Redes Neuronales Recurrentes

Clase IV - Introducción al Aprendizaje Profundo

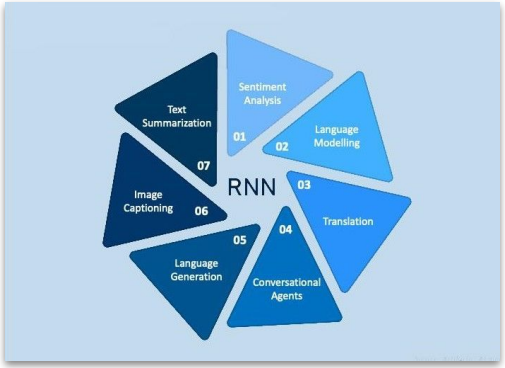
Repaso de lo que vimos hasta ahora

- Introducción y Construcción de Redes Neuronales
- Backpropagation y Optimización de Hiperparametros
  - Aplicadas a problemas de regresión y clasificación
- Redes Convolucionales aplicadas a imágenes.

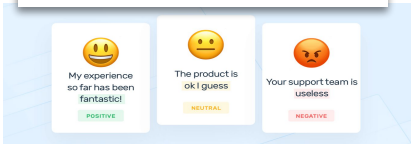
Las redes neuronales que vimos en la primera parte de la materia **no son capaces de manejar datos secuenciales**, consideran sólo el punto o ejemplo actual y tampoco pueden memorizar o tener en cuenta la información de los inputs previos.

Redes Neuronales Recurrentes (RNN)

Aplicaciones



Sentiment analysis



Time series prediction

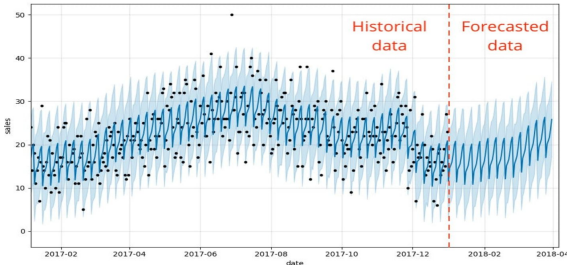


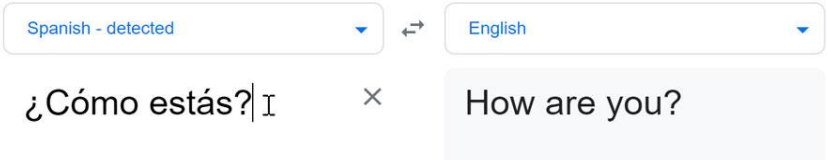
Image captioning

"man in black shirt is playing guitar."

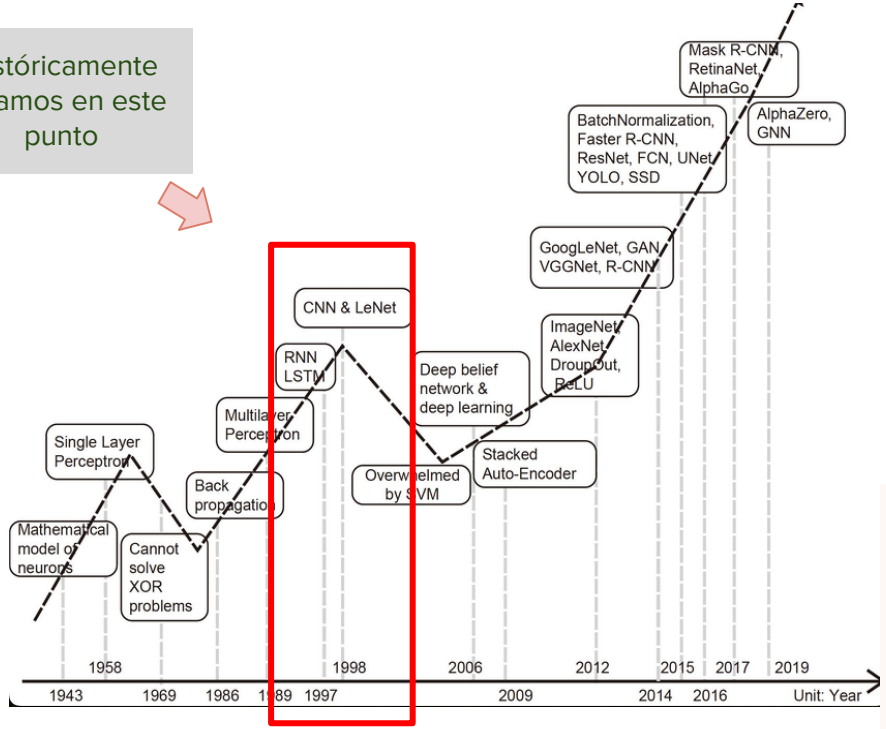
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

Machine Learning Translation



Históricamente estamos en este punto



› Neural Comput. 1997 Nov 15;9(8):1735-80. doi: 10.1162/neco.1997.9.8.1735.

Long short-term memory

S Hochreiter [1], J Schmidhuber

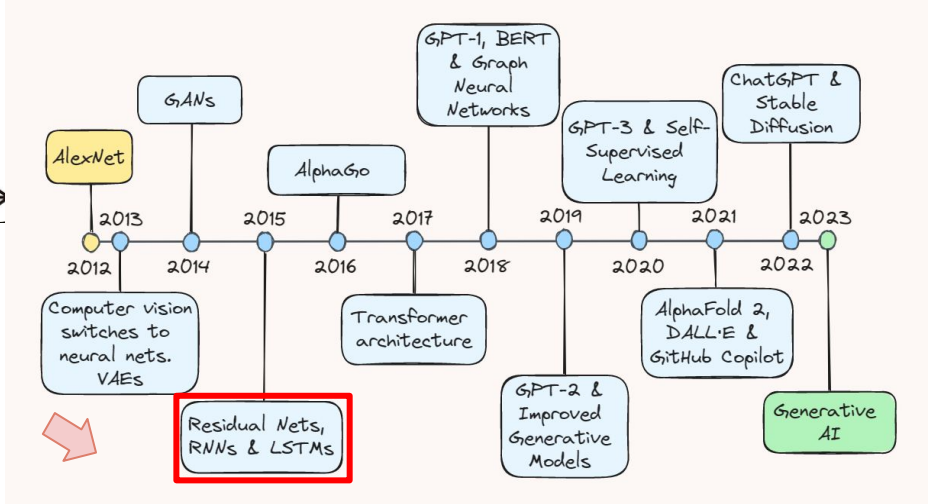
Letter | Published: 09 October 1986

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Cite this article

Y aquí!



Secuencia de Datos y Redes Recurrentes

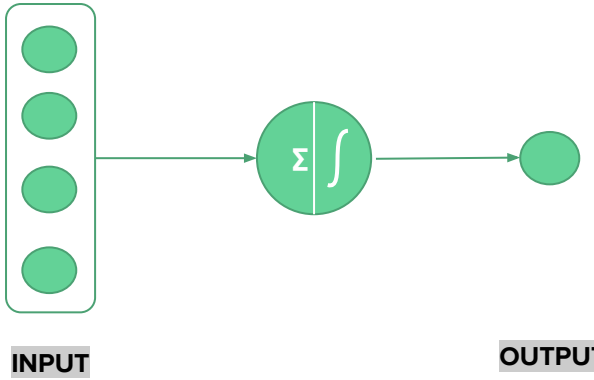
Una secuencia de datos es un conjunto de datos **dados en un orden específico** de tal manera que el orden de esos datos es importante para realizar predicciones o interpretar los datos.

