Técnicas de Aprendizaje Automático - Machine Learning

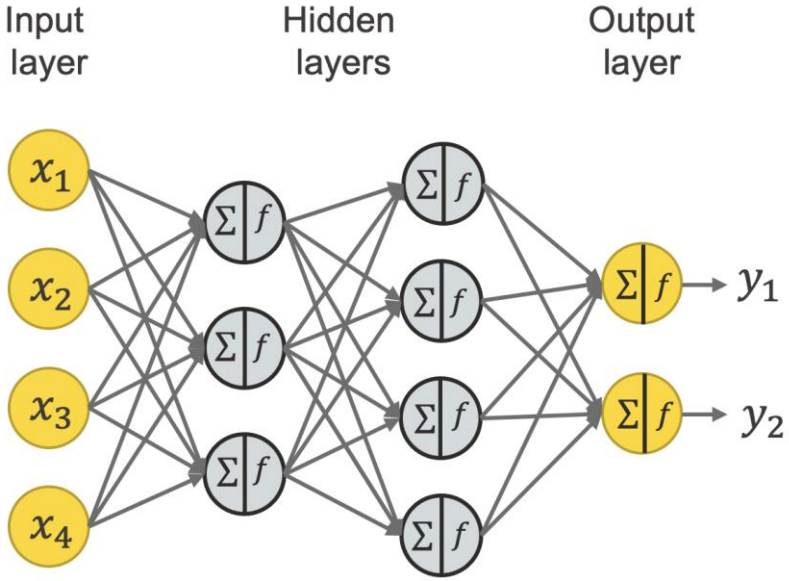
Tema 9: Redes Neuronales Recurrentes

María Inés Pisarello – Sebastián Yair Suaid – Diego Acosta Coden

Faculta de Ciencias Exactas y Naturales y Agrimensura
UNNE - 2023

Repaso de lo visto anteriormente sobre redes

En las redes comunes, los input son vectores de datos de una sola dimensión, de tamaño nx1. Los datasets de entrenamiento y validación son tabulares. Cada fila es un ejemplo, señal, observación y cada columna puede representar una característica o muestra.



03.ancho-fijo-tablas-html-xml.R | ibex35

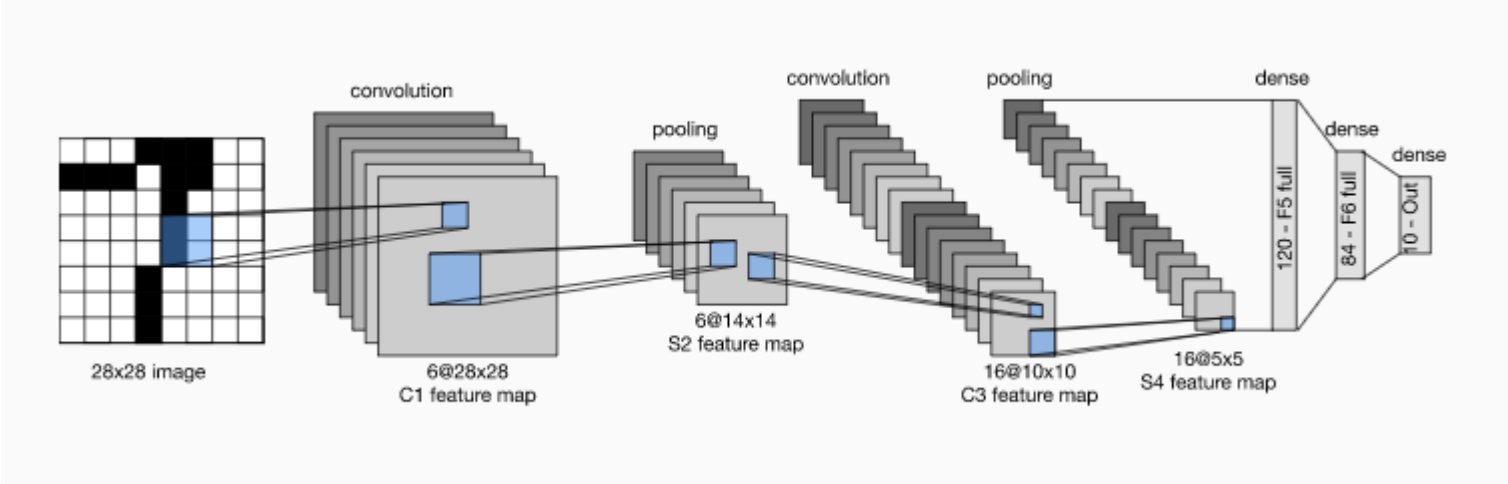
Filter

	Nombre	Cotización	Var %	Máx.	Mín.	Var % año	Vol. (mill. €)	Capit. (mill. €)	PER	Rent/Div	Hora
1	ACS	36,41	0,69 %	36,45	36,00	7,63	14,59	11.222	11,07	5,29	17:35
2	ACERINOX	8,37	1,41 %	8,37	8,26	-3,42	5,89	2.260	14,44	5,91	17:35
3	AENA SA	165,00	0,55 %	165,15	163,70	21,55	15,11	24.750	17,78	4,44	17:35
4	AMADEUS	67,06	1,12 %	67,06	66,08	10,22	27,28	28.370	24,29	1,87	17:35
5	ACCIONA	91,60	-0,81 %	92,65	91,30	23,95	3,41	4.988	17,77	3,94	17:35
6	BBVA	4,89	-0,02 %	4,93	4,84	5,38	67,31	32.342	7,34	5,41	17:35
7	BANKIA	1,78	0,88 %	1,79	1,75	-30,57	11,73	5.422	8,42	7,26	17:35
8	BANKINTER	6,29	-0,22 %	6,35	6,26	-10,40	11,83	5.649	10,52	4,78	17:35
9	CAIXABANK	2,55	0,35 %	2,57	2,53	-19,50	28,09	15.227	8,13	5,71	17:35
10	CELLNEX TELE...	38,84	1,52 %	39,21	38,11	73,47	40,53	14.958	339,33	0,23	17:35
11	CIE AUTOMOTIVE	22,56	1,26 %	22,56	22,14	5,22	3,07	2.910	9,96	3,04	17:35
12	COLONIAL	11,42	0,09 %	11,46	11,37	40,38	4,36	5.802	40,63	1,90	17:35
13	ENDESA	23,91	-0,54 %	24,09	23,86	18,78	11,92	25.315	16,68	6,03	17:35
14	ENAGAS	20,87	-0,86 %	21,21	20,83	-11,61	18,83	4.972	11,76	7,68	17:35

Showing 1 to 14 of 35 entries

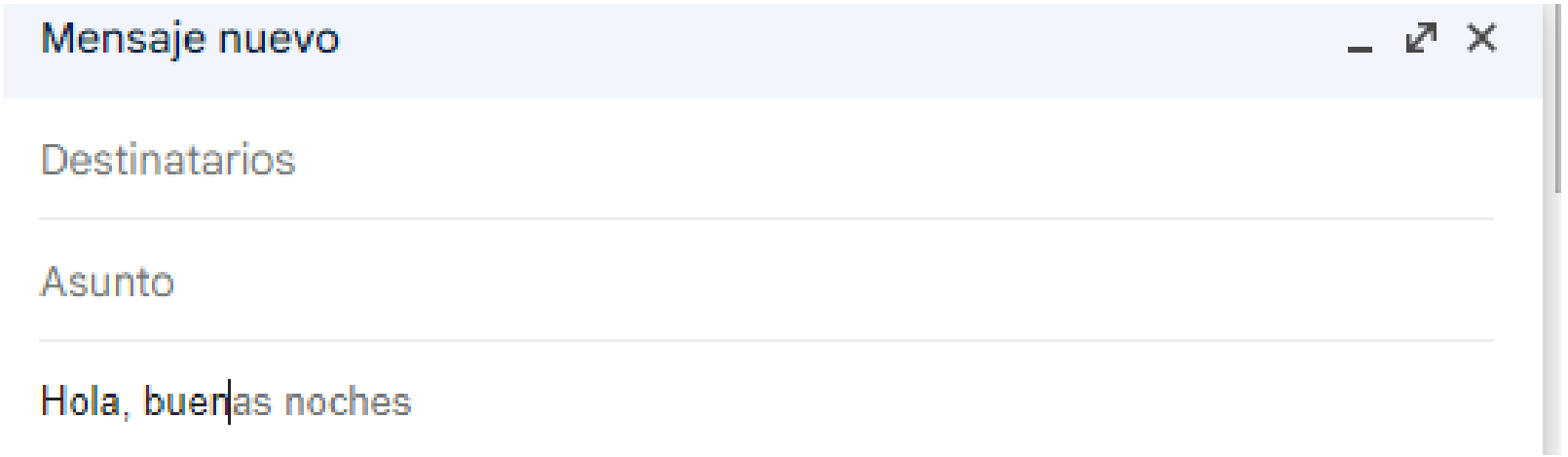
Repaso de lo visto anteriormente sobre redes

En las redes convolucionales, los elementos de nuestra red son tensores de segundo o tercer orden. Debido a que los parámetros de la red se adaptan para tamaños de entradas y salidas de datos determinadas, el tamaño de los datos de entrada es constante.



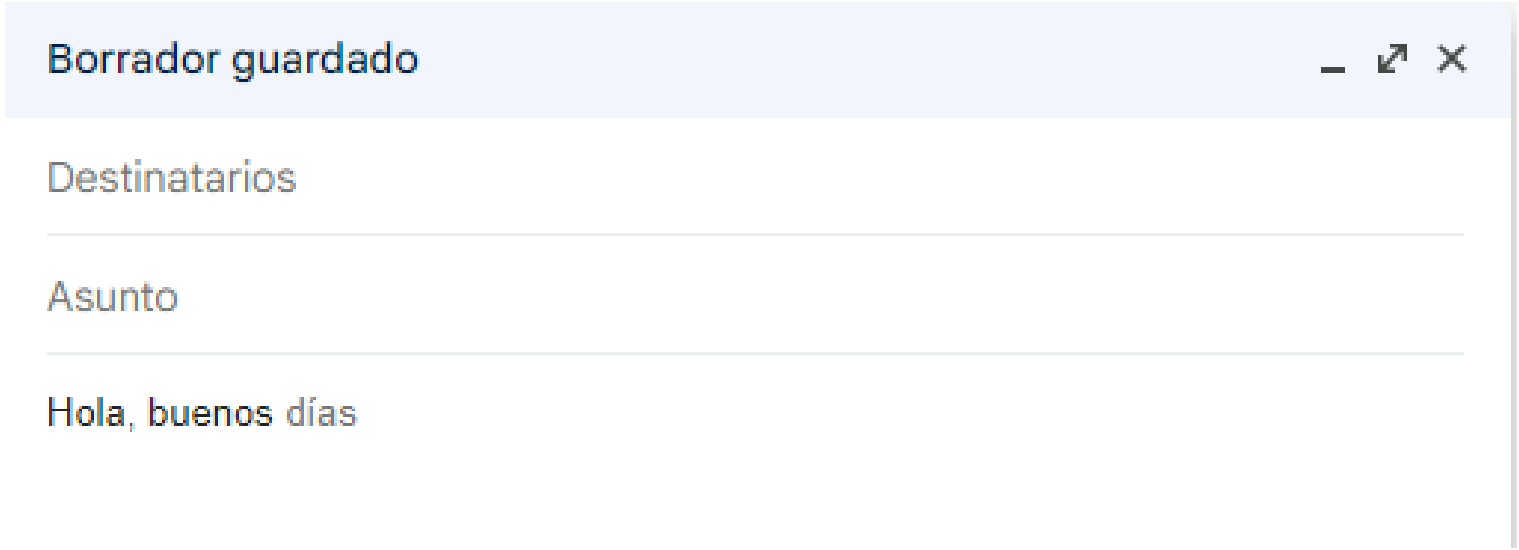
Ejemplo introductorio de redes recurrentes

En la actualidad son muy comunes los predictores de texto (correo electrónico, apps de mensajería, barras de búsqueda, editores de texto) que al mismo tempo que el usuario escribe, predicen cuál será la próxima palabra que podría llegar a elegir.

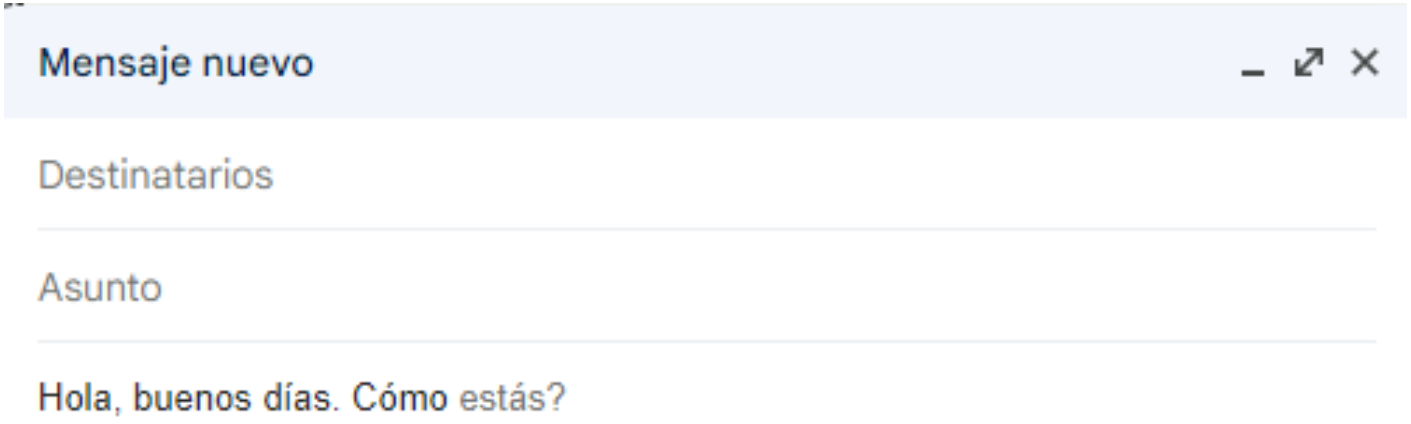


Ejemplo introductorio de redes recurrentes

Podemos pensar en estos predictores como sistemas que analizan la información que está disponible (el texto ya escrito) y generan una sugerencia que puede ser correcta o incorrecta.

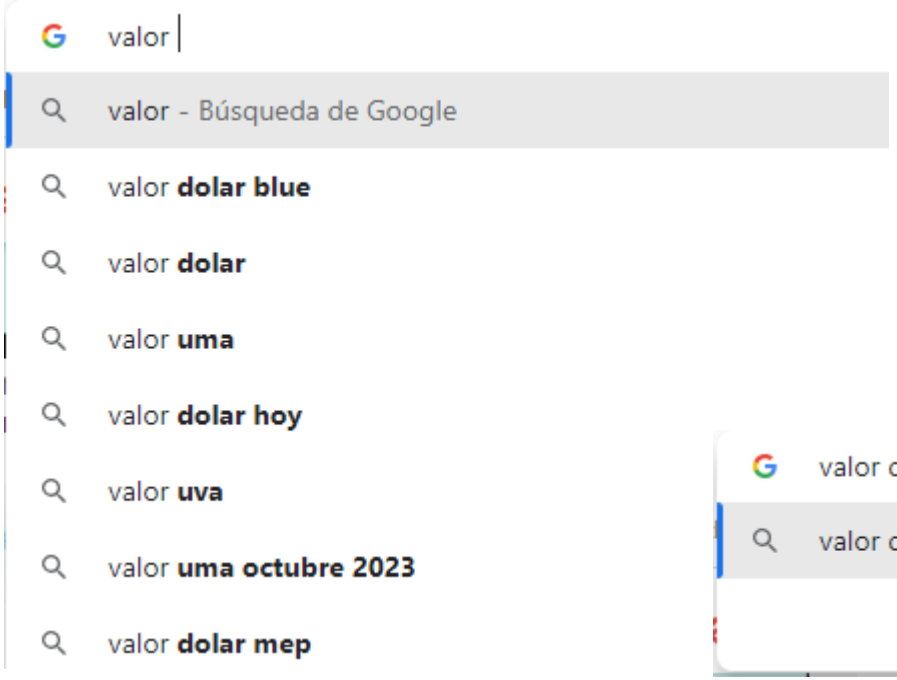


Ejemplo introductorio de redes recurrentes

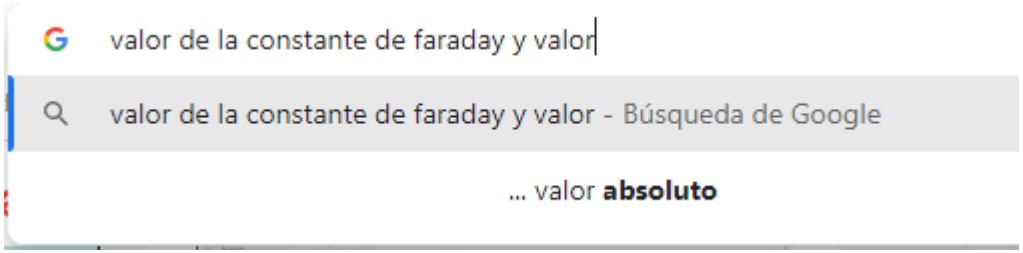


Ejemplo introductorio de redes recurrentes

Un aspecto importante de estos modelos de predicción, es que **NO** realizan un mapeo entre una palabra y la predicción de la palabra que le debe seguir.



Por lo tanto, la predicción puede variar, aun cuando la palabra en el momento "actual" sea la misma.



