Tema 6.2
Redes recurrentes

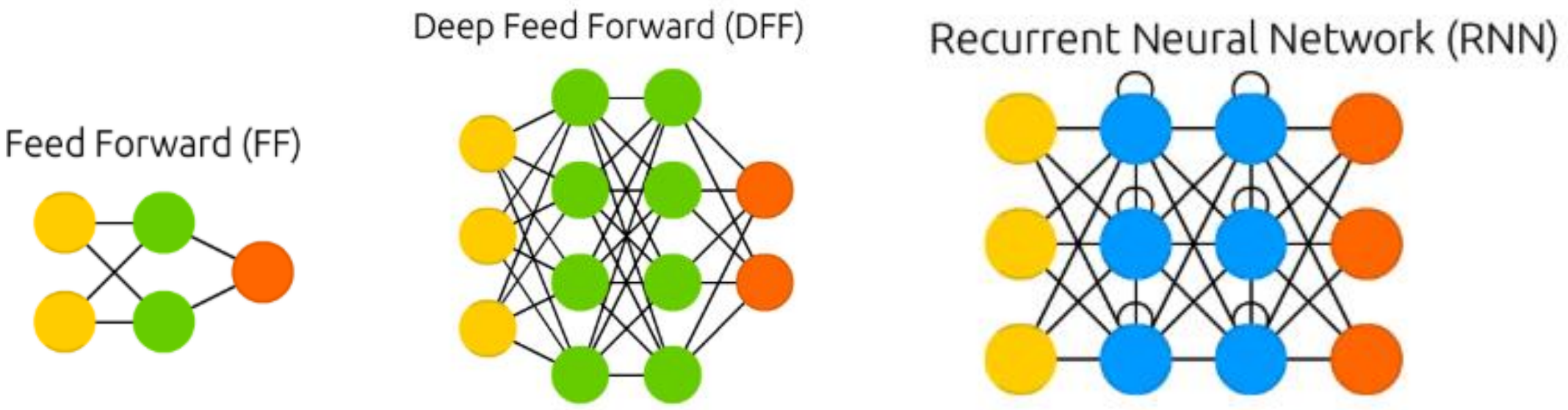
Miguel Ángel Martínez del Amor
Deep Learning
Departamento Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Contenido

- Redes recurrentes neuronales (RNN)
- Long Short-Term Memory (LSTM)
- Gated Recurrent Units (GRU)

Redes recurrentes

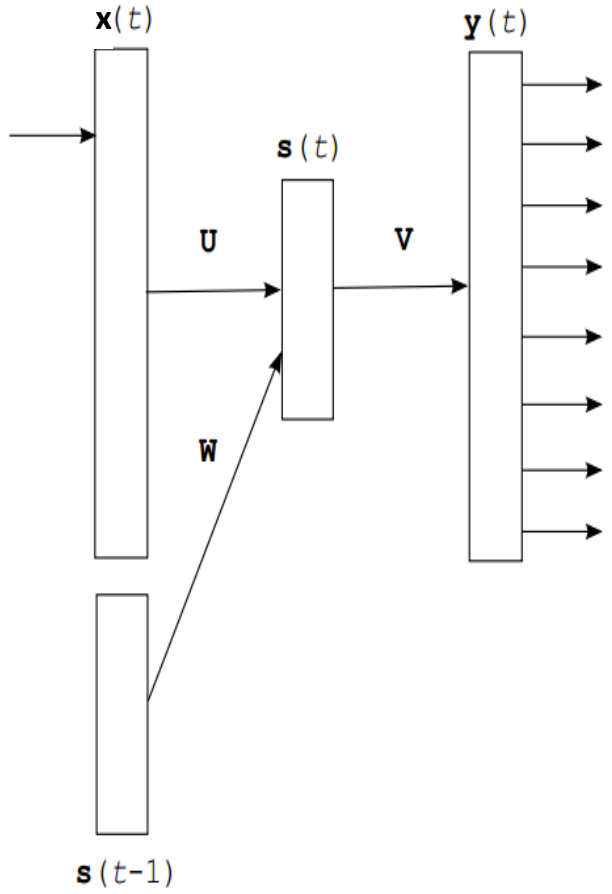
- Redes feed-forward (FF) vs recurrentes (RNN)



http://www.asimovinstitute.org/neural-network-zoo/

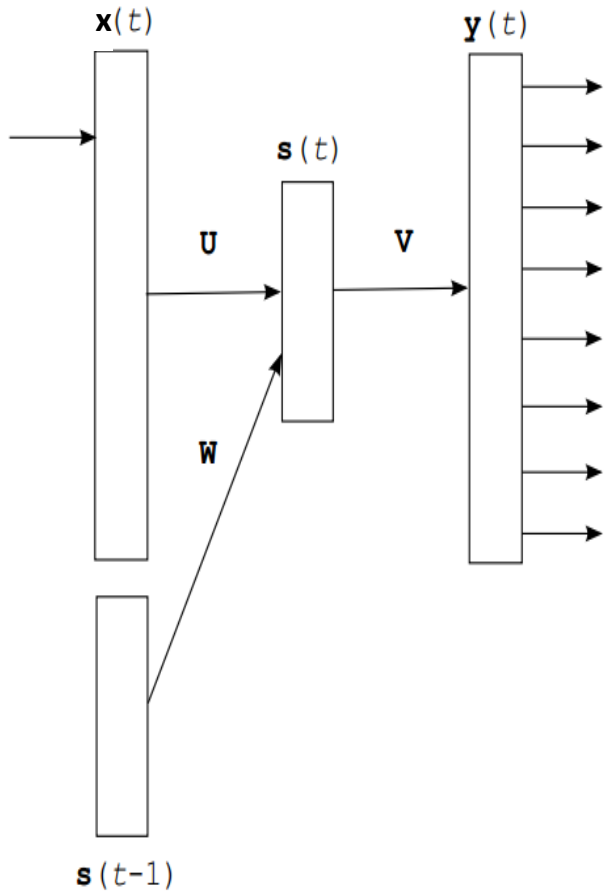
Redes recurrentes

- Son un framework simple y genérico para modelos secuenciales
- **Arquitectura**:
  - Capa de entrada
  - Capa **oculta** con conexiones recurrentes
  - Capa de salida
- En teoría, la capa oculta puede aprender a representar **memoria ilimitada**
- También llamado red Elman (*Finding structure in time*, Elman 1990)



