

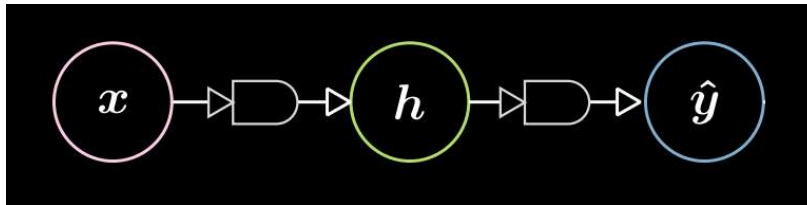
# Deep learning

*Modelos de  
secuencias*



# Introducción

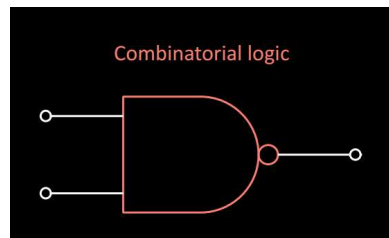
El grafo computacional de los modelos vistos hasta ahora es lineal.  
Redes convolucionales son redes feedforward con convoluciones



La entrada es ingresada a la red neuronal,  
la cual produce una salida

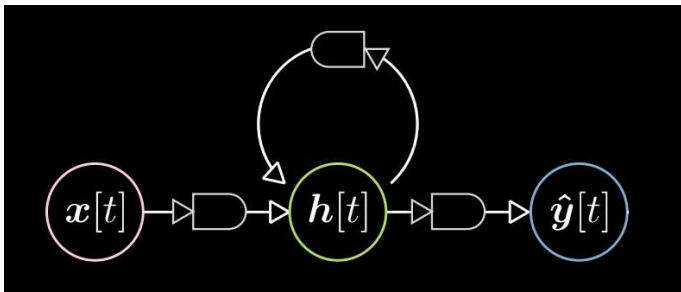
Función  $h$  es una combinación de una operación lineal  
y una función de activación no lineal.

En lógica sería algo como esto:



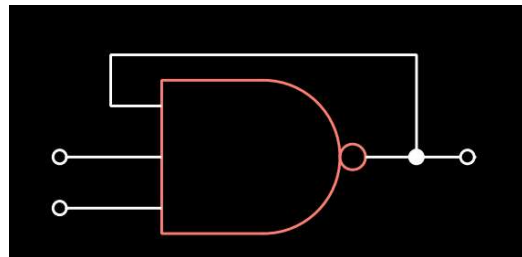
# Introducción

Una red neuronal recurrente es una red con conexiones recursivas  
La salida se convierte en entrada de la red.



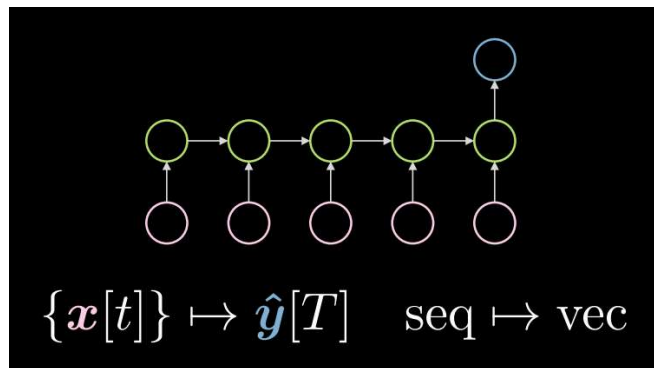
Existe un componente temporal para distinguir en cual recurrencia estamos durante el proceso

En lógica sería algo como esto

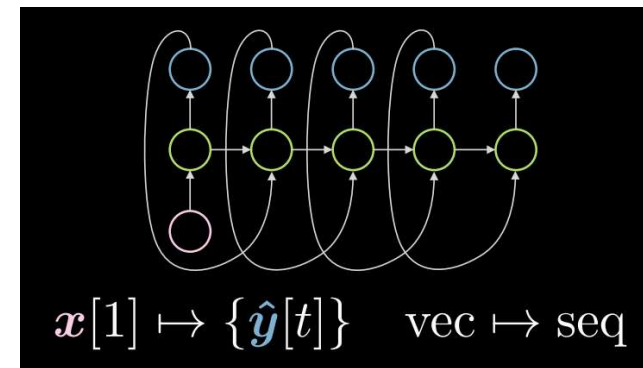


# Tipos de RNN

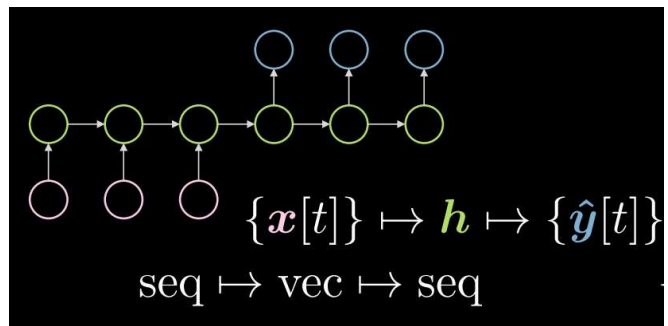
Sequence to vector - classification



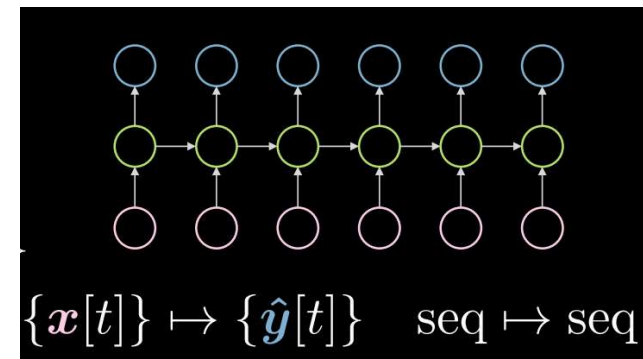
Vector to sequence – image captioning



Sequence to sequence - translation



Sequence to sequence – video analysis



# Ejemplo – Image captioning

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A skateboarder does a trick on a ramp.



A dog is jumping to catch a frisbee.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.



A refrigerator filled with lots of food and drinks.



A herd of elephants walking across a dry grass field.



A close up of a cat laying on a couch.



A red motorcycle parked on the side of the road.



A yellow school bus parked in a parking lot.



Describes without errors

Describes with minor errors

Somewhat related to the image

Unrelated to the image

# Ejemplo – Learning to execute

Predict the output of Python programs

- **Input:**

```
j=8584
for x in range(8):
    j+=920
    b=(1500+j)
    print((b+7567))
```

- **Target:** 25011.

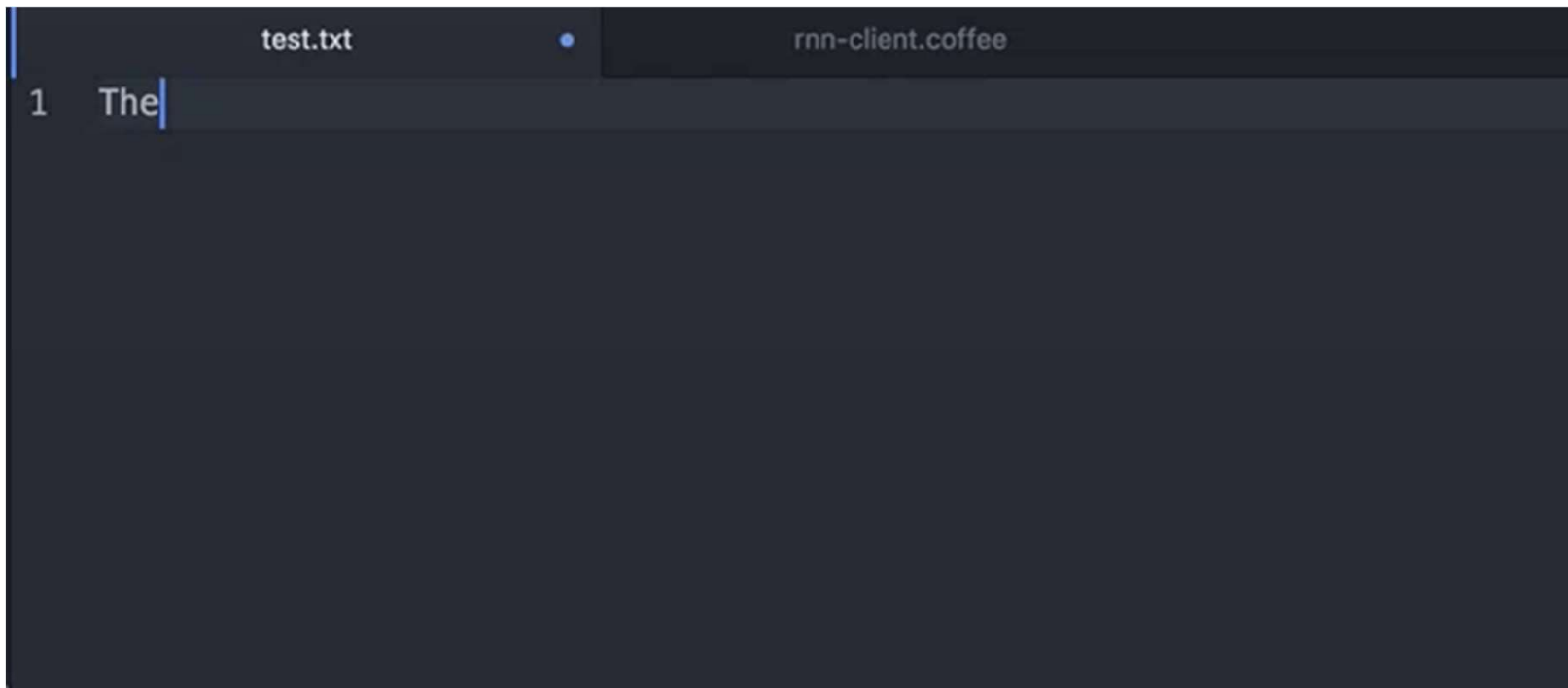
- **Input:**

```
i=8827
c=(i-5347)
print((c+8704) if
2641<8500 else 5308)
```

- **Target:** 12184.