Las Redes Neuronales Recurrentes son un tipo de red neuronal **diseñada específicamente para trabajar con datos secuenciales**.

A diferencia de las redes neuronales tradicionales (feedforward), las RNN tienen **conexiones cíclicas** que permiten que la información persista a lo largo del tiempo.

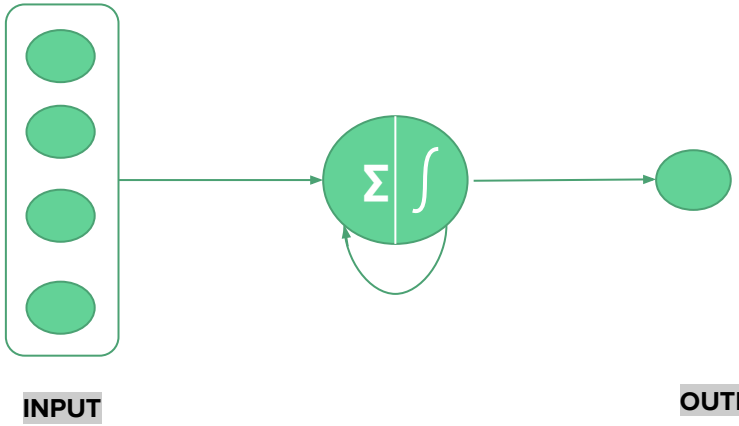
Redes Recurrentes

Unidades neuronales “tradicionales”



Procesan datos de manera independiente, sin tener en cuenta la relación entre diferentes entradas. El flujo de información va en una sola dirección: de la capa de entrada a la capa de salida.

Unidades recurrentes



Tienen una estructura interna donde **cada nodo tiene una conexión recurrente a sí mismo**, lo que permite que la red almacene información de entradas pasadas. Esta característica es útil para tareas que involucran secuencias o datos con dependencias temporales, como el procesamiento del lenguaje natural o la predicción de series temporales.

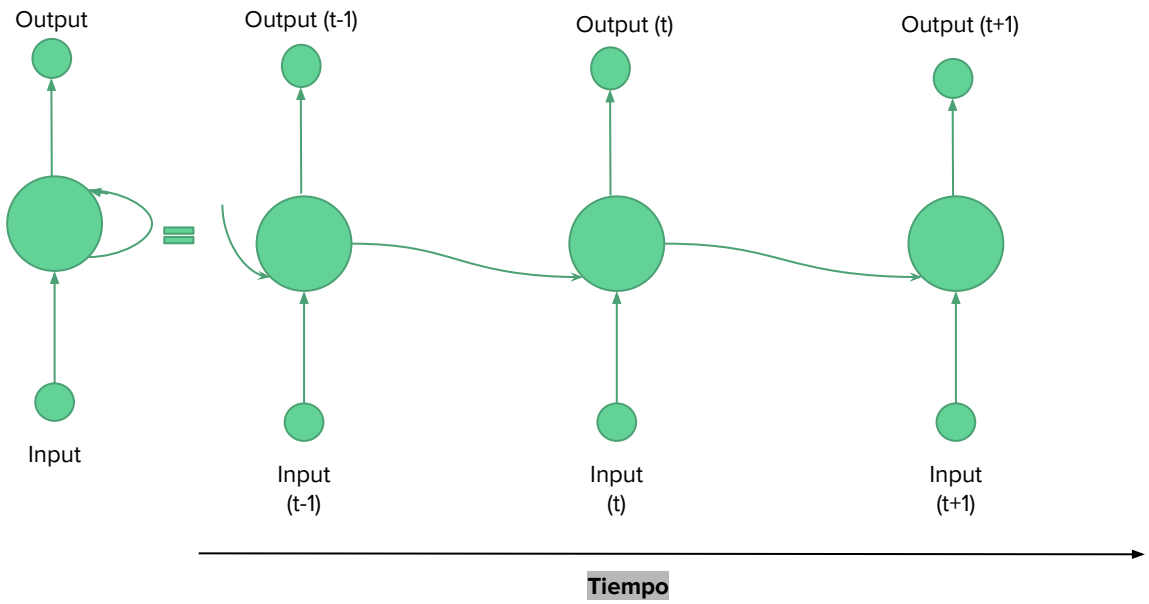
Estructura de una RNN

Cada neurona de una RNN no solo recibe una **entrada externa** en un tiempo determinado, sino también una entrada interna que es el **estado oculto** (hidden state) generado por las entradas anteriores.

Este estado oculto actúa como una memoria que se actualiza en cada paso de tiempo.

Paso de tiempo: Es una iteración de la secuencia de datos. Cada paso de tiempo procesa una entrada en un momento específico en la secuencia.

Por ejemplo cada palabra de la secuencia es un paso de tiempo diferente.



Estructura de una RNN

1. Input Layer:

La entrada es una secuencia de datos donde cada punto corresponde con un ***paso de tiempo específico o posición en la secuencia***. En cada paso de tiempo t, la red RNN recibe como input un vector x(t).

2. Recurrent Unit:

Son la base de las Redes Recurrentes.

- Mantienen un estado interno, conocido como estado oculto o memoria, que le permite a la red mantener la información de los pasos de tiempos anteriores.
- Procesan los inputs secuencialmente, tomando en cuenta el estado actual y el estado oculto anterior y producen una actualización del estado oculto para el siguiente paso de tiempo.

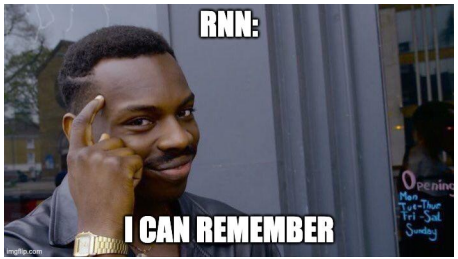
3. Hidden State:

Es un vector que representa la memoria de la red o el estado interno en un paso de tiempo particular.

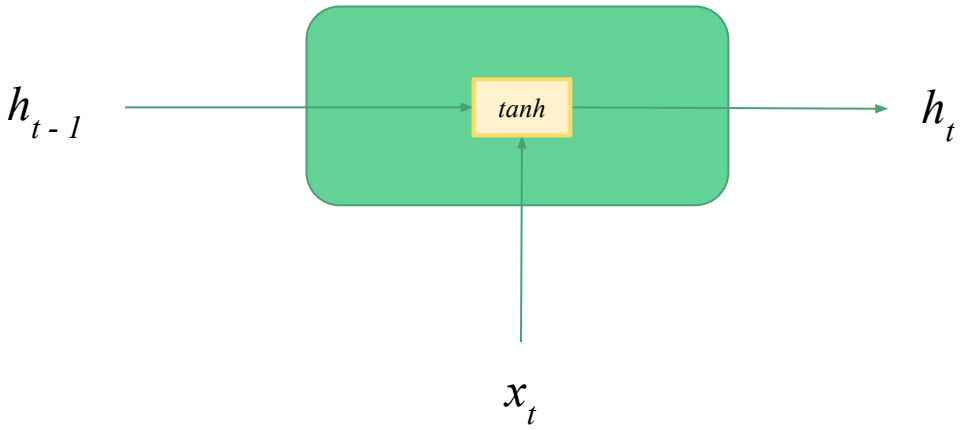
- Captura la información de los pasos de tiempos previos y colabora en el cálculo del paso de tiempo actual.
- Es actualizado en cada paso de tiempo en base al input actual y el estado oculto anterior, permitiendo a la red capturar patrones y dependencias en los datos.

4. Output Layer:

Produce la predicción en base al procesamiento secuencial de los datos. La salida puede variar dependiendo de la tarea.