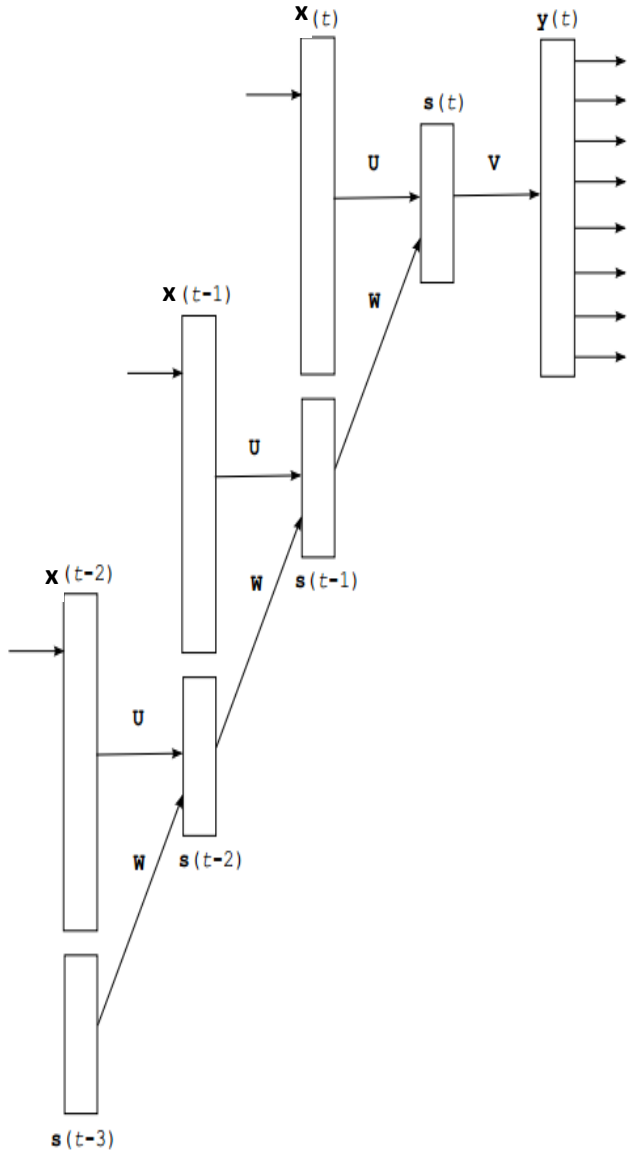
Redes recurrentes

- Podemos procesar una secuencia de vectores **x** aplicando una fórmula recurrente en cada instante de tiempo.
- Capa oculta: es un **estado oculto (s)**, que se actualiza y se emplea para el siguiente instante.
- Tres conjuntos de pesos: ***U, W, V***
- Sesgos o bias: ***b, c*** (no en la figura)
- $s(t) = f(b + Ux_t + Ws_{t-1})$, donde *f* es la función de activación sigmoide o tanh
- $\mathrm{y}(t) = g(c + Vs_t)$, donde *g* es habitualmente la función softmax



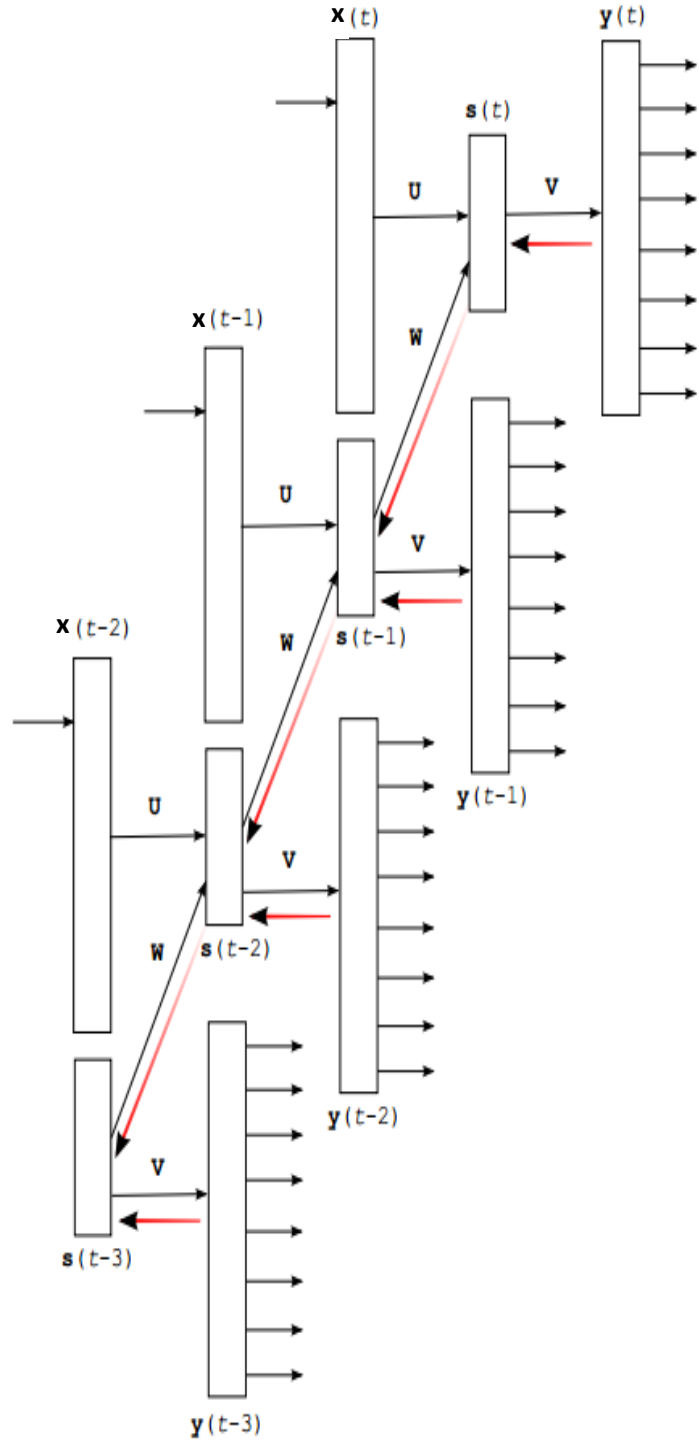
Redes recurrentes

- **Entrenamiento** de redes recurrentes:
  - ***Backpropagation Through Time (BPTT)***
  - El valor de salida depende en el estado en la capa oculta, que depende de los estados previos en la capa oculta, y éste de todas las entradas previas.
  - Una RNN se puede ver como una red feedforward muy profunda con pesos compartidos
- Intuitivamente: **desenrollamos** la RNN en el tiempo, obteniendo una red profunda con pesos **U** y **W** compartidos.
- Desenrollar tan solo unos cuantos pasos suele ser suficiente (*Truncated BPTT*)