# Ejemplo – Text completion

Rnn-writer



A screenshot of a code editor with two tabs: 'test.txt' and 'rnn-client.coffee'. The 'test.txt' tab is active, showing line 1 with the text 'The' followed by a blue cursor. The editor has a dark theme.

# Ejemplo – Text generation

SYSTEM PROMPT  
(HUMAN-WRITTEN)

*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

MODEL COMPLETION  
(MACHINE-  
WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."



# Deep learning

*Secuencias*



# Modelos auto-regresivos

Usemos como ejemplo datos de precios del mercado

En cada tiempo  $t$ , observamos un precio denotado como  $x_t$

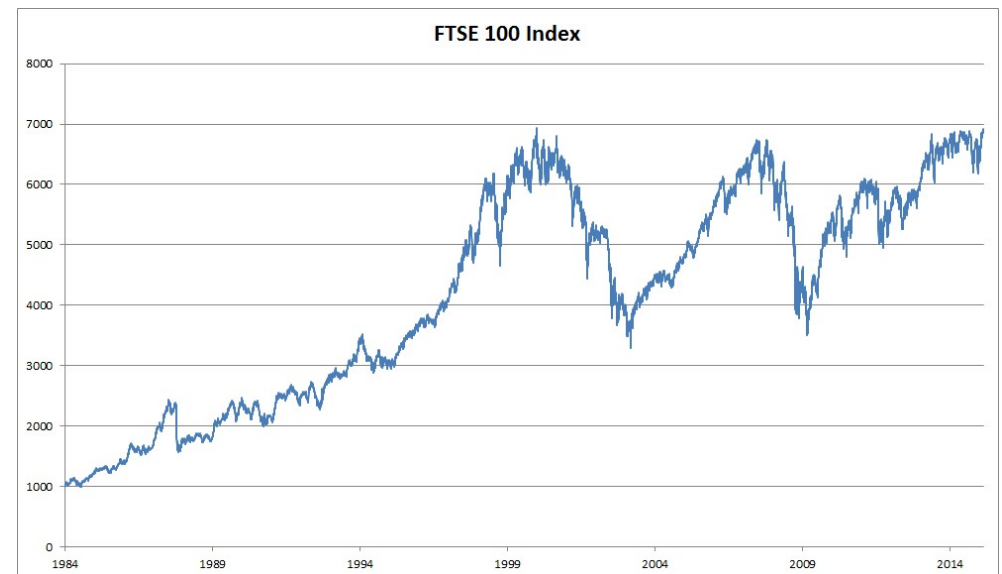
Si ninguna información externa es usada para predecir un valor, la única data disponible es el histórico de los precios.

$$P(x_t | x_{t-1}, \dots, x_1)$$

Calcular esta distribución es difícil, pero podríamos reducir el problema a estadísticas claves de la distribución

$$\mathbb{E}[(x_t | x_{t-1}, \dots, x_1)]$$

Se puede estimar con regresión lineal (modelos auto-regresivos)



# Modelos auto-regresivos

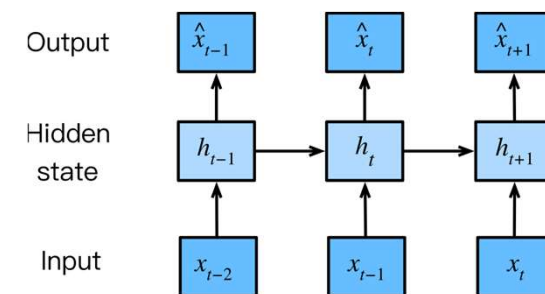
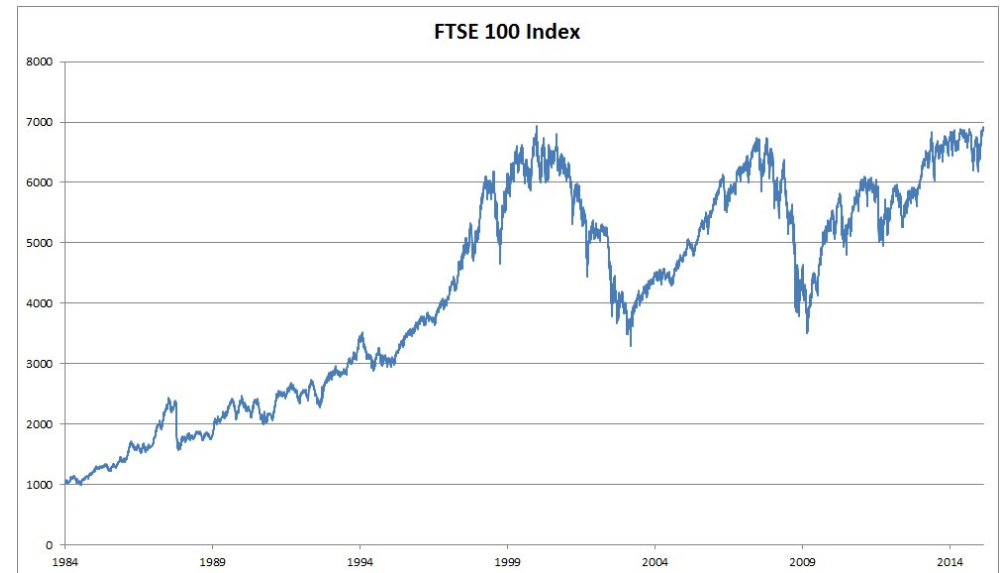
Un problema de los modelos auto-regresivos es que, dependiendo del valor de  $t$ , uno tiene más o menos valores históricos. Por lo tanto el modelo debe soportar *input features* de longitud variable.

Dos estrategias:

- Restringir el cómputo a una ventana histórica de longitud  $\tau$
- Suponer que podemos computar una función latente  $h_t$  que resuma la información histórica

$$\hat{x}_t = P(x_t|h_t)$$

$$h_t = g(h_{t-1}, x_{t-1})$$



# Texto a Secuencias

Un texto es una secuencia de palabras, caracteres o bloques de palabras.

Pre-procesamiento típico:

- Cargar texto como strings en memoria
  - Dividir los strings en tokens (palabras, caracteres, etc.)
  - Construir un diccionario de vocabulario para asociar cada elemento del vocabulario con un índice numérico
  - Convertir el texto a secuencia de índices numéricos
- 
- Una vez el texto se representa como una secuencia de índices numéricos, debemos encontrar la forma de representar cada palabra (cada índice) como un vector característico (Word embedding)