La neurona recurrente en detalle



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

$$h_t = tanh(W_h[h_{t-1}; x_t] + b_h)$$

Activación tnh que genera el vector final

Capa densa que procesa la concatenación de la nueva y el estado anterior.

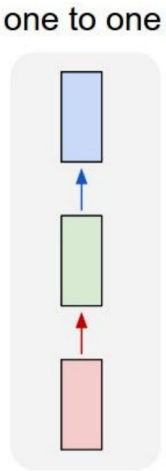
h: Estado oculto (hidden state)
x: Feature o input

Inputs & Outputs

La flexibilidad de las RNN

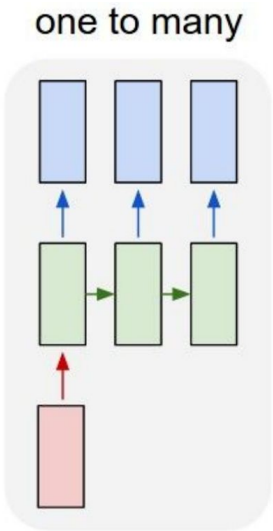
One to One

Son las redes feed forward tradicionales. Tienen una única entrada y una única salida. Útiles para problemas de clasificación.



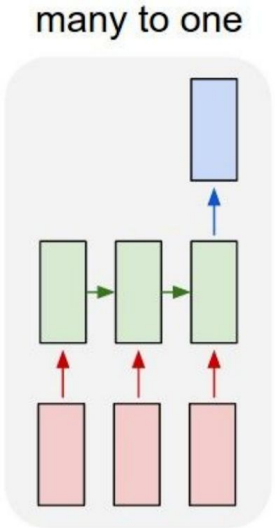
One to many

Un solo dato de entrada genera una secuencia de salidas. Muy útil para la *generación de subtítulos a partir de una imagen*.



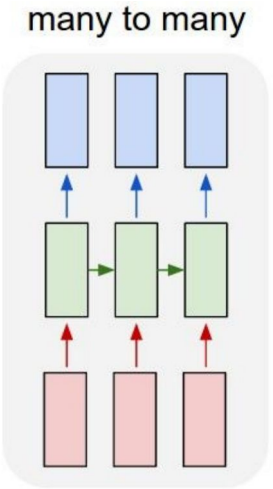
Many to one

Una secuencia de datos de entrada produce una única salida. Muy útil para *Análisis de Sentimientos.*



Many to many

Toma una secuencia de entrada y produce otra secuencia de salidas. La longitud de la salida puede o no coincidir con la longitud de entrada. Muy útil en *reconocimiento de voz*.



Inputs & Outputs

La flexibilidad de las RNN

One to One

Son las redes feed forward tradicionales. Tienen una única entrada y una única salida. Útiles para problemas de clasificación.

One to many

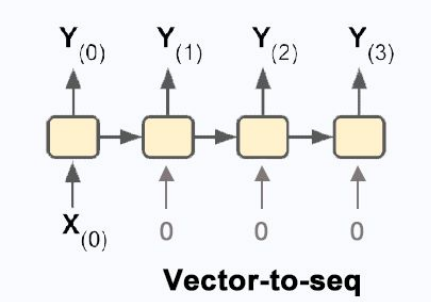
Un solo dato de entrada genera una secuencia de salidas. Muy útil para la *generación de subtítulos a partir de una imagen*.

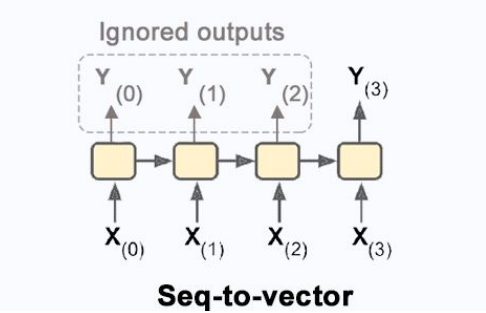
Many to one

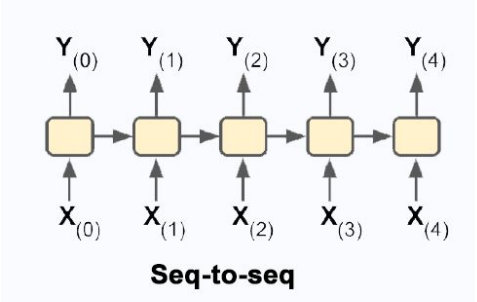
Una secuencia de datos de entrada produce una única salida. Muy útil para *Análisis de Sentimientos*.

Many to many

Toma una secuencia de entrada y produce otra secuencia de salidas. La longitud de la salida puede o no coincidir con la longitud de entrada. Muy útil en *reconocimiento de voz*.

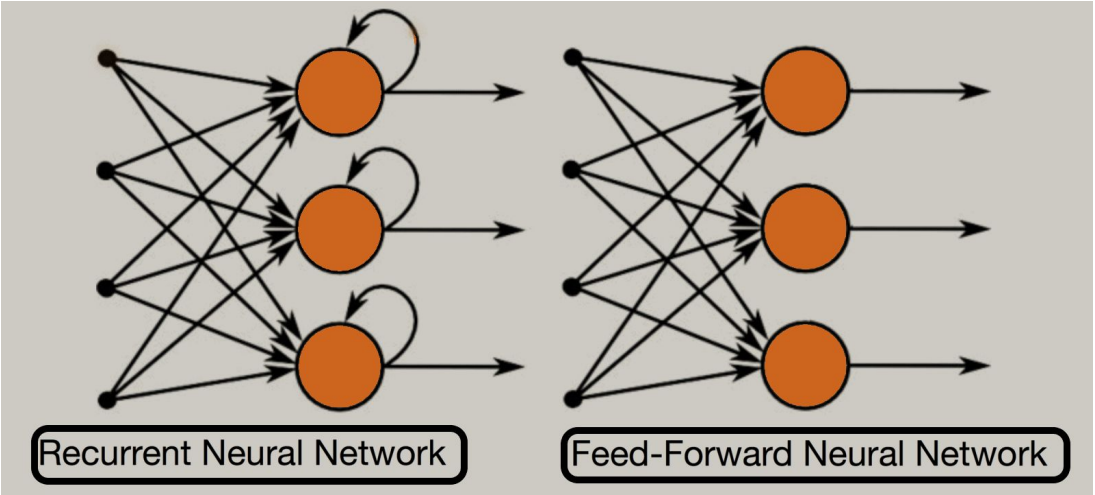






Aprendizaje

Mientras que otras redes *"viajan"* en una dirección lineal durante el feed *forward pass* o durante el *backpropagation*, la red recurrente sigue una **relación de recurrencia** en lugar de un paso de un paso hacia adelante y utiliza la **retropropagación a través del tiempo (BTT)** para aprender.



