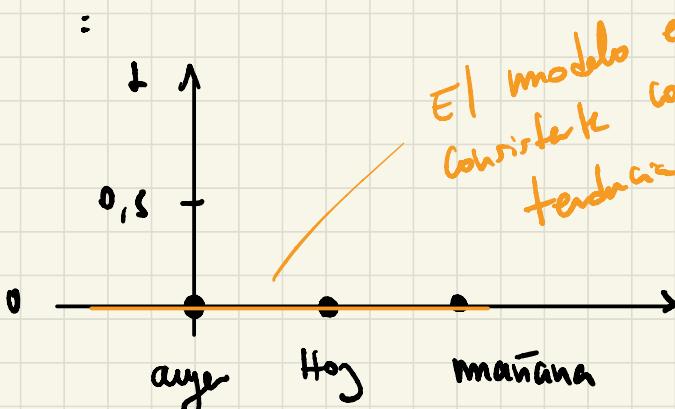
Predicció!  
Valor  
de  
Mañana

$$\text{Si usamos } w_1 = 1,8 \quad w_3 = 1,1$$

$$w_2 = -0,5$$

$$b_1, b_2 = 0$$

Y tenemos



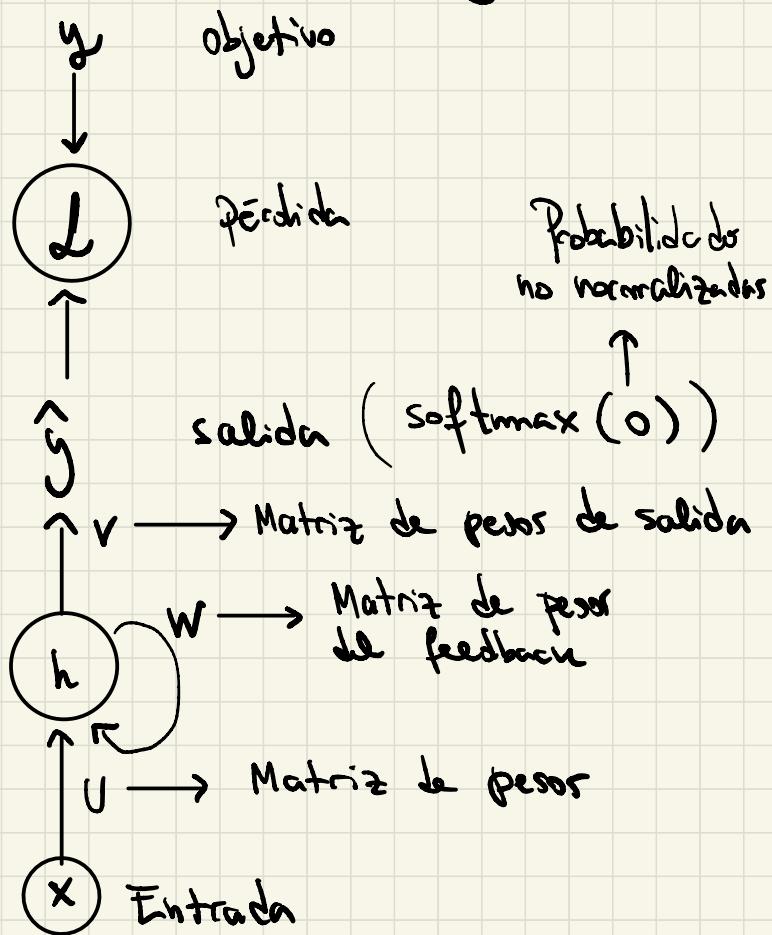
El modelo es  
constante con la  
tendencia del dato!