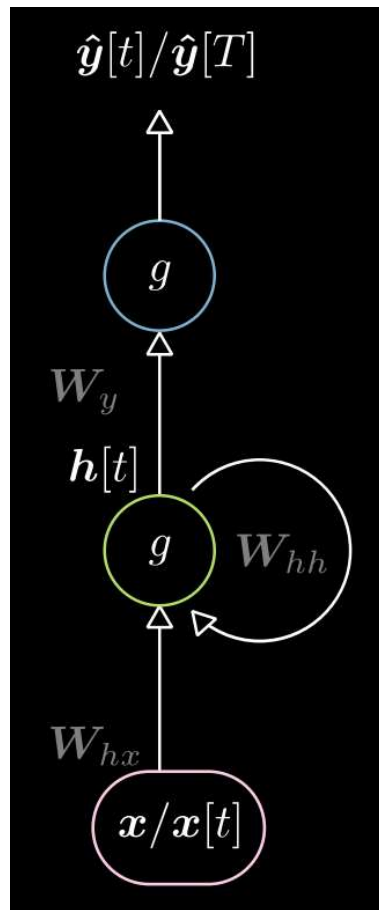




# Redes neuronales

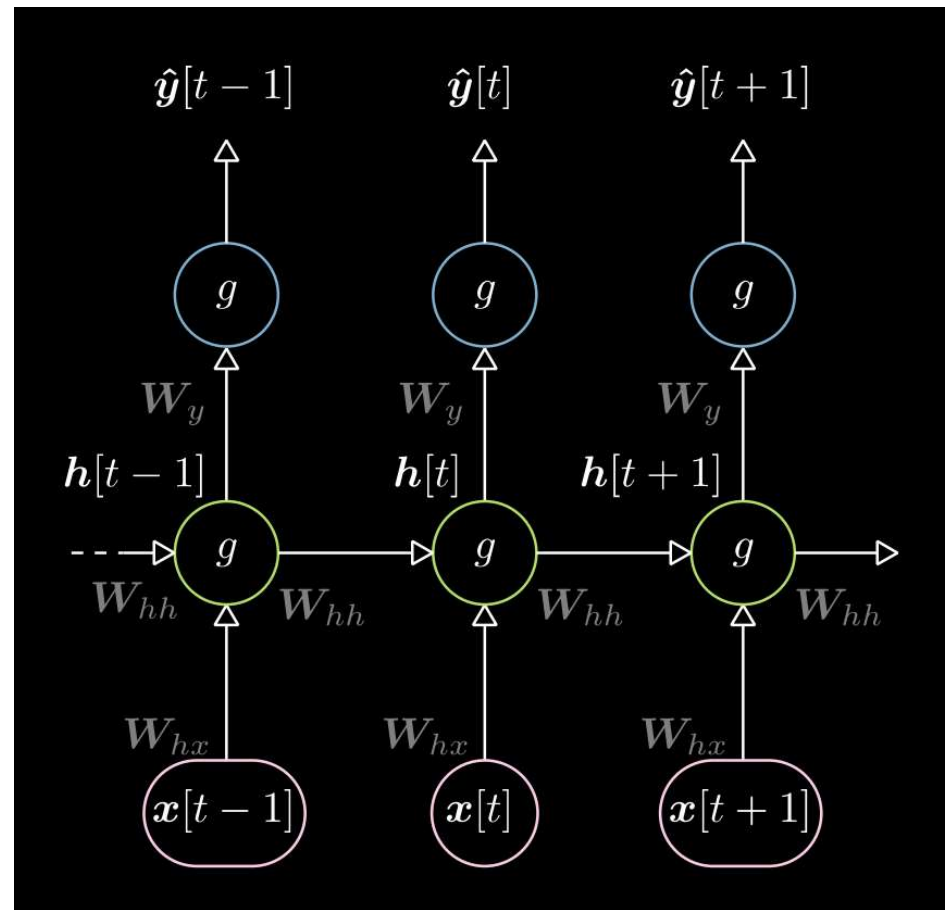
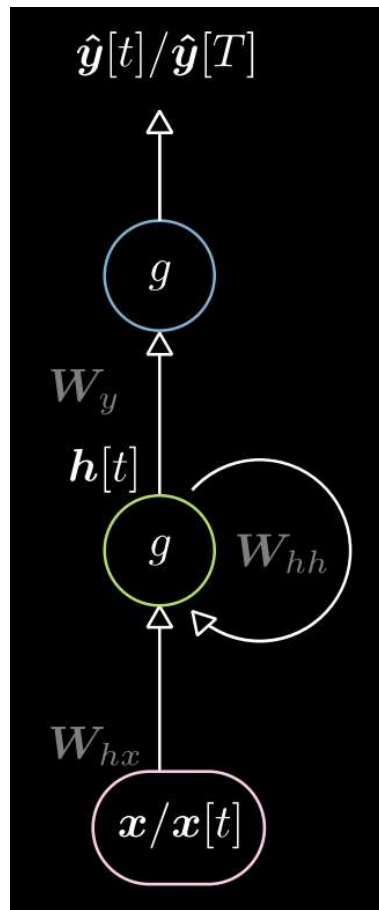
*Recurrent Neural  
Networks*

# Recurrent neural network - funcionalidad



$$\begin{aligned} \mathbf{h}[t] &= g(\mathbf{W}_h [\mathbf{x}^{[t]} \mathbf{h}_{[t-1]}] + \mathbf{b}_h) \\ \mathbf{h}[0] &\doteq \mathbf{0}, \mathbf{W}_h \doteq [\mathbf{W}_{hx} \mathbf{W}_{hh}] \\ \hat{\mathbf{y}}[t] &= g(\mathbf{W}_y \mathbf{h}[t] + \mathbf{b}_y) \end{aligned}$$

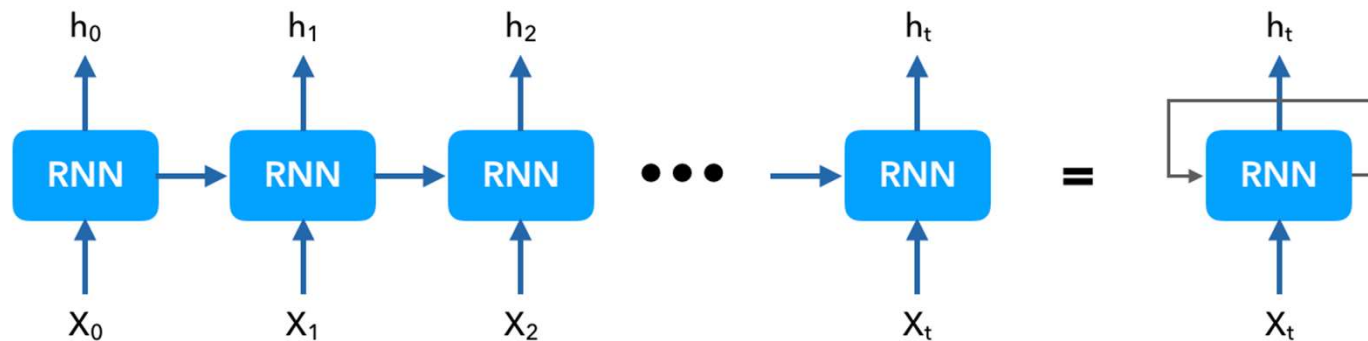
# Recurrent neural network - funcionalidad



# Recurrent neural networks

Para una secuencia de tamaño  $T$ , la red recurrente puede ser vista como una red feedforward de  $T$  capas, en donde la usa la misma matriz de pesos en cada capa.

Para secuencias largas, la red podría llegar a ser muy profunda (con respecto al tiempo)





# Backpropagation a través del tiempo

Tratemos de entender cómo funcionaría el backpropagation si lo aplicamos sobre la versión desplegada de una RNN. Asumimos una versión reducida de una RNN

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

$$o_t = g(h_t, w_o)$$

Durante el proceso, tenemos una cadena de valores  $\{\dots, (x_{t-1}, h_{t-1}, o_{t-1}), (x_t, h_t, o_t), \dots\}$  que se computan recurrentemente.

- La pasada forward consiste en iterar sobre la variable de tiempo  $t$ , calculando los valores en función de los valores en  $t - 1$
- La discrepancia entre las salidas obtenidas y las deseadas queda como

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T} \sum_{t=1}^T l(y_t, o_t)$$

# Backpropagation a través del tiempo

Para backpropagation, necesitamos una forma de computar el gradiente de  $L$  con respecto a  $w_h$

$$L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) = \frac{1}{T} \sum_{t=1}^T l(y_t, o_t)$$

Aplicamos regla de la cadena

$$\begin{aligned} \frac{\partial L}{\partial w_h} &= \frac{1}{T} \sum_{t=1}^T \frac{\partial l(y_t, o_t)}{\partial w_h} \\ &= \frac{1}{T} \sum_{t=1}^T \frac{\partial l(y_t, o_t)}{\partial o_t} \frac{\partial g(h_t, w_o)}{\partial h_t} \frac{\partial h_t}{\partial w_h} \end{aligned}$$

El tercer término es el complicado

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h}$$

# Backpropagation a través del tiempo

Esto tiene la forma de una ecuación de recurrencia

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_h} \qquad a_t = b_t + c_t a_{t-1}$$

La solución es

$$a_t = b_t + \sum_{i=1}^{t-1} \left( \prod_{j=i+1}^t c_j \right) b_i$$

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left( \prod_{j=i+1}^t \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

Cadena de productos de gradientes muy larga, cuando  $t$  es grande.

# Backpropagation a través del tiempo

Asumimos una RNN sin parámetro bias para evitar complicaciones de notaciones y la función de activación es la función identidad  $\phi(x) = x$

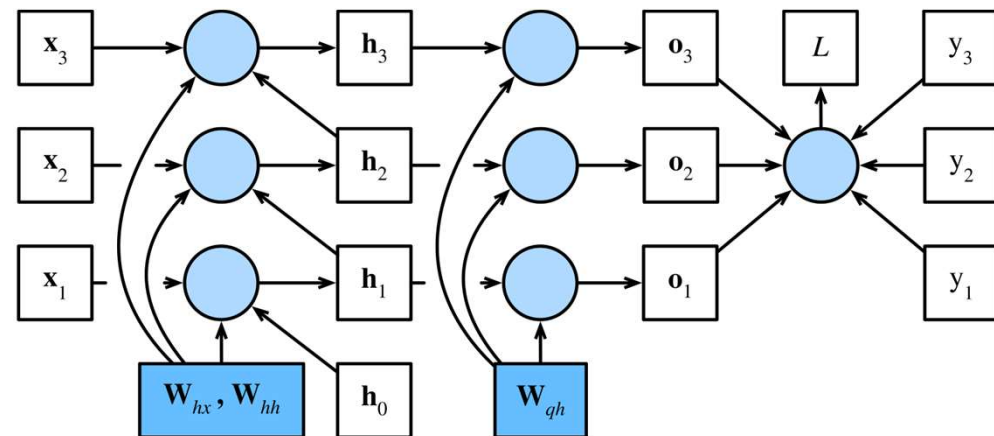
En un determinado tiempo  $t$ , el estado oculto  $h_t$  y la salida  $o_t$  se calculan como