Estructura de Red Recurrente

Se puede procesar una secuencia **X** de vectores aplicando una función de recurrencia:

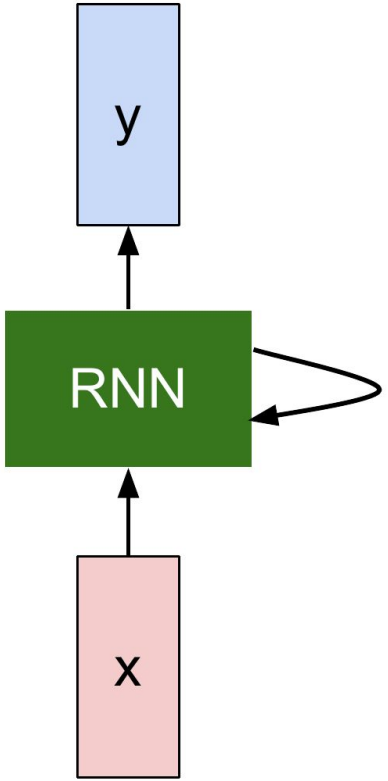
$$h_t = f_W(h_{t-1}, x_t)$$

Nuevo estado

Estado antiguo

Vector de input en el instante de tiempo t

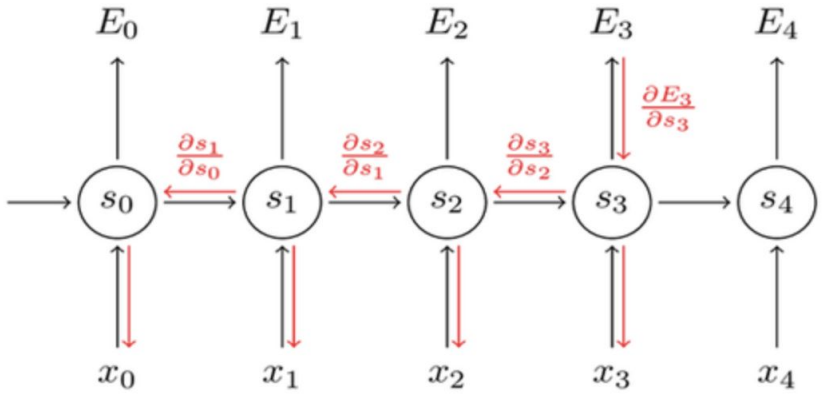
Alguna función con parámetros W



Créditos http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Backpropagation Through Time (BTT)

1. Se le proporciona a la red un paso único de la entrada.
2. Luego se calcula su estado actual utilizando un conjunto de entradas actuales y el estado anterior.
3. El ***estado oculto actual*** se convierte en el ***estado oculto anterior*** para el siguiente paso de tiempo.
4. Se pueden ir tantos pasos de tiempo según el problema y unir la información de todos los estados anteriores.
5. Una vez que se completan todos los pasos de tiempo, el estado actual final se utiliza para calcular la salida.
6. Luego, la salida se compara con la salida real, es decir, la salida objetivo y se genera el error.
7. Luego, el error se propaga hacia atrás a la red para actualizar los pesos y, por lo tanto, la red (RNN) se entrena mediante la propagación hacia atrás a través del tiempo.



How is back propagation in recurrent NNS different from FNNs?

RNN

Ventajas

- Manejan secuencias de longitud variable
- Recuerdan inputs pasados
- Comparten parámetros en todos los pasos de tiempo (reduce el número de parámetros para aprender y permite mejores generalizaciones).
- Son flexibles a distintas tareas (texto, discurso, secuencia de imágenes).

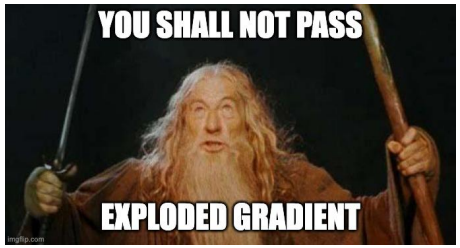
Desventajas

- Vanishing & Exploding Gradients
- Complejidad Computacional
- Dificultad para capturar dependencias a largo plazo.
- Dificultad para elegir la arquitectura (es aplicable a una variedad tareas pero... ¿qué arquitectura elijo para MI tarea?).
- Dificultad para interpretar la salida (¿cómo realiza las predicciones internamente?).

Exploding & Vanishing Gradients

Backpropagation “se mueve” desde la capa de salida hasta la capa de entrada, propagando el error del gradiente.

Para redes neuronales muy profundas es muy probable encontrarse con problemas de desvanecimiento (vanishing) o explosión (exploding) de los gradientes.



Cuando llegamos a las capas iniciales:

- Los gradientes usualmente se vuelven muy pequeños (el cambio en indistinguible)
- Los gradientes explotan 💣

Exploding & Vanishing Gradients: Posibles Soluciones

Vanishing

- Inicializar los pesos no aleatoriamente (Xavier, Kaiming, etc):
  - `torch.nn.init.xavier_normal_()`
  - `torch.nn.init.kaiming_normal_()`
  - `torch.nn.init.orthogonal_()`
- Usar LSTM en lugar de las redes recurrentes comunes

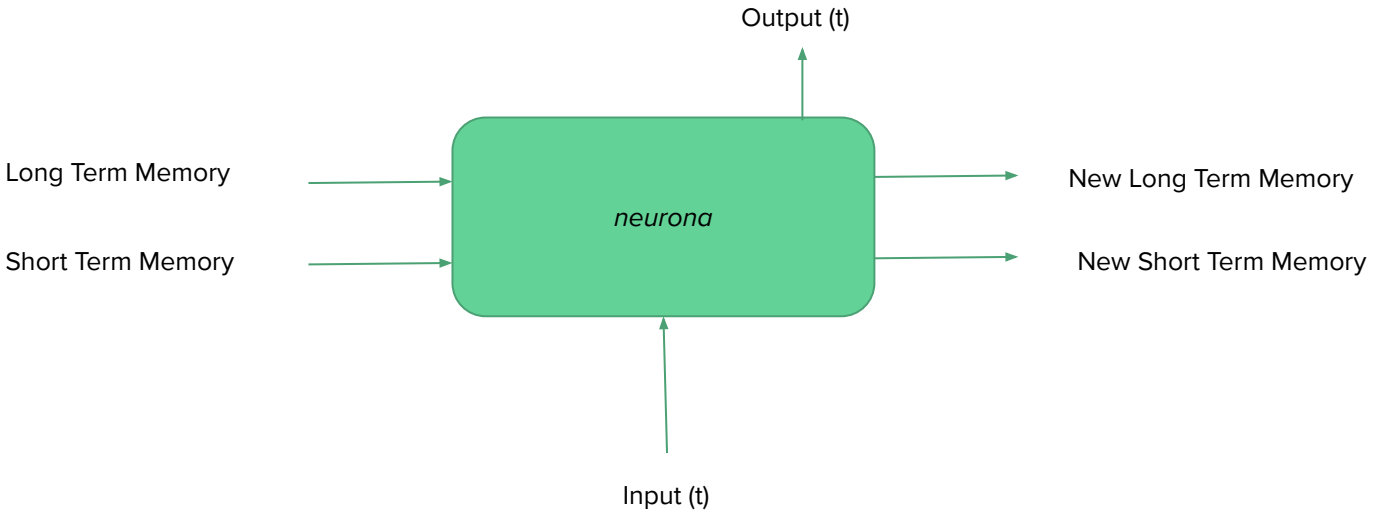
```
import torch.nn as nn
lstm = nn.LSTM(input_size, hidden_size, num_layers)
for name, param in lstm.named_parameters():
    if 'weight' in name:
        nn.init.xavier_normal_(param)
```