Ejemplo introductorio de redes recurrentes

Estos sistemas entonces deben ser capaces de mirar para atrás, y que la información contenida en las entradas anteriores tenga una influencia en la forma en la que el sistema o modelo calcule la salida actual.

Para que la salida de este sistema tenga en cuenta información pasada, el mismo debe tener la capacidad de almacenar **memoria**. Para esto debemos introducir un nuevo concepto en el análisis de modelos basados en redes neuronales, y es el concepto de **estado**.

Otro punto a tener en cuenta, es que, a diferencia de las redes ya vistas, en estos casos los datos de entrada son de **longitud variable**.

Redes neuronales recurrentes

Definimos entonces un nuevo tipo de redes, a las cuales le damos el nombre de **redes neuronales recurrentes (RNN)**.

En las redes comunes, podemos decir que cada input es **independiente** una de otra. En el dataset, los vectores representan datos que pueden comportarse de manera similar o tener una dinámica determinada.

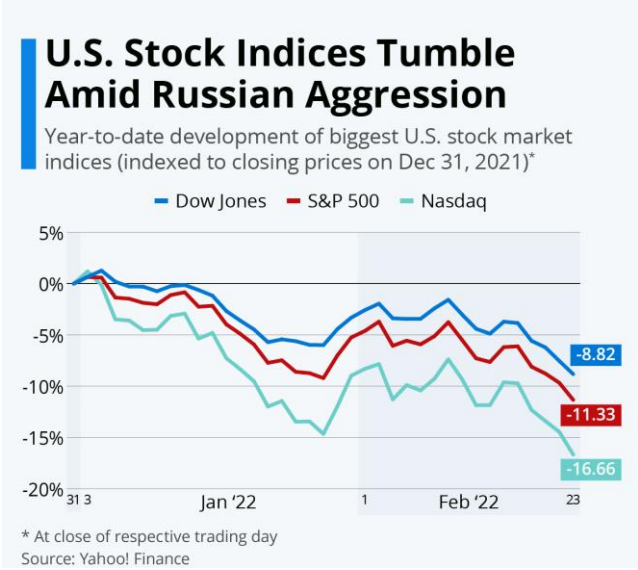
Sin embargo, una vez que la red está entrenada, la salida no depende de cuáles input se introducen, o del orden en que lo hagan.

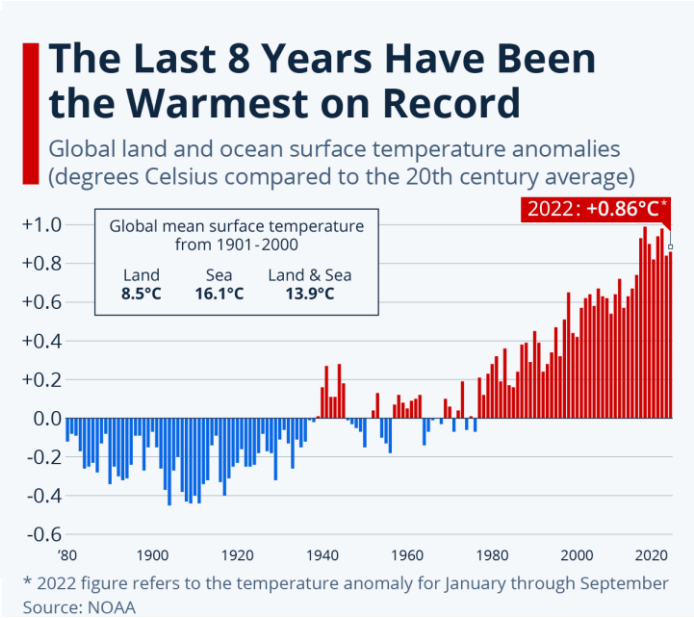
Por lo tanto, podemos decir que las redes comunes almacenan información pero no tienen forma de almacenar memoria.

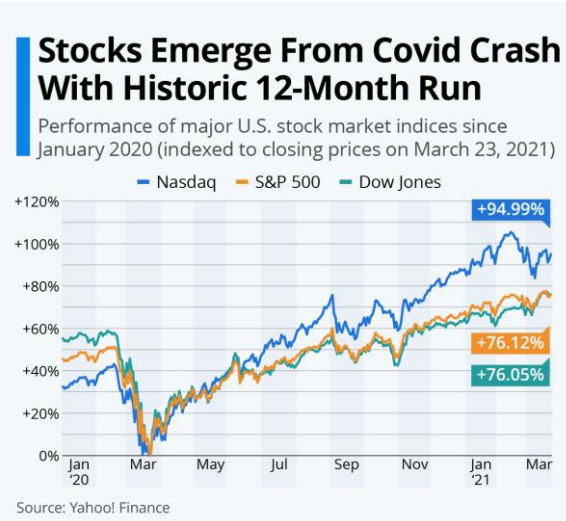
Redes neuronales recurrentes

Las redes recurrentes pueden utilizarse para predecir el funcionamiento de sistemas dinámicos, como ser las variables meteorológicas de una región, las fluctuaciones de la bolsa de valores, la evolución de una población, o señales de voz y video.

En estos casos, los datos con los que se trabajan son de tipo **secuenciales**.

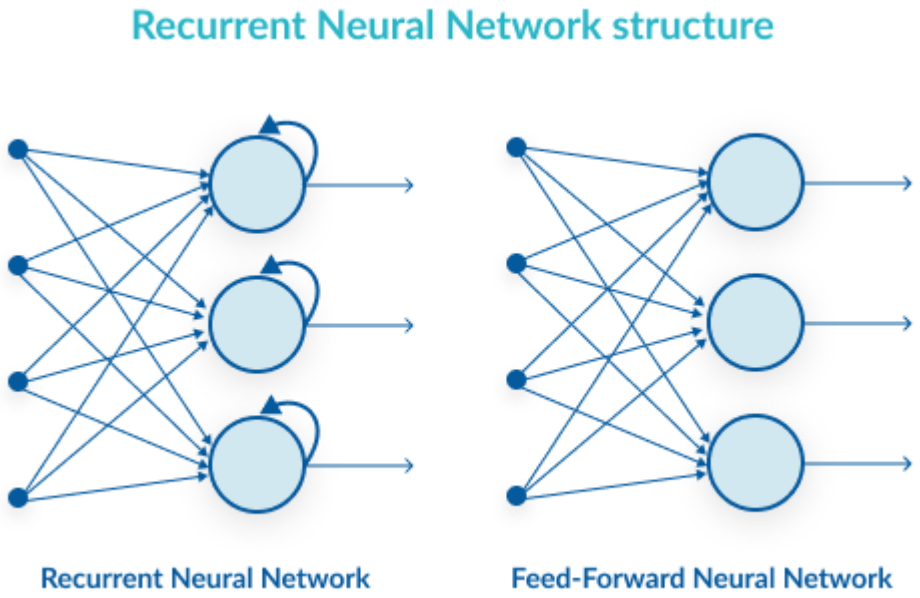






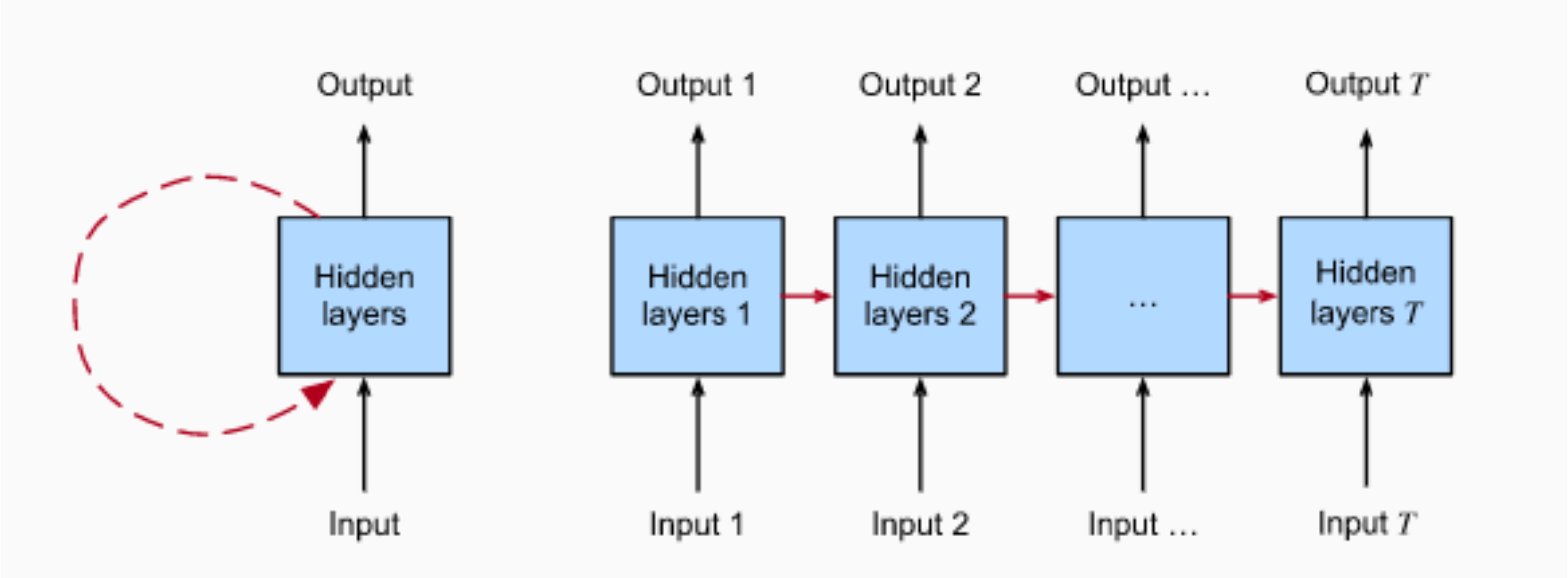
Redes neuronales recurrentes

Para que estas redes puedan cumplir con los requisitos nombrados anteriormente, se debe alterar la arquitectura dada a las redes comunes. Naturalmente la forma de hacerlo es incluyendo loops recursivos donde los datos se realimenten, es decir, la salida de un nodo se comporta también como una de las entradas del nodo.

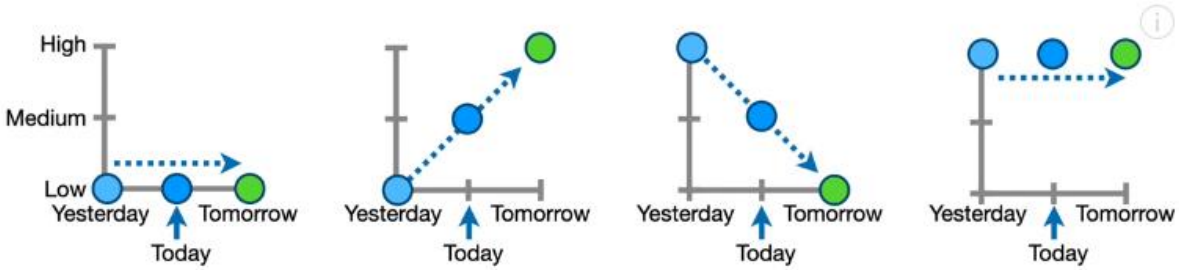


Redes neuronales recurrentes

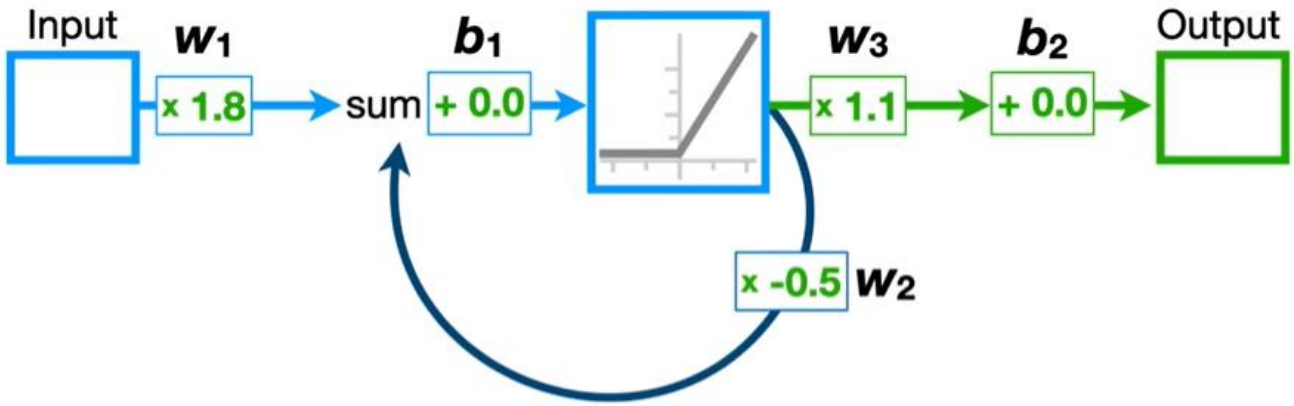
En las redes comunes, los datos se computan de forma sincrónica, mientras que en las recurrentes, se computan en *time steps* discretos, de forma secuencial. Analizamos el funcionamiento de un nodo con un loop recursivo sobre si mismo, desplegando el mismo para verlo como una red convencional.