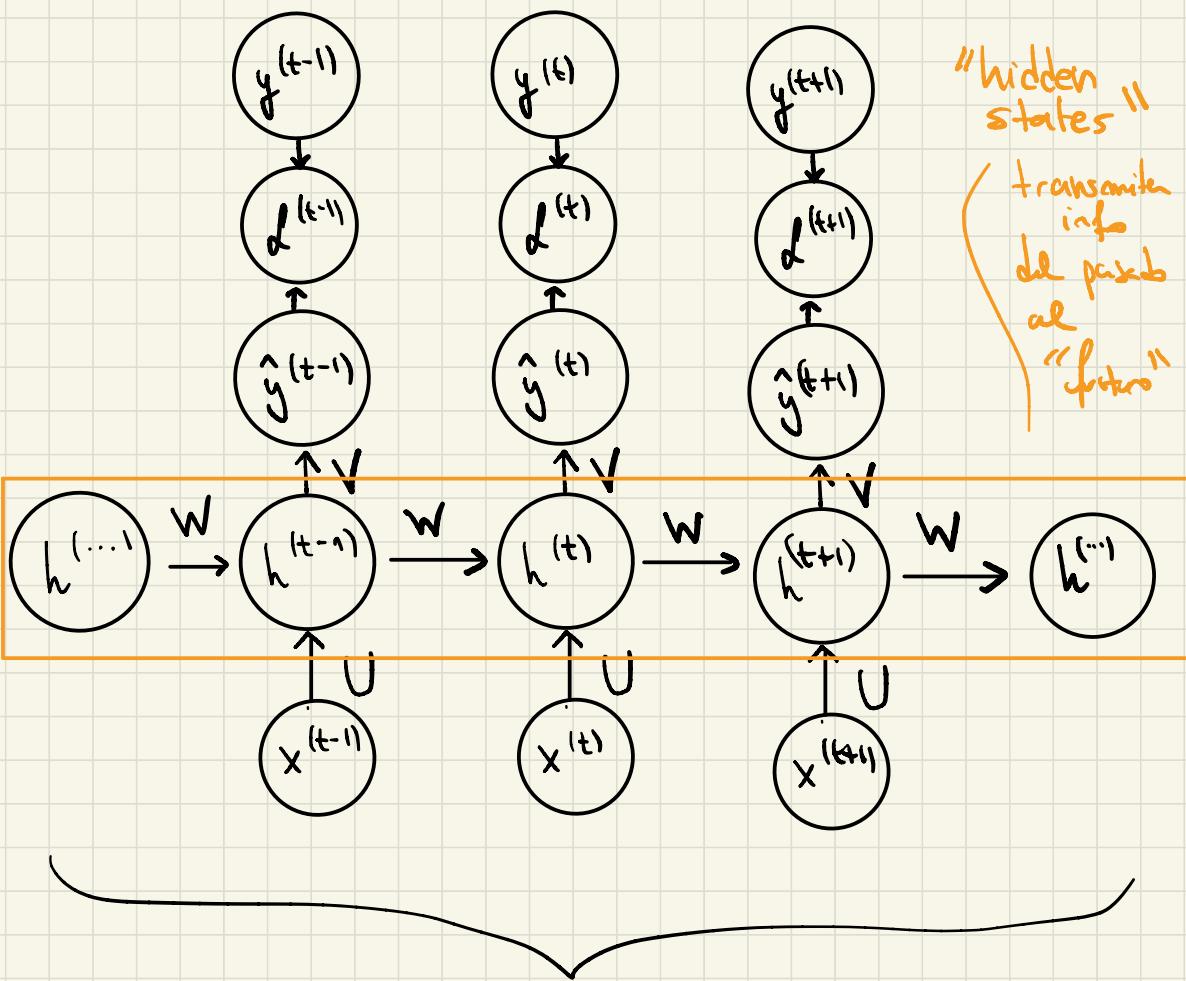
Observación

: Usamos los mismos parámetros  $b_1, w_1, \dots$  etc en el proceso de recurrencia  $\therefore$  No agregamos nuevos parámetros.

Un grafo computacional que representa el cálculo de la pérdida de entrenamiento de una red que transforma  $X$  (input) en la salida  $\hat{y}$  dado un objetivo/target  $y$ :

Forma no expandida del grafo





## Grafo Computacional Extendido

Representado en una ecuación :

$$a^{(t)} = b + W h^{(t-1)} + U x^{(t)} \quad \text{--- tanh}$$

$$h^{(t)} = f(a^{(t)}) \quad f: \text{función de activación}$$

$$\hat{y}^{(t)} = \Delta(c + V h^{(t)}) \quad \Delta: \text{Softmax}$$

Otro ejemplo :

Supongamos que tenemos un vocabulario de N palabras, dadas por :

$$\text{Vocab} = \{ \text{"hola"} : 0, \text{"Ella"} : 1, \text{"El"} : 2, \dots \}$$

Cada palabra está asociada a un número, en una lista de números enteros. Podemos representar la palabra "hola" por un "one-hot vector" de la forma :

$$\text{"hola"} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{N \times 1} \longrightarrow \begin{array}{l} \text{Posición 0} \\ \text{Posición 1} \end{array}$$

$$\text{"Ella"} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{N \times 1} \quad \text{etc.}$$

• Supongamos que queremos completar una frase dada por:

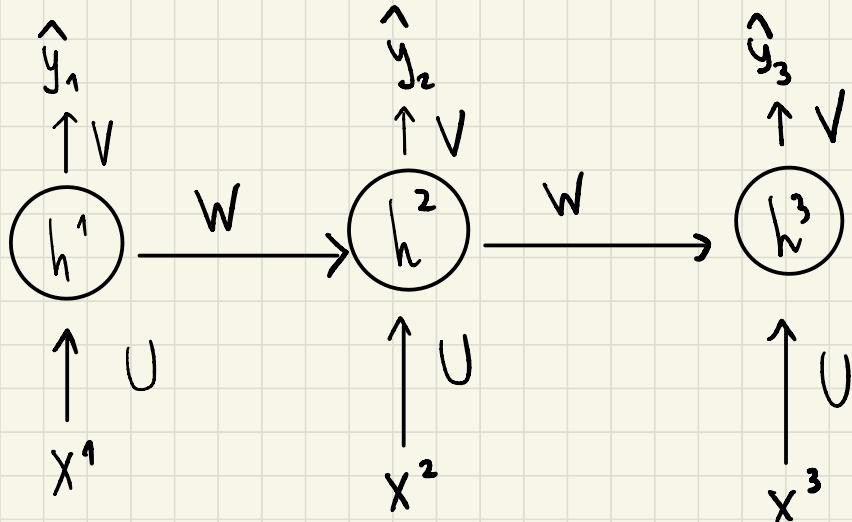
Cómo construir un \_\_\_\_\_

Todas las palabras de esta frase están representadas por un vector:

Cómo  $\equiv x_1$

construir  $\equiv x_2$

un  $\equiv x_3$



Cómo

construir

un

$$h^t = f_h \left[ U X^+ + W \underbrace{h^{t-1}}_{\text{Pasado}} + b_h \right]$$

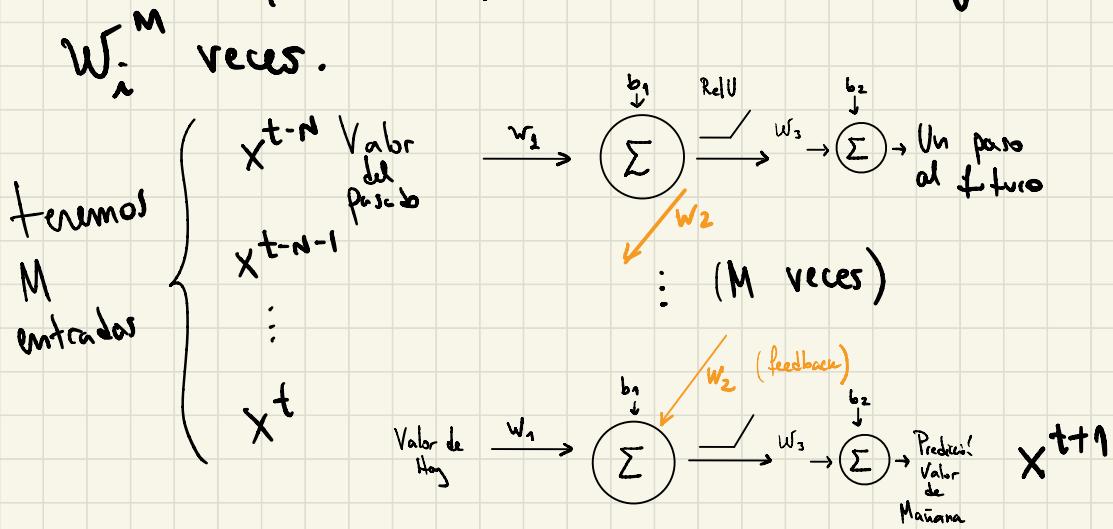
$$\hat{y}^t = f_s \left[ V h^t + b_s \right]$$

Mismas eq



- $h$  termina siendo una representación de lo que hemos visto en la historia de la secuencia.
- Debido a que  $h$  captura a través del método de aprendizaje la info a lo largo de la secuencia es más probable que luego completar la frase con mayor tasa de éxito que otros modelos.
- Por ejemplo : Cómo construir un "perro" un "perrito".