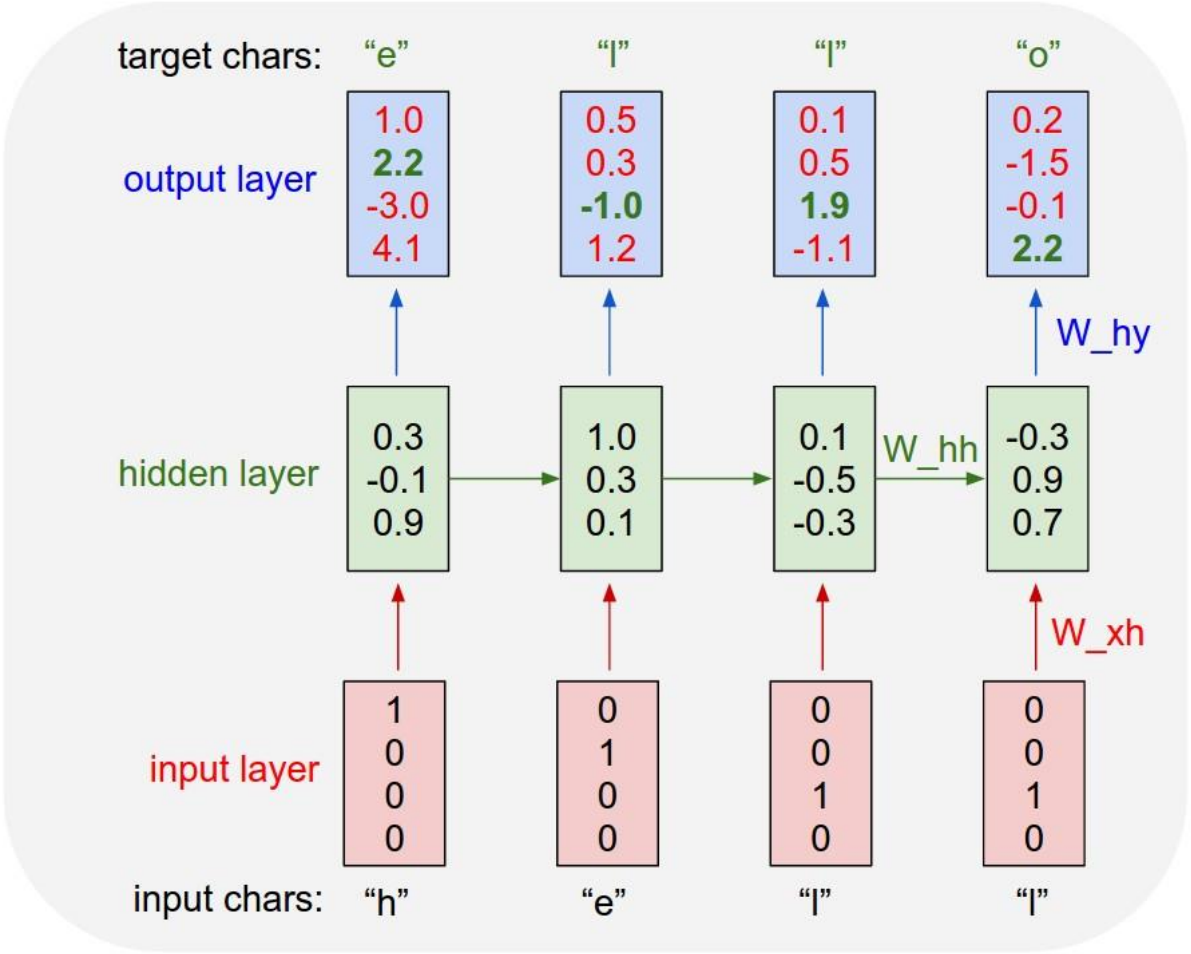
Redes recurrentes

- **Entrenamiento** de redes recurrentes:
  - Se suele entrenar unfolded RNN usando **BPTT** + SGD
  - En la práctica, se suele limitar el número de pasos de **desenrollado a 5 – 10**.
  - Es computacionalmente más eficiente propagar gradientes después de **batch** de ejemplos.