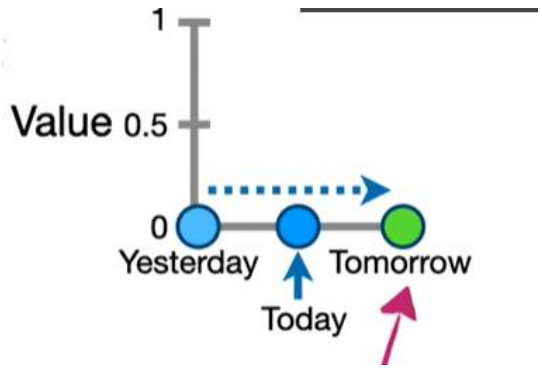
Ejemplo simple de una red recurrente

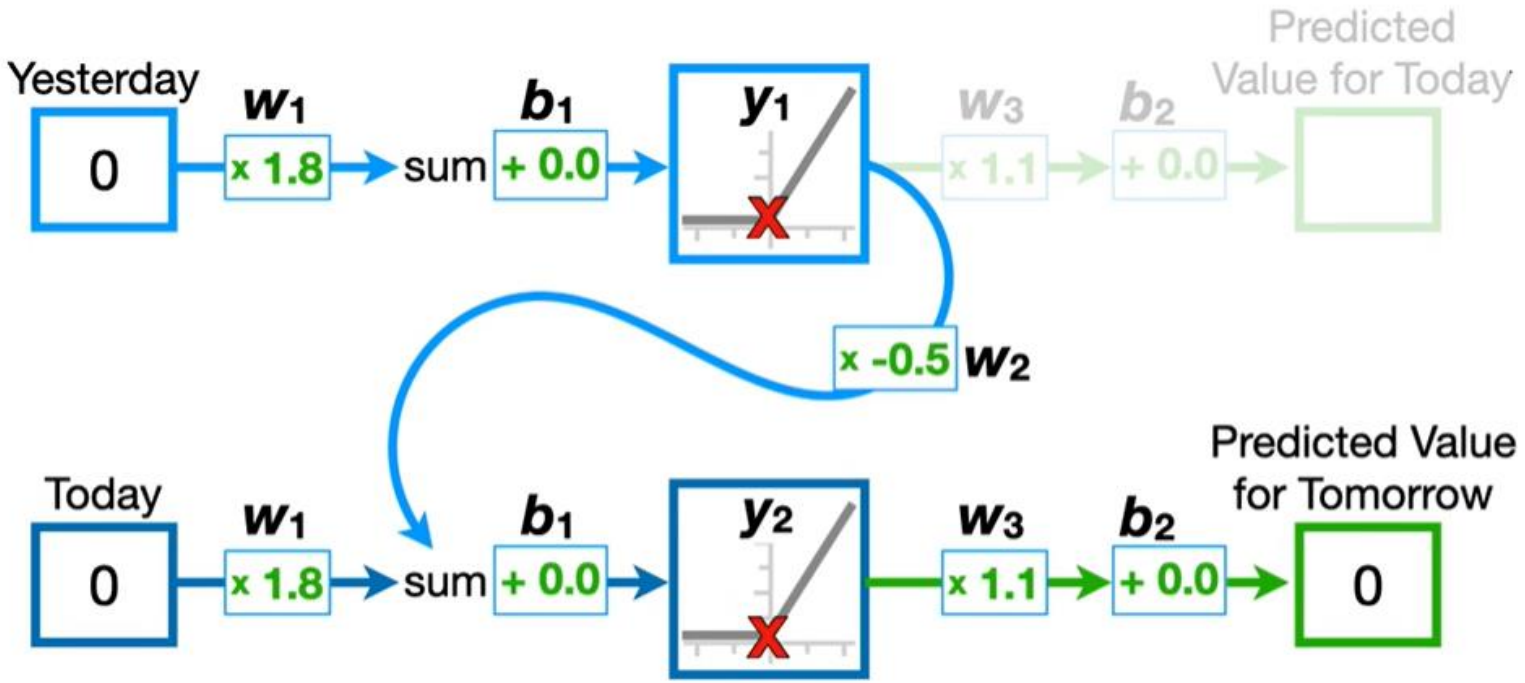


Ejemplo: Predicción de la evolución en los Mercados

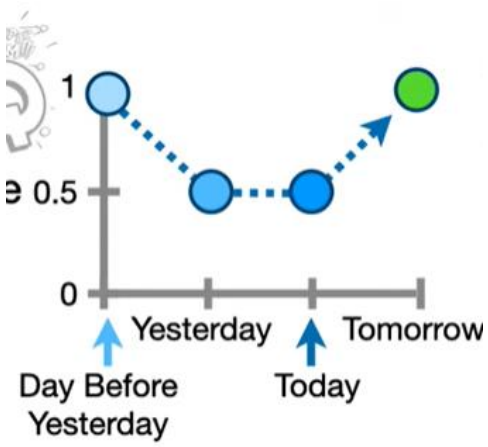


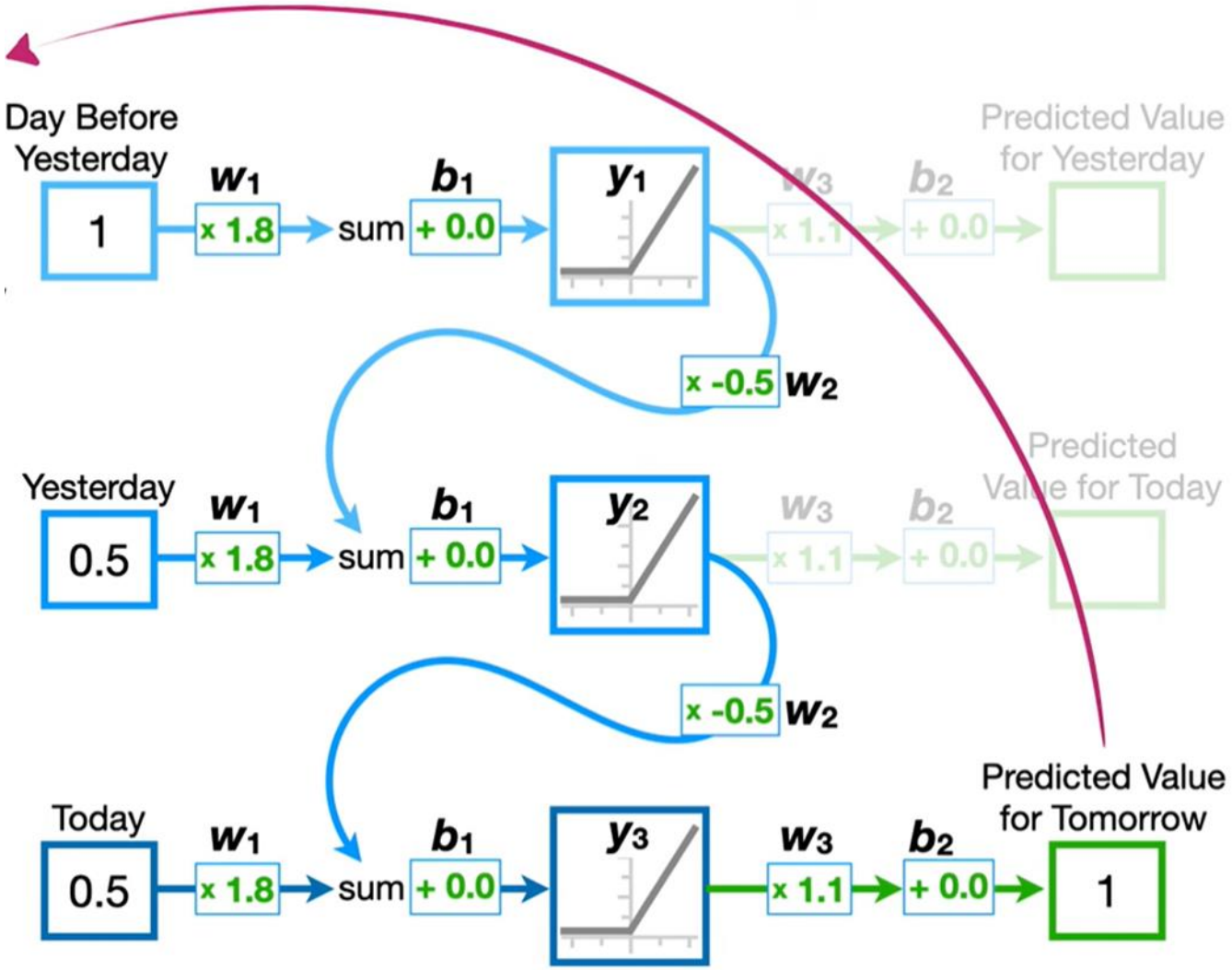
Ejemplo simple de una red recurrente



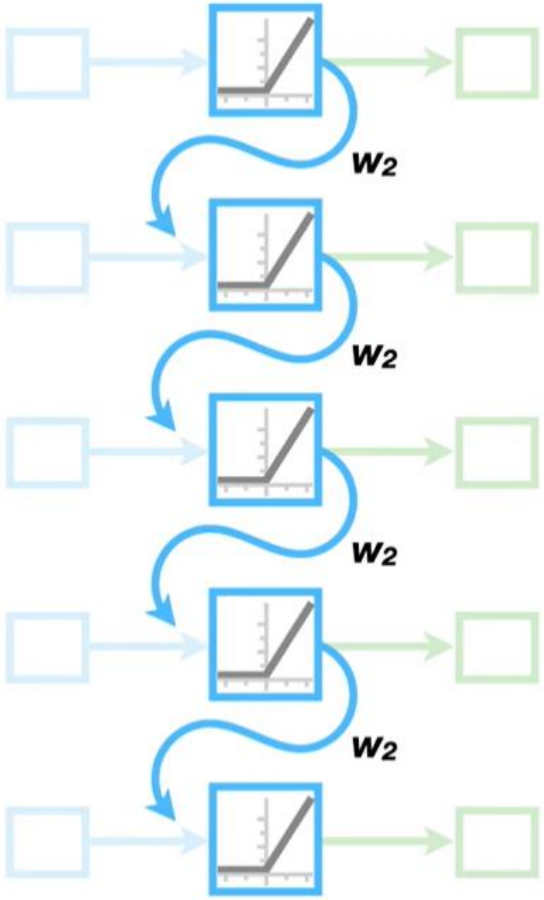


Ejemplo simple de una red recurrente





Ejemplo simple de una red recurrente



Entrenamiento: BPTT

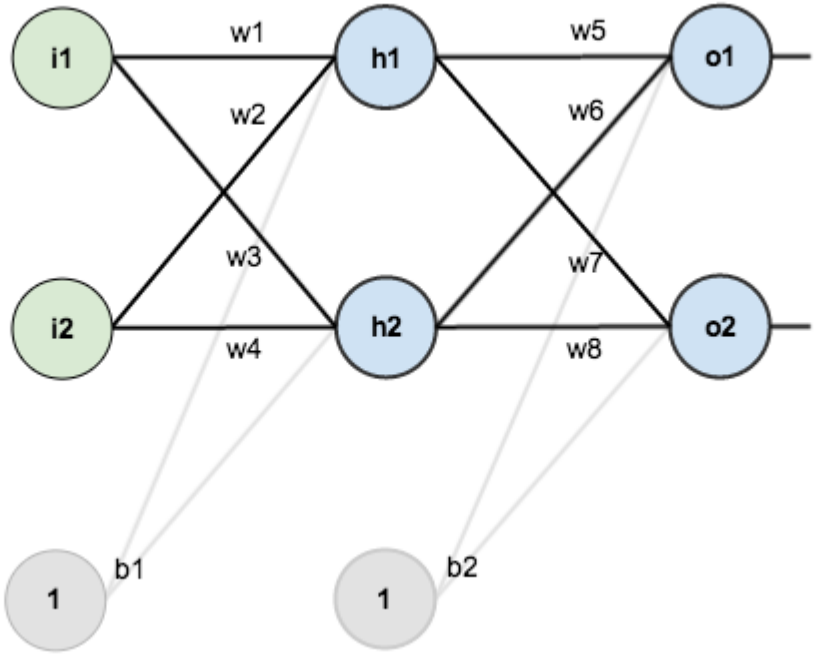
Así como en las redes completamente conectadas, la actualización de los parámetros de la red se realiza mediante el descenso del gradiente, basado en la aplicación de un algoritmo de **backpropagation**.

La propagación hacia atrás calcula los gradientes de la función de pérdida respecto de cada parámetro a entrenar. Los cálculos necesarios para hacerlo involucran derivadas parciales y la aplicación de la regla de la cadena.

En redes recurrentes, dicho algoritmo se extiende para dar cuenta de la recursión, y se denomina **backpropagation through time (BPTT)**.

Backpropagation

Veamos como funcionaba el algoritmo de backpropagation en las redes comunes. Si en esta red queremos calcular la componente del gradiente para los parámetros w5 y w1, es decir $\partial L / \partial W5$ y $\partial L / \partial W1$, debemos propagar el error desde atrás hacia adelante.

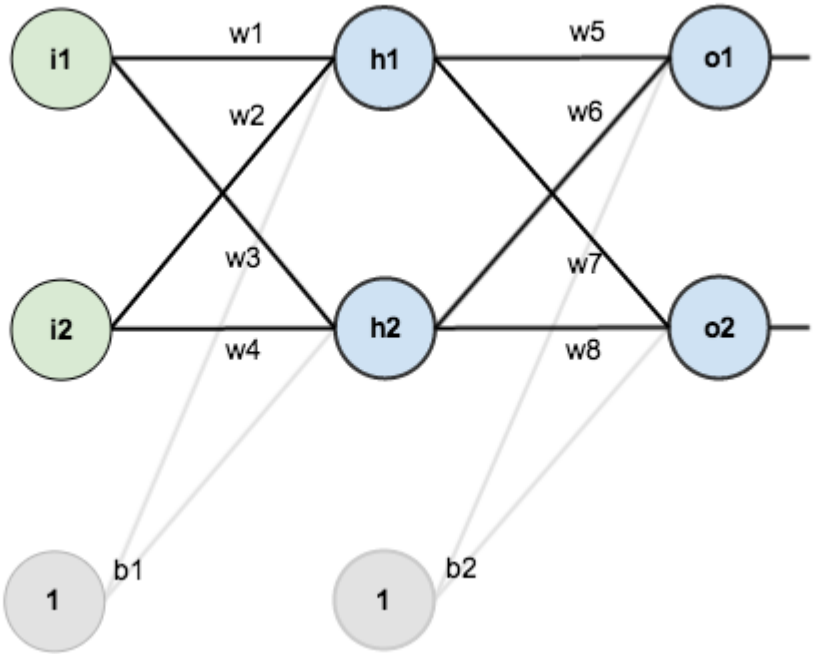


Backpropagation

$$\frac{\partial L}{\partial W5} = \frac{\partial L}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial in_{o1}} \frac{\partial in_{o1}}{\partial w5}$$

$$L = \frac{1}{2}(target_{o1} - out_{01})^2 + \frac{1}{2}(target_{o2} - out_{02})^2$$

$$\frac{\partial L}{\partial out_{o1}} = -(target_{o1} - out_{01})$$

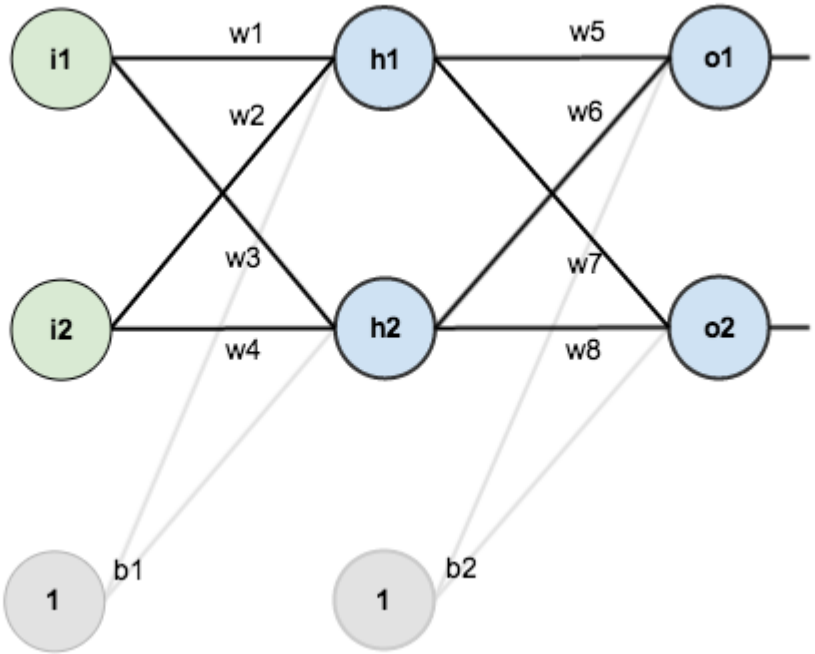


Backpropagation

$$\frac{\partial L}{\partial W5} = \frac{\partial L}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial in_{o1}} \frac{\partial in_{o1}}{\partial w5}$$

$$out_{o1} = \frac{1}{1 + e^{-in_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial in_{o1}} = out_{01}(1 - out_{01})$$

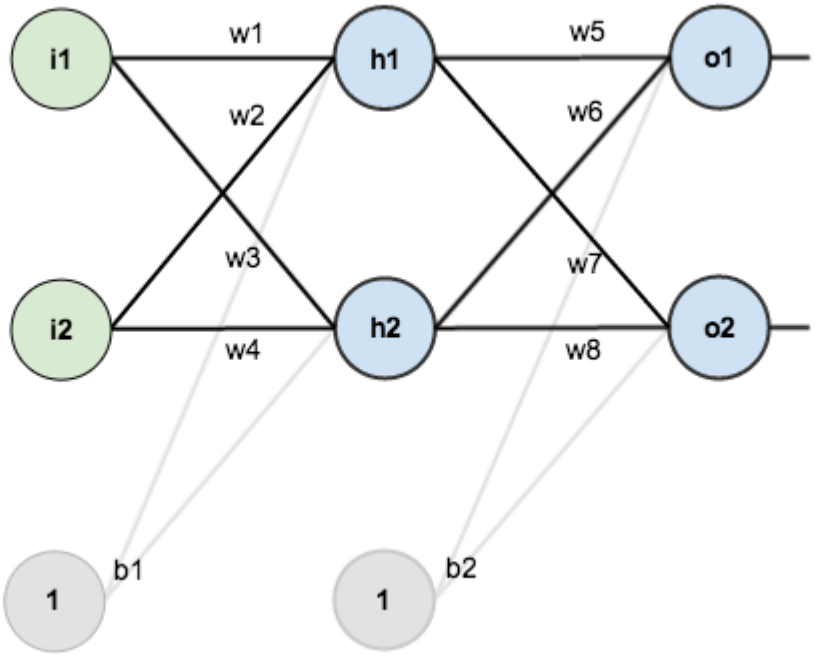


Backpropagation

$$\frac{\partial L}{\partial W5} = \frac{\partial L}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial in_{o1}} \frac{\partial in_{o1}}{\partial w5}$$

$$in_{o1} = w_5 out_{h1} + w_6 out_{h2} + b_2$$

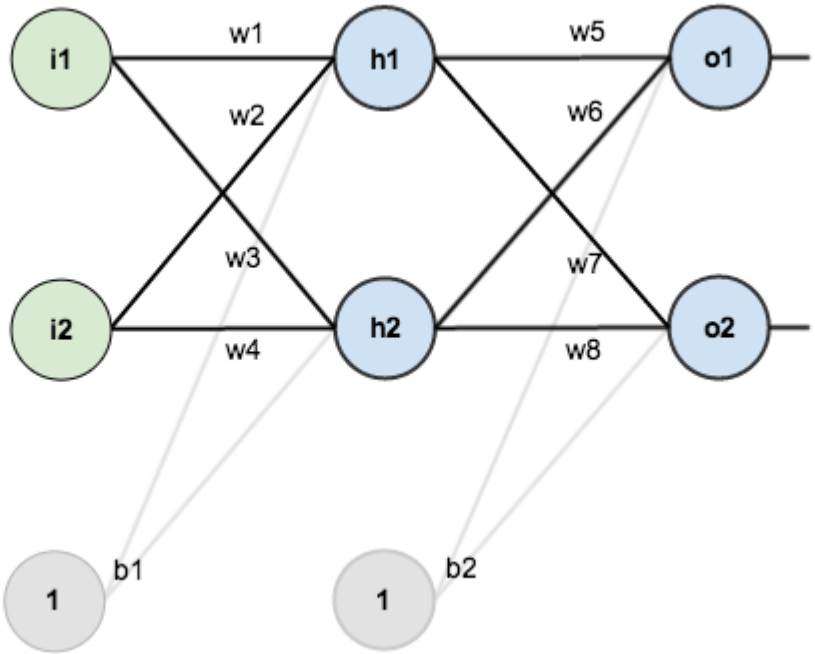
$$\frac{\partial in_{o1}}{\partial w5} = out_{h1}$$



Backpropagation

$$\frac{\partial L}{\partial W1} = \frac{\partial L}{\partial out_{h1}} \frac{\partial out_{h1}}{\partial in_{h1}} \frac{\partial in_{h1}}{\partial w1}$$

$$\frac{\partial L}{\partial out_{h1}} = \frac{\partial E_{O1}}{\partial out_{o1}} + \frac{\partial E_{O2}}{\partial out_{o2}}$$