Exploding

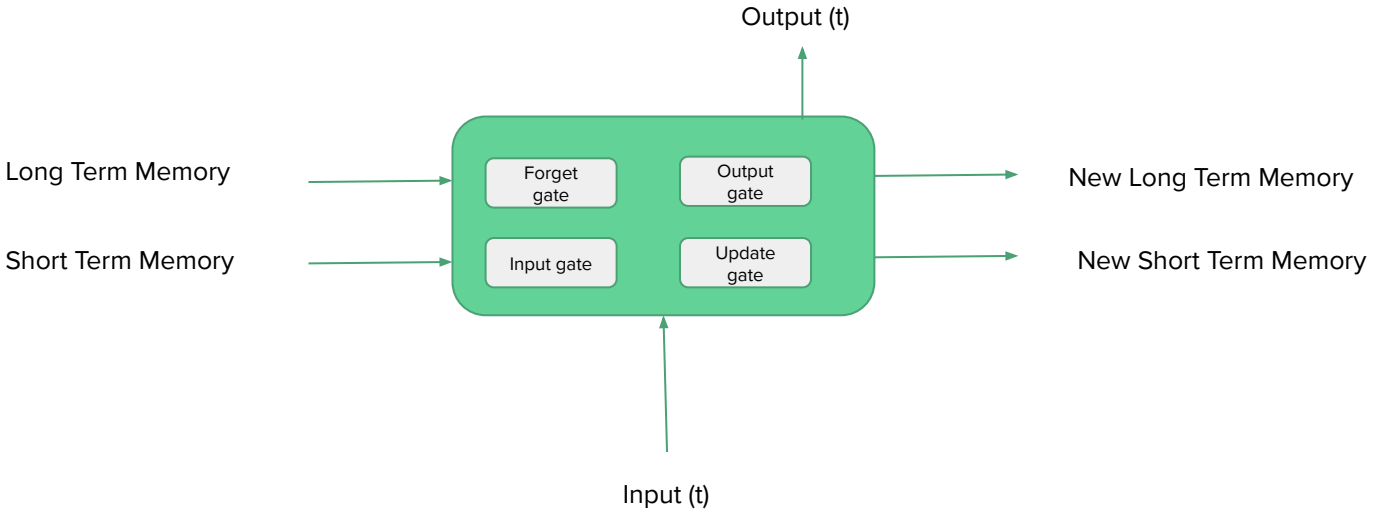
- Parar el backpropagation después de cierto punto.
- Poner un máximo en la norma del gradiente.
- Penalizar o reducir artificialmente el gradiente.

```
clip = gradient_clip
if clip is not None:
  nn.utils.clip_grad_norm_(model.parameters(), clip)
```

LSTM (Long Short-Term Memory)

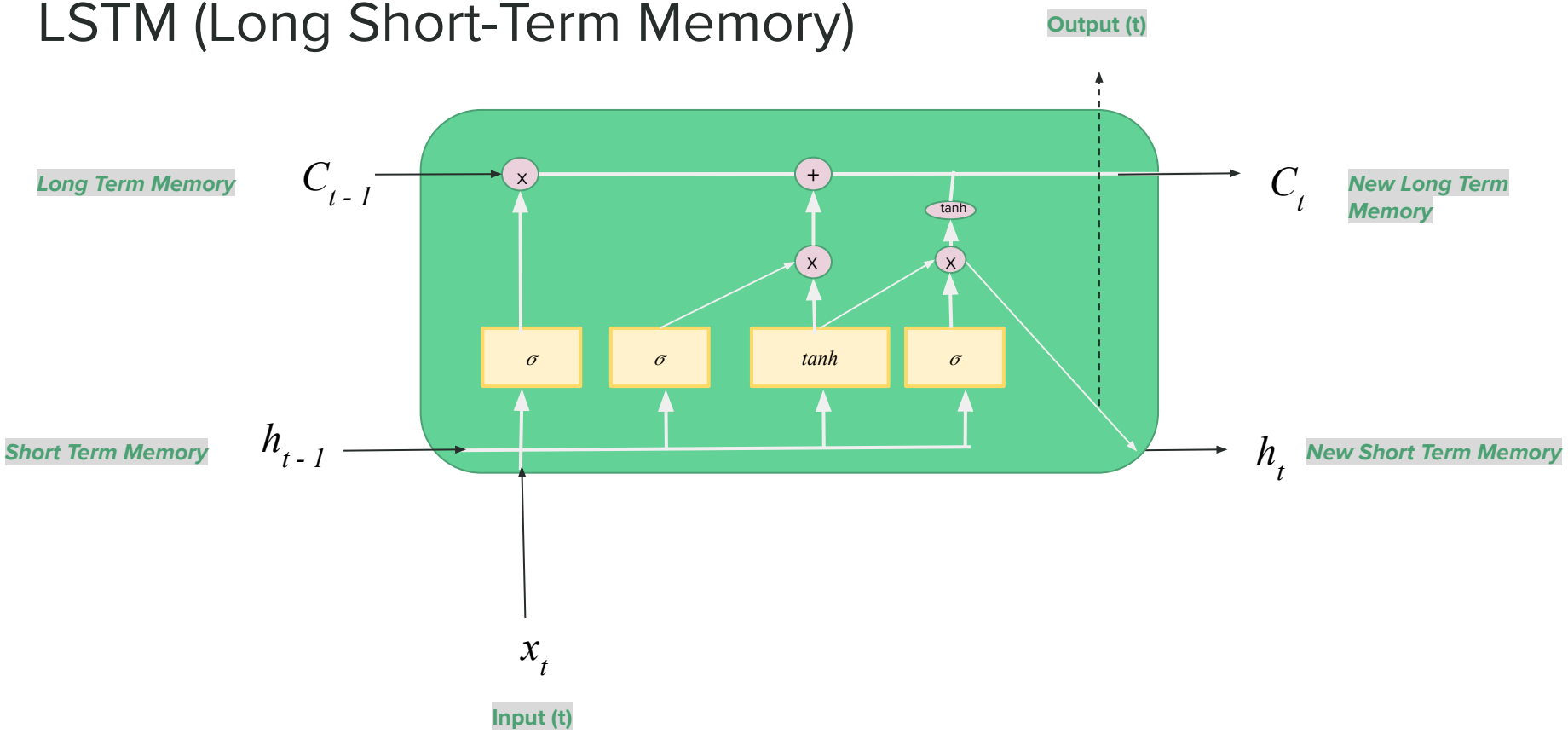


LSTM (Long Short-Term Memory)