- Estas arquitecturas presentan un problema grave y es uno de los motivos por la que no es actualmente utilizada : Gradiéntes Exploratorios / Gradiéntes que desaparecen.
- Recordemos que para calcular el gradiénte y actualizar los pesos debemos utilizar la regla de la cadena.
- Si uno de los pesos  $w_i > 1$  y repetimos la recurrencia  $M$  veces, el valor de entrada afectado por el peso  $w_i$  será amplificado  $w_i^M$  veces.



- Para entrenar estos redes se usa una variante del Backpropagation llamada Backpropagation Through Time (BPTT).
- En general nuestras actualizaciones de gradiente serán las mismas:

$$V^{k+1} = V^k - \alpha \frac{\partial L}{\partial V}$$

$$W^{k+1} = W^k - \alpha \frac{\partial L}{\partial W}$$

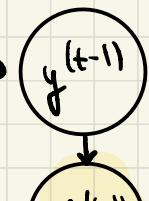
$$U^{k+1} = U^k - \alpha \frac{\partial L}{\partial U}$$

Pero el gradiente de  $L$  será una suma de los gradientes calculados en cada tiempo  $t$ :

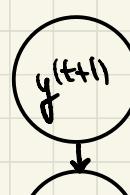
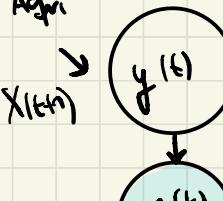
$$\frac{\partial L}{\partial V} = \sum_{t=1}^T \frac{\partial L_t}{\partial V}$$

$T$ : total de pasos de tiempo

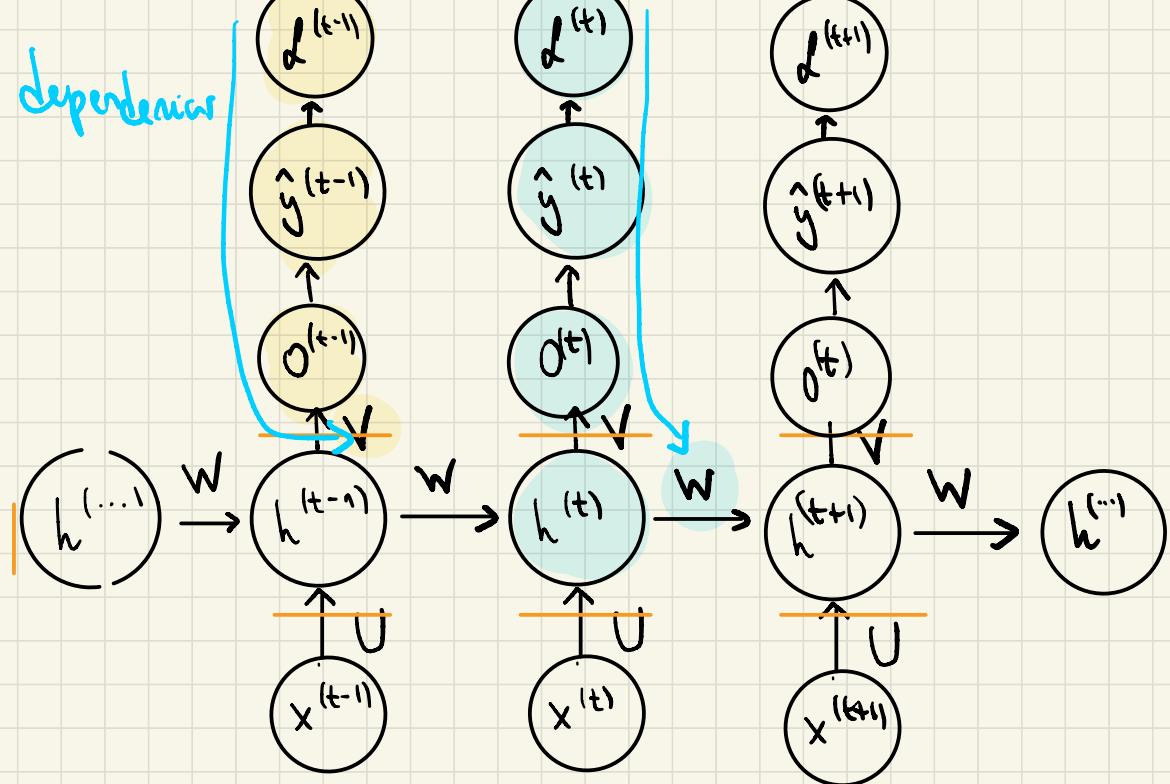
Aqui esperamos  
 $x^{(t)}$



Aqui  
 $x^{(t)}$



dependencias



Gradientes :

