Redes Neuronales Recurrentes

Long Short Term Memory Networks (y GRU)

Long Short Term Memory (LSTM)

- Uno de los métodos más utilizados como variante de las Redes de Elman para permitir el aprendizaje de dependencias de largo plazo.
- Propuesta Oficialmente por Hochreiter y Schmidhuber en 1997.

LONG SHORT-TERM MEMORY

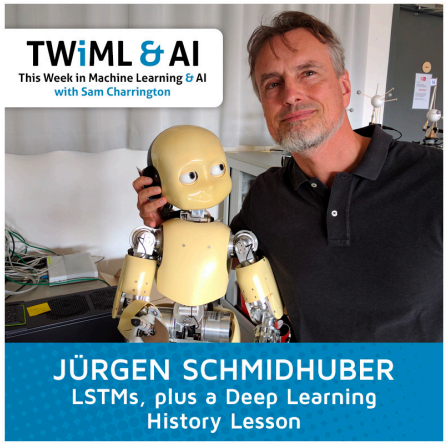
NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.



Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural Computation* 9.8 (1997): 1735-1780.

Long Short Term Memory (LSTM)

- Ideas fundamentales ya presentes en la tesis de magíster de Hochreiter de 1991.

DIPLOMARBEIT
IM FACH INFORMATIK

Untersuchungen zu dynamischen neuronalen Netzen

Josef Hochreiter Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
hochreit@kiss.informatik.tu-muenchen.de

Aufgabensteller: Professor Dr. W. Brauer
Betreuer: Dr. Jürgen Schmidhuber

15 Juni 1991

Josef Hochreiter, Untersuchungen zu dynamischen neuronalen Netzen.
Institut für Informatik. Technische Universität München 1991.

Long Short Term Memory (LSTM)

- Constant Error Carrusels (CECs)
- La idea fundamental consiste en utilizar recurrencias condicionales (o sensibles al contexto) en vez de recurrencias fijas.

Kapitel 4

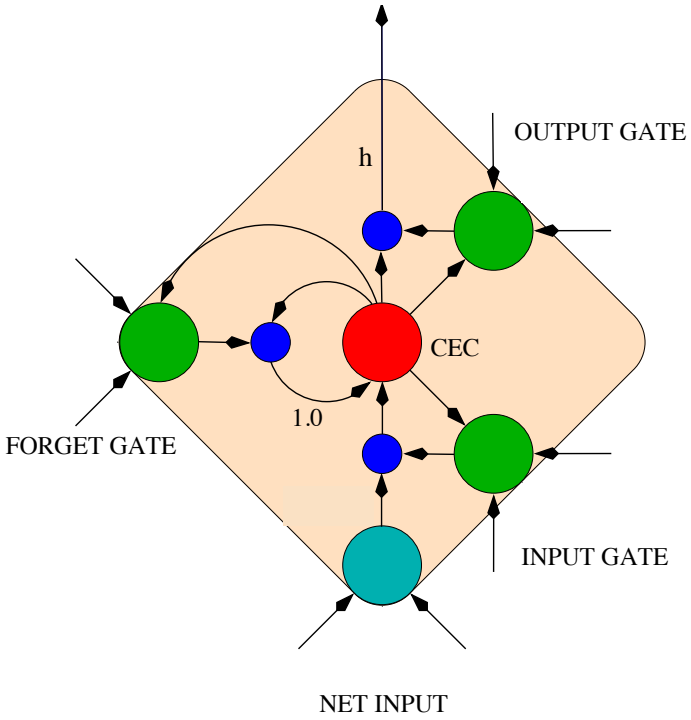
Konstanter Fehlerrückfluß

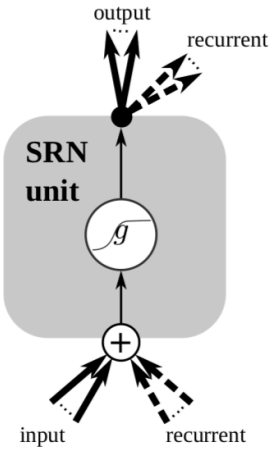
Eine andere Möglichkeit, um zu verhindern, daß der Fehler beim Zurückpropagieren durch die Zeit zu klein wird, ist den Fehlerrückfluß konstant zu halten. Entsteht der Fehler $e_i(t)$ zum Zeitpunkt t an dem Ausgabenknoten i und wird dieser zu dem Eingabeknoten k zum Zeitpunkt $t-l$ zurückgereicht, so soll der Fehler, der dort ankommt, genau so groß sein, wie der Fehler, der an dem Eingabeknoten k zum Zeitpunkt $t-l-1$ ankommt. Somit werden für das Zustandekommen eines Fehlers an den Ausgabeknoten alle vergangenen Eingaben gleich bewertet und nicht, wie im herkömmlichen Backpropagation, die letzten Eingaben viel stärker gewichtet. Dies ist bei Aufgaben, bei denen vergangene Eingaben Auswirkungen auf die gewünschte Ausgabe haben, sehr hilfreich und führt zu starker Konvergenzbeschleunigung. Im Unterschied zum Zeitüberbrücker-System muß hier die Eingabesequenz nicht reduzierbar sein, d. h. es braucht keine Struktur in der Eingabesequenz vorhanden sein. Dieses Verfahren ist sinnvoll, falls abstraktere Eingaben vorliegen, die keine offensichtliche Struktur besitzen, wie z. B. Namen von Unterprogrammen, die ausgeführt wurden, wichtige Situationen, die in der Vergangenheit vorhanden waren oder Zusammenfassung von vergangenen Zuständen der Umgebung.

Josef Hochreiter, Untersuchungen zu dynamischen neuronalen Netzen. Institut für Informatik. Technische Universität München 1991.

Celdas LSTM

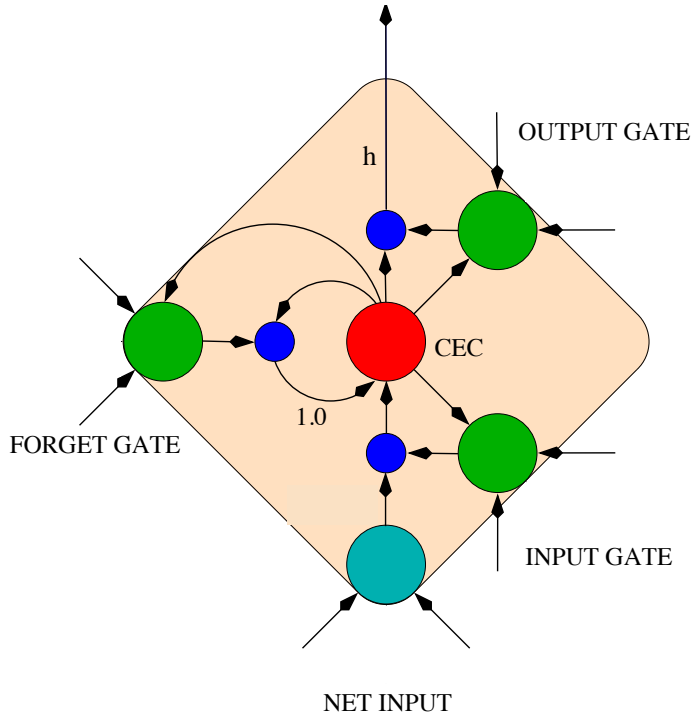
- Esta idea que se implementa mediante la introducción de unidades denominadas “celdas”, que sustituyen las neuronas de una RNN tradicional.