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1}$$

$$\mathbf{o}_t = \mathbf{W}_{qh}\mathbf{h}_t$$

El los calculado sobre todas las instancias de  $t$  nos queda como

$$L = \frac{1}{T} \sum_{t=1}^T l(y_t, \mathbf{o}_t)$$



# Backpropagation a través del tiempo

Aplicando el procedimiento de cálculo de los gradientes de la función los, tenemos:

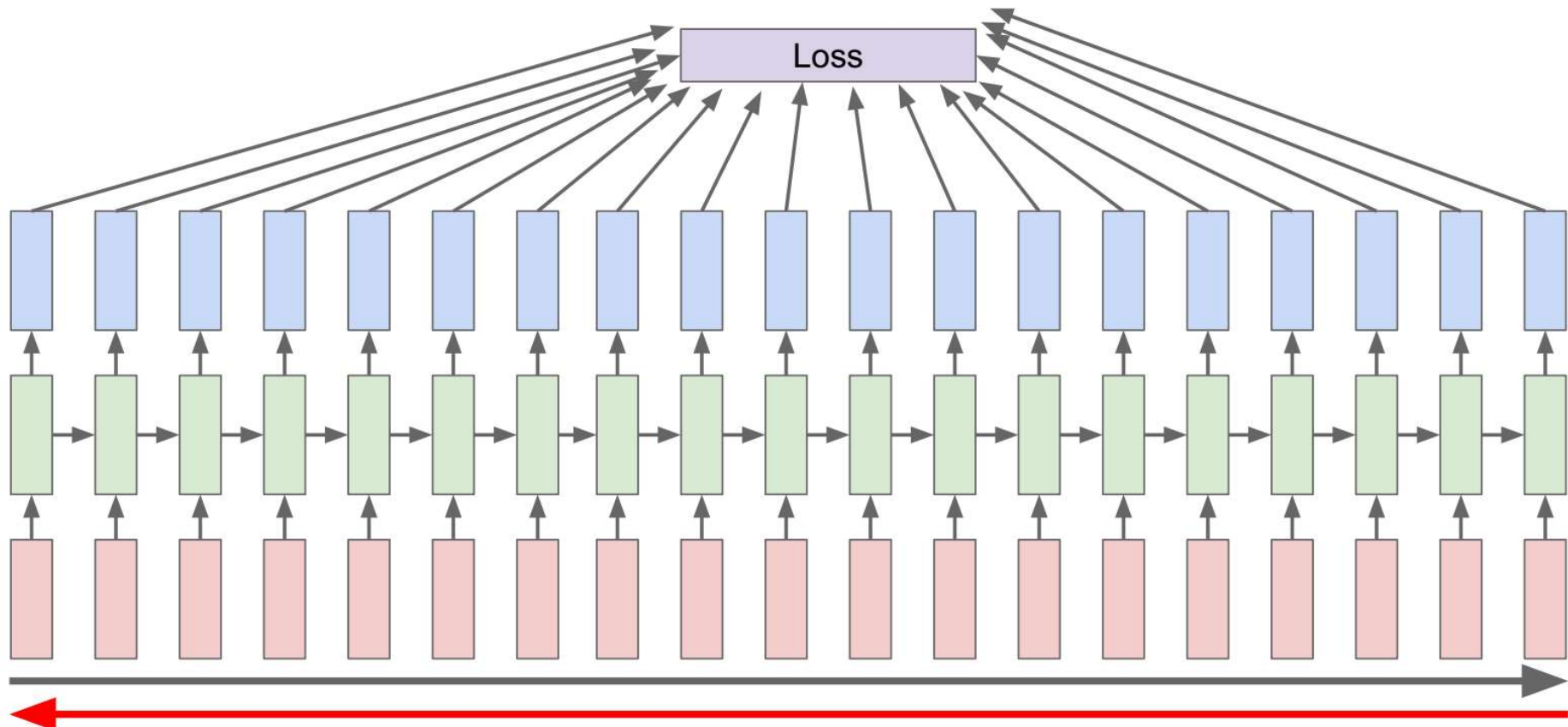
$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^T (\mathbf{W}_{hh}^\top)^{T-i} \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}$$

Potencias muy grandes de  $\mathbf{W}$  pueden fácilmente provocar gradientes que se desvanecen o que se vuelven muy grandes.

# Backpropagation through time

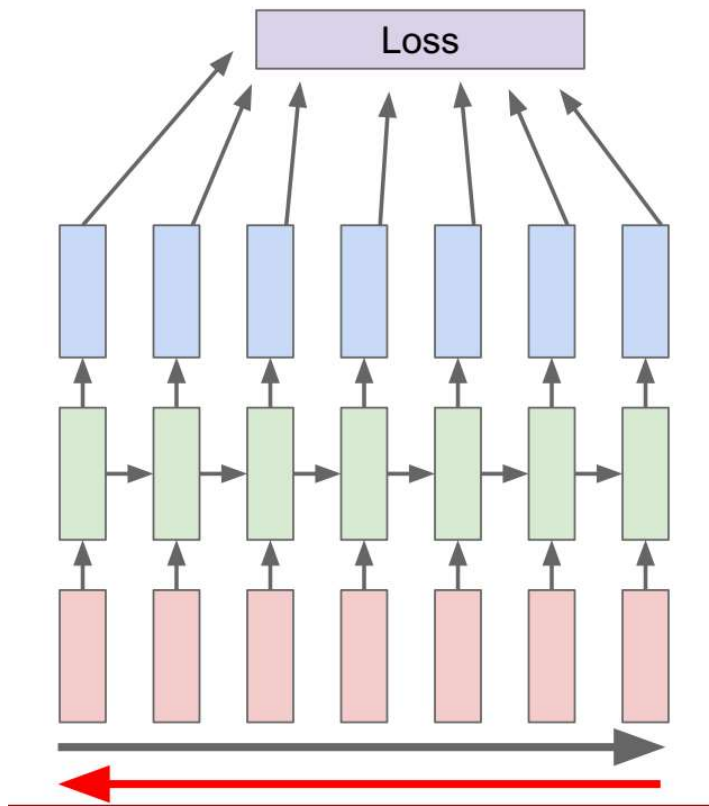
The loss is computed after consuming the entire sequence.

The backward goes also through the entire sequence.



# Backpropagation through time

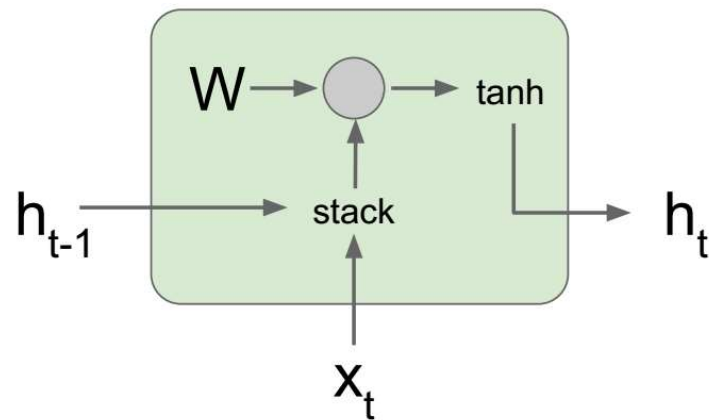
Derivative of the loss with respect to weight matrix is a chain of multiplication of the weight matrix.  
Problem with gradients.



Truncated version of BTT

Backpropagation uses a temporal window, instead of the entire sequence.

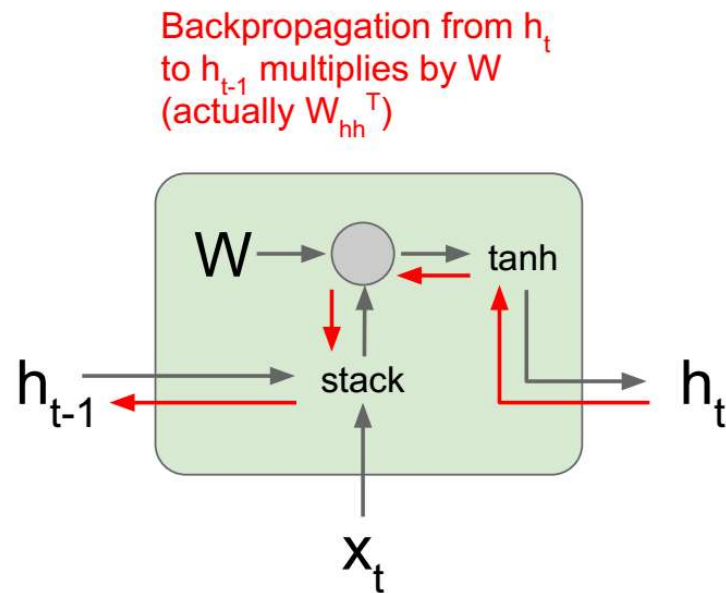
# Flujo de gradiente RNN



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\ &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

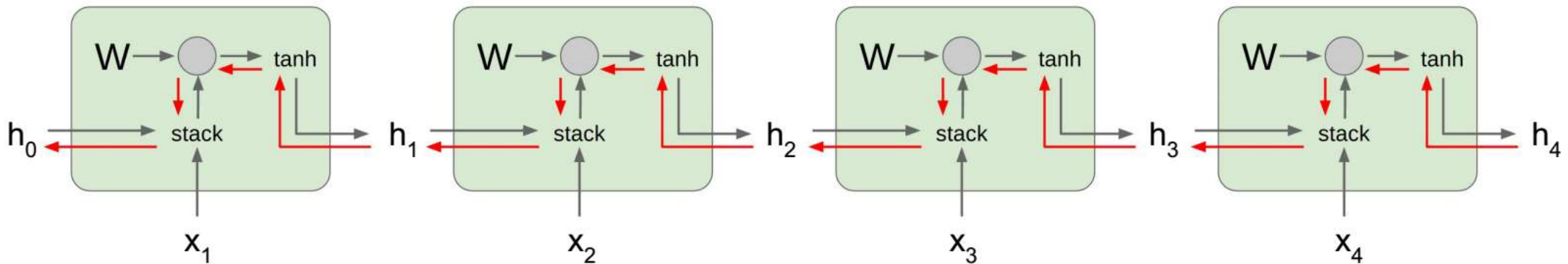


# Flujo de gradiente RNN



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\ &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

# Flujo de gradiente RNN



El gradiente contiene muchos factores de  $W$  y la función de activación  $\tanh$

Gradientes grandes son resueltos de alguna forma con gradient clipping.