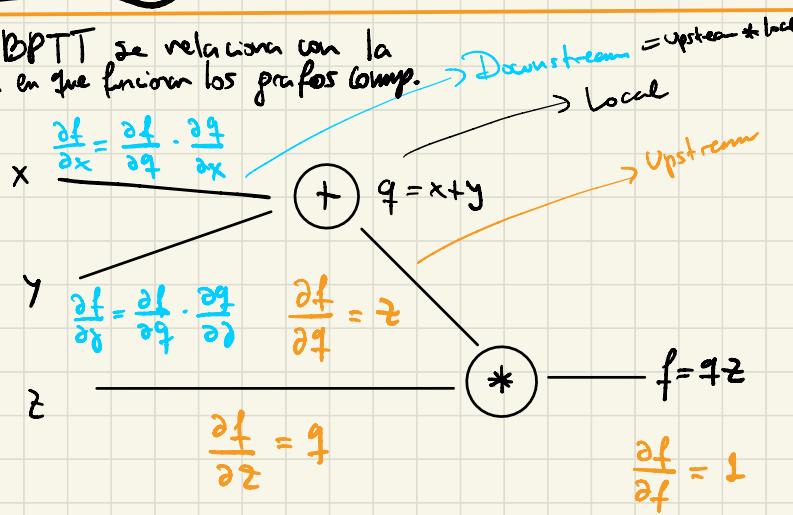
$$\frac{\partial f_t}{\partial v} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial v}$$

que depende  
de  $h_{t-1}$ ,  
 $h_{t-2}, \dots$   
 $h_0$

$$\frac{\partial f_t}{\partial w} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial w}$$

## Observación:

El BPPT se relaciona con la  
manera en que funcionan los grafos como:



## Upstream / Downstream / Local Gradients

$\frac{\partial f}{\partial x}$  no se puede calcular directamente

Requiere del uso de la regla de la cadena

Se denominan en inglés downstream gradient

$\frac{\partial f}{\partial q}$  Se denominan upstream gradient

$\frac{\partial q}{\partial x}$  Se denominan gradiente local

- Trabajar con grafos es efectivo computacionalmente.

- Esta terminología nos permite entender BPPT

\* El error es retropropagado a través del tiempo

- En nuestro modelo RNN existe un loop que puede traernos problemas al momento de calcular gradientes downstream pues, estamos multiplicando muchas veces los términos por los mismos parámetros.
- Si  $W > 1$  explota y si  $W < 1$  desaparece el gradiente.



Esto nos deja en problemas a la hora de propagar el gradiente. Problemas de memoria a largo plazo!

Observemos en detalle las ventas :

$$h_t = \sigma(U x_t + W h_{t-1} + b)$$

$$\hat{y}_t = \text{softmax} \left( \underbrace{\nabla h_t + c}_{\partial_t} \right)$$

Asumamos una función de pérdida dada por:

$$\mathcal{L}(y, \hat{y}) = \sum_{t=1}^T L_t(\hat{y}_t, y_t)$$

Vamos a  $t+1$  para el caso de  $W$ :

$$\frac{\partial f_{t+1}}{\partial W} = \frac{\partial f_{t+1}}{\partial \hat{y}_{t+1}} \cdot \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial W}$$

\*  $h_{t+1}$  depende parcialmente de  $h_t$

Es decir :

$$\frac{\partial f_{t+1}}{\partial W} = \frac{\partial f_{t+1}}{\partial \hat{y}_{t+1}} \cdot \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

Y a su vez  $h_t$  depende de  $h_{t-1}$ ,  
entonces podemos calcular en el tiempo  $t+1$ :

$$\frac{\partial f_{t+1}}{\partial W} = \sum_{k=1}^{t+1} \frac{\partial f_{t+1}}{\partial \hat{y}_{t+1}} \cdot \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

este es  
otra regla de  
lubricación.

Por ejemplo :

$$\xrightarrow{\text{Si } t=2} \sum \frac{\partial f_2}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial h_2} \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W} + \\ \quad // \quad // \quad \frac{\partial h_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial W}$$

¿Cómo revienta o desaparece el gradiente?

$$\frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_{k+1}}{\partial h_k}$$

- Estas derivadas son : derivo un vector respecto a un vector.



Esto es una matriz

- El cálculo de estas derivadas se escapa del curso.

Esta matriz Jacobiana es dada por :

$$\frac{\partial h_{t+1}}{\partial h_t} = \text{diag} \left( \underbrace{\sigma' \left( Ux_{t+1} + Wh_t + b \right)}_{\tilde{M}} W \right)$$

Y vamos a tener algo de la forma :

$$\tilde{M} = M \cdot (M + \Delta_1 M) \cdot (M + \Delta_2 M) \dots = \prod_{i=1}^t (M + \Delta_i M)$$

Si los autovalores de  $\tilde{M}$  se afectan tal que su mayor autovalor  $|\lambda| > 1$  entonces:

$$T(M + \Delta_i M) \longrightarrow \text{satura el gradiente}$$

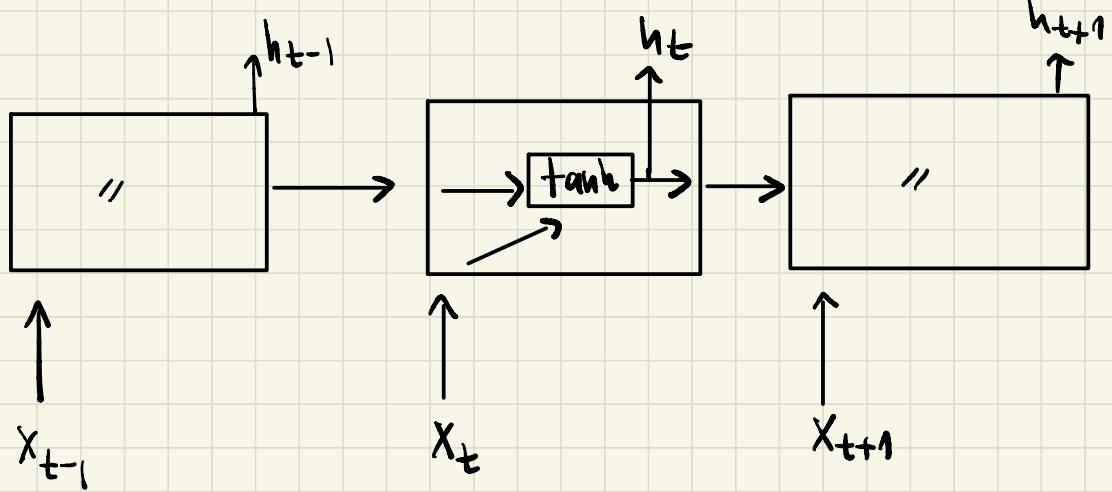
$$\text{Si } |\lambda| < 1 \longrightarrow \text{Cesa el gradiente}$$