Celdas LSTM - CEC

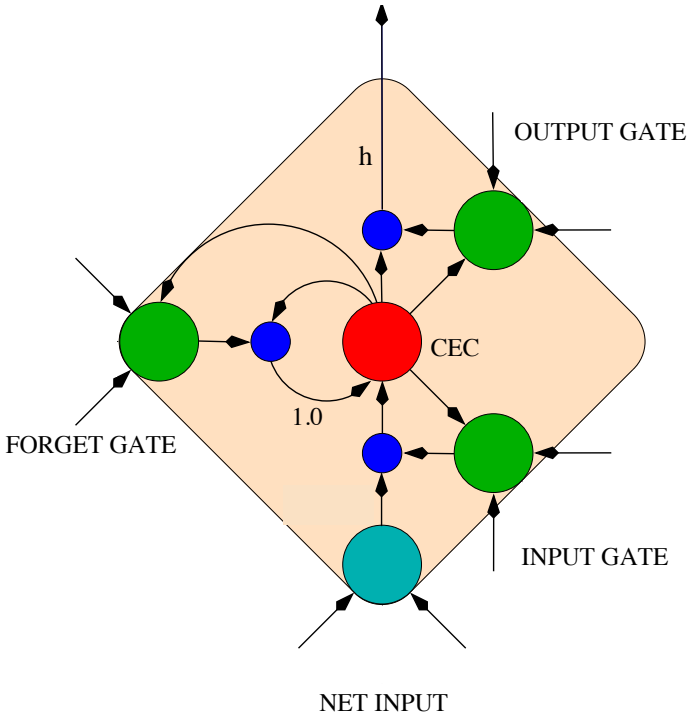
- La celda va a mantener un estado interno que denotaremos c_t y que representa la memoria de la celda. La ideal del CEC es actualizar la memoria interna mediante una recurrencia constante, sin pesos.



$$c_t = \boxed{1}\, c_{t-1} + \boxed{\text{nueva info}}$$

Celdas LSTM - Input Gate

- La unidad puede escribir nueva información en la memoria pero el paso de esa información se controla usando una compuerta $i_t \in [0,1]$

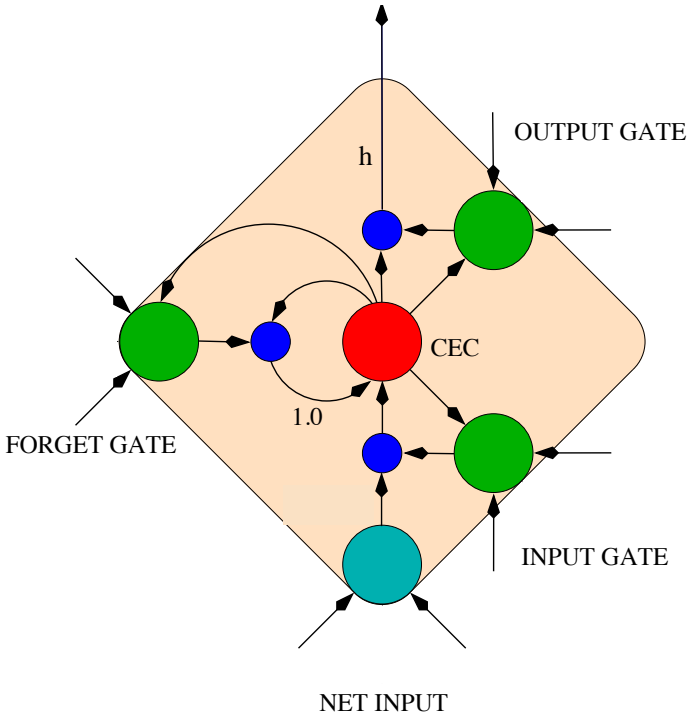


$$c_t = \boxed{1}\, c_{t-1} + \boxed{i_t} \odot \boxed{\text{nueva info}}$$

Si la compuerta está en 1, deja pasar la información.
Si la compuerta está en 0, bloquea la información.

Celdas LSTM - Activación

- La información por almacenar se calcula usando una capa densa.
- El estado de la compuerta de entrada se determina usando una capa densa.



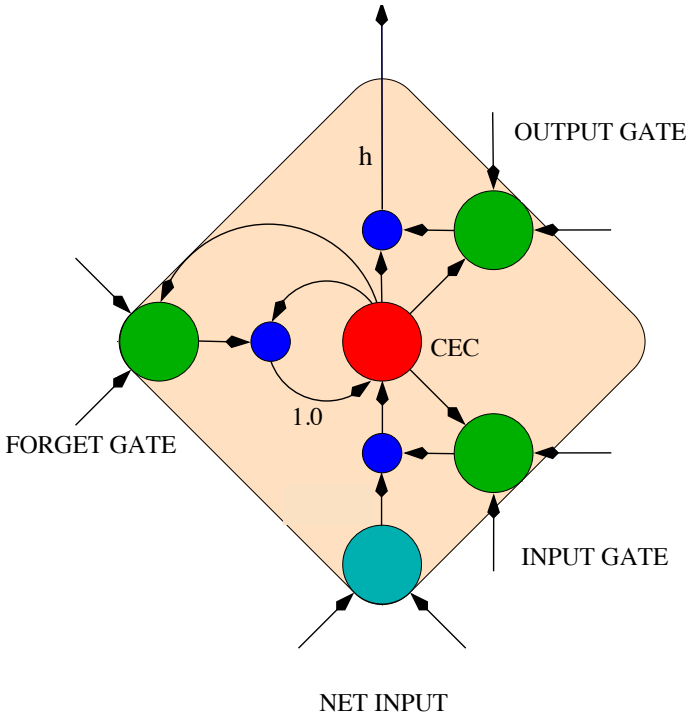
$$c_t = \boxed{1}\, c_{t-1} + \boxed{i_t} \odot \boxed{\text{nueva info}}$$

nueva info = $a_t = \sigma_a(Ux_t + Wh_{t-1})$

$i_t = \sigma_a(U_i x_t + W_i h_{t-1})$

Celdas LSTM - Output Gate

- En un determinado tiempo, la unidad puede liberar información o no, y esto se controla usando una compuerta de salida $o_t \in [0,1]$.



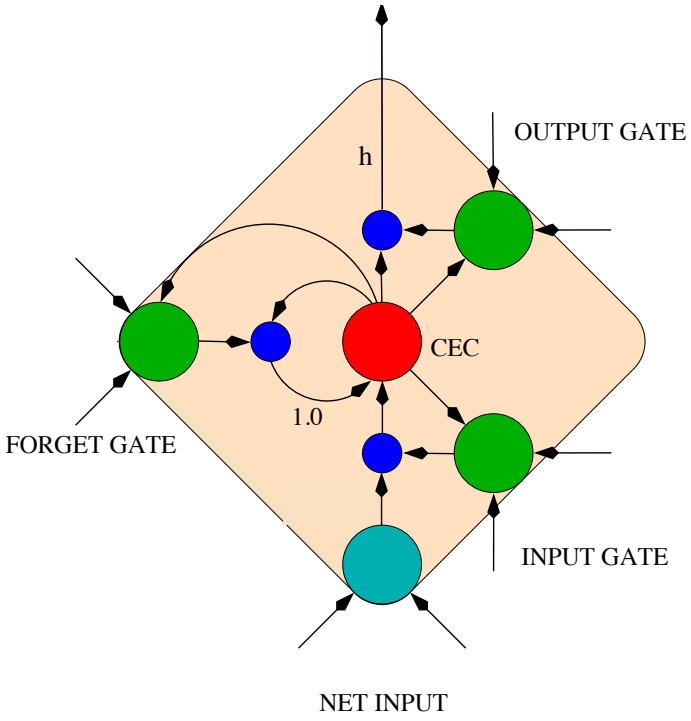
$$h_t = o_t \odot$$ info por mostrar

$$o_t = \sigma_o(U_o x_t + W_o h_{t-1})$$

Si la compuerta está en 1, la celda muestra información. **Si la compuerta está en 0, la celda no participa en los cálculos de la red de ese tiempo.**

Celdas LSTM - Salida

- Si la compuerta está abierta, el estado a mostrar se determina a partir de su memoria interna c_t.