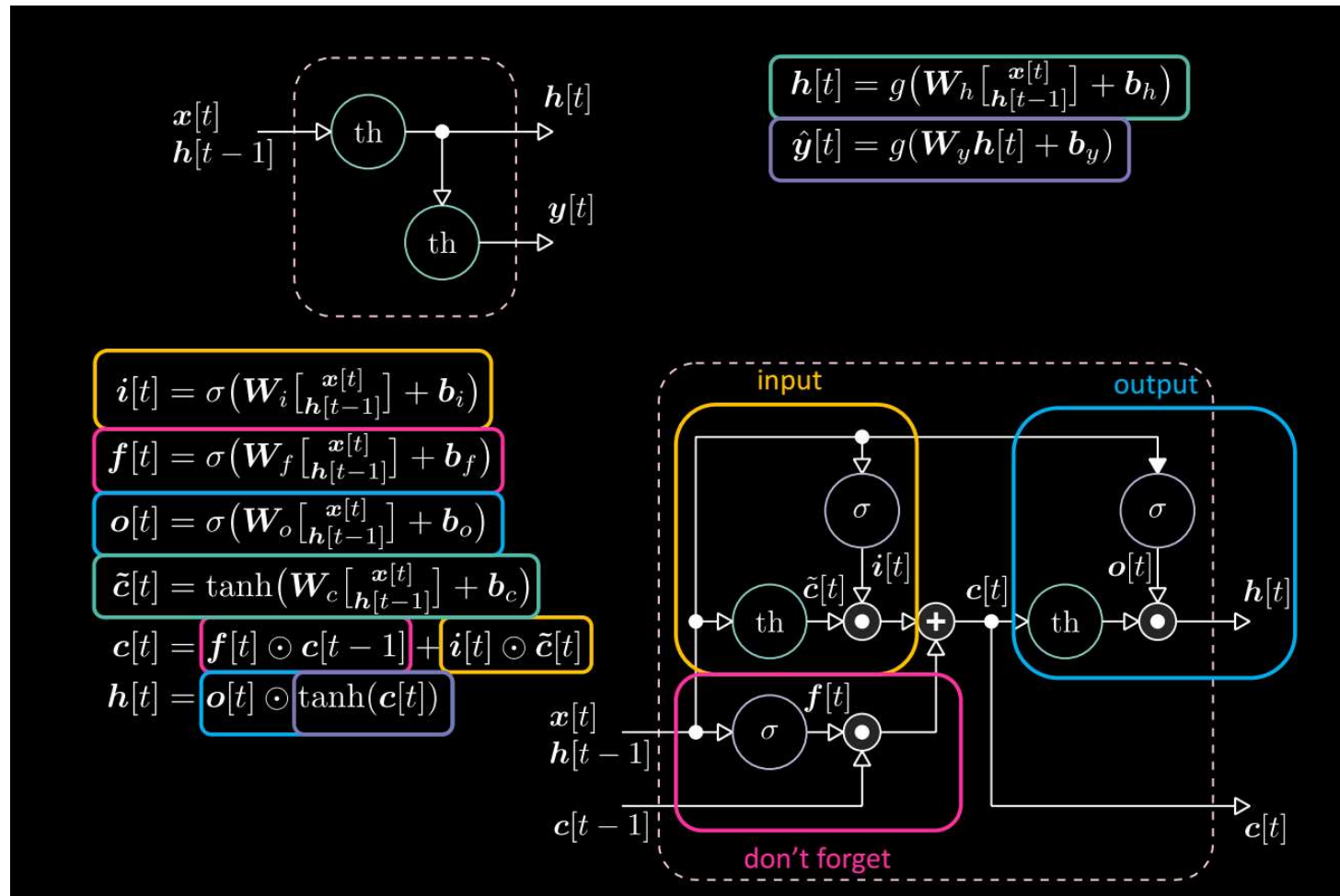
# Long short-term memory (LSTM)

El problema con Backpropagation a través del tiempo hace que sea difícil aprender dependencias entre términos de secuencias largas

Tenemos que resolver el problema de los gradientes desvanecientes.

En 1997, Hochreiter y Schmidhuber propusieron la red LTSM.

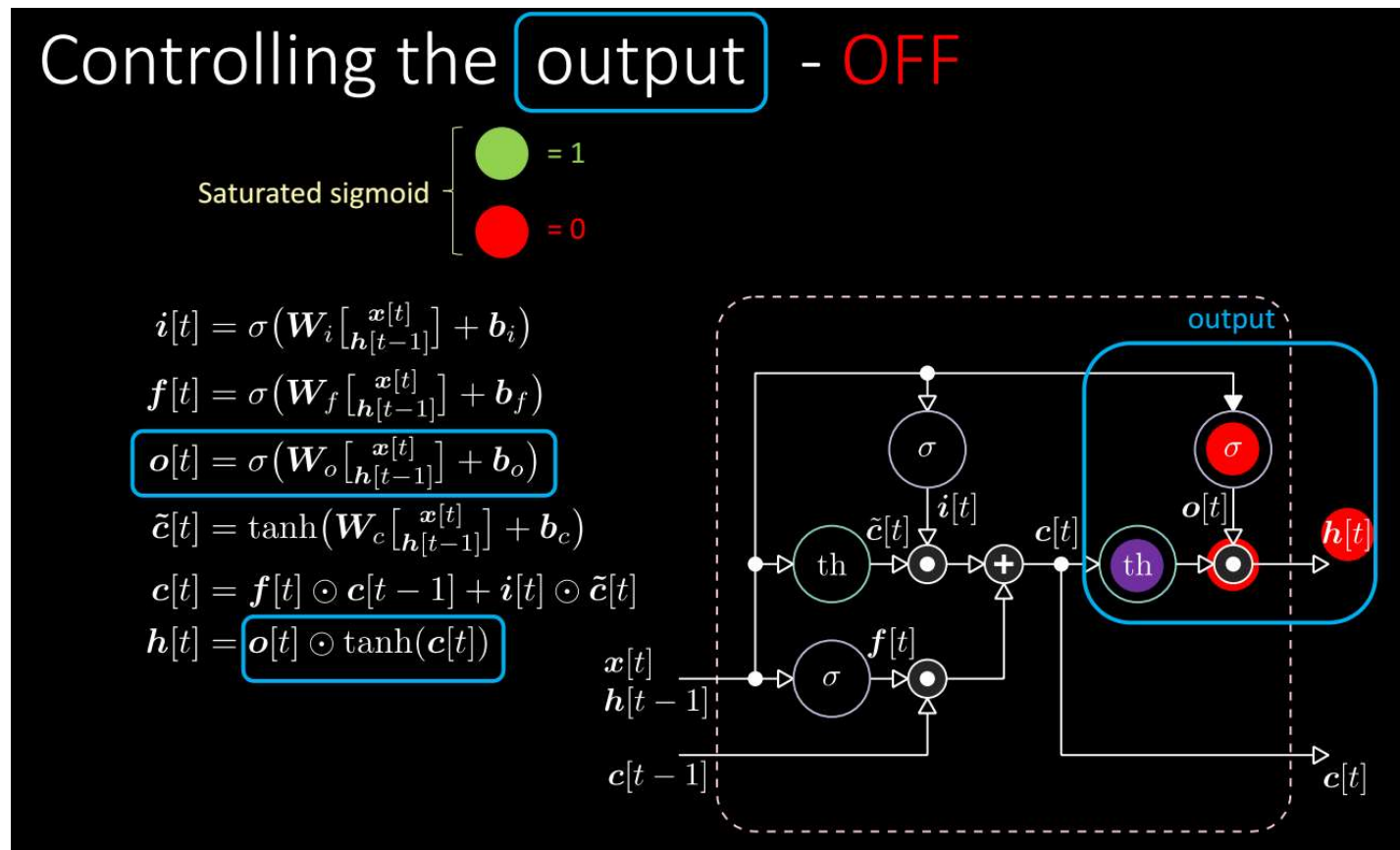
# Long short-term memory (LSTM)