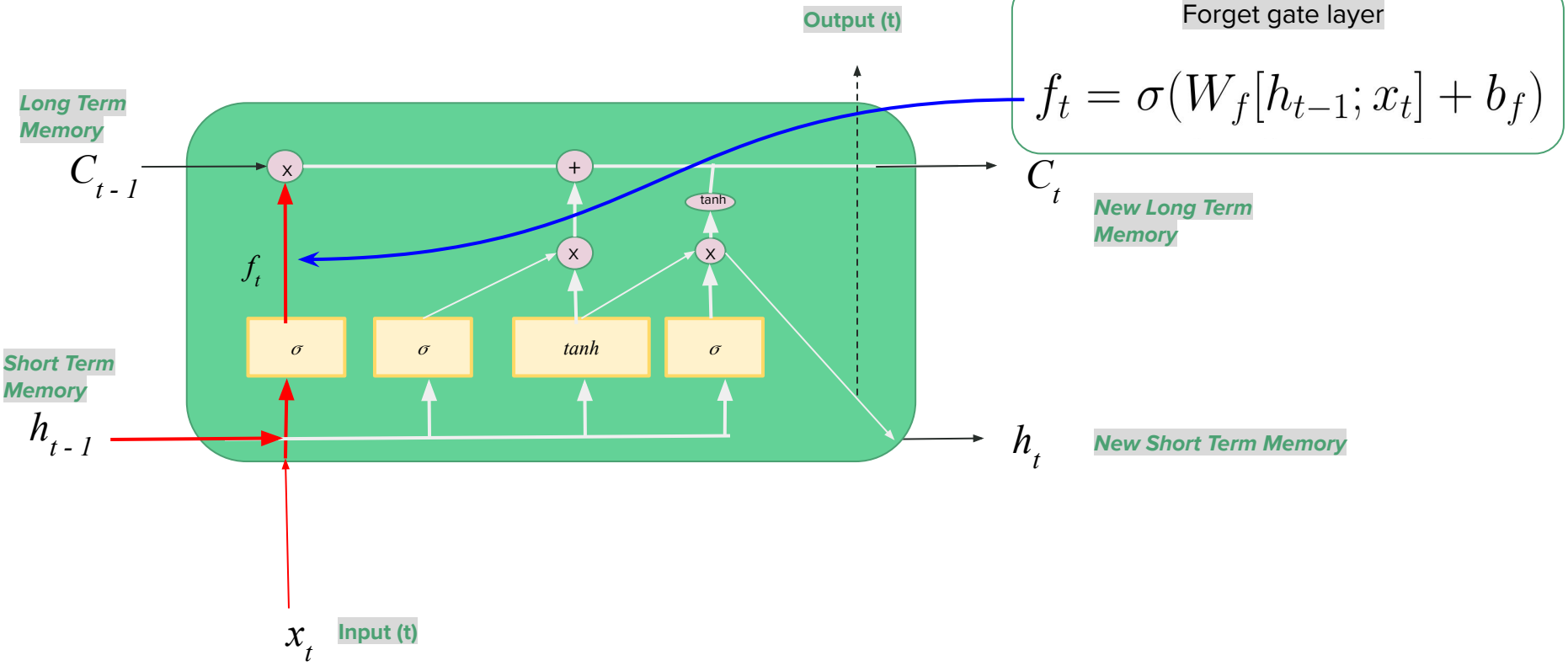
- ***Input gate:*** *Cuánta información vamos a almacenar en la memoria.*
- ***Forget gate:*** *Qué información vamos a olvidar.*
- ***Output gate:*** *Qué parte del estado interno es expuesto a la siguiente unidad LSTM.*
- ***Update gate:***

LSTM (Long Short-Term Memory)



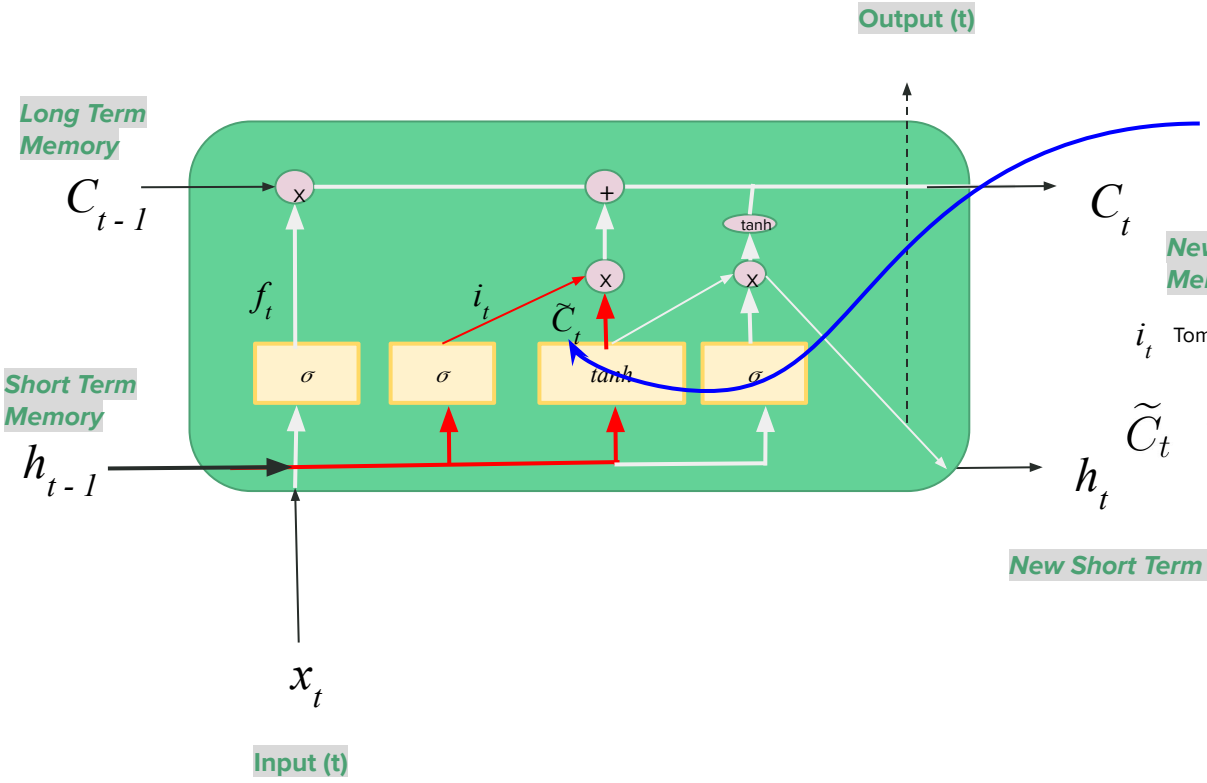
LSTM (Long Short-Term Memory)

Step 1: ¿Qué vamos a olvidar?



LSTM (Long Short-Term Memory)

Step 2: ¿Qué información vamos a almacenar en el paso t?



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

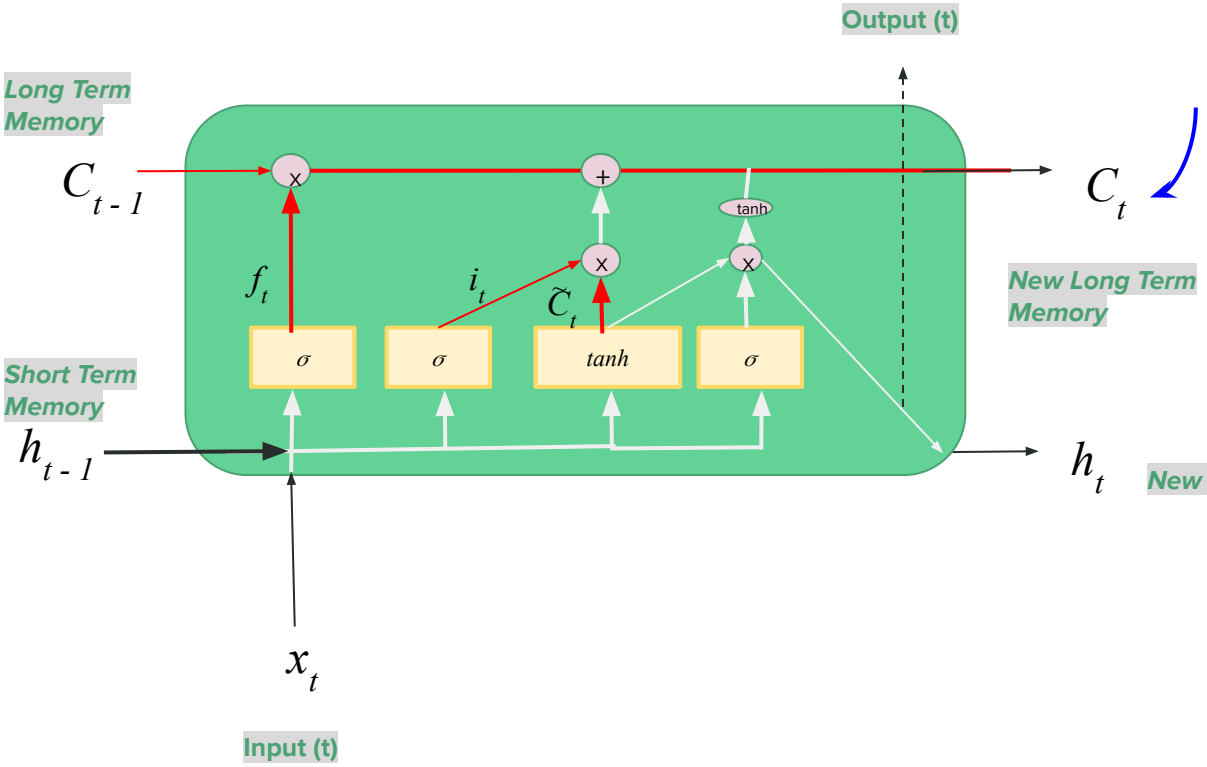
$$\tilde{C}_t = tanh(W_C[h_{t-1}; x_t] + b_C)$$

i_t Toma los valores de entrada y decide qué valores van a hacer actualizados

$\tilde{C}_t$ Crea un vector de nuevos candidatos

LSTM (Long Short-Term Memory)