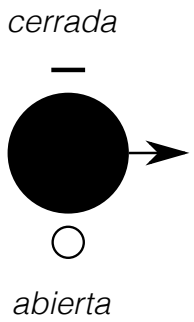


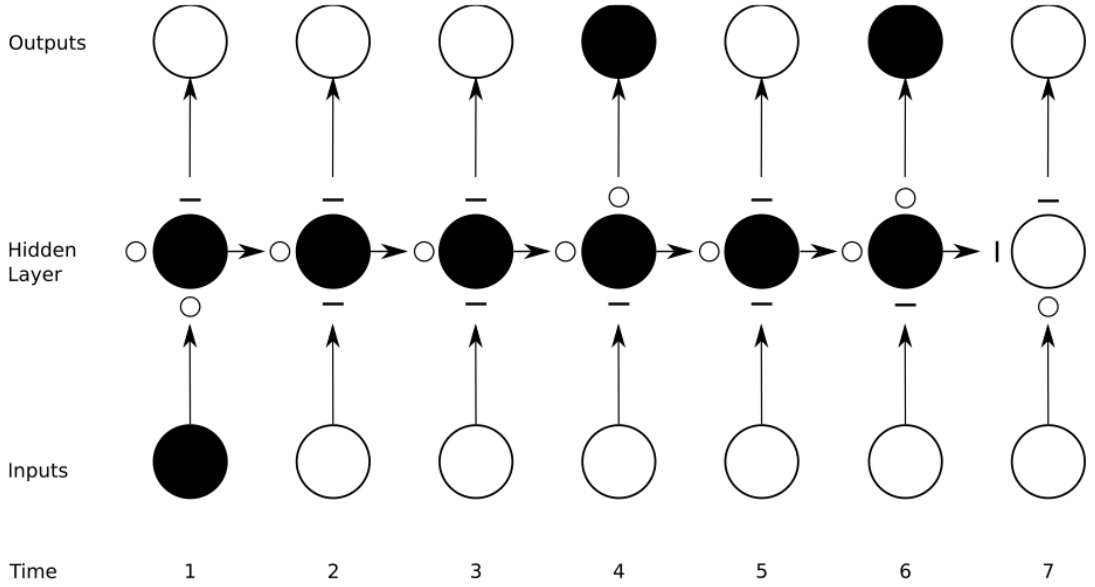
$$h_t = o_t \odot \sigma_h(c_t)$$

$$o_t = \sigma_o(U_o x_t + W_o h_{t-1})$$

Celdas LSTM

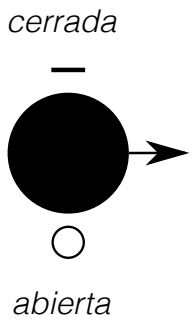
- Sincronizando adecuadamente las compuertas de entrada y salida, a unidad puede implementar skip connections a cualquier tiempo futuro en que recordar información de pasado sea importante.

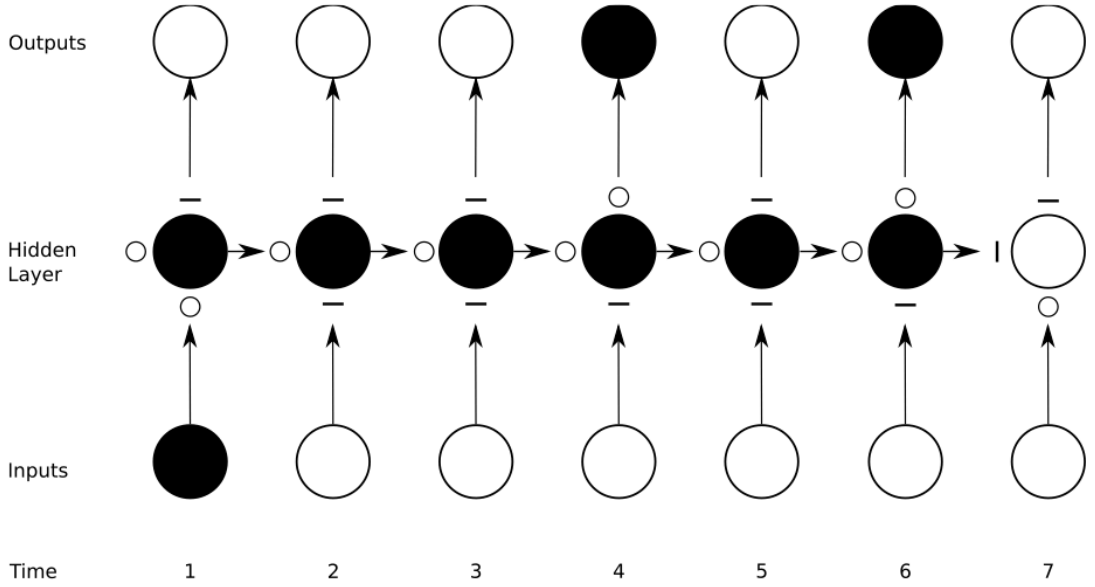




Celdas LSTM

- Sincronizando adecuadamente las compuertas de entrada y salida, a unidad puede implementar skip connections a cualquier tiempo futuro en que recordar información de pasado sea importante.





Forget Gate

- Si las compuertas de entrada no se cierran perfectamente, la unidad puede acumular información indefinidamente y eventualmente explotar. Para remediar esto, Gers propone en 1999 el uso de una compuerta de olvido $f_t \in [0,1]$ que permita resetear el estado interno cuando sea necesario.

Learning to Forget: Continual Prediction with LSTM

Technical Report IDSIA-01-99
January, 1999

Felix A. Gers felix@idsia.ch
Jürgen Schmidhuber juergen@idsia.ch
Fred Cummins fred@idsia.ch
IDSIA, Corso Elvezia 36
6900 Lugano, Switzerland
www.idsia.ch

Abstract

Long Short-Term Memory (LSTM, Hochreiter & Schmidhuber, 1997) can solve numerous tasks not solvable by previous learning algorithms for recurrent neural networks (RNNs). We identify a weakness of LSTM networks processing continual input streams that are not *a priori* segmented into subsequences with explicitly marked ends at which the network's internal state could be reset. Without resets, the state may grow indefinitely and eventually cause the network to break down. Our remedy is a novel, adaptive "forget gate" that enables an LSTM cell to learn to reset itself at appropriate times, thus releasing internal resources. We review illustrative benchmark problems on which standard LSTM outperforms other RNN algorithms. All algorithms (including LSTM) fail to solve continual versions of these problems. LSTM with forget gates, however, easily solves them in an elegant way.

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." (1999): 850-855.

Forget Gate

- El carrusel se modifica así:

$$c_t = f_t \odot c_{t-1} + i_t \odot a_{t-1}$$

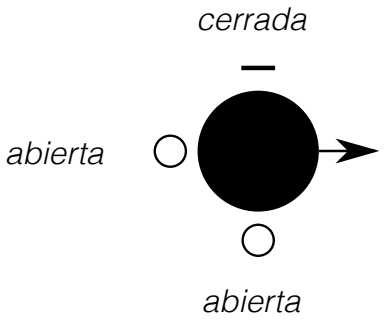
nueva info $a_t = \sigma_a(U x_t + W h_{t-1})$

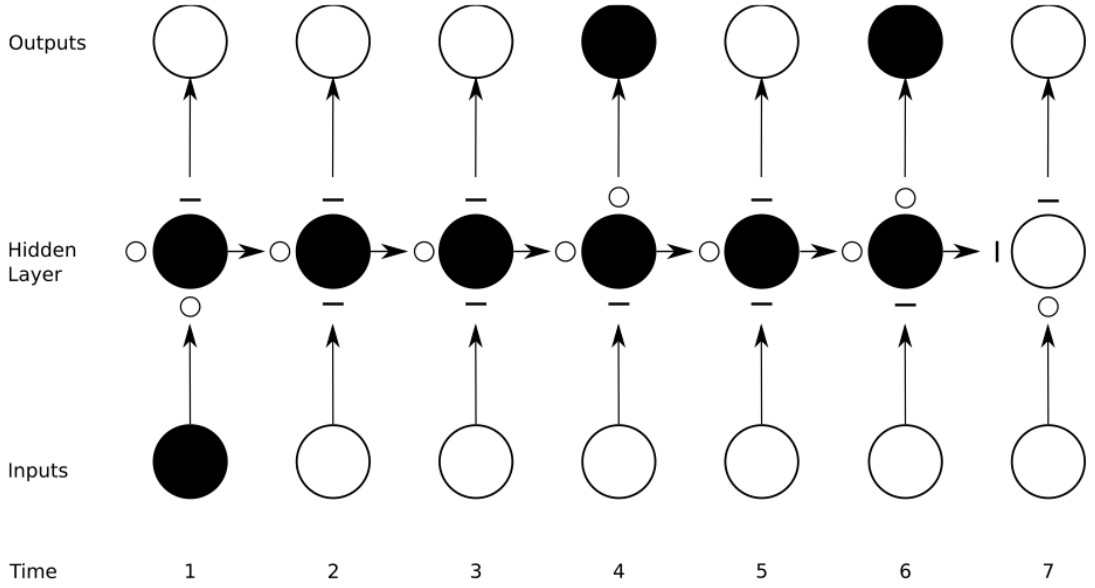
olvido $f_t = \sigma_f(U_f x_t + W_f h_{t-1})$

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." (1999): 850-855.