Además de la entrada y la salida, este modelo introduce una memoria  $c[t]$

# Long short-term memory (LSTM)

La salida es controlada por la función de activación de  $o[t]$



# Long short-term memory (LSTM)

La salida es controlada por la función de activación de  $o[t]$

## Controlling the output - ON

Saturated sigmoid  $\begin{cases} \text{green circle} = 1 \\ \text{red circle} = 0 \end{cases}$

$$i[t] = \sigma(\mathbf{W}_i[\mathbf{x}[t]] + \mathbf{b}_i)$$

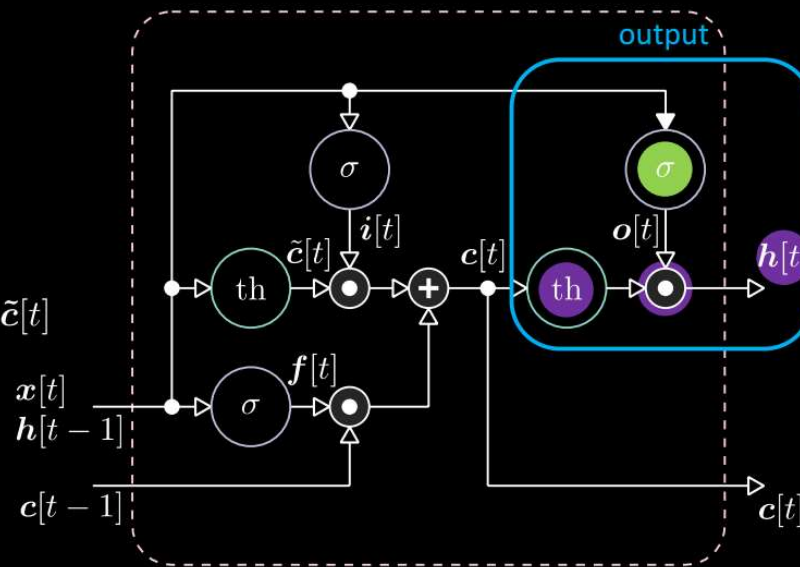
$$f[t] = \sigma(\mathbf{W}_f[\mathbf{x}[t]] + \mathbf{b}_f)$$

$$o[t] = \sigma(\mathbf{W}_o[\mathbf{x}[t]] + \mathbf{b}_o)$$

$$\tilde{c}[t] = \tanh(\mathbf{W}_c[\mathbf{x}[t]] + \mathbf{b}_c)$$

$$c[t] = f[t] \odot c[t-1] + i[t] \odot \tilde{c}[t]$$

$$h[t] = o[t] \odot \tanh(c[t])$$



# Long short-term memory (LSTM)

Controlling the **memory** - reset

Saturated sigmoid  $\left\{ \begin{array}{l} \text{green circle} = 1 \\ \text{red circle} = 0 \end{array} \right.$

$$i[t] = \sigma(W_i[x[t]] + b_i)$$

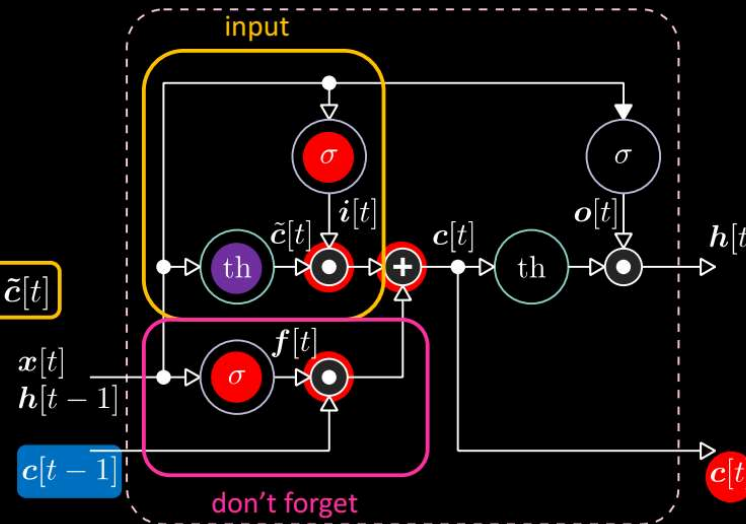
$$f[t] = \sigma(W_f[h[t-1]] + b_f)$$

$$o[t] = \sigma(W_o[h[t-1]] + b_o)$$

$$\tilde{c}[t] = \tanh(W_c[x[t]] + b_c)$$

$$c[t] = f[t] \odot c[t-1] + i[t] \odot \tilde{c}[t]$$

$$h[t] = o[t] \odot \tanh(c[t])$$



La memoria es reseteada si el valor de activación del gate forget y el gate input se desvanecen.

# Long short-term memory (LSTM)

## Controlling the **memory** - keep

Saturated sigmoid  $\begin{cases} \text{green circle} = 1 \\ \text{red circle} = 0 \end{cases}$

$$i[t] = \sigma(W_i[x[t]] + b_i)$$

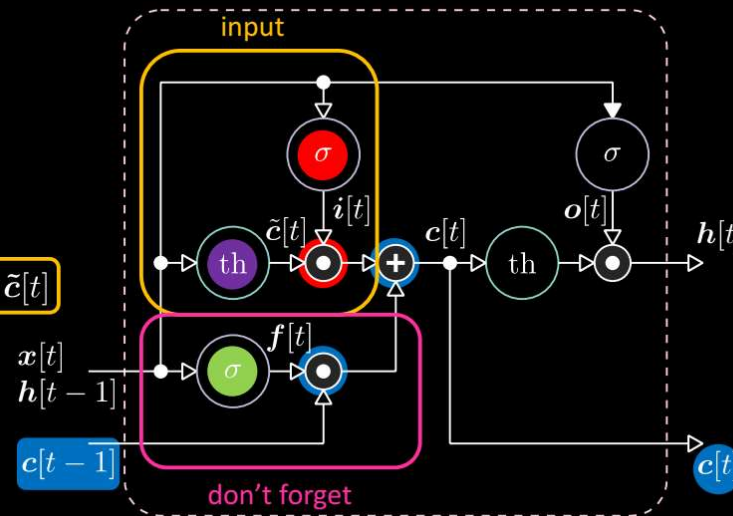
$$f[t] = \sigma(W_f[h[t-1]] + b_f)$$

$$o[t] = \sigma(W_o[h[t-1]] + b_o)$$

$$\tilde{c}[t] = \tanh(W_c[x[t]] + b_c)$$

$$c[t] = f[t] \odot c[t-1] + i[t] \odot \tilde{c}[t]$$

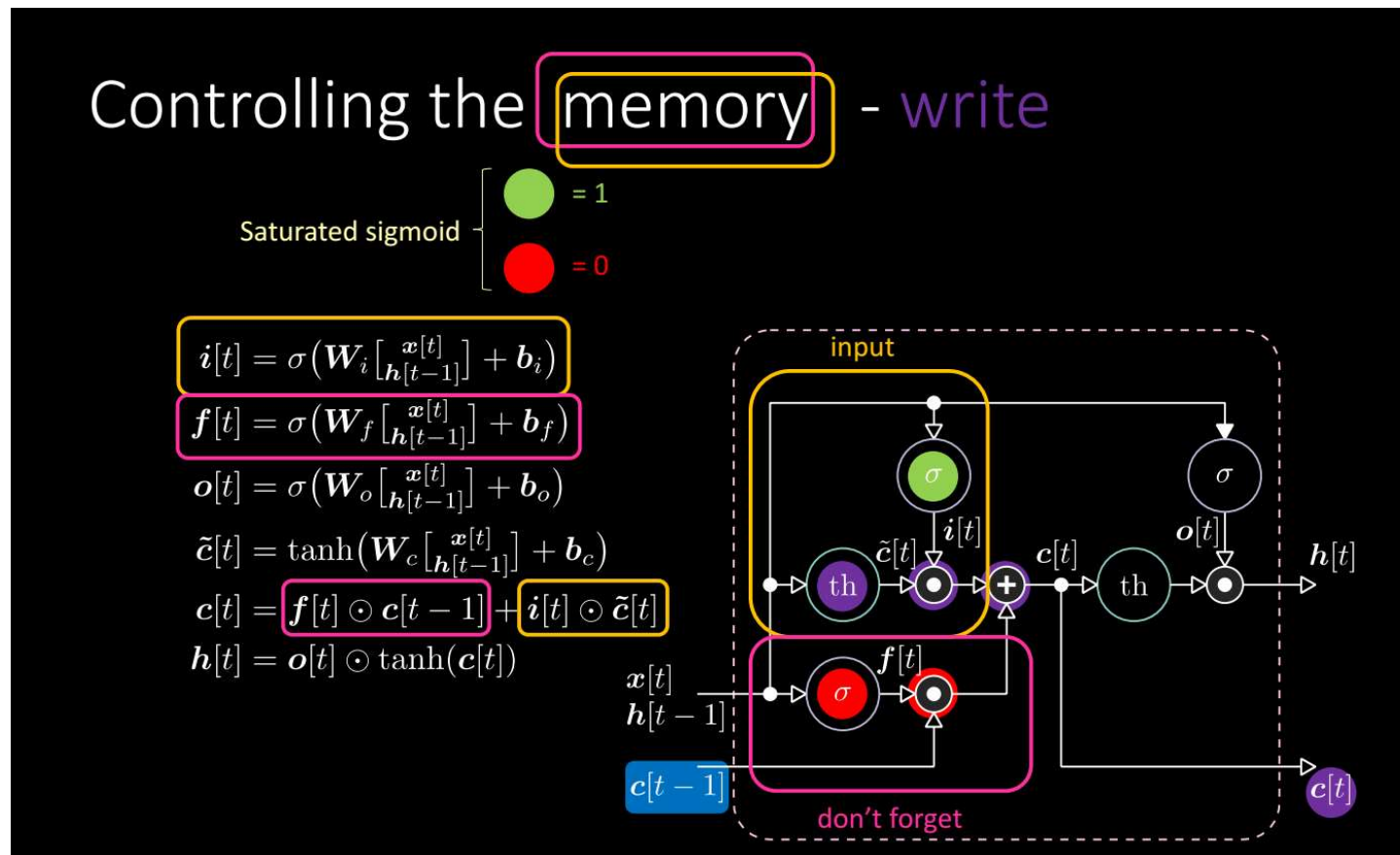
$$h[t] = o[t] \odot \tanh(c[t])$$



La memoria se mantiene si el valor de activación del forget gate se satura y el valor de activación del input se desvanece.