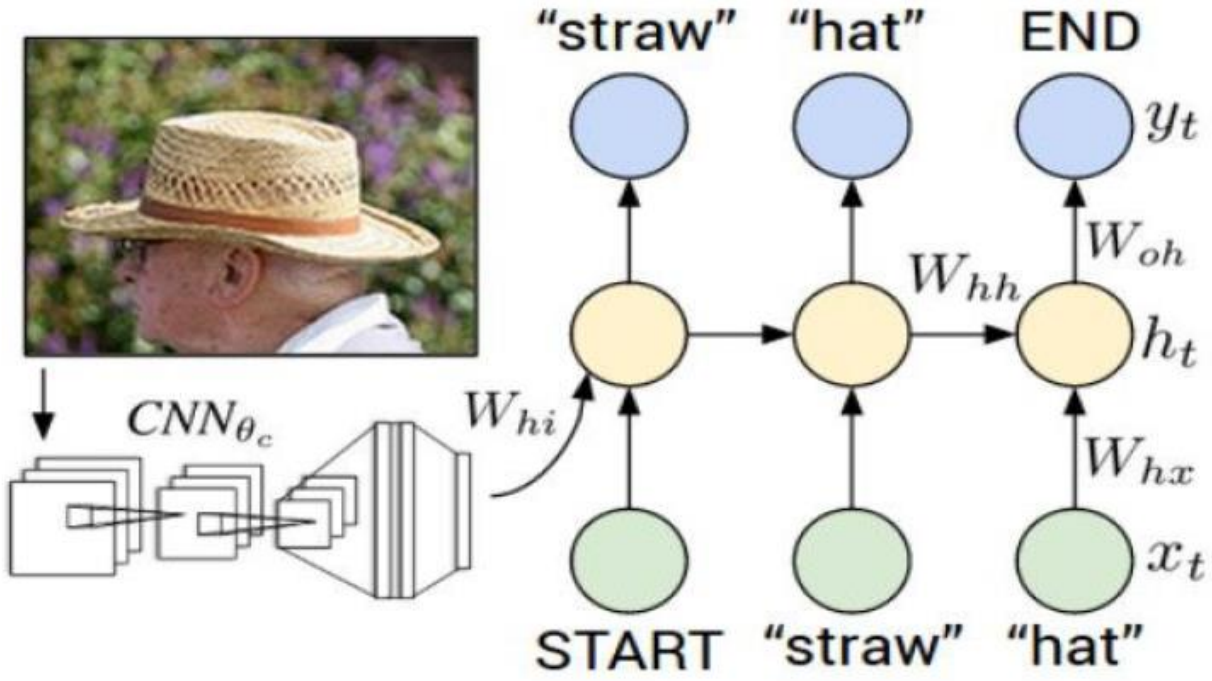
Redes recurrentes

- Ejemplo 1:
  - Entrenar que diga “**hello**”
  - Vocabulario de 4 letras {h,e,l,o}
  - Hay que optimizar la red para que las salidas sean como los valores en verde.



Redes recurrentes

- Ejemplo 2: [Karpathy, Fei-Fei 2015]
  - **Image captioning**
  - La red convolucional como extractor de características (VGG16)
  - Se inserta en la red recurrente a través de la capa oculta de forma ponderada con unos pesos.
  - Tokens especiales: START, END



Redes recurrentes

- Ejemplo 2 (image captioning)

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

Redes recurrentes

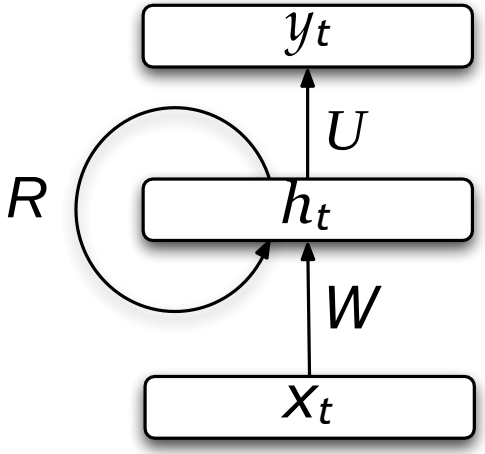
- Problema del **desvanecimiento del gradiente**: [Bengio et al 1994]
  - Conforme propagamos los gradientes atrás en el tiempo, usualmente su magnitud decrece drásticamente.
  - En práctica, aprender **dependencias a largo plazo** en el tiempo en los datos es **difícil** con una arquitectura RNN simple.
- Problema de la **explosión del gradiente**:
  - A veces, los gradientes comienzan a crecer exponencialmente durante backpropagation a través de los pesos recurrentes
  - Sucede rara vez, pero el efecto es catastrófico: cambios bruscos en los pesos, destruyendo lo que hayas aprendido hasta el momento.
  - Una razón principal por las que las RNN son supuestamente **inestables**.
  - Solución simple: normalizar o restringir los valores de los gradientes.

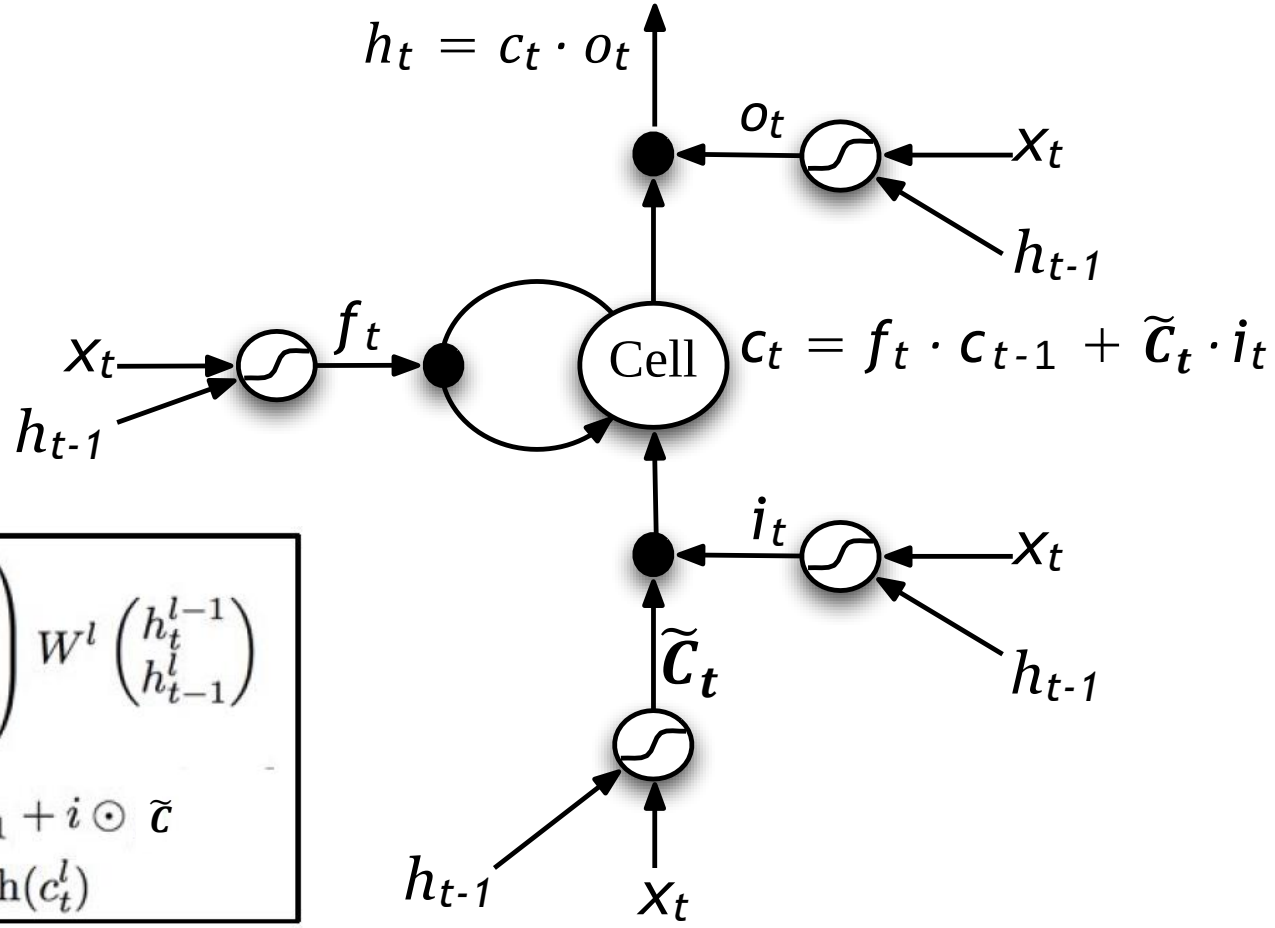
Long Short-Term Memory (LSTM)