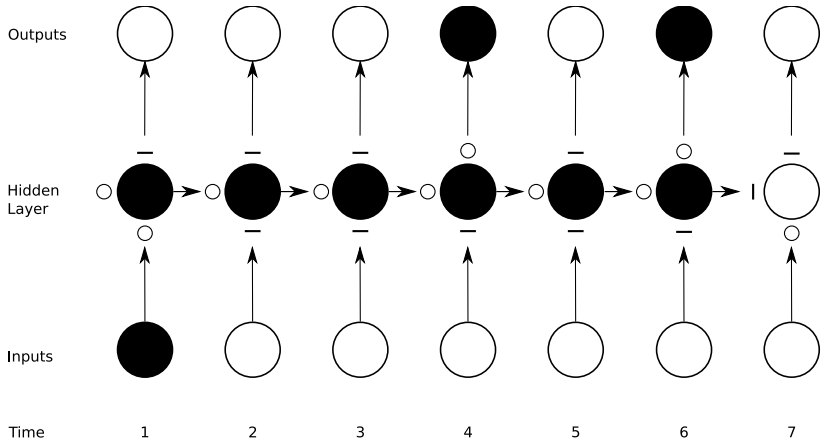
Forget Gate

- Esta compuerta permite implementar múltiples skip connections en la red re-utilizando la misma unidad.





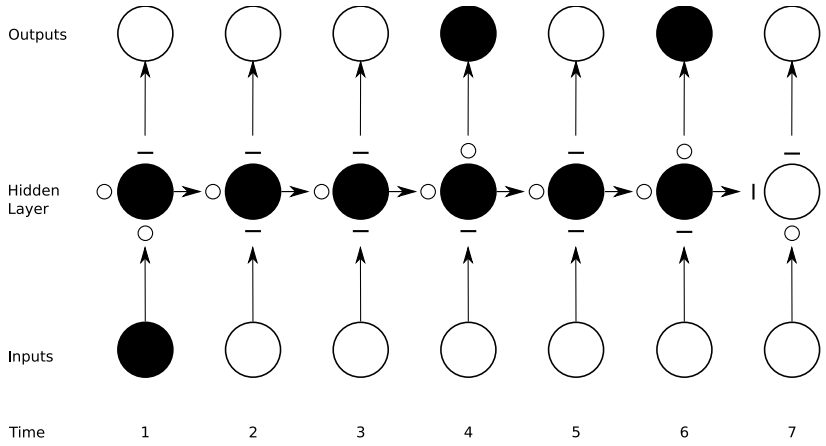
LSTM - Skip Connections



$$a_1 = \sigma_a\left(Ux_1 + Wh_0\right) = \sigma_a\left(Ux_1\right)$$
$$i_1 = 1$$
$$o_1 = 0$$
$$f_1 = 1$$
$$c_1 = f_1 \odot c_0 + i_1 \odot a_1 = \sigma_a\left(Ux_1\right)$$
$$h_1 = o_1 \odot \sigma_h\left(c_1\right) = 0$$

$$a_2 = \sigma_a\left(Ux_2 + Wh_1\right) = \sigma_a\left(Ux_2\right)$$
$$i_2 = 0$$
$$o_2 = 0$$
$$f_2 = 1$$
$$c_2 = f_2 \odot c_1 + i_2 \odot a_2 = c_1 = \sigma_a\left(Ux_1\right)$$
$$h_2 = o_2 \odot \sigma_h\left(c_2\right) = 0$$

LSTM - Skip Connections

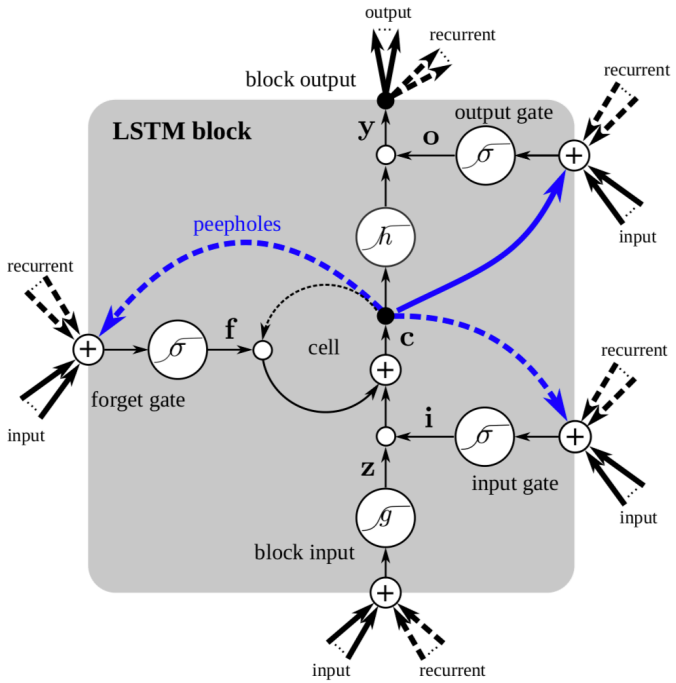


$$a_3 = \sigma_a\left(Ux_3 + Wh_2\right) = \sigma_a\left(Ux_3\right)$$
$$i_3 = 0$$
$$o_3 = 0$$
$$f_3 = 1$$
$$c_3 = f_3 \odot c_2 + i_3 \odot a_3 = c_2 = c_1 = \sigma_a\left(Ux_1\right)$$
$$h_3 = o_3 \odot \sigma_h\left(c_2\right) = 0$$

$$a_4 = \sigma_a\left(Ux_4 + Wh_3\right) = \sigma_a\left(Ux_4\right)$$
$$i_4 = 0$$
$$o_4 = 1$$
$$f_4 = 1$$
$$c_4 = f_4 \odot c_3 + i_4 \odot a_4 = c_3 = c_2 = c_1 = \sigma_a\left(Ux_1\right)$$
$$h_4 = o_4 \odot \sigma_h\left(c_4\right) = \sigma_h\left(\sigma_a\left(Ux_1\right)\right) \approx \sigma_a(Ux_1)$$

Peephole LSTM

- Un poco más tarde Gers propone otra variante en que el estado de las compuertas se controla usando la memoria interna en vez de los estados que muestra la celda al exterior.



Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." *Journal of machine learning research* (2002): 115-143.