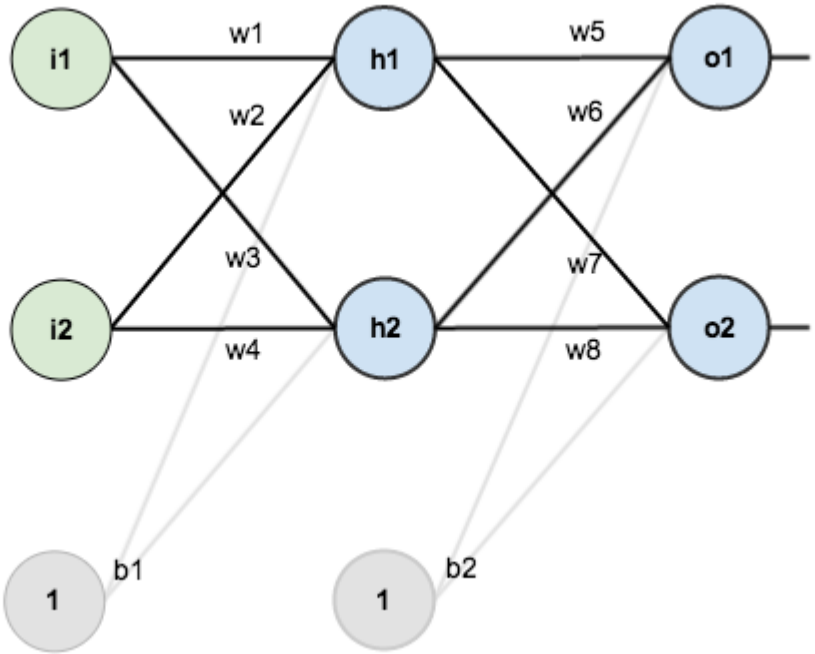


Backpropagation

$$\frac{\partial L}{\partial W1} = \frac{\partial L}{\partial out_{h1}} \frac{\partial out_{h1}}{\partial in_{h1}} \frac{\partial in_{h1}}{\partial w1}$$

$$out_{h1} = \frac{1}{1 + e^{-in_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial in_{h1}} = out_{h1}(1 - out_{h1})$$

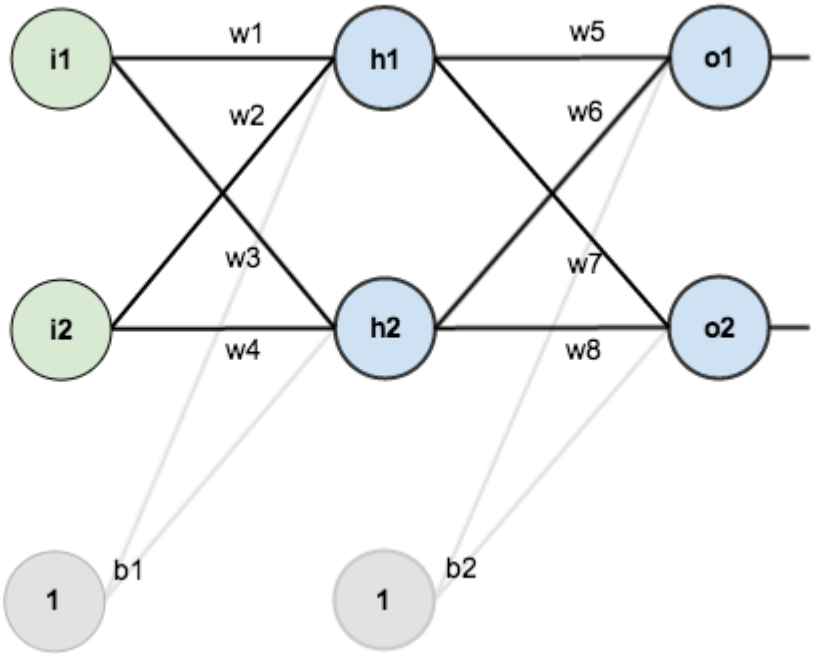


Backpropagation

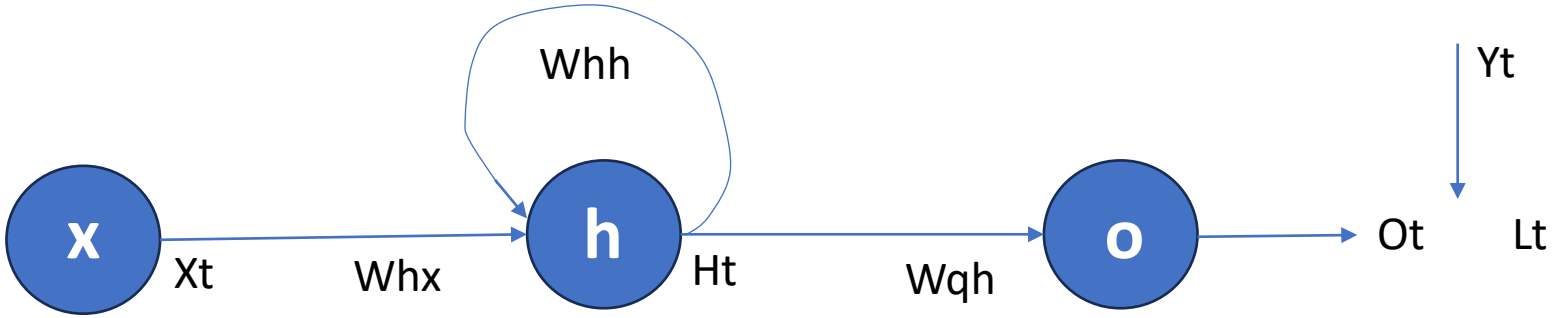
$$\frac{\partial L}{\partial W1} = \frac{\partial L}{\partial out_{h1}} \frac{\partial out_{h1}}{\partial in_{h1}} \frac{\partial in_{h1}}{\partial w1}$$

$$in_{h1} = w_1 i1 + w_2 i2 + b_1$$

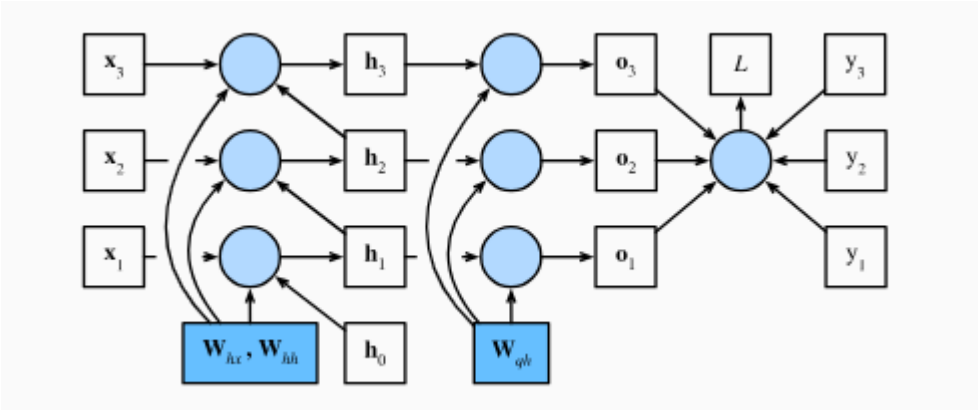
$$\frac{\partial in_{h1}}{\partial w1} = i1$$



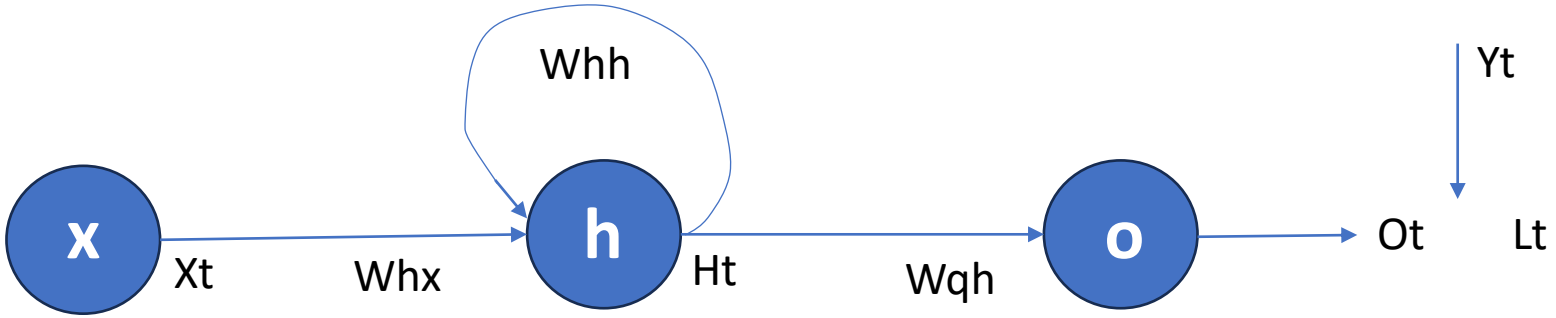
Backpropagation through time



Tomemos el caso de una red donde los parámetros a entrenar son: Whx, Whh y Wqh. Supongamos que no hay bias y que la función de activación es la función identidad.



Backpropagation through time

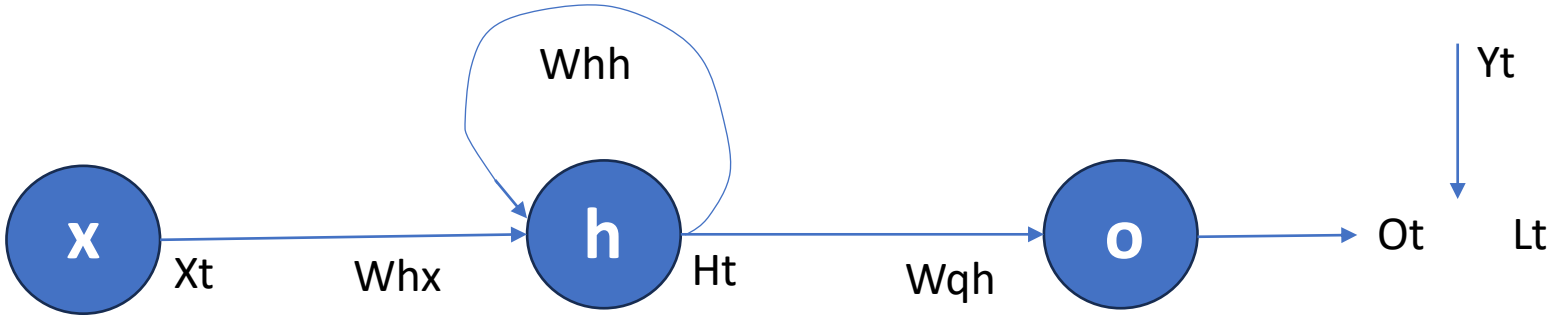


$$\mathbf{h}_t = \mathbf{W}_{\text{hx}}\mathbf{x}_t + \mathbf{W}_{\text{hh}}\mathbf{h}_{t-1},$$
$$\mathbf{o}_t = \mathbf{W}_{\text{qh}}\mathbf{h}_t,$$

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

$$\partial L/\partial \mathbf{W}_{\text{qh}}, \quad \partial L/\partial \mathbf{W}_{\text{hx}}, \quad \partial L/\partial \mathbf{W}_{\text{hh}},$$

Backpropagation through time

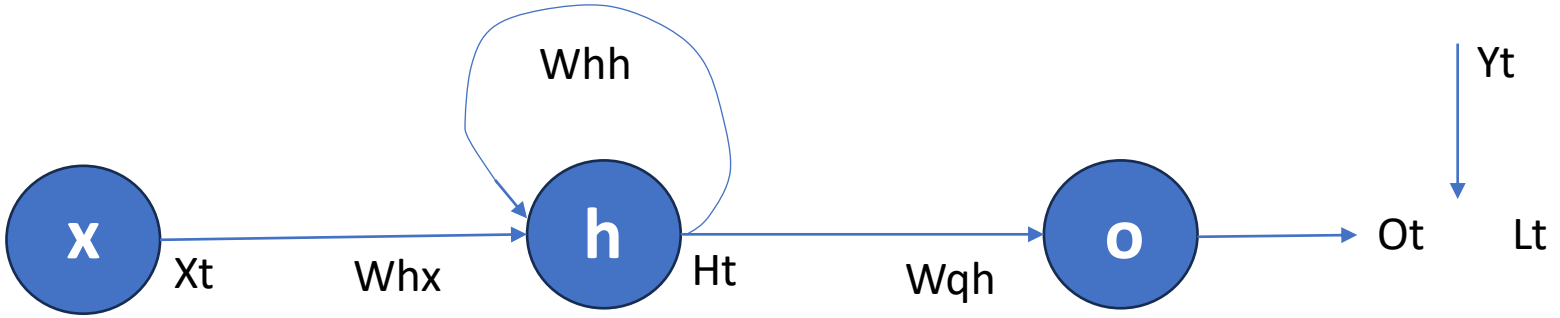


$$\partial L/\partial \mathbf{W}_{\mathrm{qh}}.$$

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{\partial l(\mathbf{o}_t, y_t)}{T \cdot \partial \mathbf{o}_t} \in \mathbb{R}^q.$$

$$\frac{\partial L}{\partial \mathbf{W}_{\mathrm{qh}}} = \sum_{t=1}^{T} \mathrm{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{\mathrm{qh}}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t} \mathbf{h}_t^\top,$$

Backpropagation through time



Next, as shown in Fig. 9.7.2, at the final time step T, the objective function L depends on the hidden state $\mathbf{h}_T$ only via $\mathbf{o}_T$. Therefore, we can easily find the gradient $\partial L/\partial \mathbf{h}_T \in \mathbb{R}^h$ using the chain rule:

$$\frac{\partial L}{\partial \mathbf{h}_T} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_T}, \frac{\partial \mathbf{o}_T}{\partial \mathbf{h}_T}\right) = \mathbf{W}_{\text{qh}}^\top \frac{\partial L}{\partial \mathbf{o}_T}. \tag{9.7.13}$$

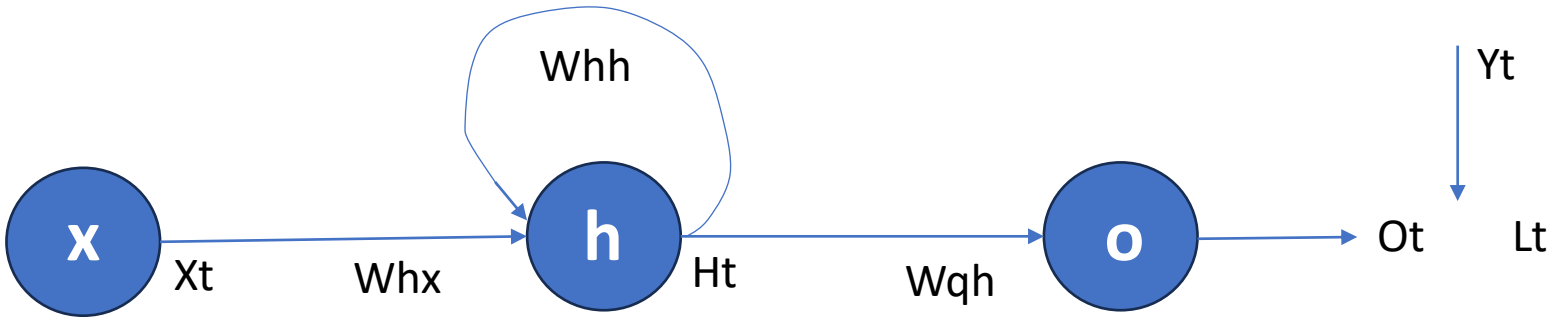
It gets trickier for any time step $t < T$, where the objective function L depends on $\mathbf{h}_t$ via $\mathbf{h}_{t+1}$ and $\mathbf{o}_t$. According to the chain rule, the gradient of the hidden state $\partial L/\partial \mathbf{h}_t \in \mathbb{R}^h$ at any time step $t < T$ can be recurrently computed as:

$$\frac{\partial L}{\partial \mathbf{h}_t} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_{t+1}}, \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right) + \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right) = \mathbf{W}_{\text{hh}}^\top \frac{\partial L}{\partial \mathbf{h}_{t+1}} + \mathbf{W}_{\text{qh}}^\top \frac{\partial L}{\partial \mathbf{o}_t}. \tag{9.7.14}$$

For analysis, expanding the recurrent computation for any time step $1 \leq t \leq T$ gives

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} \left(\mathbf{W}_{\text{hh}}^\top\right)^{T-i} \mathbf{W}_{\text{qh}}^\top \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}. \tag{9.7.15}$$

Backpropagation through time



$\partial L/\partial \mathbf{W}_{\text{hx}}$, $\partial L/\partial \mathbf{W}_{\text{hh}}$,

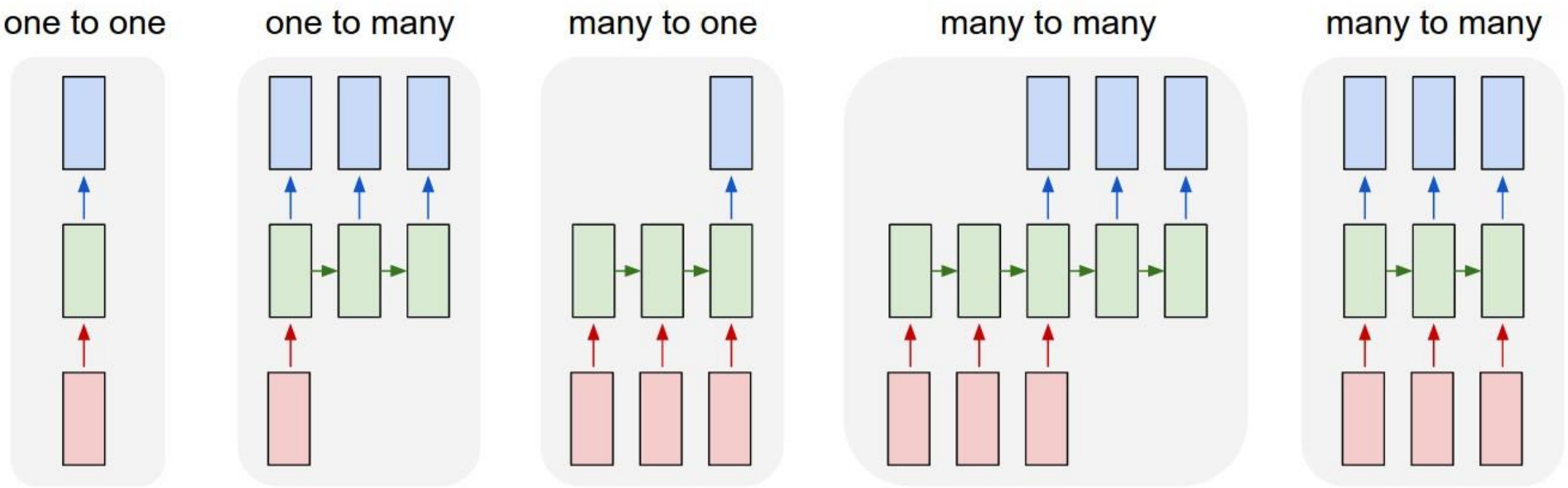
Finally, Fig. 9.7.2 shows that the objective function L depends on model parameters $\mathbf{W}_{\text{hx}}$ and $\mathbf{W}_{\text{hh}}$ in the hidden layer via hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_T$. To compute gradients with respect to such parameters $\partial L/\partial \mathbf{W}_{\text{hx}} \in \mathbb{R}^{h \times d}$ and $\partial L/\partial \mathbf{W}_{\text{hh}} \in \mathbb{R}^{h \times h}$, we apply the chain rule giving

$$\frac{\partial L}{\partial \mathbf{W}_{\text{hx}}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{\text{hx}}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{x}_t^\top, \tag{9.7.16}$$

$$\frac{\partial L}{\partial \mathbf{W}_{\text{hh}}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{\text{hh}}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{h}_{t-1}^\top,$$

where $\partial L/\partial \mathbf{h}_t$ which is recurrently computed by (9.7.13) and (9.7.14) is the key quantity that affects the numerical stability.