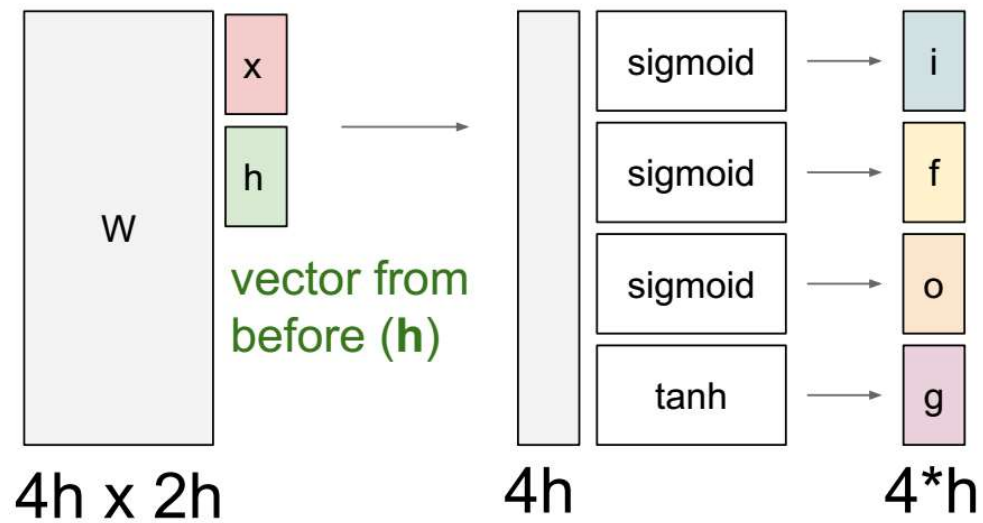


# Long short-term memory (LSTM)



La memoria es actualizada si el valor de activación del forget gate se desvanece y el valor de activación del input se satura.

# Long short-term memory (LSTM)

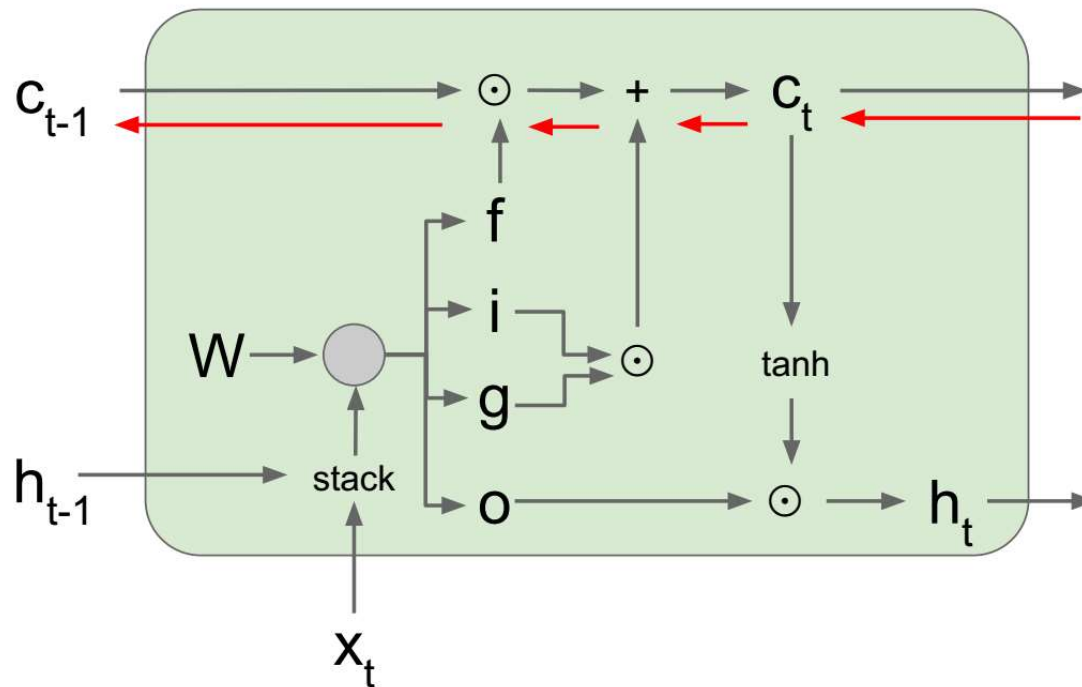


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

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

# Long short-term memory (LSTM)



Backpropagation from  $c_t$  to  $c_{t-1}$  only elementwise multiplication by  $f$ , no matrix multiply by  $W$

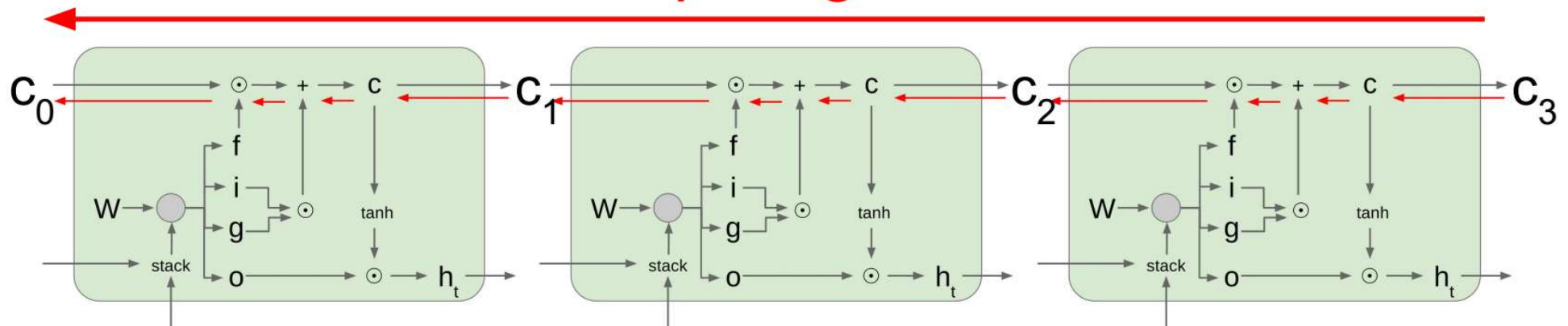
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

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

# Long short-term memory (LSTM)

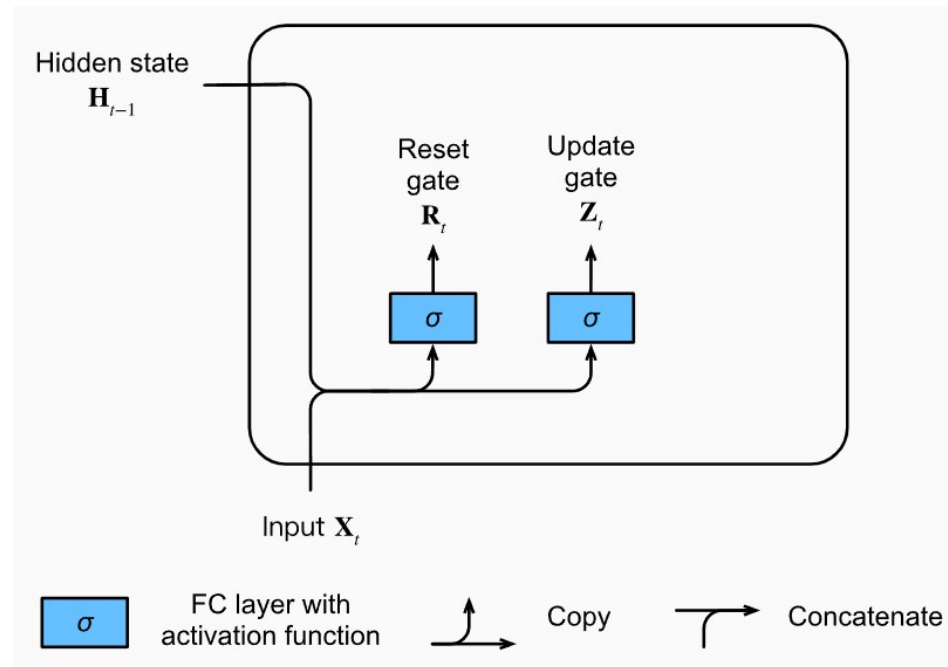
Uninterrupted gradient flow!



La memoria ayuda a preservar dependencias largas en los datos.