- **Objetivo**: paliar el problema de desvanecimiento del gradiente en las redes recurrentes simples [Hochreiter y Schmidhuber 1997]
- Es una variante de una RNN, pero con una vía secundaria que permite pasar información a un instante de tiempo más lejano
- Incluye **células de memoria explícitas** para guardar las activaciones a corto plazo (**short-term**)
- Versiones multi capa trabajan bien en tareas que tienen dependencias a medio plazo

Long Short-Term Memory (LSTM)

- Esquema de una unidad LSTM





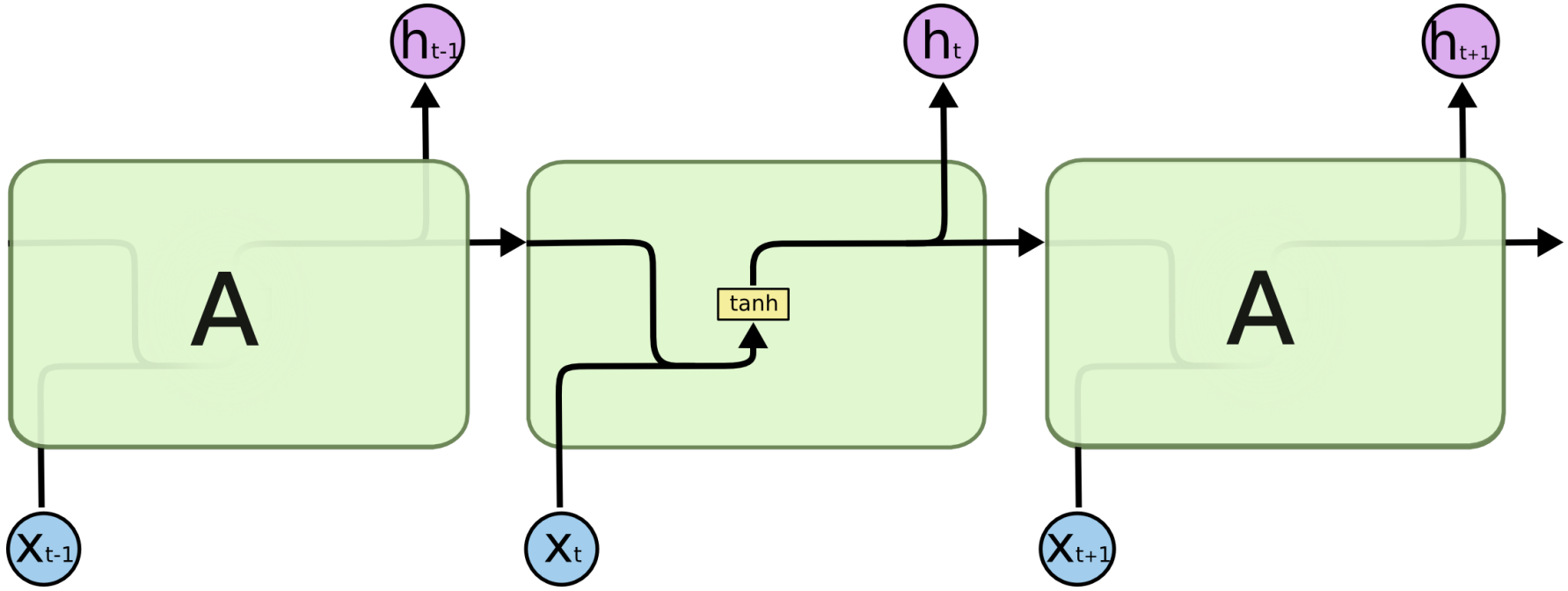
$$\begin{pmatrix} i \\ f \\ o \\ \tilde{c} \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot \tilde{c}$$

$$h_t^l = o \odot \tanh(c_t^l)$$

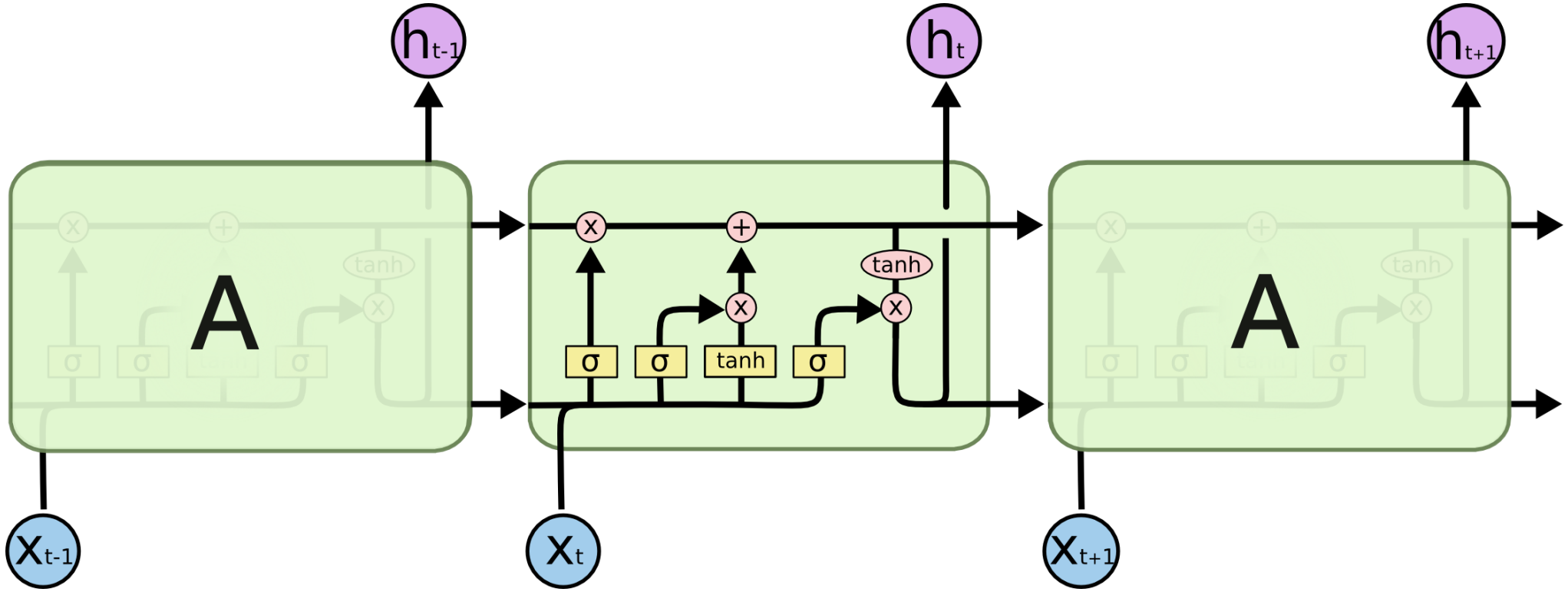
Long Short-Term Memory (LSTM)