GRU

- Mucho más recientemente, se han propuesto un sin número de variantes.
- Entre las más relevantes está el modelo GRU (Gated Recurrent Unit) que (buscando eficiencia) elimina la compuerta de entrada combinándola con la de olvido e identifica el estado de la celda con el estado mostrado al exterior.

$$z_t = \sigma_z(U_z x_t + W_z h_{t-1} + b_z)$$

$$r_t = \sigma_z(U_r x_t + W_r h_{t-1} + b_r)$$

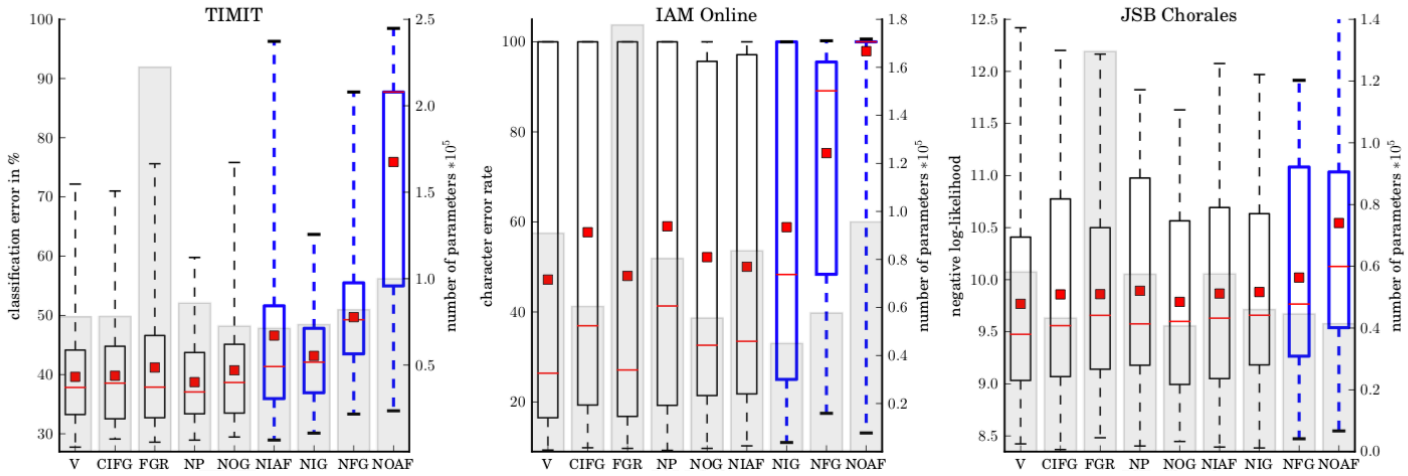
$$a_t = \sigma_a(U x_t + W(r_t \odot h_{t-1}))$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot a_t$$

Cho, Kyunghyun, et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014.

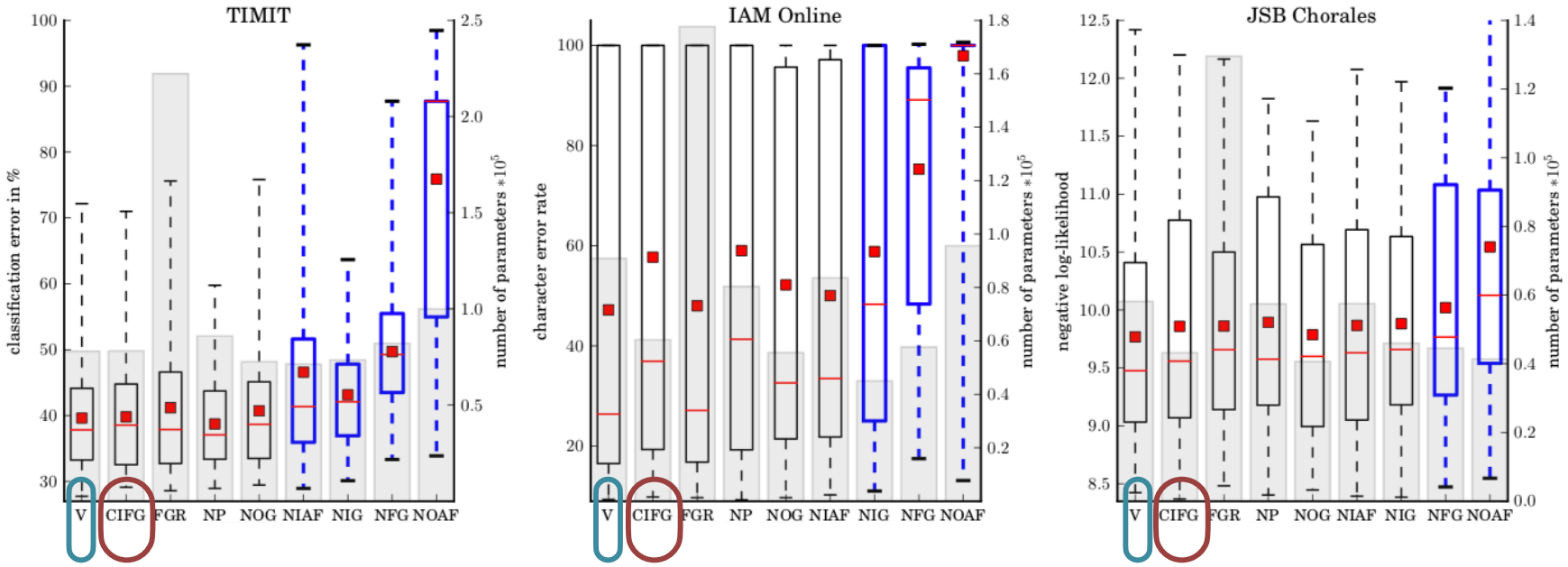
LSTM Variants

- Recientemente Greff compara 8 variantes del modelo original en diferentes tareas de aprendizaje de secuencias, concluyendo que ninguna de ellas mejora significativamente su capacidad de predicción



Greff, Klaus, et al. "LSTM: A search space odyssey." *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2016): 2222-2232.

LSTM Variants



GRU Vanilla LSTM

Greff, Klaus, et al. "LSTM: A search space odyssey." *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2016): 2222-2232.

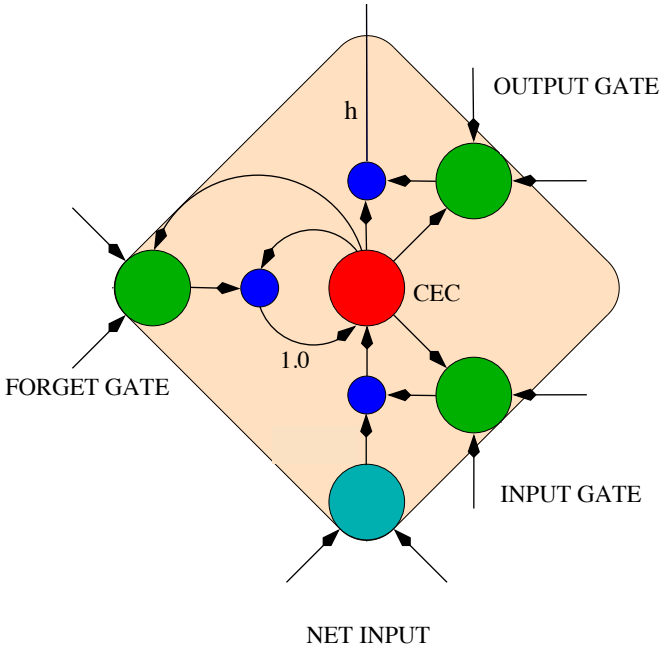
LSTM Variants

- Una compuerta de olvido y la función de activación interesante es que la compuerta de olvido así como la no-linealidad tipo (squashing?) en la salida de la ceda parecen ser fundamentales.

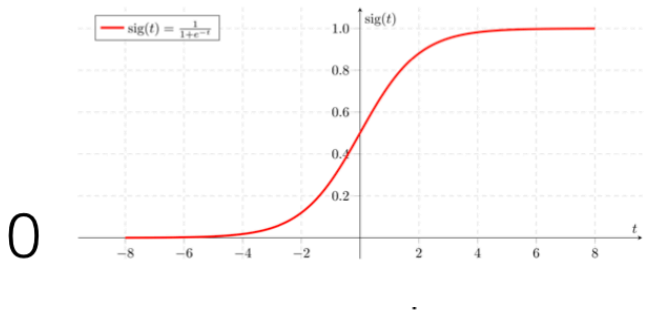
The first important observation based on Figure 3 is that removing the output activation function (NOAF) or the forget gate (NFG) significantly hurt performance on all three datasets. Apart from the CEC, the ability to forget old information and the squashing of the cell state appear to be critical for the LSTM architecture. Indeed, without the output activation function, the block output can in principle grow unbounded. Coupling the input and the forget gate avoids this problem and might render the use of an output non-linearity less important, which could explain why GRU performs well without it.