Step 3: Generar el nuevo contexto



Salida de forget gate layer | Salida de update gate layer

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

Para generar el nuevo contexto se hace una combinación pesada del contexto anterior y el actual. De esta forma la red puede seleccionar la información más relevante para olvidar y la más relevante para mantener del nuevo input

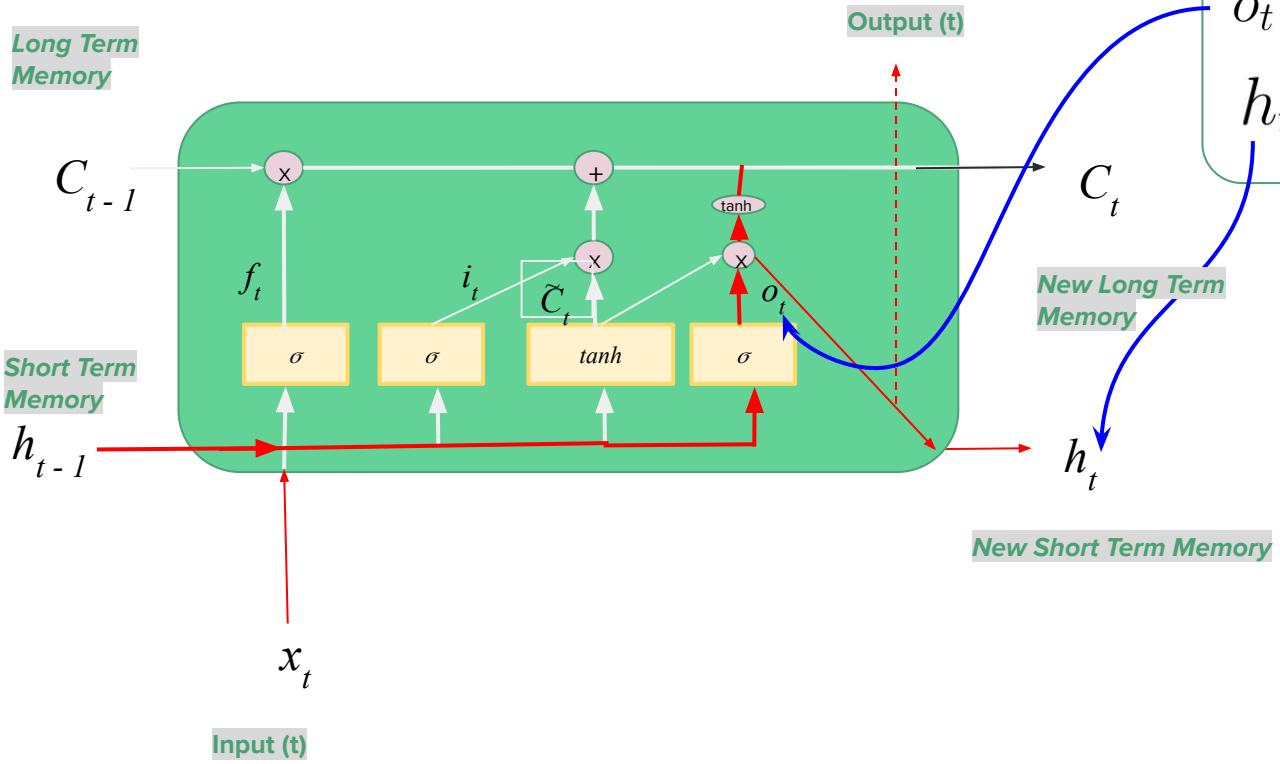
LSTM (Long Short-Term Memory)

Step 4: Decidir cuál va a ser la salida

Output gate layer

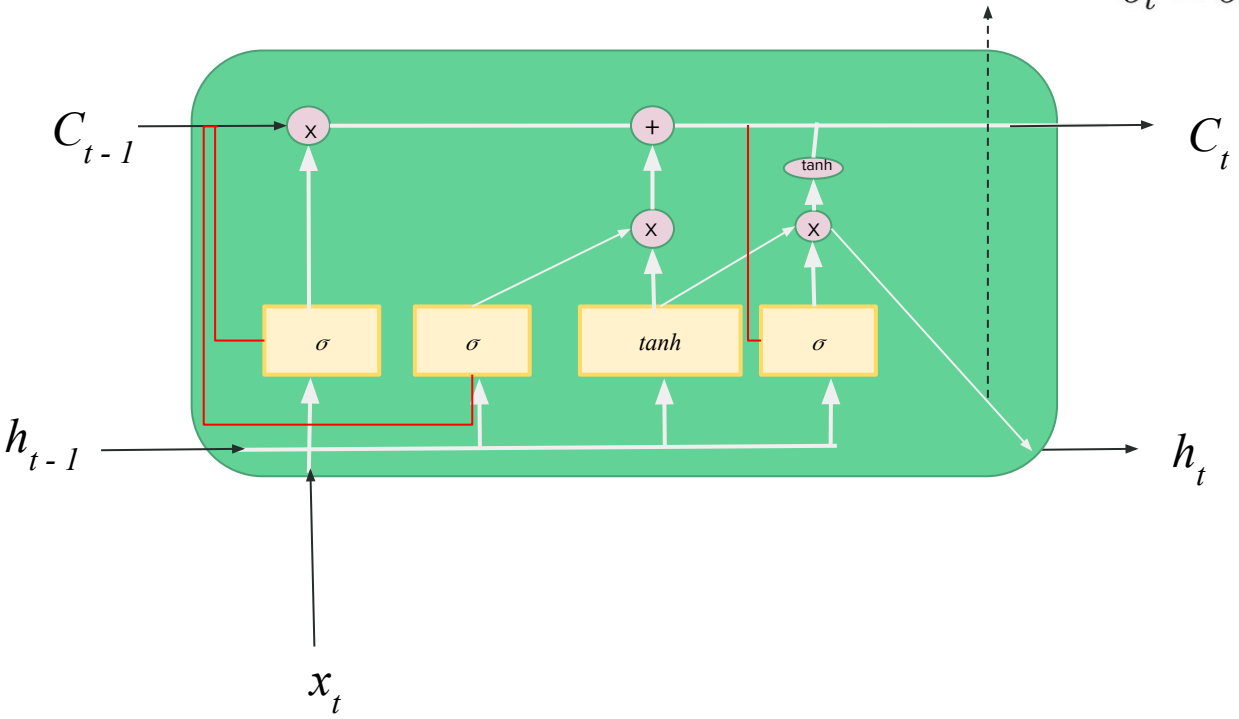
$$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$

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



LSTM con “peepholes”

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$



A modo de resumen

Problemas con las RNN estandar

- **Desvanecimiento del gradiente:** Cuando una RNN entrena a través de retropropagación, los gradientes pueden disminuir exponencialmente a medida que se propagan hacia atrás en el tiempo, haciendo que sea muy difícil para la red aprender dependencias a largo plazo (recuerdos de eventos lejanos).