- Veamos paso por paso los elementos de una unidad LSTM
- Una unidad RNN simple contiene solo una capa oculta



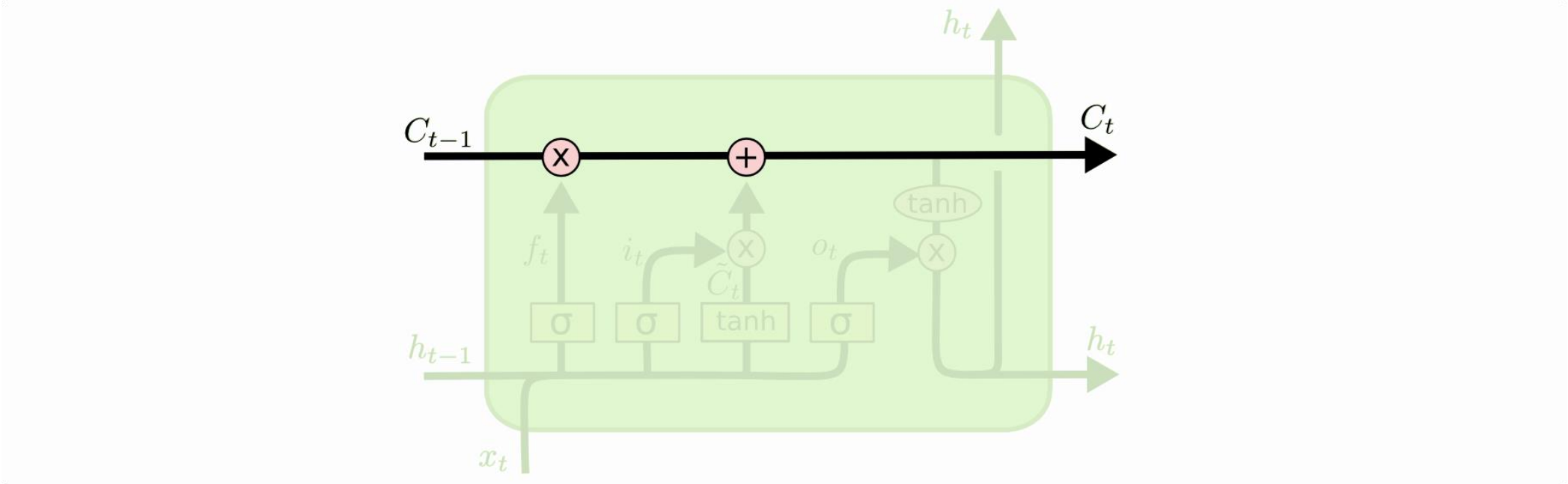
Long Short-Term Memory (LSTM)

- Una unidad LSTM contiene 4 capas ocultas y una célula de memoria



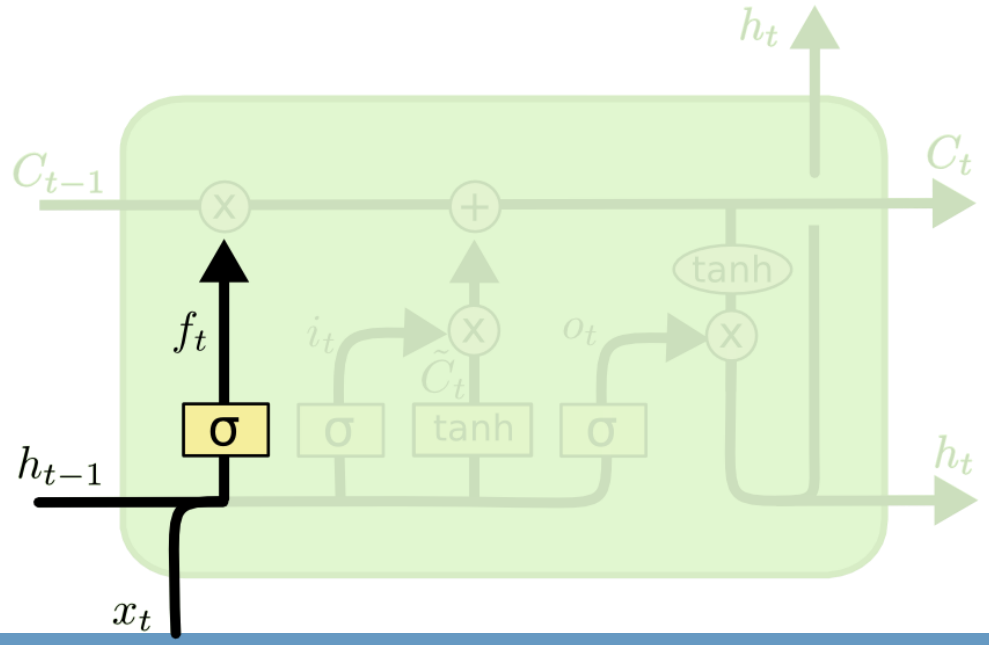
Long Short-Term Memory (LSTM)

- La célula de memoria puede llevar información a otras.
- Pensemos en ella como si fuese un contador.