Greff, Klaus, et al. "LSTM: A search space odyssey." *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2016): 2222-2232.

LSTM Estándar



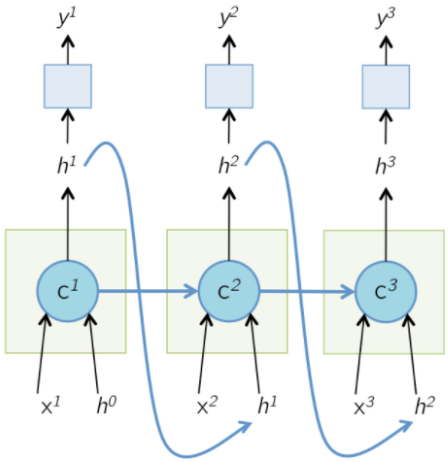
$$a_t = \sigma_a \left(U x_t + W h_{t-1} \right)$$
$$i_t = \sigma_i \left(U_i x_t + W_i h_{t-1} \right)$$
$$o_t = \sigma_o \left(U_o x_t + W_o h_{t-1} \right)$$
$$f_t = \sigma_f \left(U_f x_t + W_f h_{t-1} \right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$
$$h_t = o_t \odot \sigma_h \left(c_t \right)$$



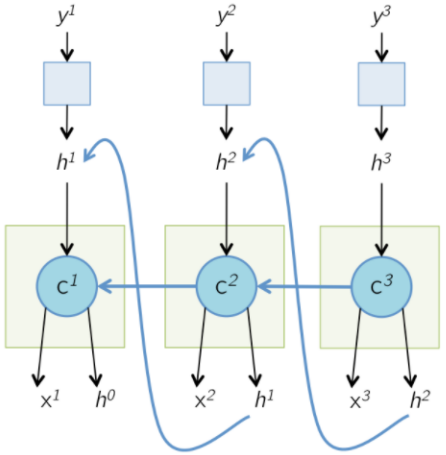
Típicamente se adopta una función de transferencia sigmoidal para las compuertas y tanh para las demás

LSTM - Aprendizaje

- Es posible entrenar una red LSTM desenrollando la red y adaptando las ecuaciones correspondientes a BPTT para aprender los pesos de cada compuerta.



forward pass



backward pass

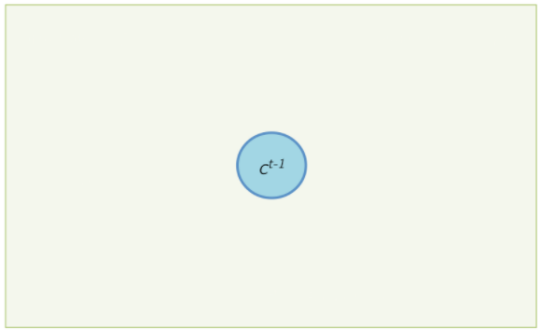
LSTM - Aprendizaje

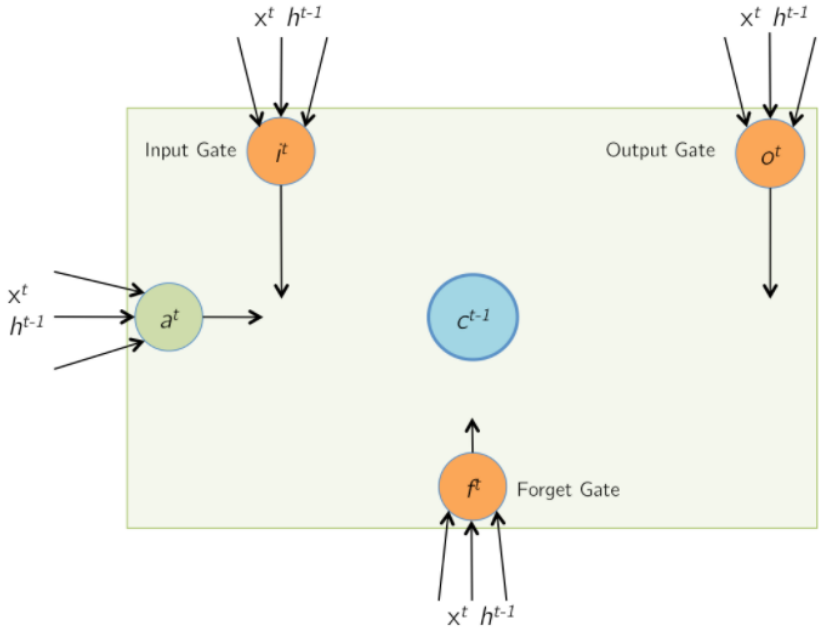
- Podemos dividir el **forward pass** en 3 pasos fundamentales para cada tiempo t (elemento de la secuencia).

1. Calcular el estado de las compuertas y la activación correspondiente al input actual y el output anterior de la celda.
2. Actualizar el estado interno de la celda a partir del estado de las compuertas y la activación actual.
3. Calcular la salida a partir del estado interno de la celda y el estado de las compuertas.

LSTM - Aprendizaje

forward pass



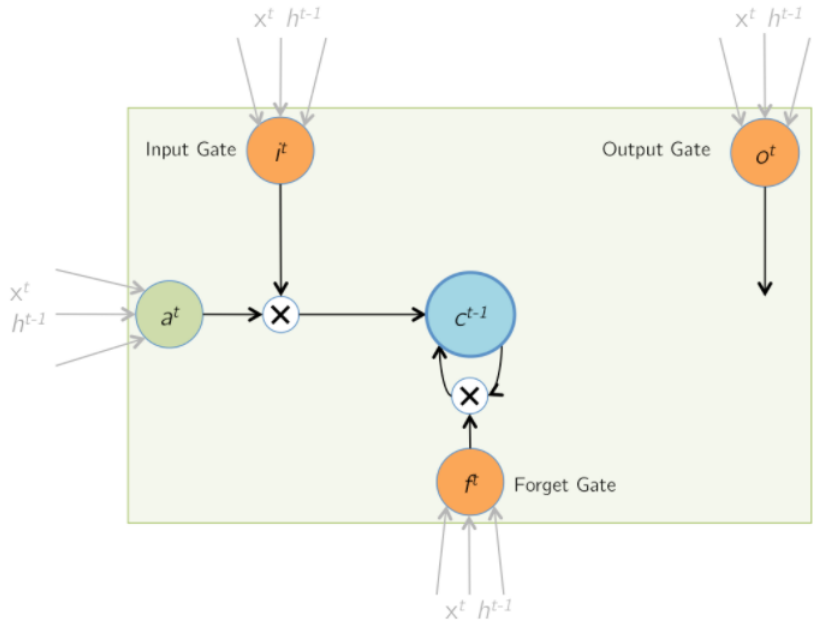


1

estado de compuertas

$$a_t = \sigma_a\left(Ux_t + Wh_{t-1}\right)$$
$$i_t = \sigma_i\left(U_i x_t + W_i h_{t-1}\right)$$
$$o_t = \sigma_o\left(U_o x_t + W_o h_{t-1}\right)$$
$$f_t = \sigma_f\left(U_f x_t + W_f h_{t-1}\right)$$