---

### Long - Short time memories

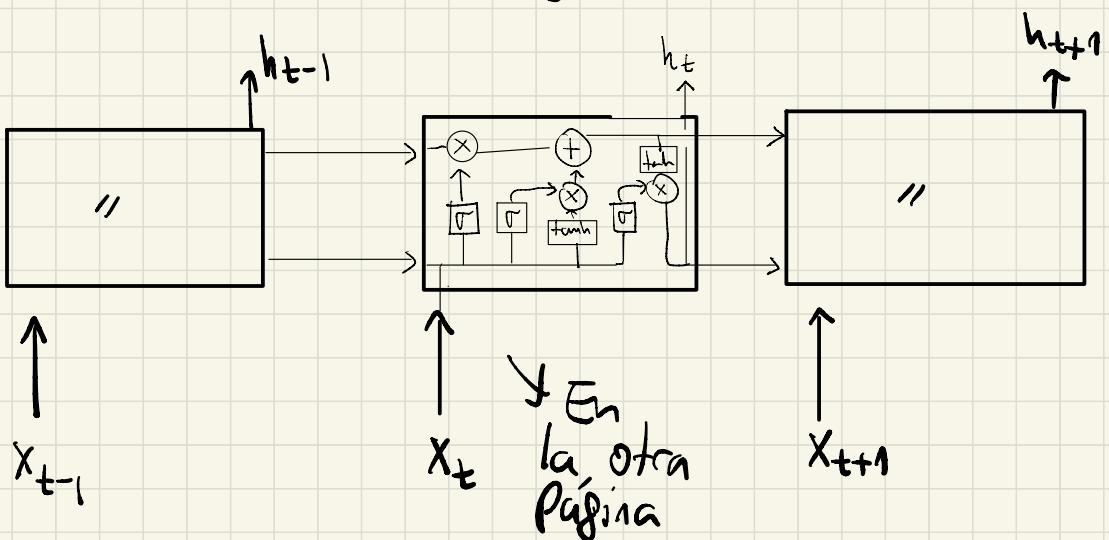
- \* Mejoran la capacidad de almacenar memorias de largo plazo.
- \* El modelo dura patrones (Patrones que dependen de cadenas largas de información).
- \* Logra sostener memorias que dependen de 100 etapas finitas sobre ese valor tiene problemas.
- \* Dos trabajos contribuyen al desarrollo de este modelo:
  - i) Hochreiter and Schmidhuber 1997
  - ii) Gers et al 2000

# Modelos de Red Recurrente ya visto

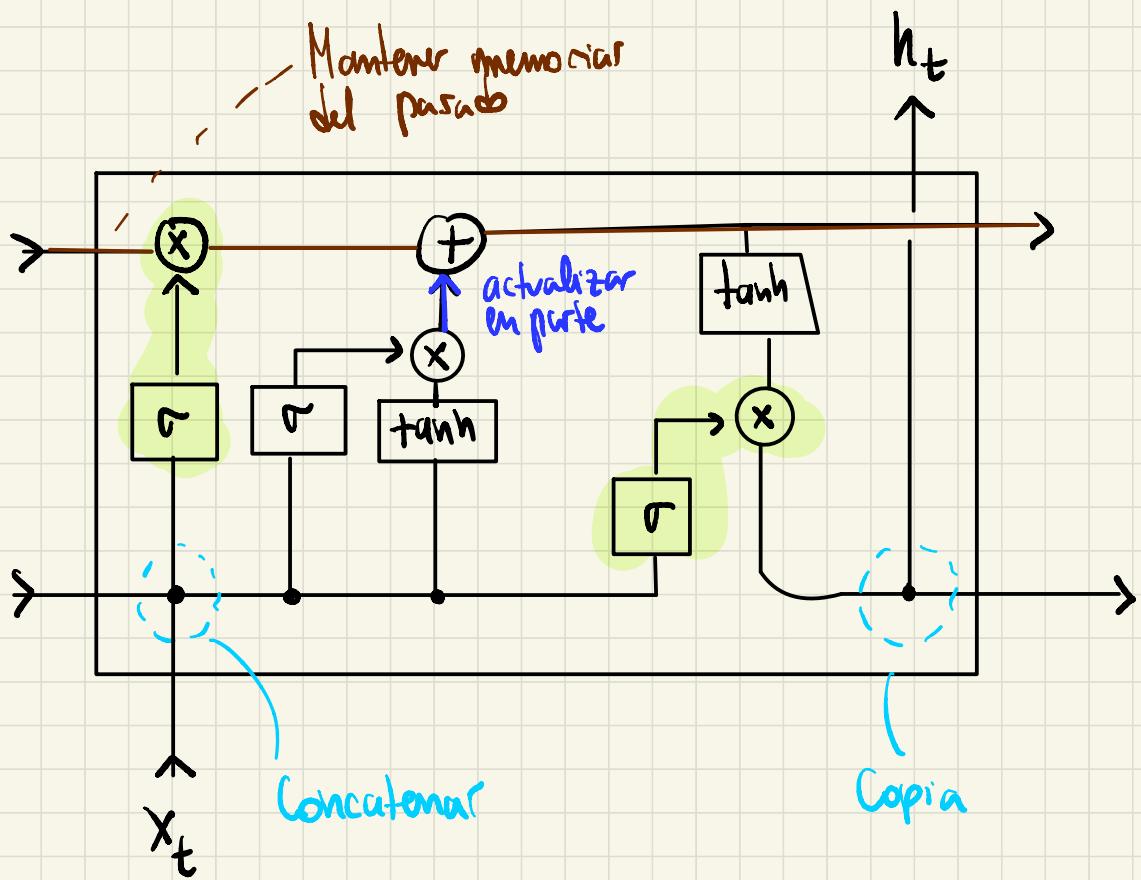


"Understanding LSTM Networks" (Diagramas de este artículo).

La evolución del modelo radica en complicar la arquitectura de la siguiente forma



Mantener memoria  
del pasado



$\times$ ;  $+$  : Operaciones punto a punto!