Long Short-Term Memory (LSTM)

- Primer paso, **Forget gate**: ¿mantener u olvidar el estado de la célula?
  - Si f_t es 0 o 1, resetea o mantiene el valor del contador C_{t-1}
  - Cálculo con sigmoide según entrada en instante t y estado h anterior.

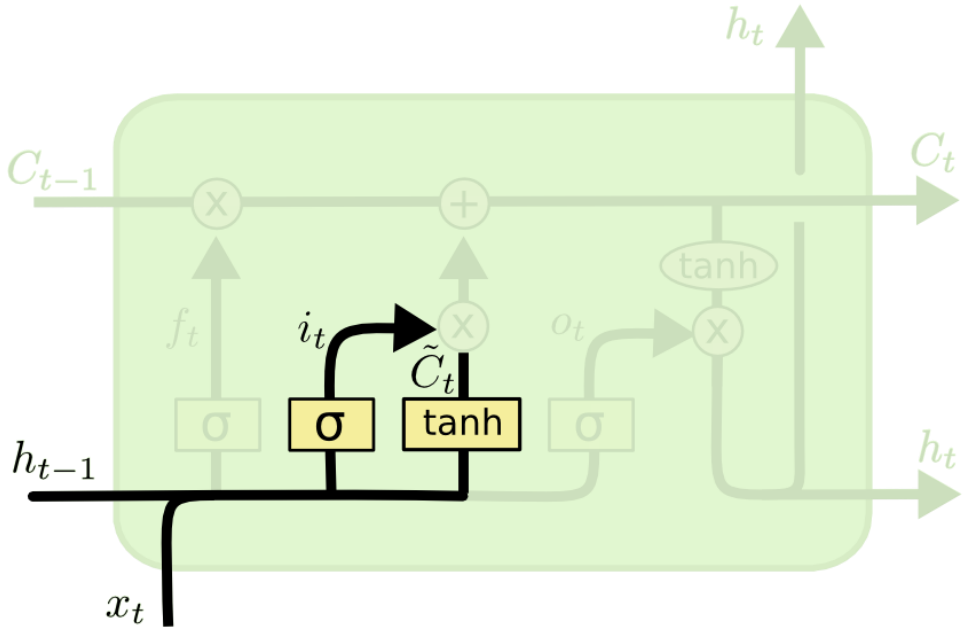


$$c_t = f_t \cdot c_{t-1} + \tilde{c}_t \cdot i_t$$

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

Long Short-Term Memory (LSTM)

- Segundo paso, (External) **Input Gate**: ¿Cuánto actualizamos la célula?
  - Si $\boldsymbol{i_t}$ is 0 o 1, actualiza o no actualiza el contador $\boldsymbol{C_t}$
  - $\boldsymbol{\tilde{C}_t}$ es el vector de nuevos valores candidatos para $\boldsymbol{C_t}$



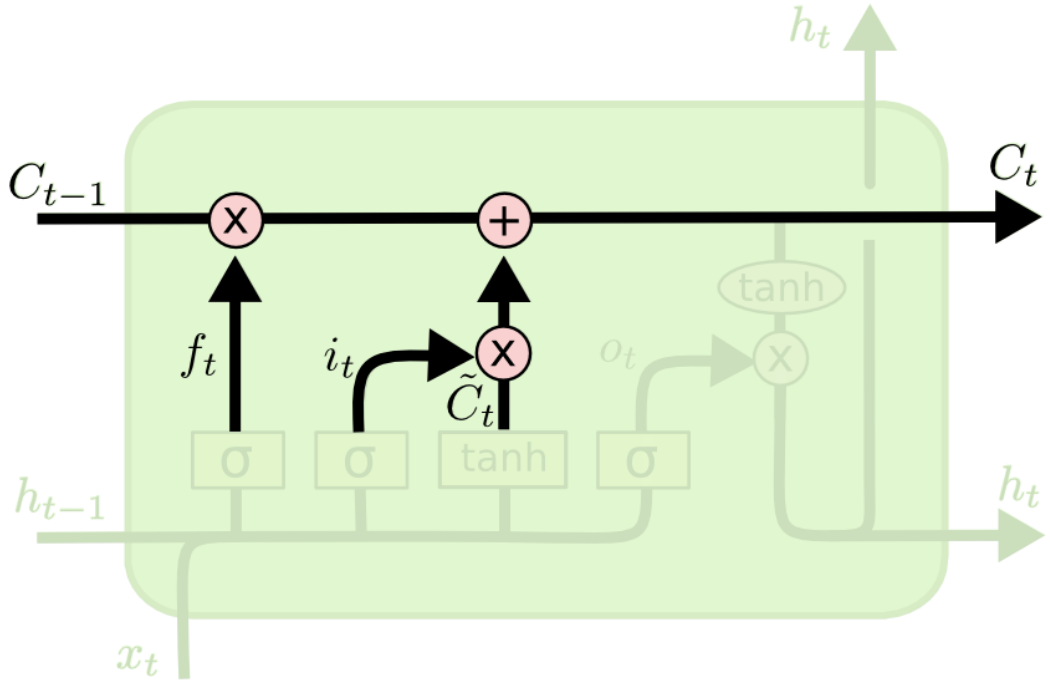
$$\boldsymbol{c_t} = f_t \cdot c_{t-1} + \boldsymbol{\tilde{C}_t \cdot i_t}$$

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

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

Long Short-Term Memory (LSTM)