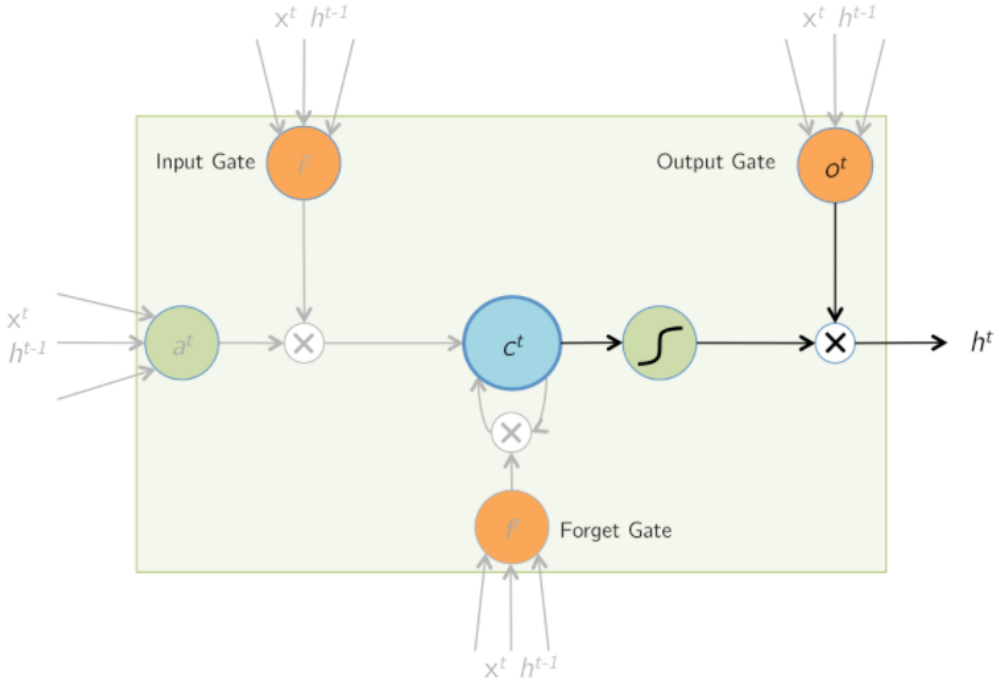
LSTM - Aprendizaje



2 actualización del estado

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

LSTM - Aprendizaje



3 salida

$$h_t = o_t \odot \sigma_h(c_t)$$

LSTM - Aprendizaje

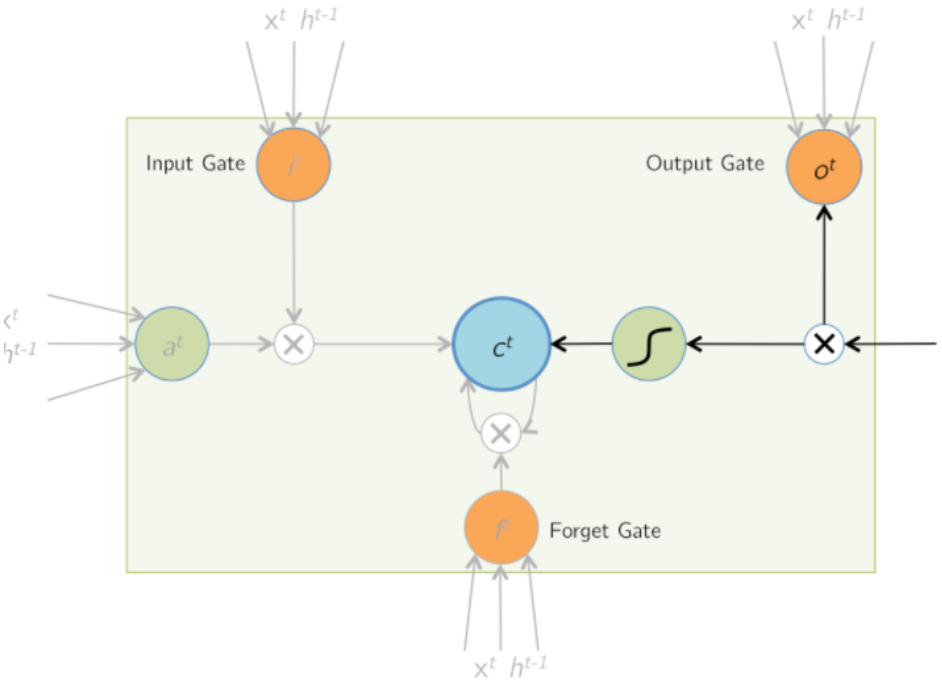
- Podemos dividir el **backward pass** en 3 pasos fundamentales que esencialmente corresponden recorrer en orden inverso los 3 pasos usados para implementar el forward pass.

    1. Calcular los gradientes correspondientes al estado interno de la celda y al estado de la compuerta de salida en el tiempo t.
    2. Calcular los gradientes correspondientes a la activación y los estados de las compuertas de entrada y olvido en el tiempo t.
    3. Pasar a la capa anterior (de la red desenrollada) el gradiente correspondiente a las señales recibidas (estado oculto y patrón de entrada).

- **Asumiremos por simplicidad que sólo se calcula un error (E) al terminar el procesamiento de la secuencia (arquitectura many to one).**