Backpropagation through time



Problemas del Vanishing / Exploding Gradient

Existe un problema que aparece en las RN debido a la forma matemática de calcular los gradientes de forma iterativa. Este problema se presenta incluso en las redes fully-connected cuando estas tienen un gran número de capas (redes profundas) y más aún en las redes recurrentes.

Se trata del problema del gradiente que se desvanece o explota, y tiene su origen en los cálculos de gradientes que involucran el producto de varios factores en un mismo término (regla de la cadena). Cuando estos factores tienen valores mayores o menores a 1, y la cadena se hace muy larga, esto tiende a hacer que los gradientes calculados tiendan hacia valores muy pequeños o muy grandes.

Problemas del Vanishing / Exploding Gradient

En redes FC de pocas capas esto no representa un problema, pero cuando el número de capas va en aumento, este fenómeno ocurre al calcular los gradientes de las capas más cercanas a la entrada (que involucran el producto de mayor número de términos).

Puesto que el valor con el que se actualizan los pesos en cada iteración depende del gradiente, esto genera que los mismos se actualicen de manera muy errática sin converger a la solución (exploding gradient) o bien se actualicen muy lentamente o incluso se llegue a frenar el aprendizaje (vanishing gradient).

Esto lleva a la red a una situación de inestabilidad.

Problemas de las redes recurrentes

For analysis, expanding the recurrent computation for any time step $1 \leq t \leq T$ gives

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} \left(\mathbf{W}_{\text{hh}}^\top\right)^{T-i} \mathbf{W}_{\text{qh}}^\top \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}. \tag{9.7.15}$$

Si vemos en esta parte de la ecuación, encontramos que aparece una potencia de orden T, que depende de la longitud de la secuencia de entrada.

Por lo tanto las redes recurrentes tienden a ser mas inestables cuando tratamos con secuencias más largas.

Problemas de las redes recurrentes

En el siguiente ejemplo podemos ver claramente como, al ser constante el valor del peso, la salida de la red se vuelve más caótica a medida que aumenta la longitud de la secuencia.

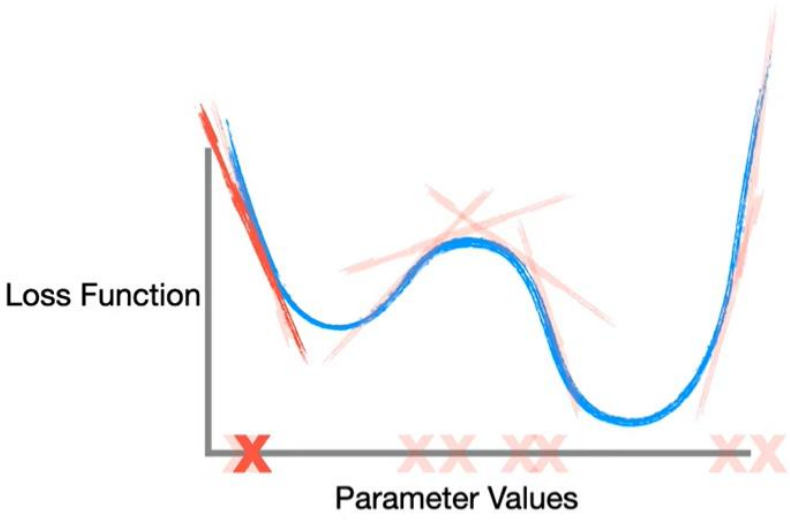
Problemas de las redes recurrentes

$w > 1$

$$\text{Input}_1 \times 2 \times 2 \times 2 \times 2$$

$$= \text{Input}_1 \times 2^4 = \text{Input}_1 \times 16$$

$$= \text{Input}_1 \times w_2^{\text{Num. Unroll}}$$

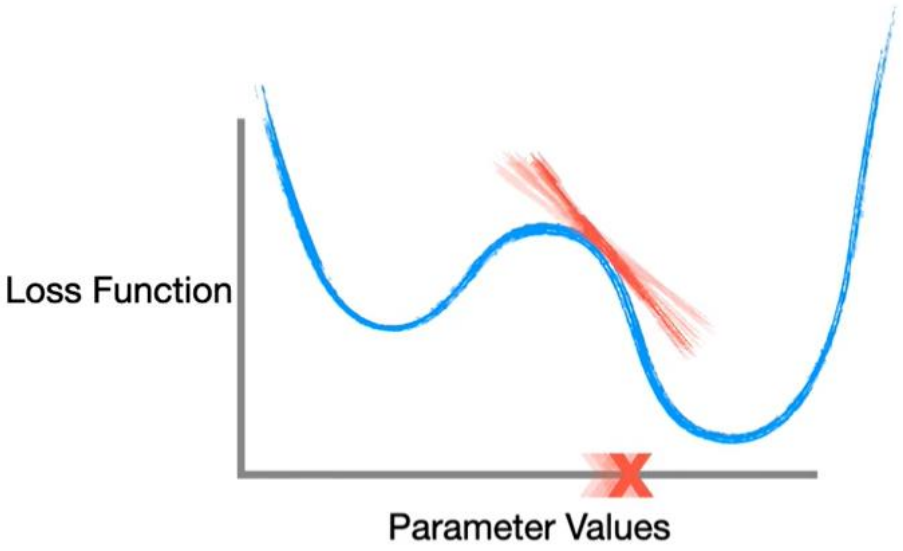


Pasos muy grandes.
Se pierde el mínimo global

Problemas de las redes recurrentes

w < 1

$\text{Input}_1 \times w_2^{\text{Num. Unroll}}$



Pasos muy pequeños.
Se agotan las iteraciones antes de alcanzar el mínimo global.

Problemas de las redes recurrentes

Estos problemas quedaron al descubierto poco después de que las primeras RNN fueran entrenadas con backpropagation, a principios de la década del 90.

Para solventar los mismos, se han propuesto modelos alternativos, con arquitecturas innovadoras y diseños complejos que resultaron exitosos en la práctica.

Estos modelos han reactivado el interés y el desarrollo sobre las RNN, y han permitido que las mismas puedan aplicarse de manera satisfactoria sobre problemas de machine learning.

Redes LSTM

Las redes LSTM (Long Short Term Memory) son un modelo de RNN propuesto en 1995 y publicado en 1997 por Sepp Hochreiter y Jürgen Schmidhuber, el cual introduce el concepto de **celda de memoria**, una unidad de cómputo que reemplaza los nodos tradicionales en las capas ocultas.

El estado de dicha celda es propagado recursivamente con un weight de valor 1, con lo que intuitivamente se evita el problema del gradiente que se desvanece.

Además, se introducen distintas compuertas que establecen de qué manera se debe actualizar el valor de la celda de memoria, y la forma en que ella afecta a la salida de la red.

Redes LSTM

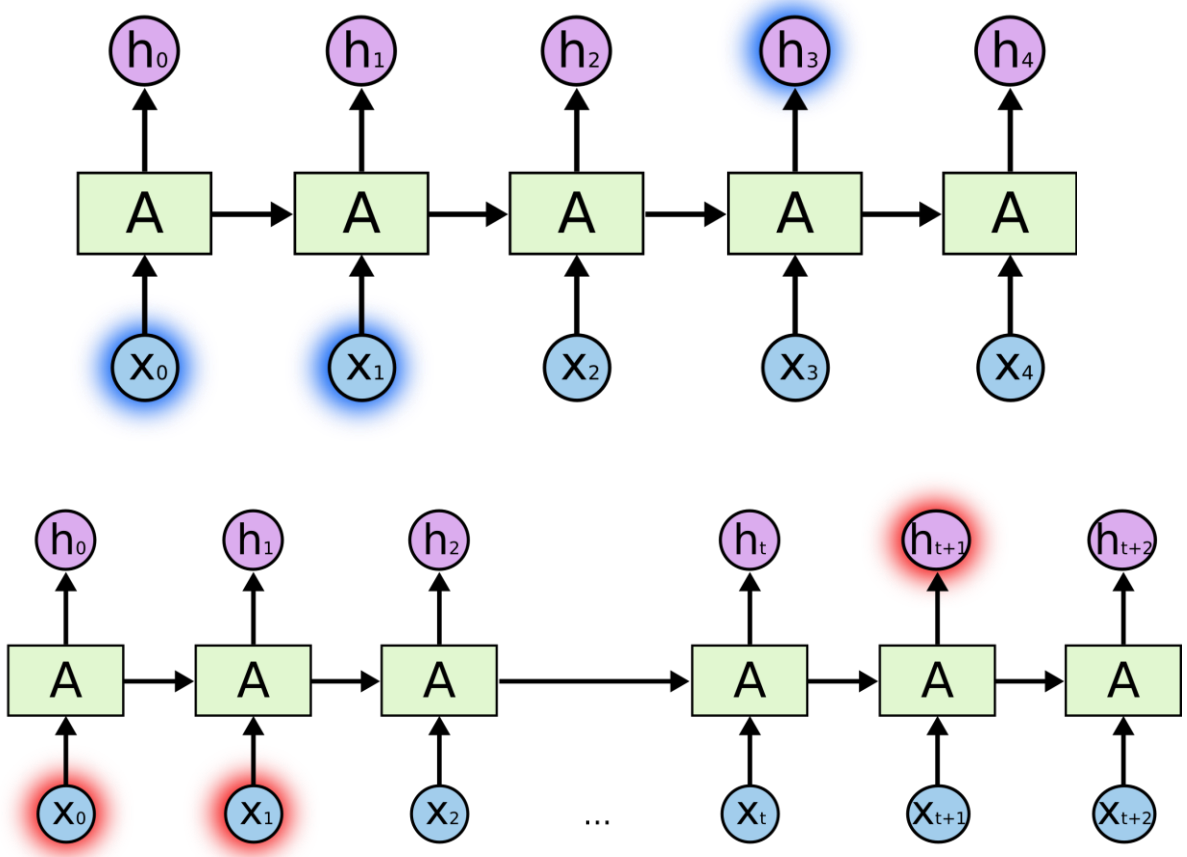
Esta arquitectura permite a la red la posibilidad de tener **memoria larga** y **memoria corta**.

En redes comunes podemos asociar la memoria larga a los pesos de la red, que son calculados en el entrenamiento y almacenan el conocimiento más general acerca de los datos.

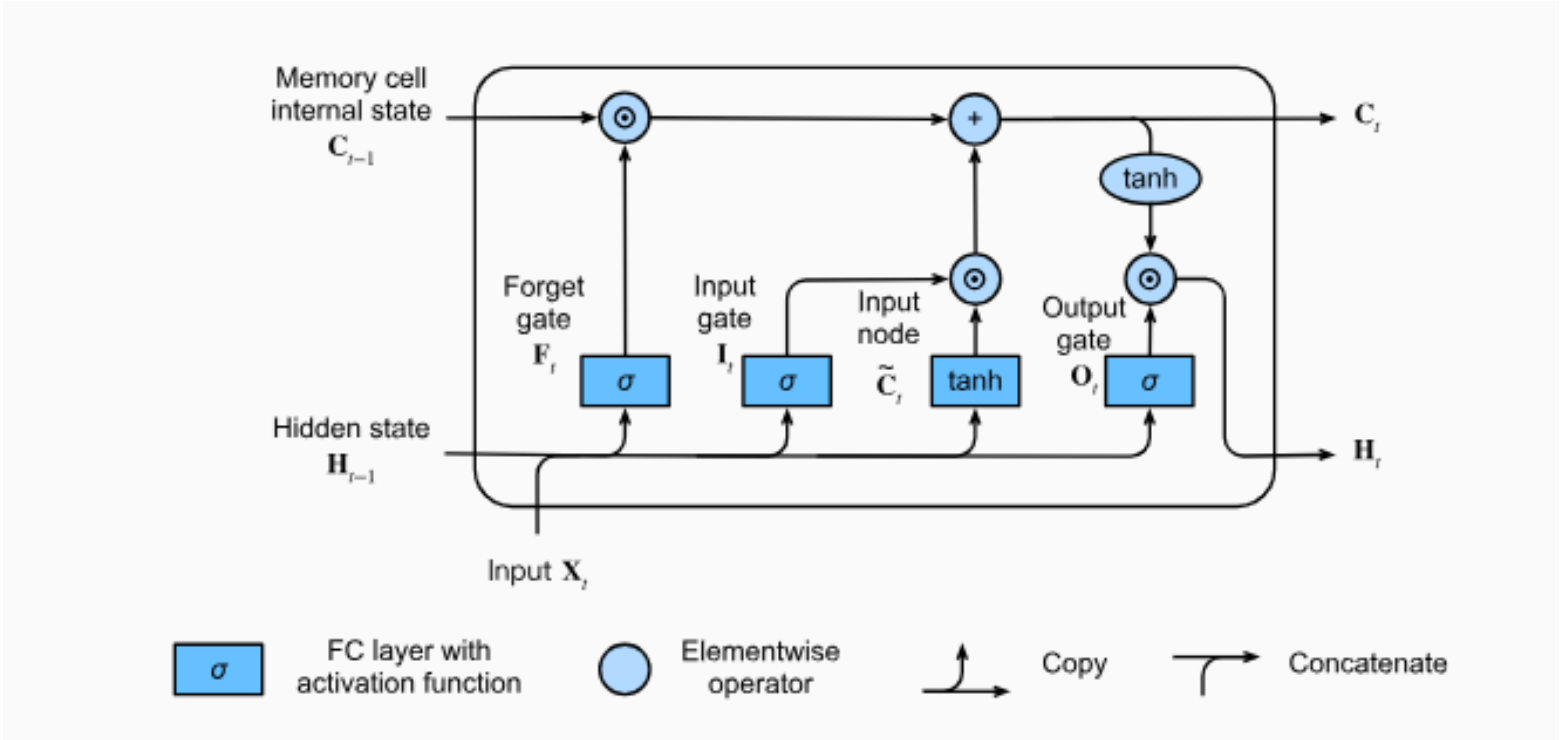
También tienen memoria corta, en la forma de las activaciones que se van dando en cada nodo, y que van propagando sucesivamente.

Las celdas de memoria les dan a las redes LSTM la posibilidad de incorporar un tipo intermedio de almacenamiento, con lo cual la información puede retenerse hasta que sea necesario recuperarla.