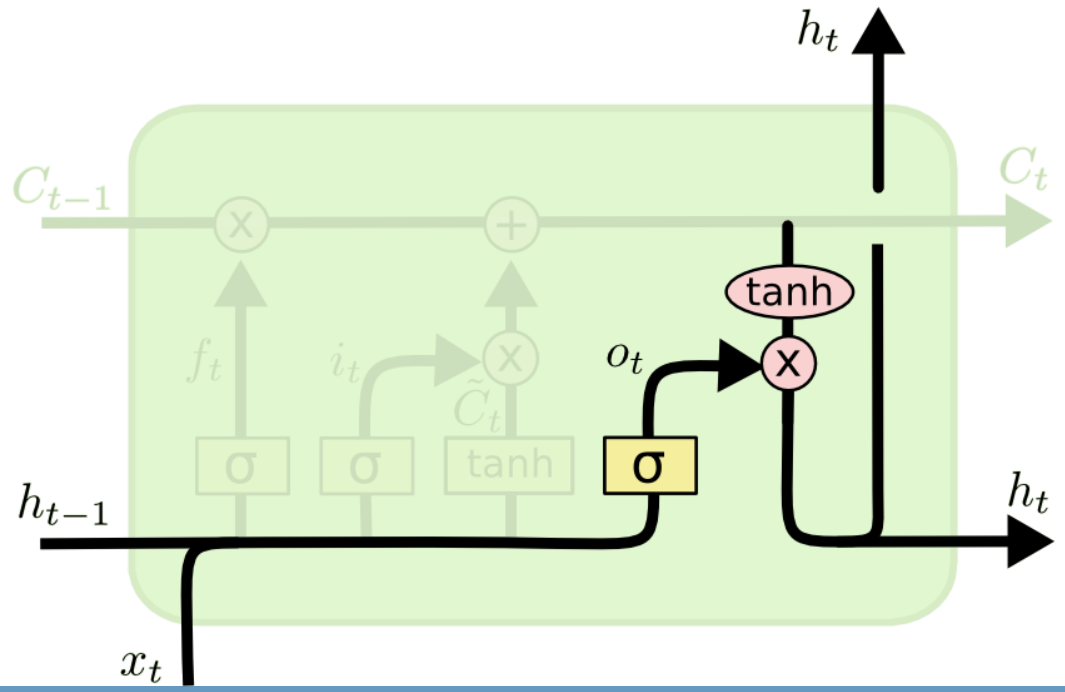
- Tercer paso, calcular el siguiente valor de la célula
  - Sumando lo recibido por el forget y el input gate



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

Long Short-Term Memory (LSTM)

- Cuarto paso, calcular la salida (siguiente estado h_t)
  - Regulado por el **Output Gate** (o_t)

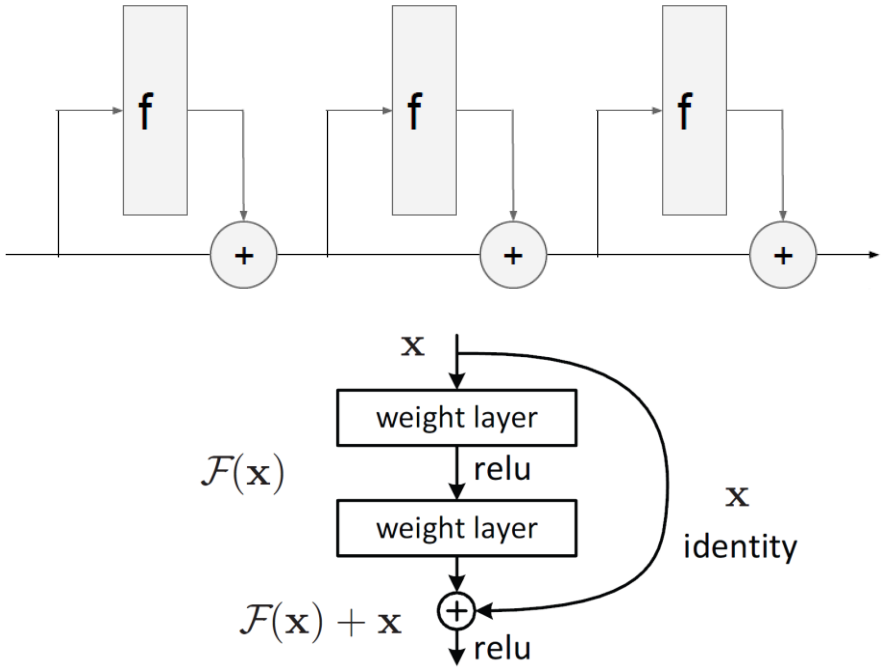


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

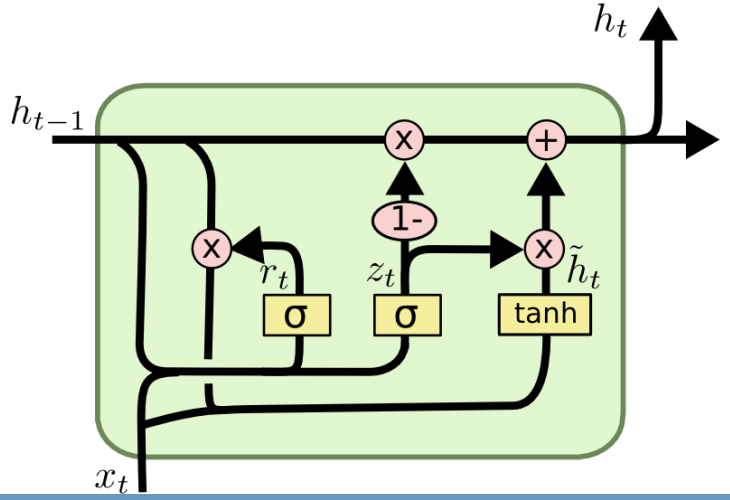
Long Short-Term Memory (LSTM)

- Utilizado en modelos de secuencias con éxito
- **Ventaja**: técnica para evitar desvanecimiento del gradiente (con idea parecida a la utilizada en ResNet)
- **Desventaja**: Requiere de muchos parámetros: $W_C\ W_f\ W_i\ W_o\ b_C\ b_f\ b_i\ b_o$
- Variantes:
  - peep-hole, coupled forget and input gates,
  - Gated Recurrent Unit (GRU)



Gated Recurrent Units (GRU)

- **Objetivo**: reducir la complejidad de las unidades [Cho et al. 2014]
- Combina el Forget y el Input gate en una simple **Update gate (z_t)**.
- **Reset gate** (r_t) indica cuanto se usa del estado anterior
- h_t es **estado y salida** (célula de memoria y estado)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Recapitulando

- Las redes **RNN** sufren del problema **vanishing** and **exploding gradient**
- Variante para paliar el problema: **LSTM**
- Variante para reducir complejidad de LSTM: **GRU**
- ¿Cuál mejor?
  - Comparativa por [Greff et al. 2015] indica que **todas muy parecidas a una LSTM estándar** (lo más crítico, la forget y la output gate)
  - Comparativa por [Jozefowicz et al. 2015] indica que **GRU mejora a la LSTM** estándar en solo algunas tareas de **modelado de lenguaje**. Pero una LSTM con un alto bias en el forget gate (p.ej. 1), funciona mejor que el resto de variantes.