$\sigma$ ; tanh : (Red + Función de activación)

: Compuestas o "Gates"

$\sigma$ : sigmoid

i) Gates



Wantas voy a  
recordar u olvidar  
de las componentes

El  
modelo  
tiene 3  
componentes

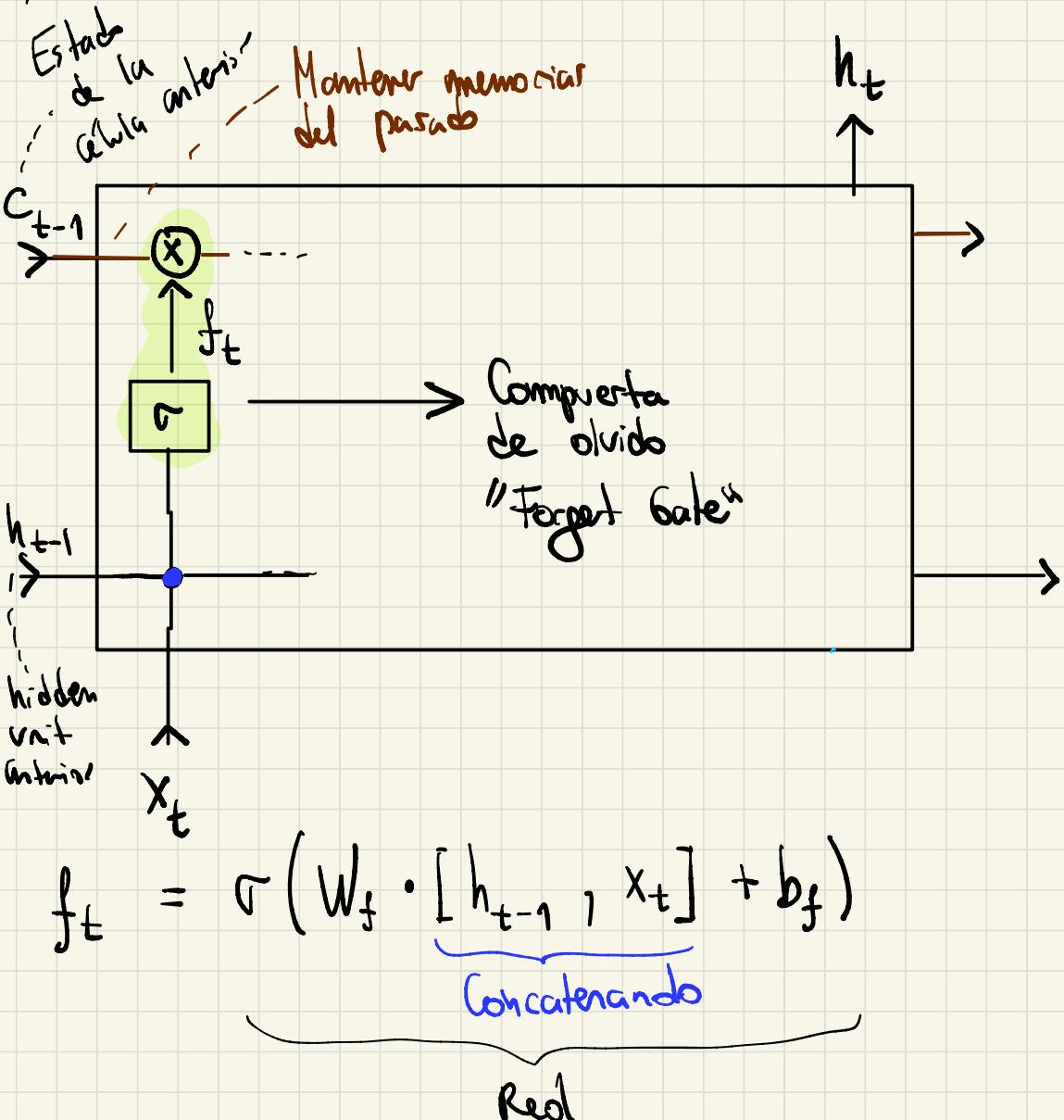
ii)



información que sufre bajas alteraciones  
y viene del "pasado".

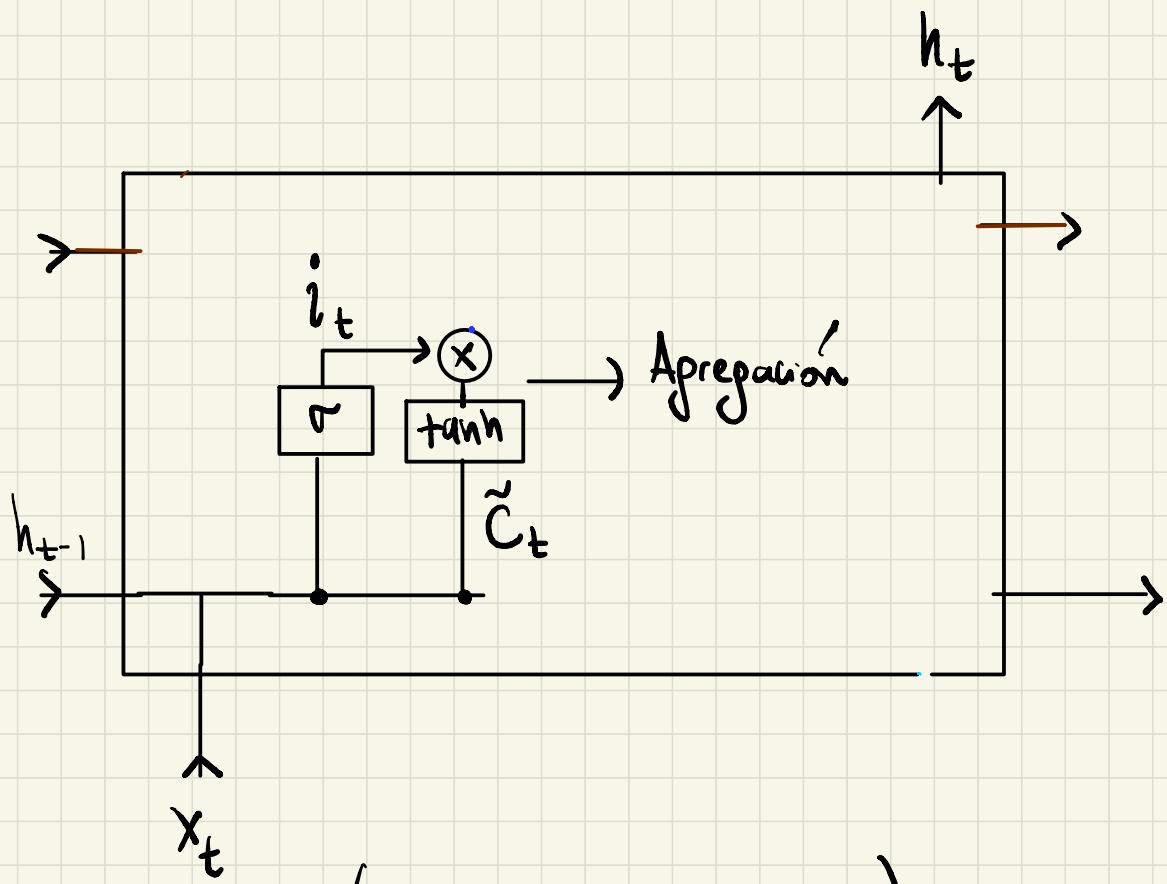
El modelo tiene  
1 linea horizontal  
para mantener info del  
pasado.