# Gated Recurrent Units (GRU)

Bajo la misma idea de LSTM, este modelo usa dos gates para controlar el proceso: reset gate y update gate. Al usar sólo dos gates, es más eficiente que LSTM, y los resultados son comparables



$$\begin{aligned}\mathbf{R}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r) \\ \mathbf{Z}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z)\end{aligned}$$

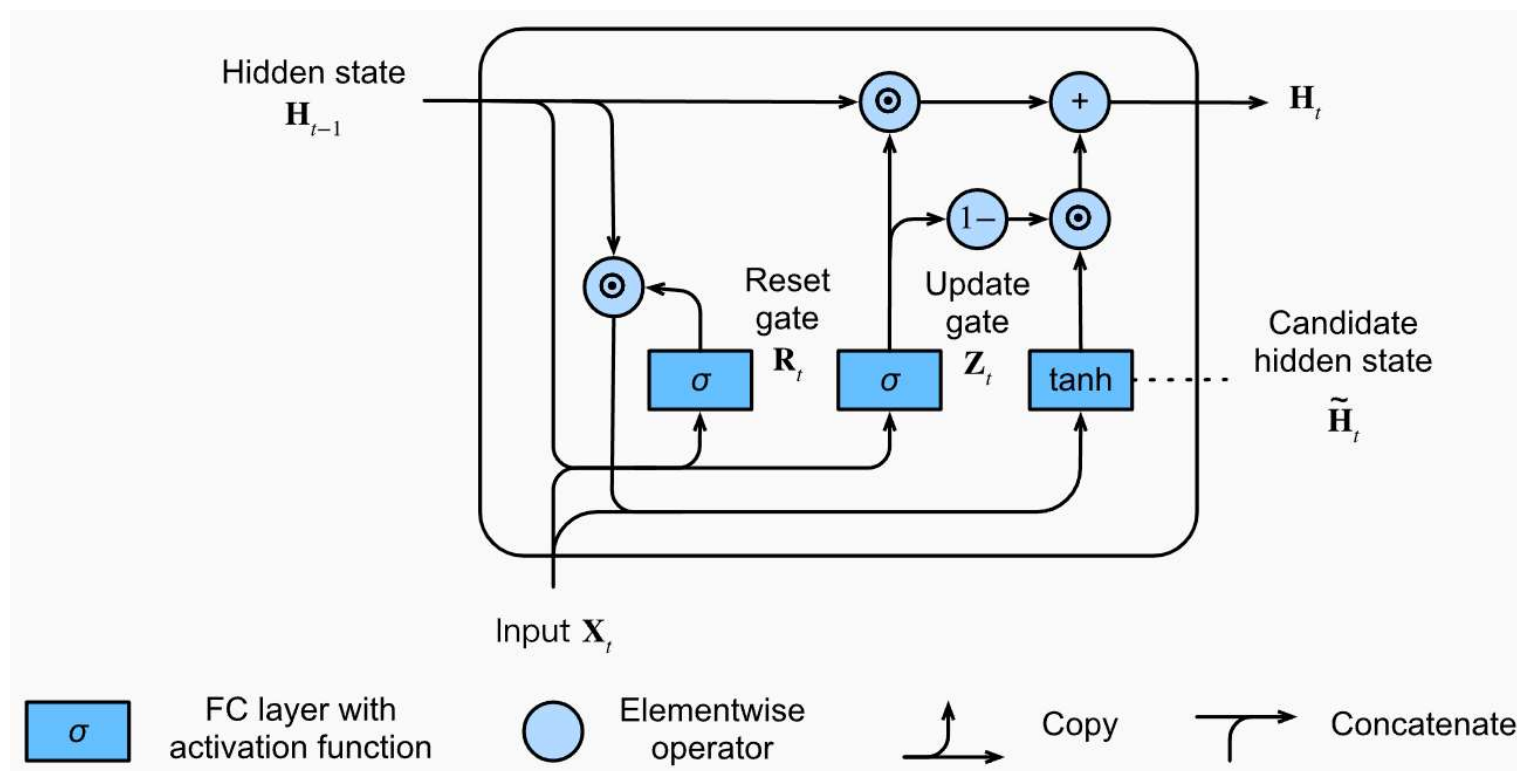
Estado oculto candidato

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

Estado oculto

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$

# Gated Recurrent Units (GRU)



# Recurrent Layers

We can add several RNN layers. These are not actually deep models.

## Multilayer RNNs

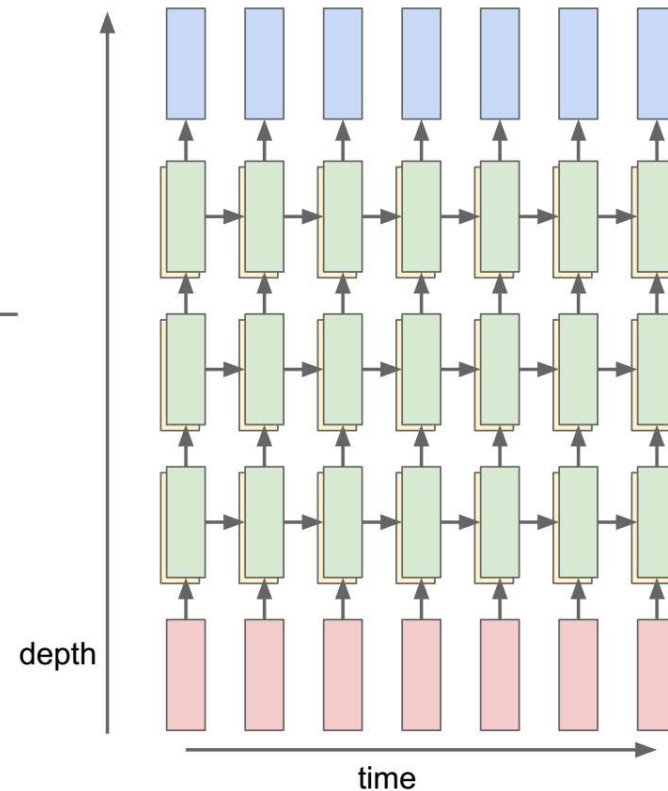
$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$        $W^l [n \times 2n]$

## LSTM:

$$W^l [4n \times 2n]$$

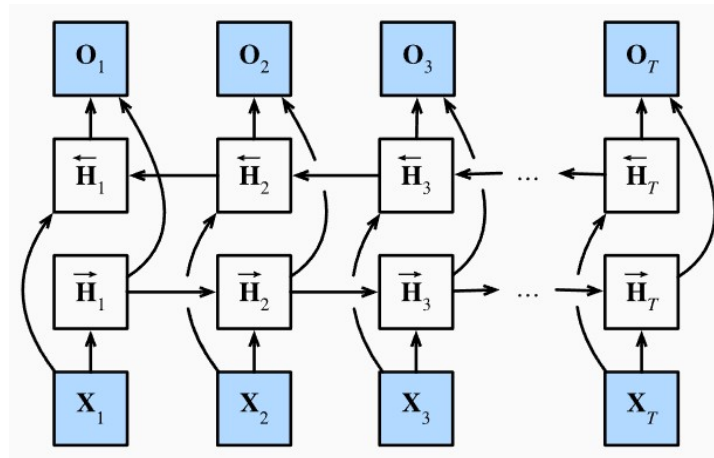
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$
$$c_t^l = f \odot c_{t-1}^l + i \odot g$$
$$h_t^l = o \odot \tanh(c_t^l)$$



# Bidirectional RNN

Una RNN (como vista hasta ahora es unidireccional).

Solo captura la dependencia en una dirección de la secuencia. Si queremos capturar las dependencias en ambos sentidos, usamos dos RNNs que consumen las secuencias en orden distinto.



$$\begin{aligned}\vec{H}_t &= \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \vec{H}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}) \\ \overleftarrow{H}_t &= \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{H}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)})\end{aligned}$$

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q$$



# Gradient clipping

LSTM y GRU lidian de alguna forma con el desvanecimiento de gradientes. Sin embargo, todavía es posible que en algunos casos particulares, los gradientes crezcan mucho, haciendo que el entrenamiento de una red diverja.

Una alternativa de solución es recortar los gradientes.

En gradiente descendente, la función de costo  $f(\mathbf{x})$  se minimiza por aplicar sucesivas actualizaciones a los parámetros, de la forma  $\mathbf{x} = \mathbf{x} - \eta \nabla f$ .

Si la función  $f$  es suficientemente suave, decimos que  $f$  es Lipschitz continua con constante  $L$  si

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L \|\mathbf{x} - \mathbf{y}\|$$

Por consiguiente:

$$|f(\mathbf{x}) - f(\mathbf{x} - \eta \nabla f)| \leq L\eta \|\nabla f\| \longrightarrow \text{Cota superior del gradiente}$$

# Gradient clipping

Si el gradiente se vuelve excesivamente grande, la optimización no converge

**Solución:** proyectar el gradiente a una hiperesfera de radio  $\theta$

$$\nabla f = \min\left(1, \frac{\theta}{\|\nabla f\|}\right) \nabla f$$

Nota: se cambia la magnitud del gradiente, por lo que ya no se sigue el gradiente original. Aún se desconoce a nivel teórico sobre los efectos secundarios de esta heurística. Sin embargo, en la práctica, funciona.