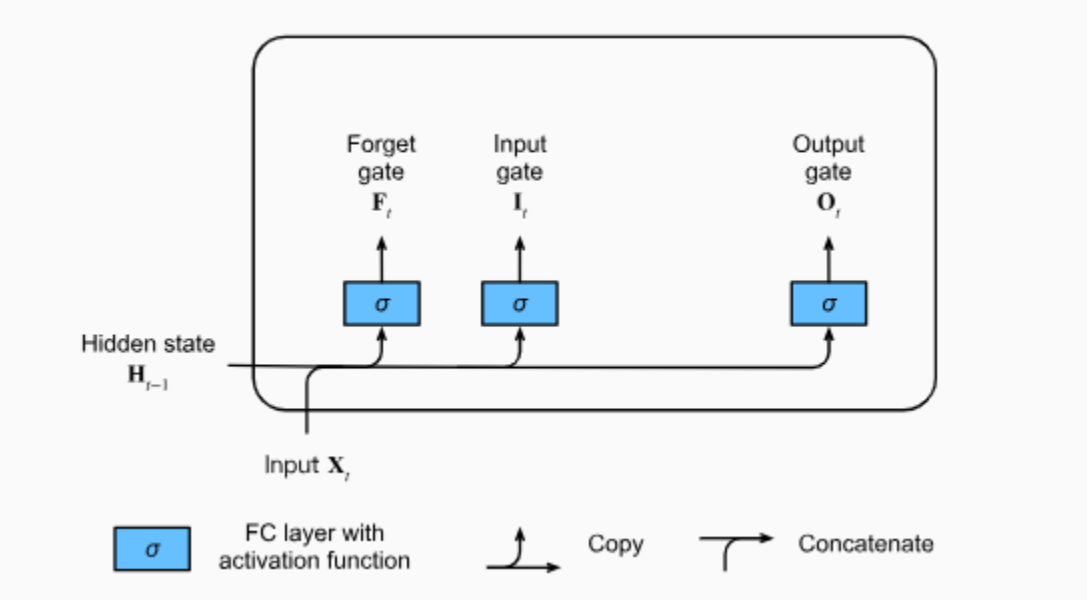
Memoria corta y memoria larga



Redes LSTM

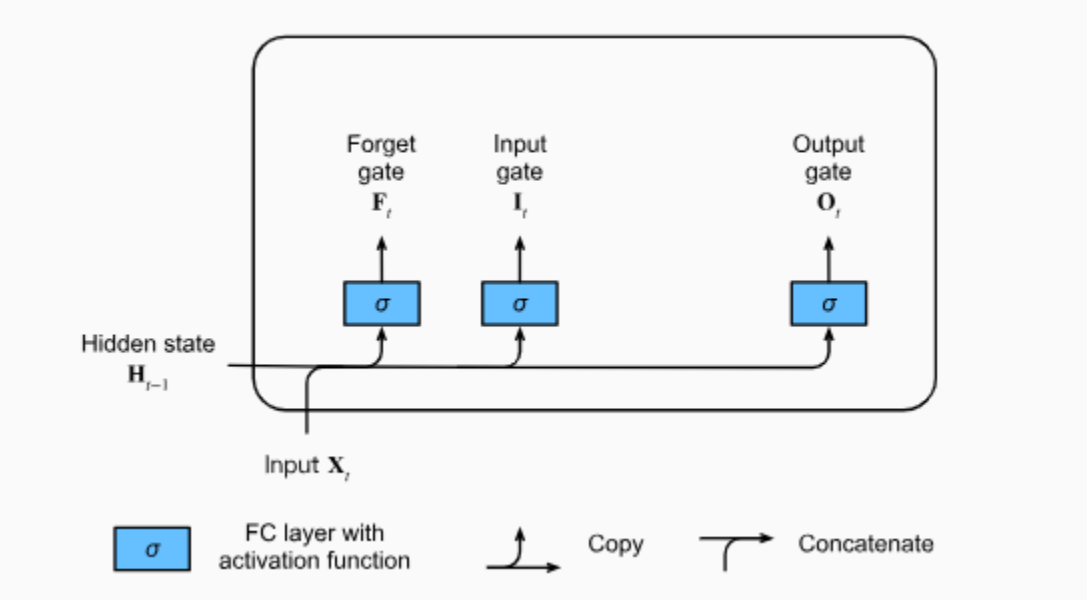

Memory cell
internal state
$\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget
gate
$\mathbf{F}_t$
Input
gate
$\mathbf{I}_t$
Input
node
$\tilde{\mathbf{C}}_t$
Output
gate
$\mathbf{O}_t$
σ
σ
tanh
σ
Hidden state
$\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$
σ
FC layer with
activation function
Elementwise
operator
Copy
Concatenate

Redes LSTM



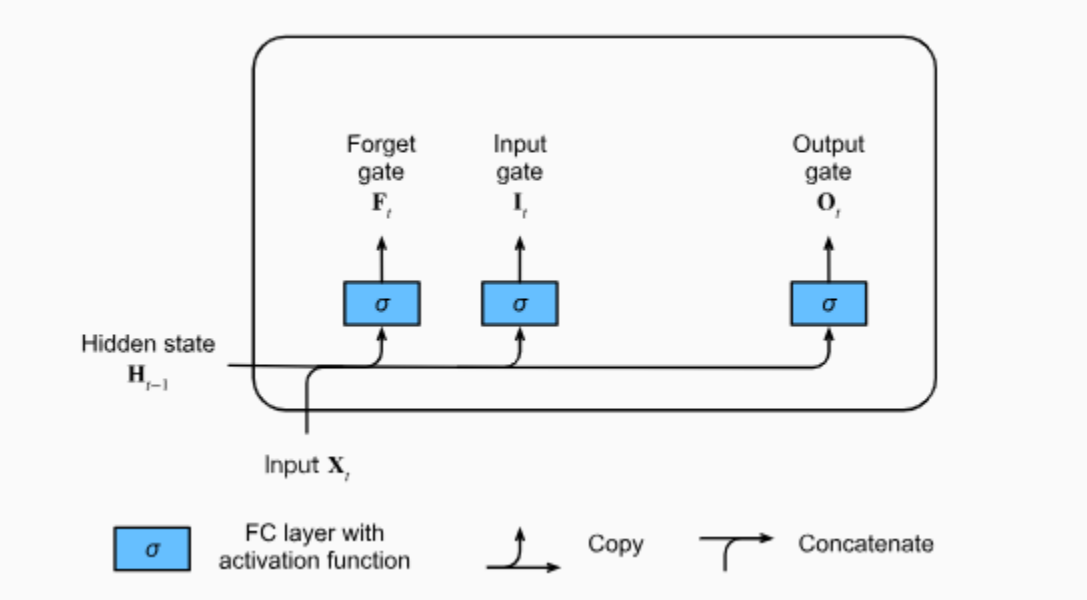
- Input gate: cuánto del valor del nodo de entrada se debe añadir al estado actual de la celda de memoria interna.
- Forget gate: si debemos mantener o eliminar el valor de la memoria.
- Output gate: si la celda de memoria debe influenciar la salida en el time step actual.

Redes LSTM



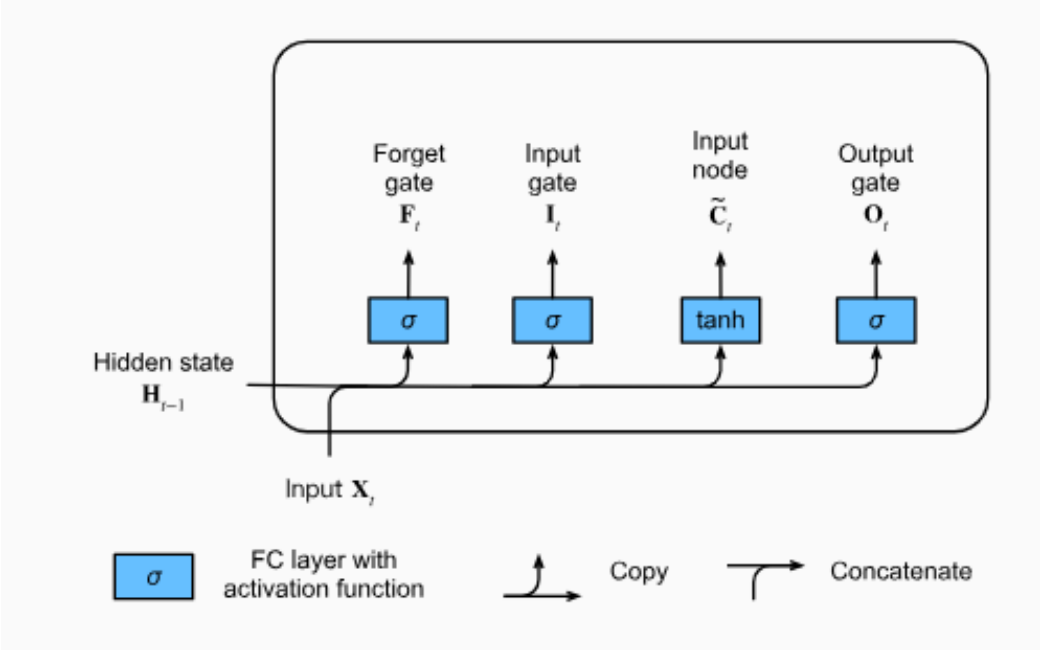
Todas las compuertas tienen una función de activación sigmoide, por lo que sus valores de salida están en el rango de 0 a 1.

Redes LSTM



$$
\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{\mathrm{xi}} + \mathbf{H}_{t-1} \mathbf{W}_{\mathrm{hi}} + \mathbf{b}_{\mathrm{i}}),\\
\mathbf{F}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{\mathrm{xf}} + \mathbf{H}_{t-1} \mathbf{W}_{\mathrm{hf}} + \mathbf{b}_{\mathrm{f}}),\\
\mathbf{O}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{\mathrm{xo}} + \mathbf{H}_{t-1} \mathbf{W}_{\mathrm{ho}} + \mathbf{b}_{\mathrm{o}}),
\end{aligned}
$$

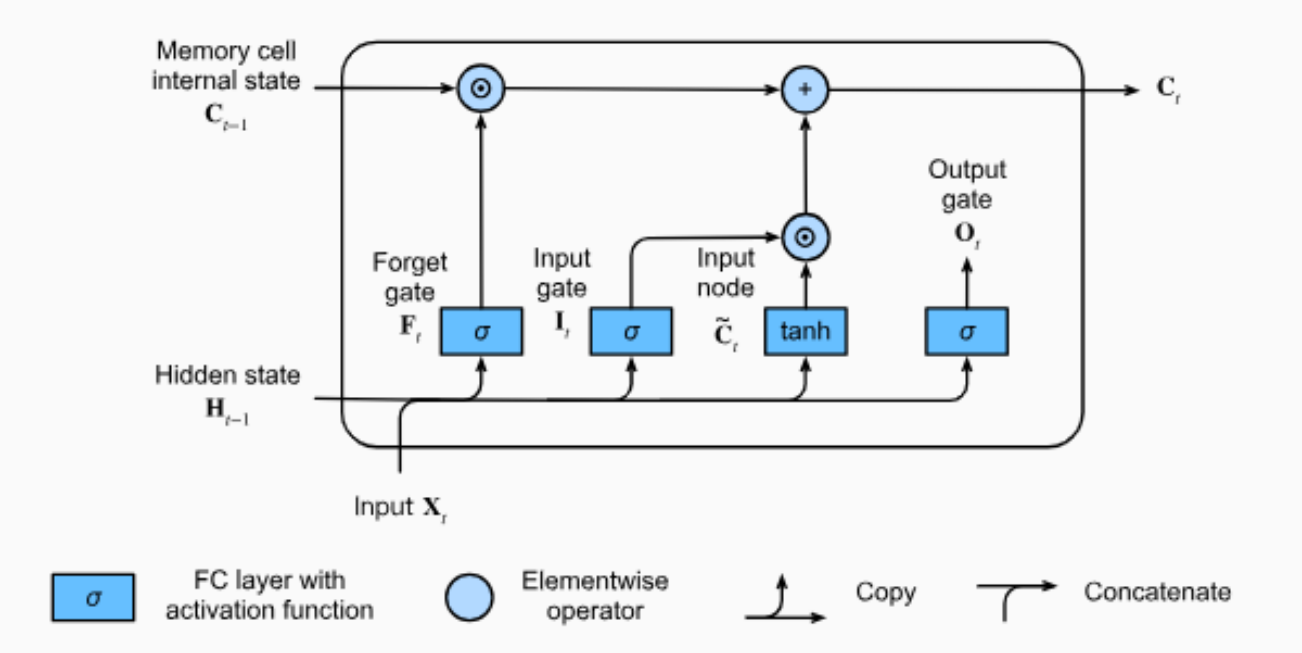
Redes LSTM



- Nodo input: a diferencia de los otros, tiene una función de activación tanh, por lo que su salida tiene valores entre -1 y 1.

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{\mathrm{xc}} + \mathbf{H}_{t-1} \mathbf{W}_{\mathrm{hc}} + \mathbf{b}_{\mathrm{c}}),$$

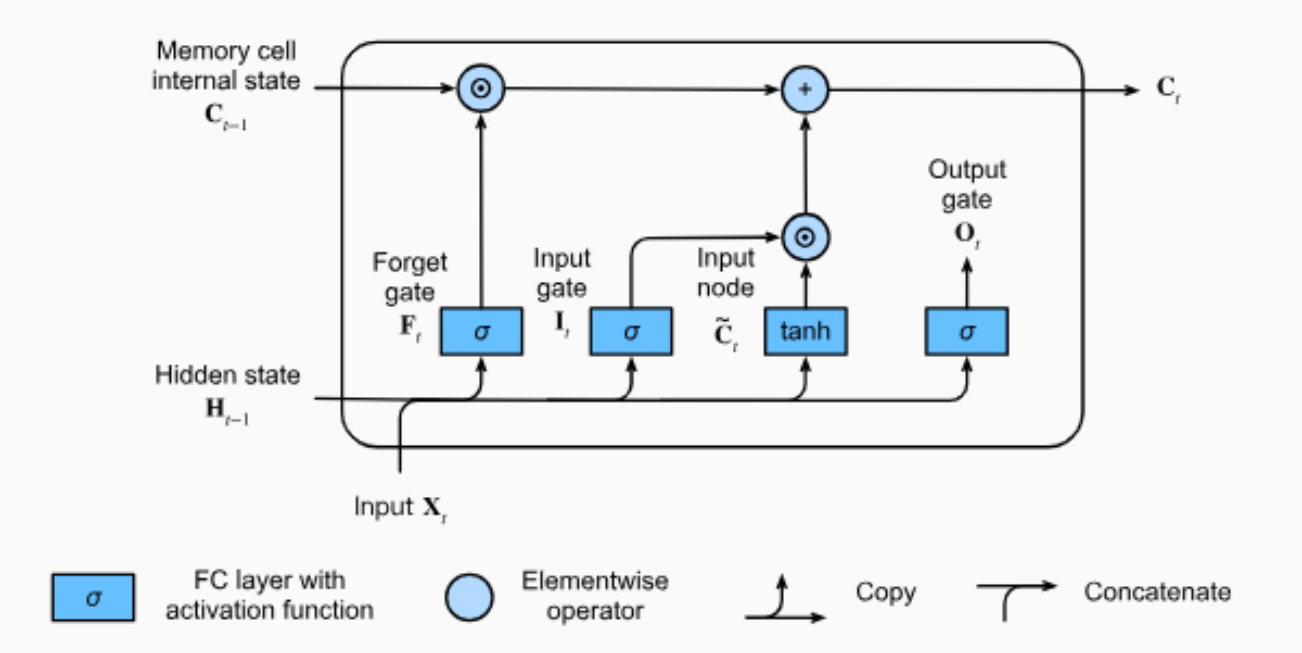
Redes LSTM



- Celda interna de memoria: se calcula mediante productos (elemento a elemento) y combinaciones de las salidas de las compuertas definidas.

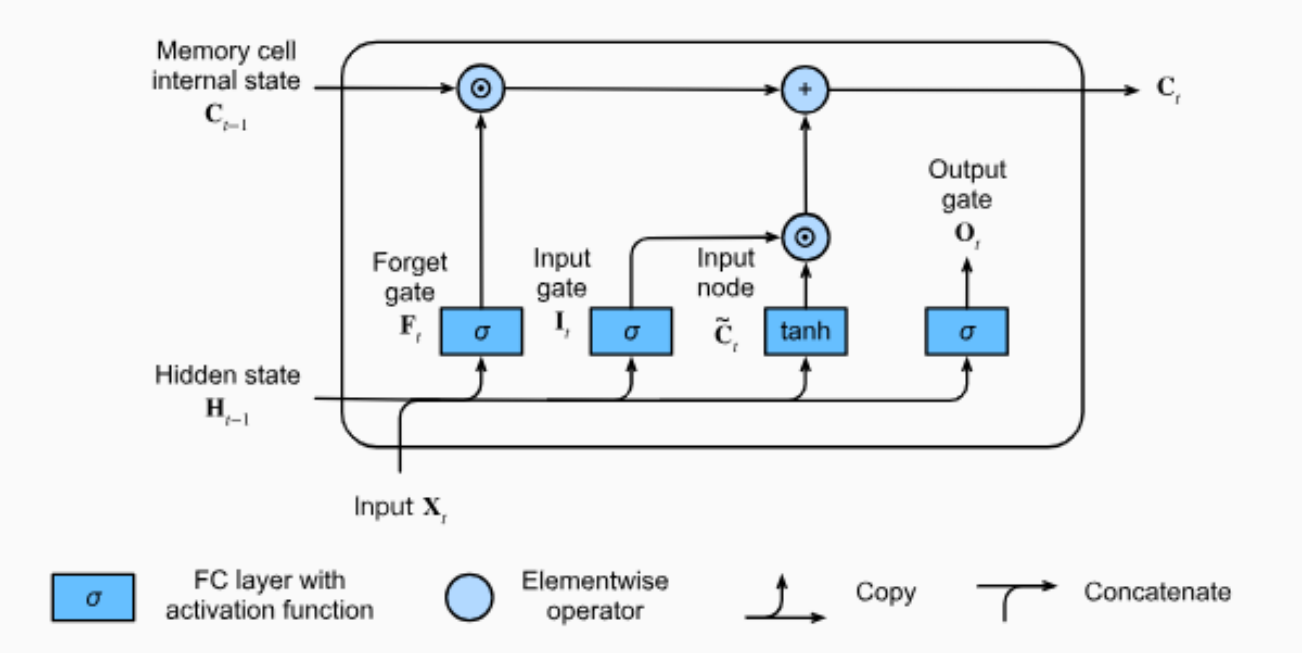
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t.$$