- **Explosión del gradiente:** Cuando los gradientes crecen exponencialmente, causando que los pesos de la red se actualicen de manera descontrolada y la red se vuelva inestable.

- **Memoria a corto plazo:** Las RNN tienen dificultades para mantener información relevante durante muchas etapas de tiempo, lo que las hace poco útiles cuando se requiere procesar secuencias largas o dependencias a largo plazo en los datos.

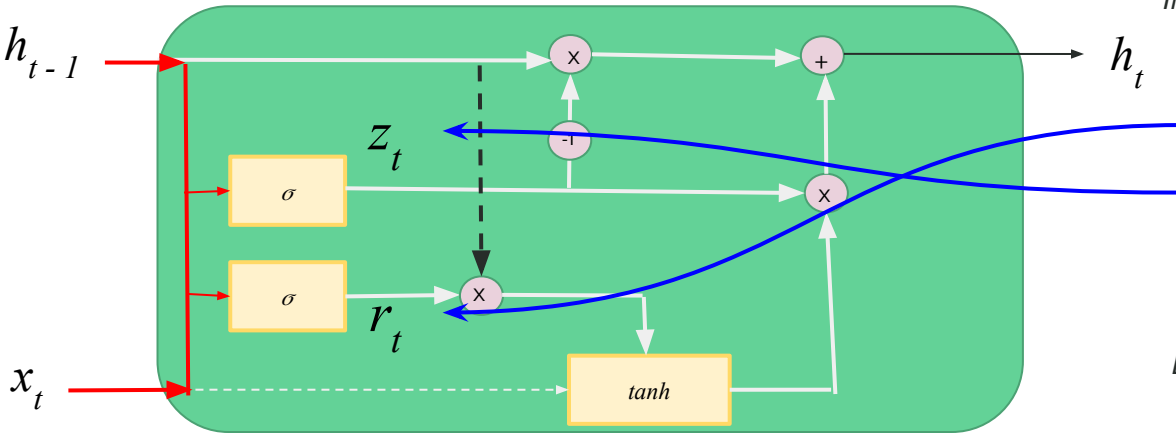
LSTM soluciones

- **Preservación de información a largo plazo:** La estructura de las LSTM permite que la información importante se mantenga y pase a lo largo de muchos pasos temporales sin ser afectada por el problema del desvanecimiento de gradiente.

- **Mejora en la estabilidad del entrenamiento:** Las LSTM son más estables en el entrenamiento, lo que las hace efectivas para tareas que implican secuencias largas, como traducción automática, análisis de sentimiento, reconocimiento de voz, etc

GRU (Gated Recurrent Unit)

Step 1



GRU condensa en una sola función la operación de olvidar y actualizar. Inicialmente se calculan z_t y r_t

$$r_t = \sigma(W_r[h_{t-1}; x_t] + b_r)$$

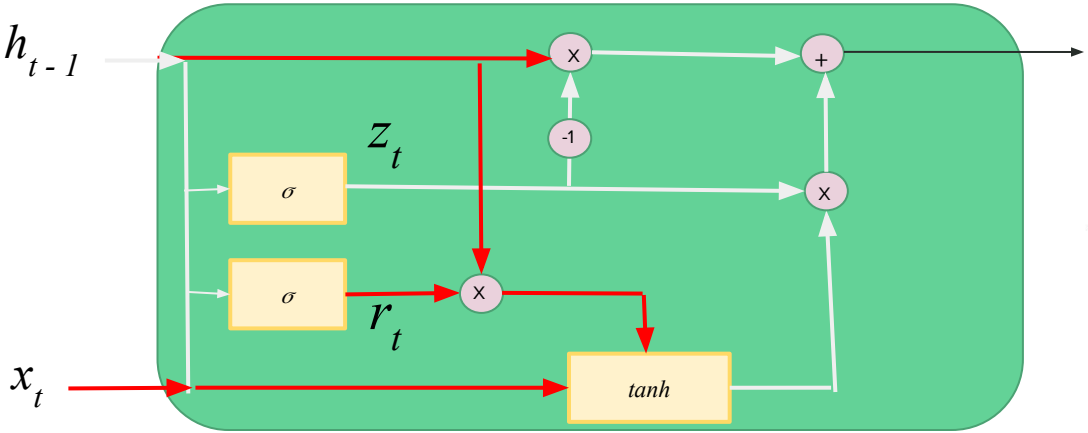
$$z_t = \sigma(W_z[h_{t-1}; x_t] + b_z)$$

Estas se conocen como:

- *r_t: reset (reiniciar) gate*
- *z_t: update (actualizar) gate*

GRU (Gated Recurrent Unit)

Step 2

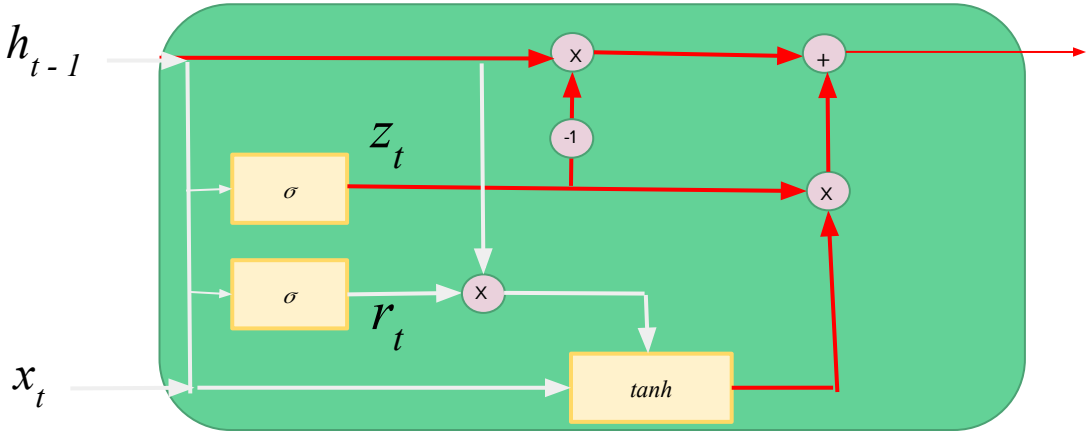


Una capa densa más una activación tanh procesa la combinación del estado anterior h_{t-1} y x_t. La intuición es que la red puede usar r_t para eliminar o conservar lo relevante del estado anterior con respecto a la nueva entrada.

$$\widetilde{h}_t = tanh(W_h[r_t * h_{t-1}; x_t] + b_h)$$

GRU (Gated Recurrent Unit)

Step 3



Finalmente, el próximo estado oculto es una combinación pesada del estado pasado y el actual.

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

RNN para texto: Sentimental analysis

Algunos puntos a tener en cuenta en la implementación

1. Una red recurrente toma una secuencia de palabras de palabras a partir de una oración o un documento como ***input***.
2. Cada palabra tiene que ser representada como un ***vector denso***, o un embedding, que capture la semántica de la palabra.
3. La red procesa una secuencia palabra a palabra, ***actualizando su estado interno*** teniendo en cuenta la palabra actual y su estado previo.
4. El estado final de la red se usa para ***predecir el sentimiento***. Este se pasa a través de una capa totalmente conectada seguida de una función de activación que devuelve la distribución de probabilidad de la secuencia (positivo, negativo o neutral). La clase con la probabilidad más alta es elegida como la predicción del modelo.