# Paso a Paso : Paso L : Olvidar



- Si  $\sigma = 1$  : No olvida
- Si  $\sigma = 0$  : Olvida por completo

Pass 2: Almacenar info del nuevo estado

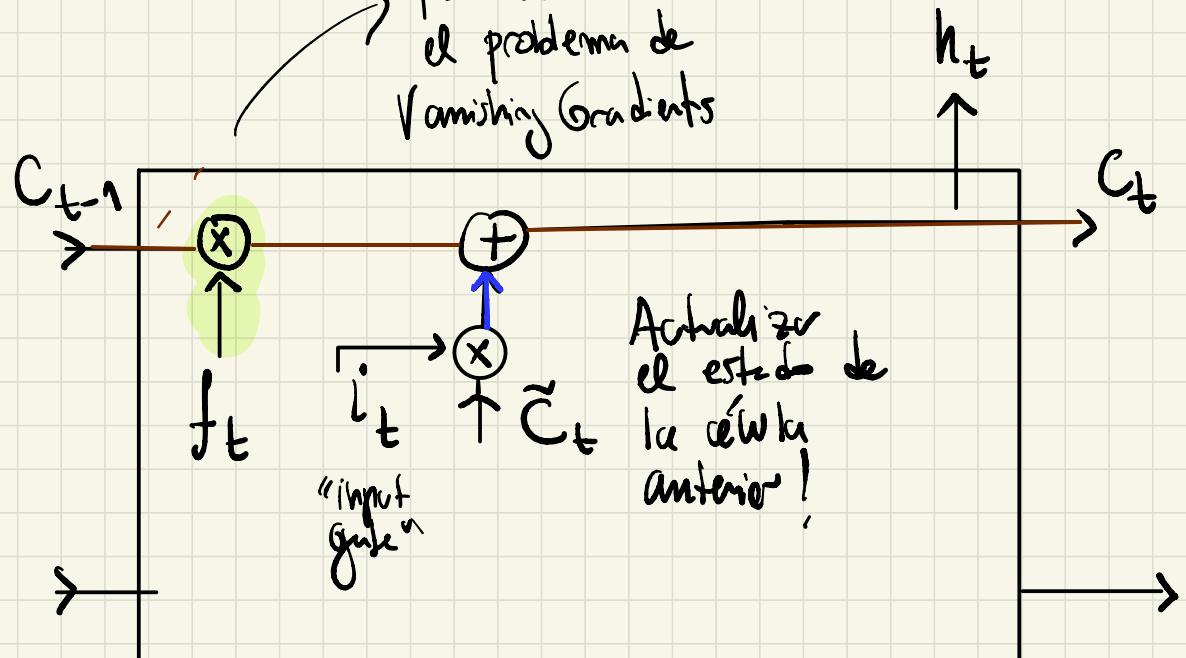


$$i_t = \sigma \left( W_i [h_{t-1}, x_t] + b_i \right)$$

$$\tilde{C}_t = \tanh \left( W_c [h_{t-1}, x_t] + b_c \right)$$

## Paso 3 : Actualizar

Permite evitar  
el problema de  
Vanishing Gradients



Actualizar  
el estado de  
la célula  
anterior!

"input gate"