Redes LSTM



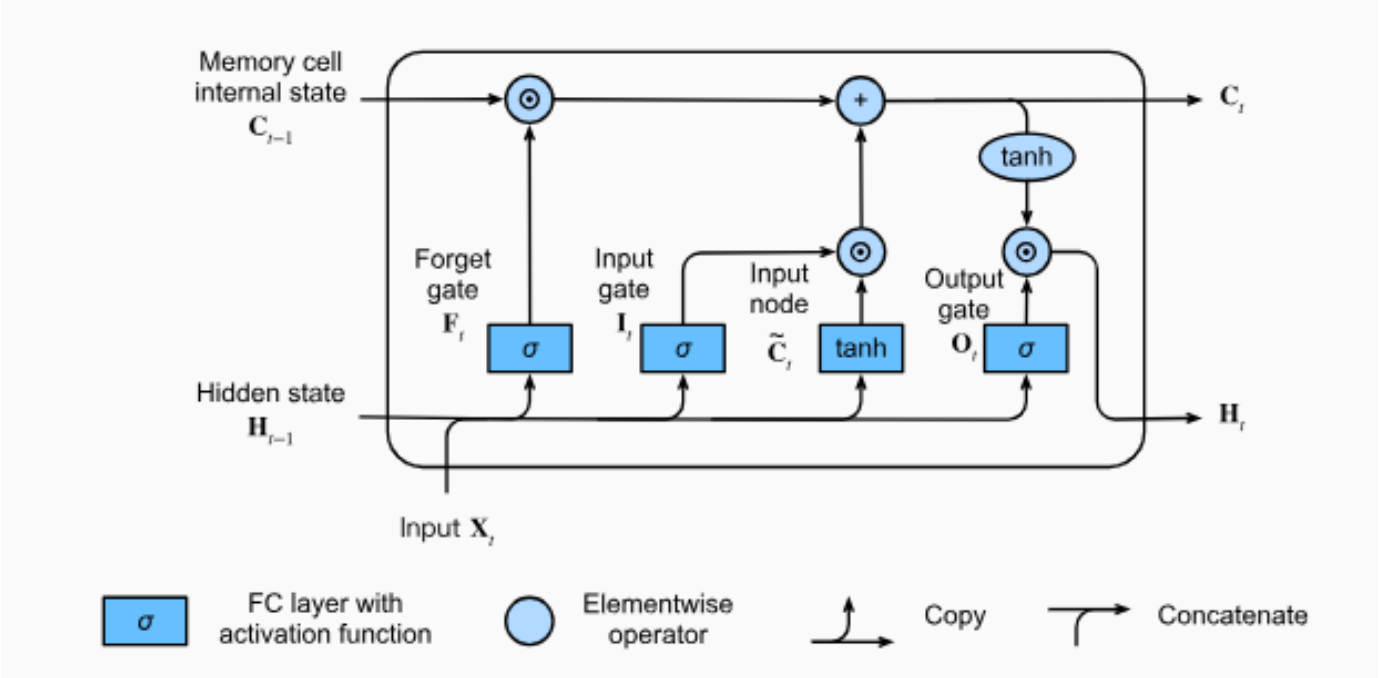
Estas compuertas le dan al modelo la flexibilidad para ser capaz de aprender cuándo mantener dicho valor, y cuándo cambiarlo, en respuesta a sucesivas inputs.

Redes LSTM



Esto resulta en modelos más fáciles de entrenar, con la posibilidad de manejar datasets con secuencias largas.

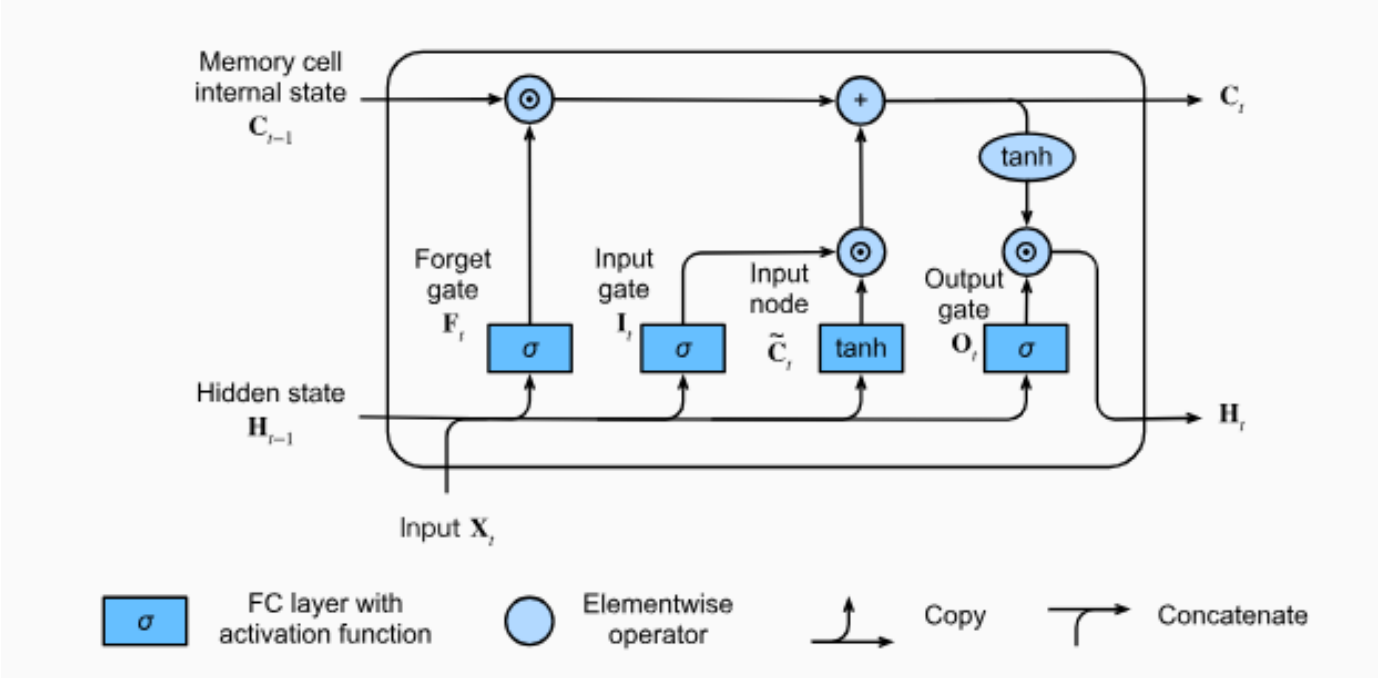
Redes LSTM



-Estado oculto (Ht): es la salida de la celda de memoria, que es vista por las otras capas. Su valor está en el intervalo de -1 a 1.

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t).$$

Redes LSTM



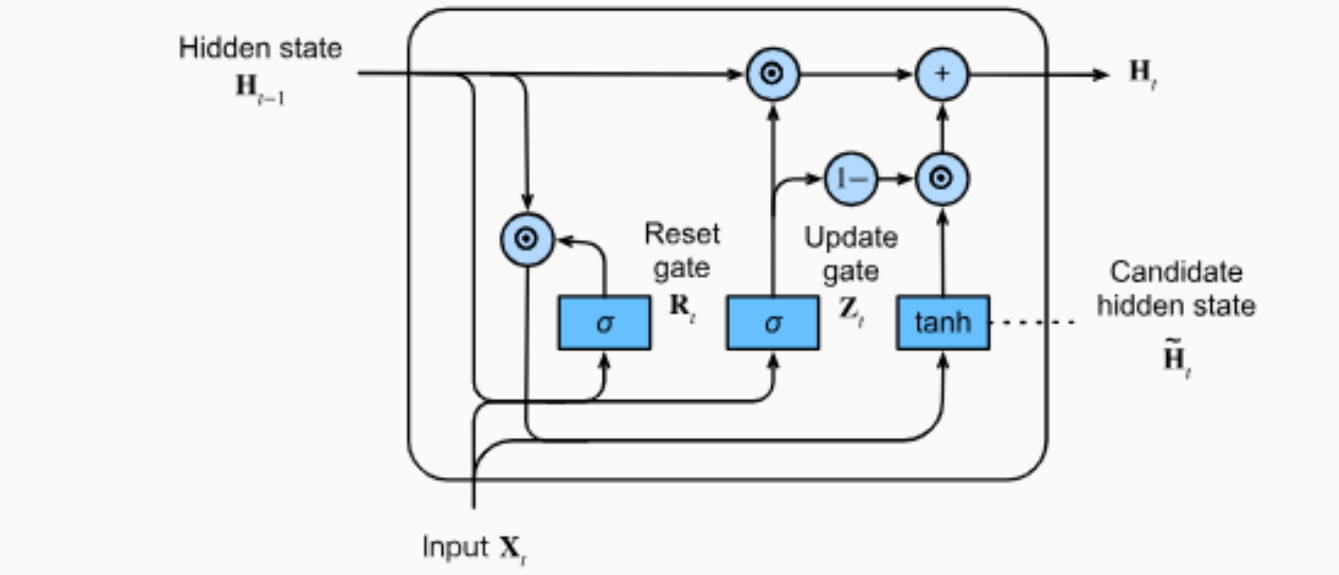
Si Ot = 1, el valor de la celda interna es visto por el resto de las capas. El momento en que Ot pasa de 0 a 1 determina el time step cuando la información de la celda de memoria impacta en el resto de la red.

Redes GRU

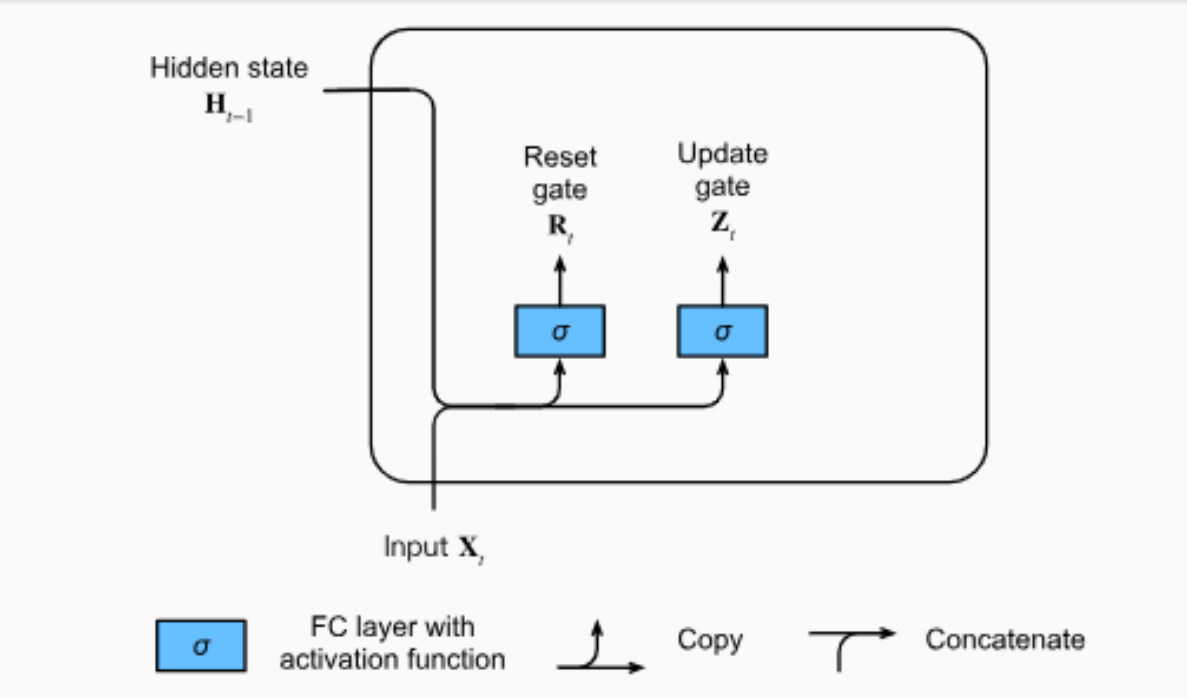
Gracias a las redes LSTM, las RNN cobraron popularidad durante la década de los 2000 y 2010. Entonces se empezó a investigar formas de simplificar la arquitectura pero manteniendo los avances logrados.

Las redes GRU (Gated Recurrent Unit), introducidas en 2014, ofrecen una versión simplificada de las redes LSTM que pueden lograr resultados comparables, pero con la ventaja de ser mas rápidas de computar.

Redes GRU

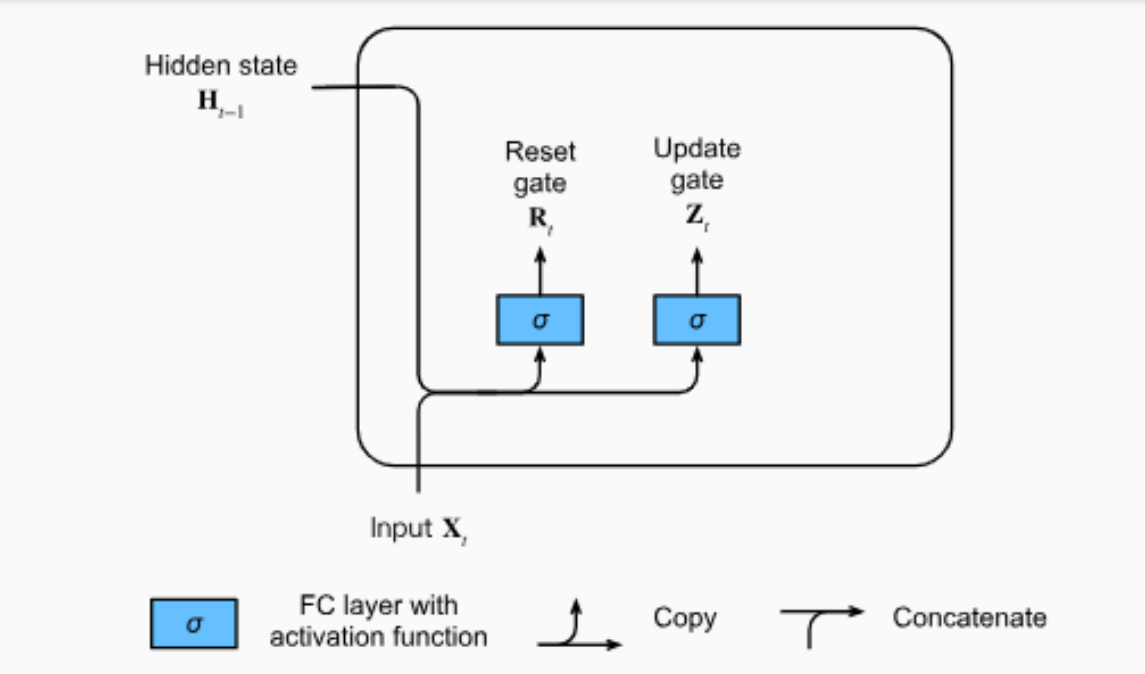

Hidden state
$\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Reset gate
$\mathbf{R}_t$
Update gate
$\mathbf{Z}_t$
Candidate hidden state
$\tilde{\mathbf{H}}_t$
σ
tanh
1−
+
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Redes GRU



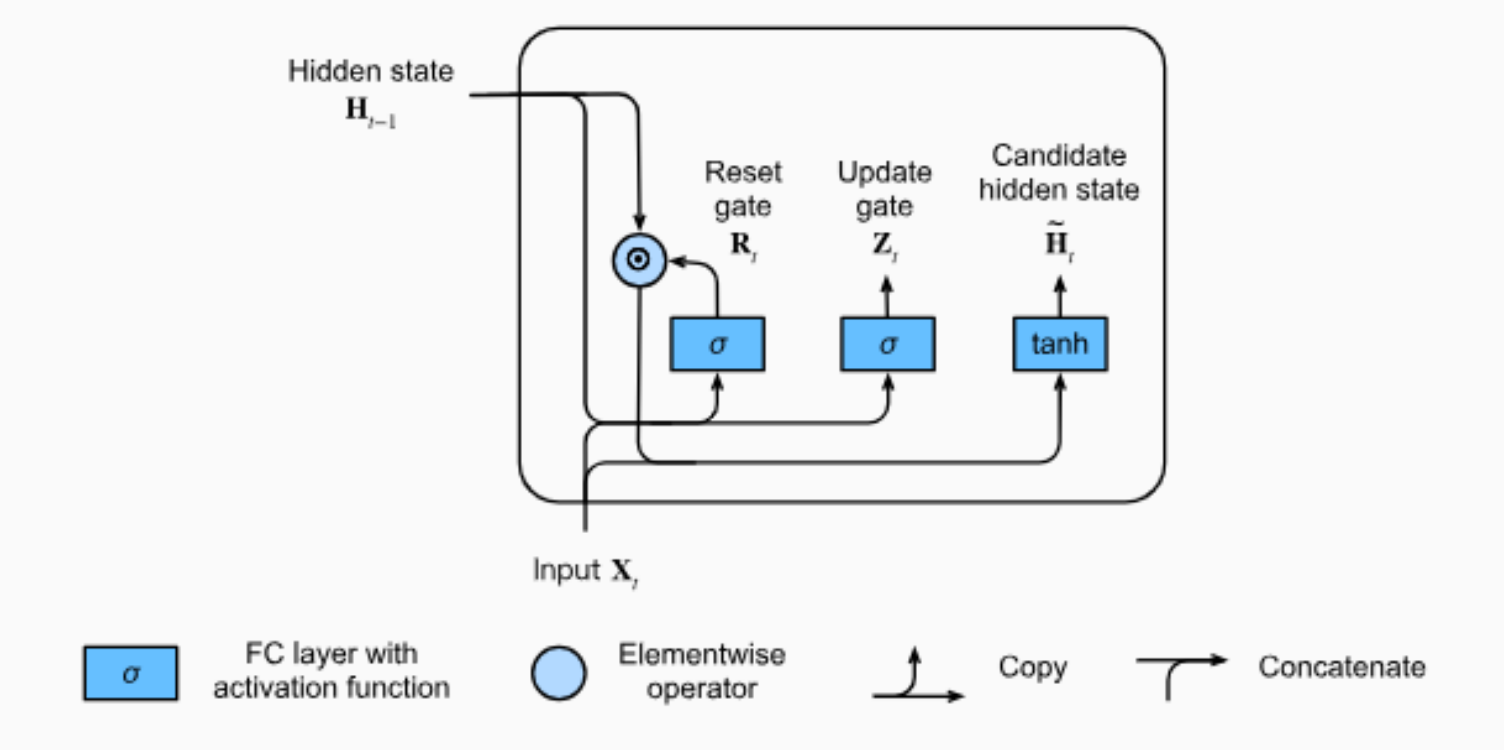
- Reset gate: controla cuánto del estado anterior se quiere recordar.
- Update gate: controla cuánto del estado nuevo es una copia del estado anterior.

Redes GRU



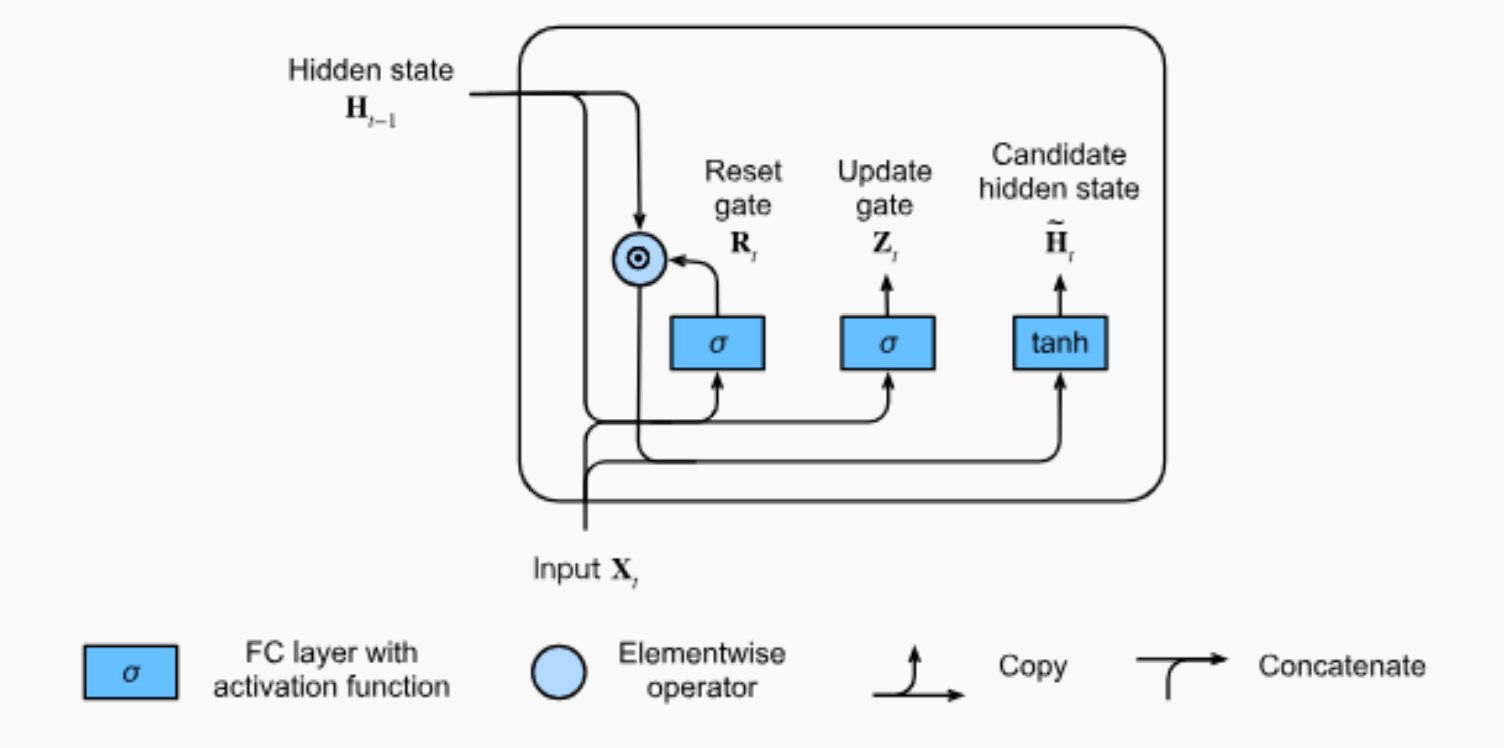
$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{\mathrm{xr}} + \mathbf{H}_{t-1} \mathbf{W}_{\mathrm{hr}} + \mathbf{b}_{\mathrm{r}}),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{\mathrm{xz}} + \mathbf{H}_{t-1} \mathbf{W}_{\mathrm{hz}} + \mathbf{b}_{\mathrm{z}}),$$

Redes GRU



- Estado oculto candidato: se calcula a partir de los valores de la entrada actual y el estado oculto anterior teniendo en cuenta la salida del reset gate. Es un estado candidato ya que no tiene en cuenta el update gate.

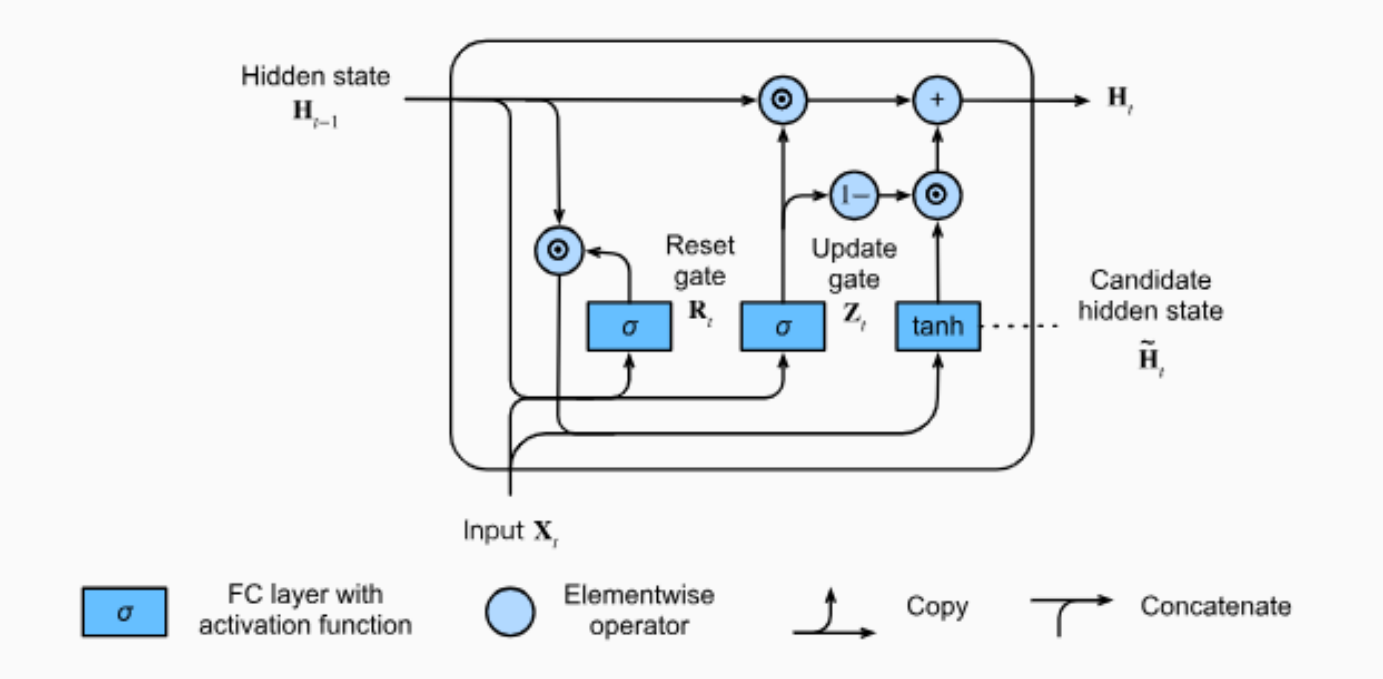
Redes GRU



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

Si Rt es cercano a 1, la salida de la compuerta se comporta como un nodo de una RNN común. Si Rt es cercano a 0, la red se comporta como si no tuviera recursión: se resetea el estado anterior.

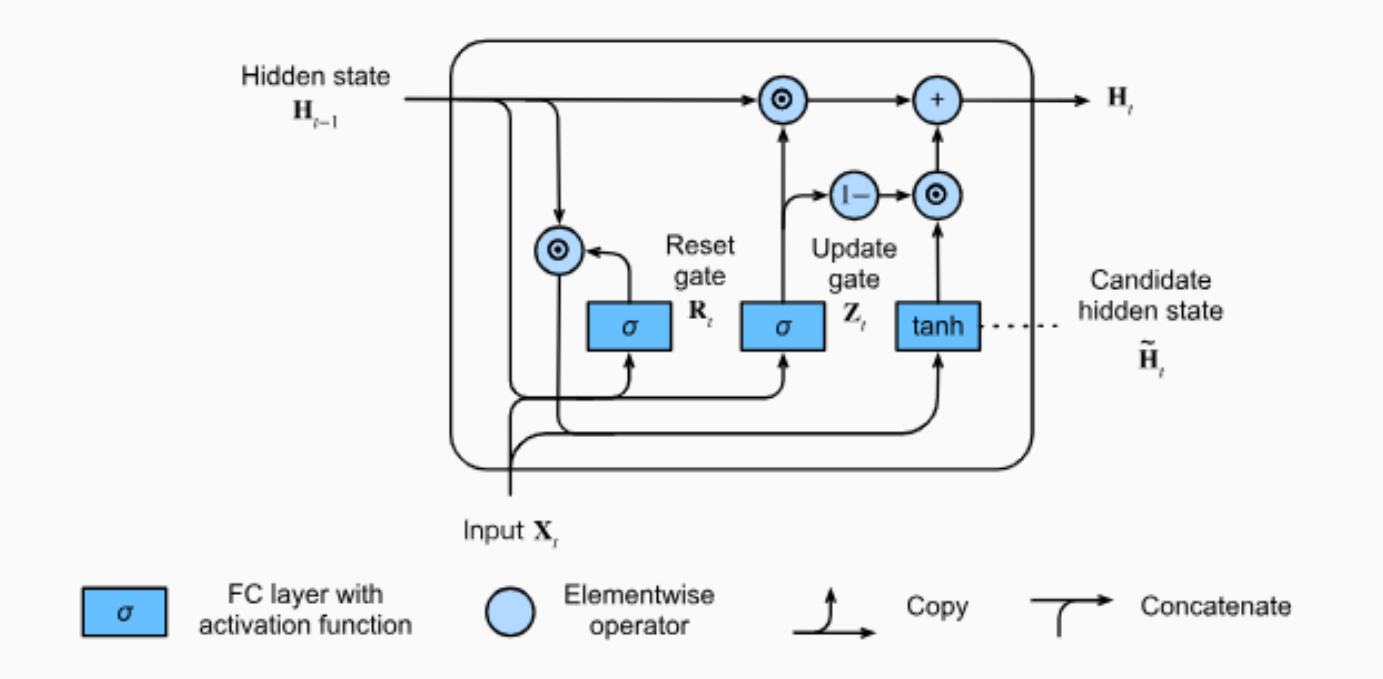
Redes GRU



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

Estado oculto: su valor la determina la update gate, en función de cuánto se parece el estado anterior Ht-1 al estado oculto candidato.

Redes GRU

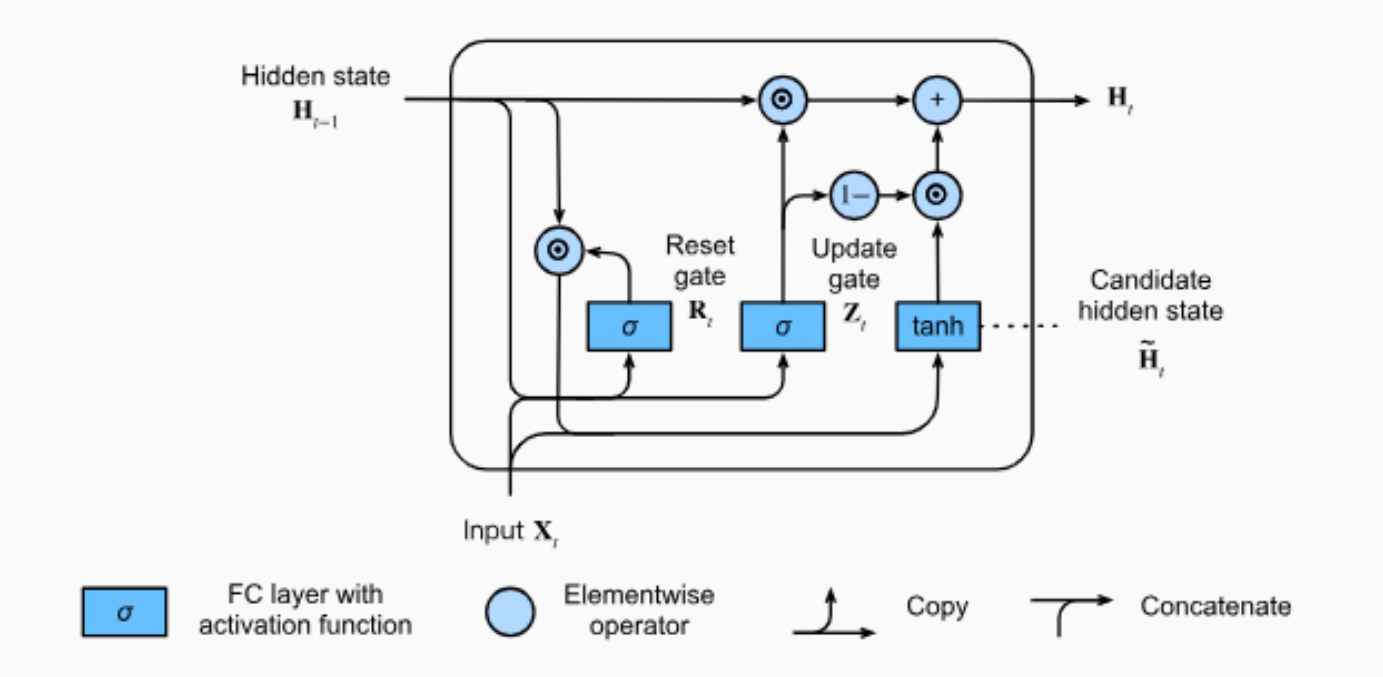


$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

Si Zt = 0: el nuevo estado es igual al estado candidato.

Si Zt = 1: el nuevo estado es igual al anterior, no se actualiza.

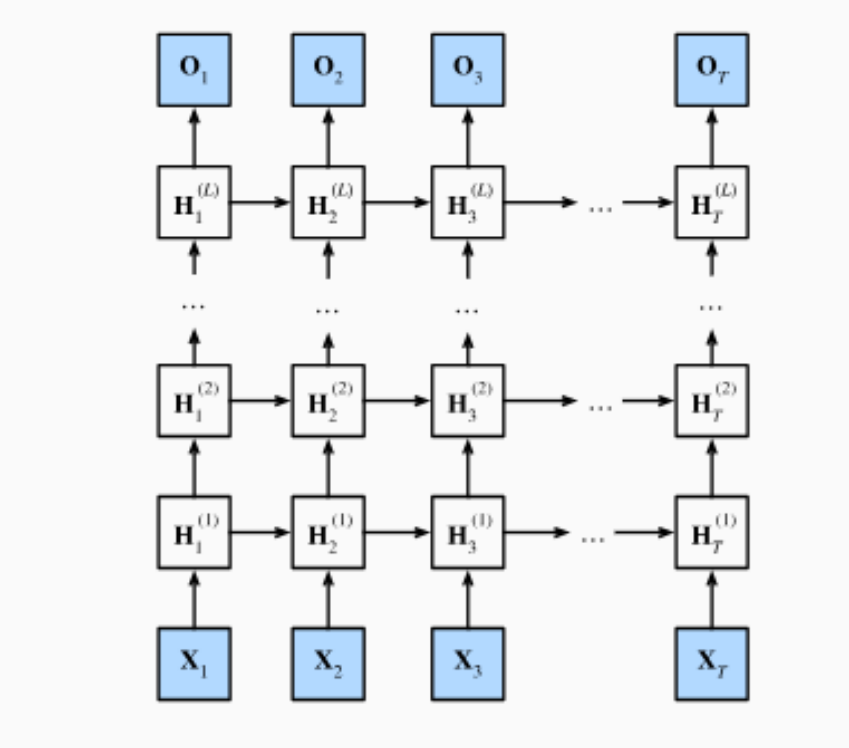
Redes GRU



Entonces el **reset gate** permite capturar dependencias de **tiempo corto** en las secuencias y el **update gate** permite capturar dependencias de **tiempo largo** en las secuencias.

Redes recurrentes profundas

Es posible construir redes que sean profundas, no solamente en la dirección del tiempo (debido a la cantidad de iteraciones que hace la red), sino también en la dirección de entrada y salida de datos. Para ello se agregan mas de una capa oculta (como en MLP y CNN), ya sea de una RNN común, o una capa de tipo LSTM o GRU.



Bibliografía

-**Dive into Deep Learning**. Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J. Cambridge University Press. 2023

https://d2l.ai/index.html

-Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.