↓ misma dimensión

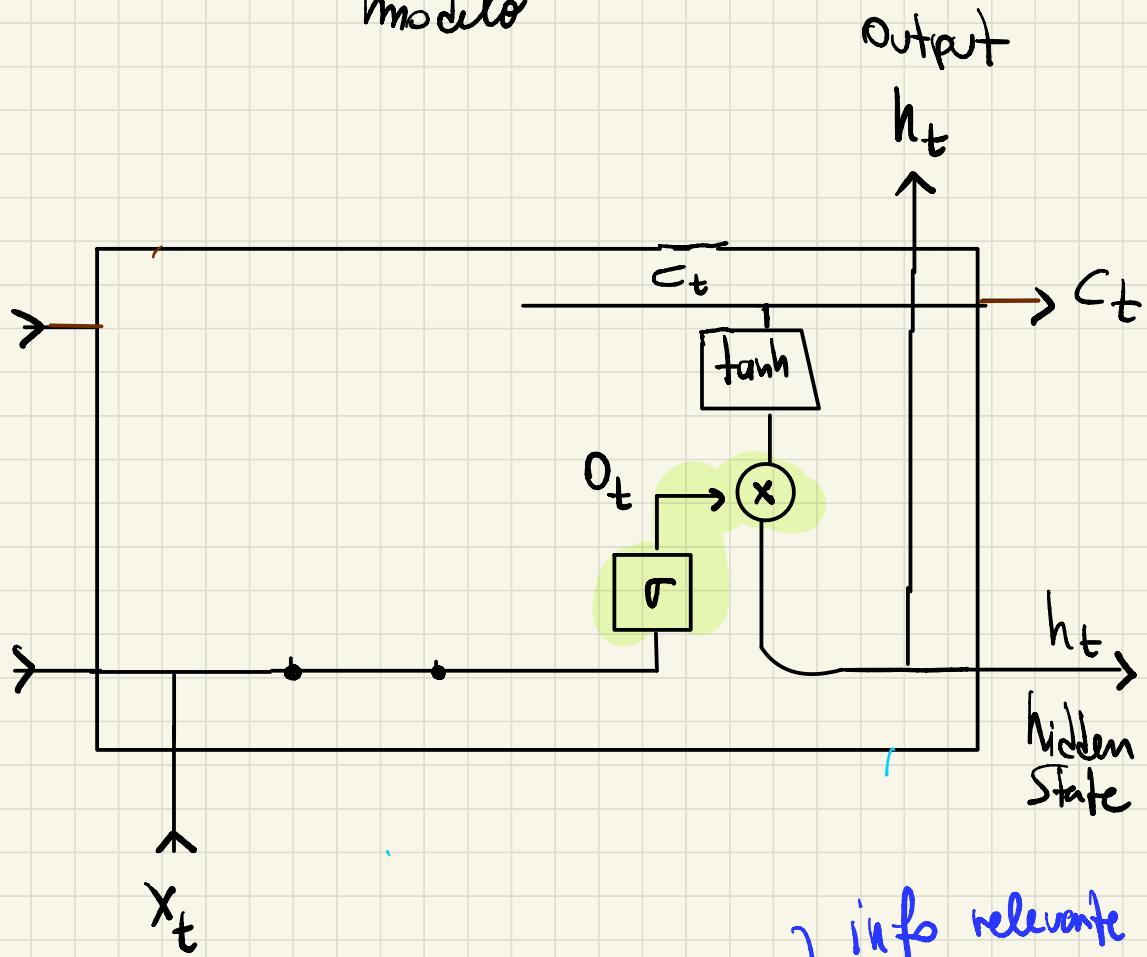
$x_t$  Misma dimensión

$$C_t = \underbrace{f_t * C_{t-1}}_{\text{olvidamos}}$$

$$+ \underbrace{i_t * \tilde{C}_t}_{\text{agregamos}}$$

actualizamos!

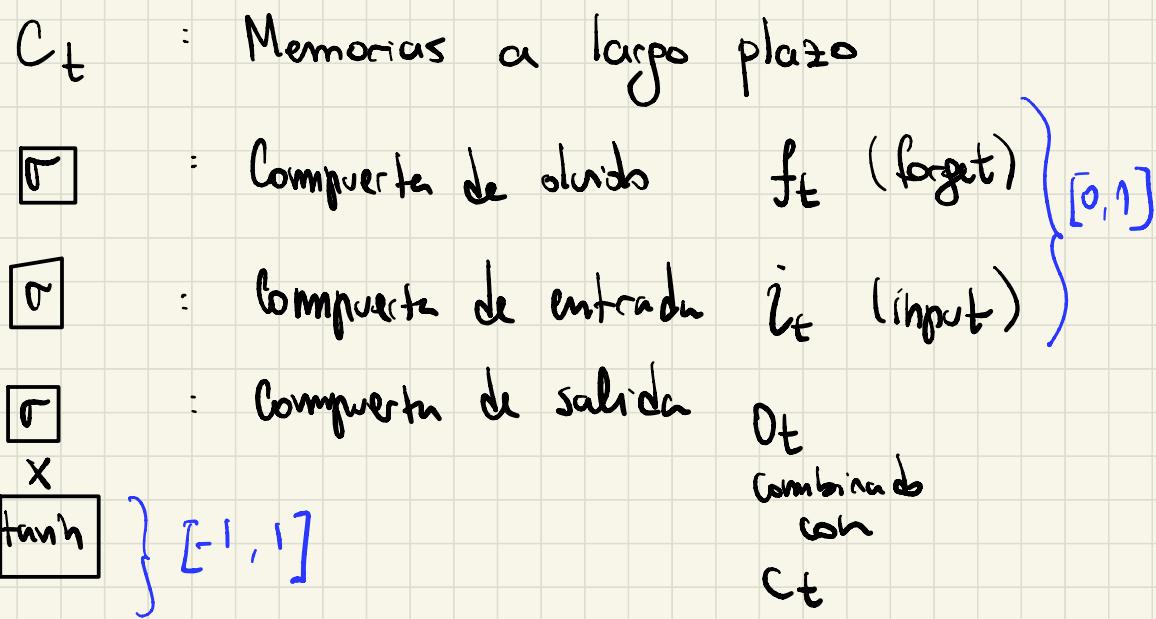
## Paso 4 : Definir una salida del modelo



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

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

} info relevante de este estado  
} influencia del estado actual y pasado sobre la salida



\* Las compuestas actúan como filtros!

\* Es una generalización / Extensión de una RNN.

\* La RNN está contenida en tanh ( $E$ , una red densa)

\* El "hidden state" es la memoria a corto plazo.

Ejemplo : Paso 1 : olvidar

$$C_{t-1} = [2, 4, 6]$$

$$f_t = [0, 1, 1]$$

$$f_t * C_{t-1} = [0, 4, 6] = C_t^f$$

↓  
Olvidaremos  
este estado!

Paso 2/3: Calcular el input + Actualizar

$$i_t = [1, 1, 0]$$

$$\tilde{C}_t = [-1/2, -1, 1/2]$$

$$C_t^i = \tilde{C}_t * i_t = [-1/2, -1, 0]$$

$$C_t = C_t^f \oplus C_t^i$$

$$= [0, 4, 6] + [-1/2, 1, 0]$$

$$= [-1/2, 5, 6]$$

info  
nueva  
del  
input

Combinación  
del estado  
anterior  
y la nueva

info  
del pasado  
representación  
del estado  
actual  
para  
el ct

Nuevos estados

$$h_t = [-1/2, 0, 1]$$

Paso 4: Salida

$$O_t = [1, 0, 1]$$

$$h_t = [1, 0, 1] * \tanh([-1/2, 5, 6])$$

$$\begin{matrix} & & & // \\ -0,5 & 1 & 1 \\ ss & ss & ss \end{matrix}$$

Existen Variantes de este modelo : GRU

---

"Gated Recurrent Unit". Tarea!

Notebook: Implementación