LSTM - Aprendizaje

backward pass



$$\frac{\partial E}{\partial h_t}$$

3 salida

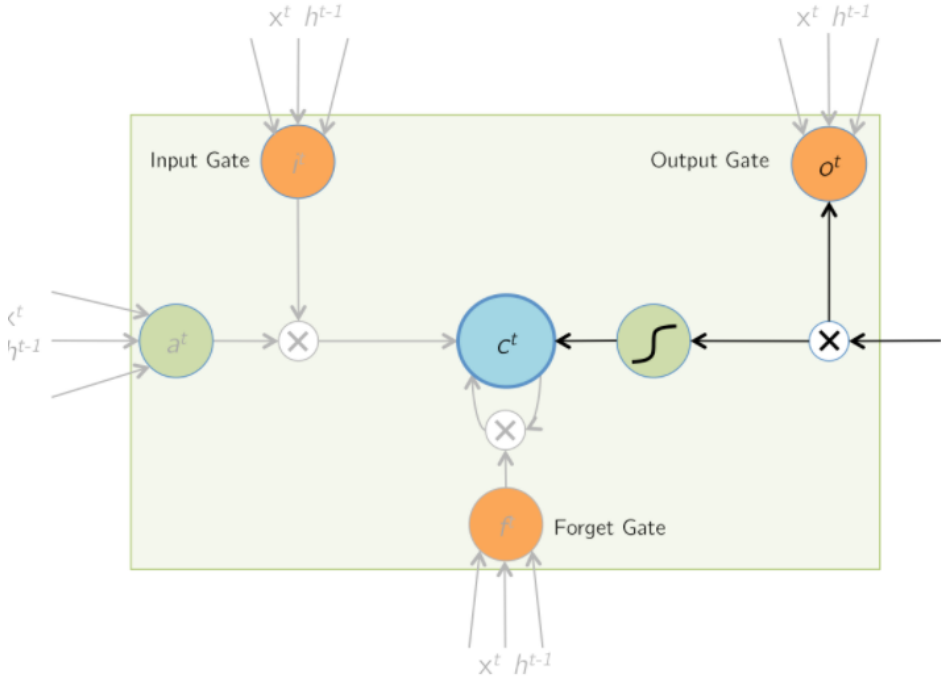
$$h_t = o_t \odot \sigma_h(c_t)$$

dado: $\frac{\partial E}{\partial h_t}$

queremos: $\frac{\partial E}{\partial o_t}, \frac{\partial E}{\partial c_t}$

LSTM - Aprendizaje $\partial E / \partial o_t$

backward pass



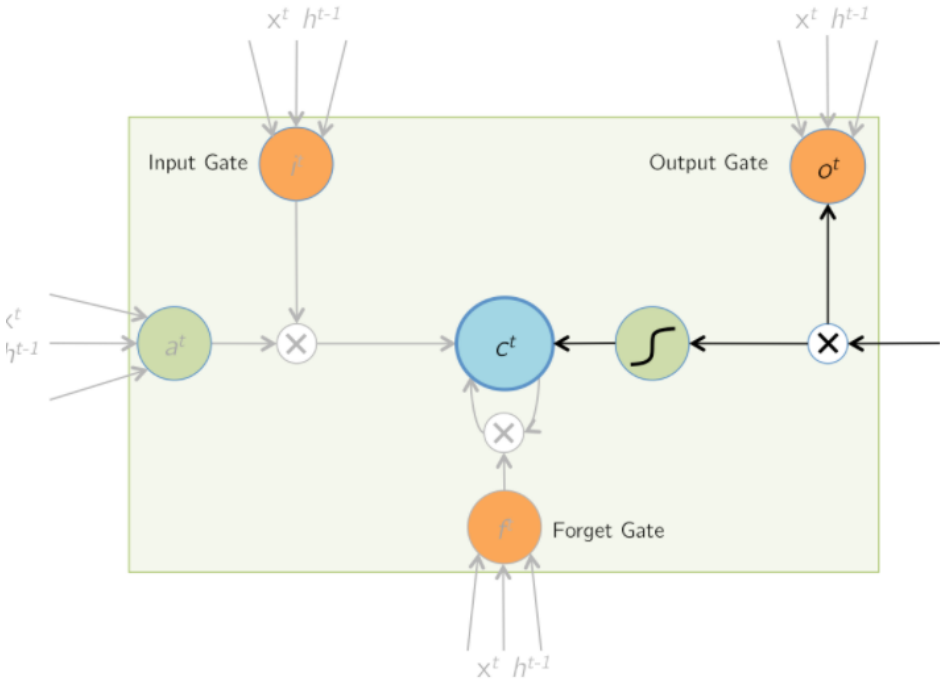
$$\frac{\partial E}{\partial h_t}$$

3 salida

$$h_t = o_t \odot \sigma_h(c_t)$$

LSTM - Aprendizaje $\partial E / \partial c_t$

backward pass



$$\frac{\partial E}{\partial h_t}$$

3 salida

$$h_t = o_t \odot \sigma_h(c_t)$$

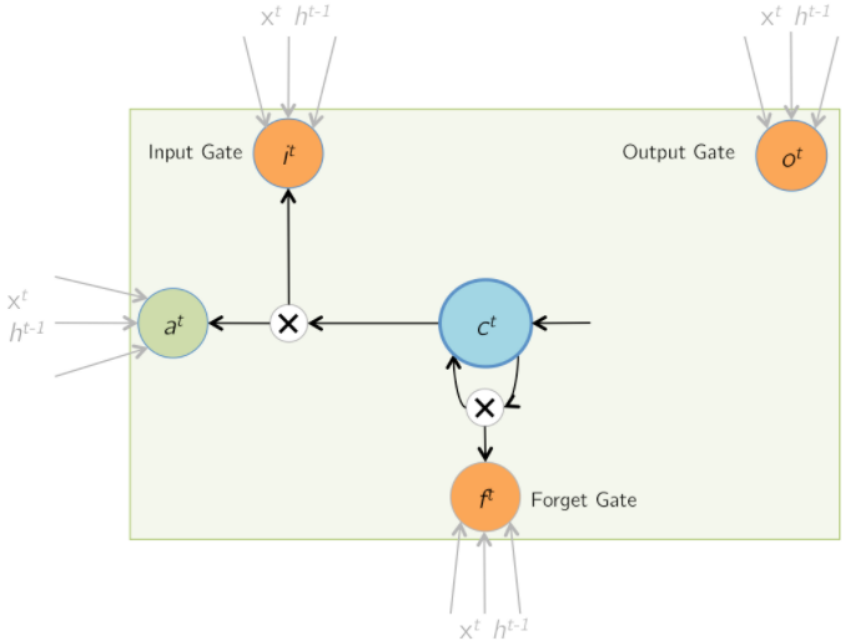
Si la completa de salida está abierta, la señal de error entra, sino se bloquea.

$$\frac{\partial E}{\partial c_t[j]} = \frac{\partial E}{\partial h_t[j]} \frac{\partial h_t[j]}{\partial c_t[j]}$$

$$\frac{\partial h_t[j]}{\partial c_t[j]} = o_t[j] \sigma_h'(c_t[j])$$

$$\Rightarrow \frac{\partial E}{\partial c_t} = \frac{\partial E}{\partial h_t} \odot o_t \odot \sigma'(c_t)$$

LSTM - Aprendizaje



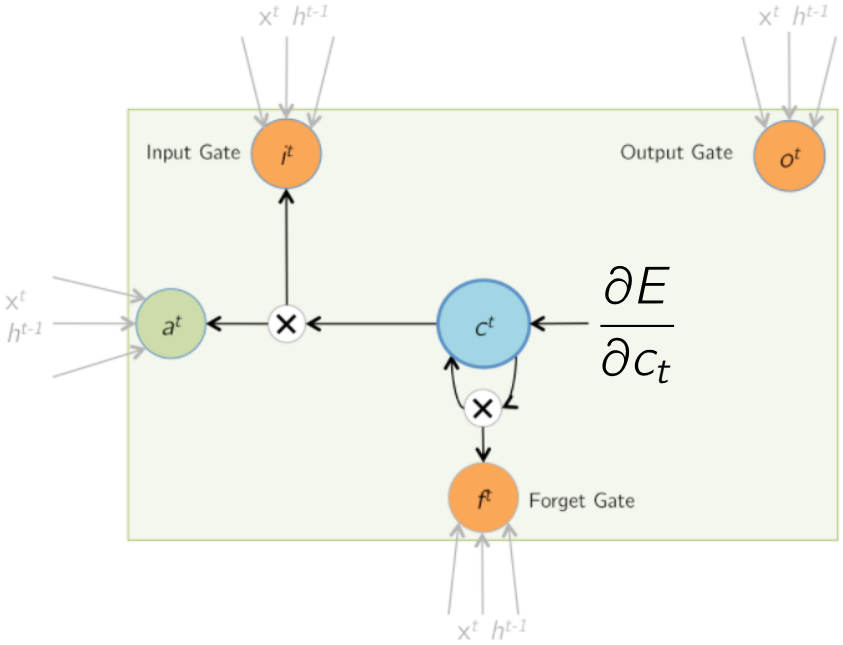
2 actualización del estado

$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$

dados: $\frac{\partial E}{\partial o_t}, \frac{\partial E}{\partial c_t}$

queremos: $\frac{\partial E}{\partial i_t}, \frac{\partial E}{\partial f_t}, \frac{\partial E}{\partial a_t}$

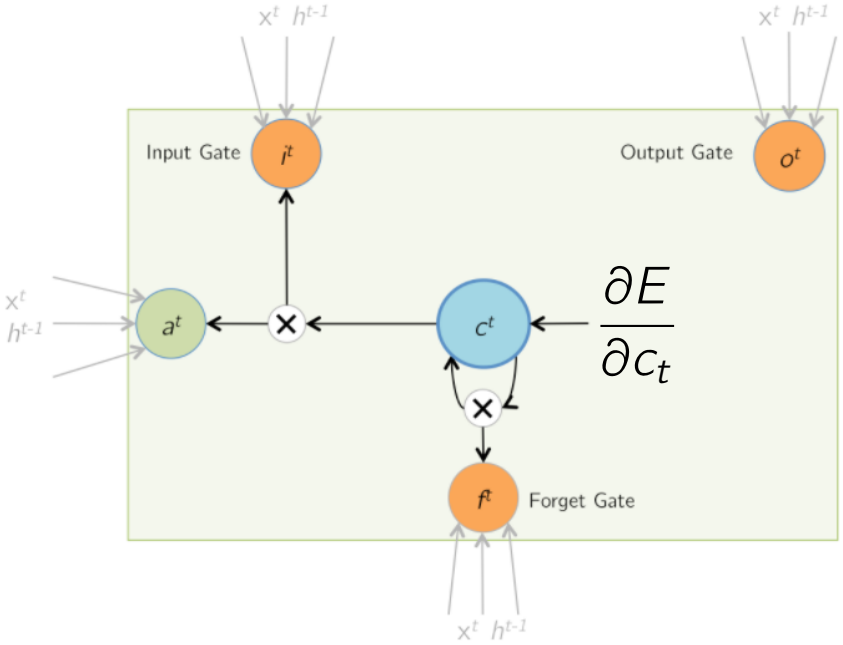
LSTM - Aprendizaje $\partial E / \partial i_t$



2 actualización del estado

$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$

LSTM - Aprendizaje $\partial E / \partial i_t$



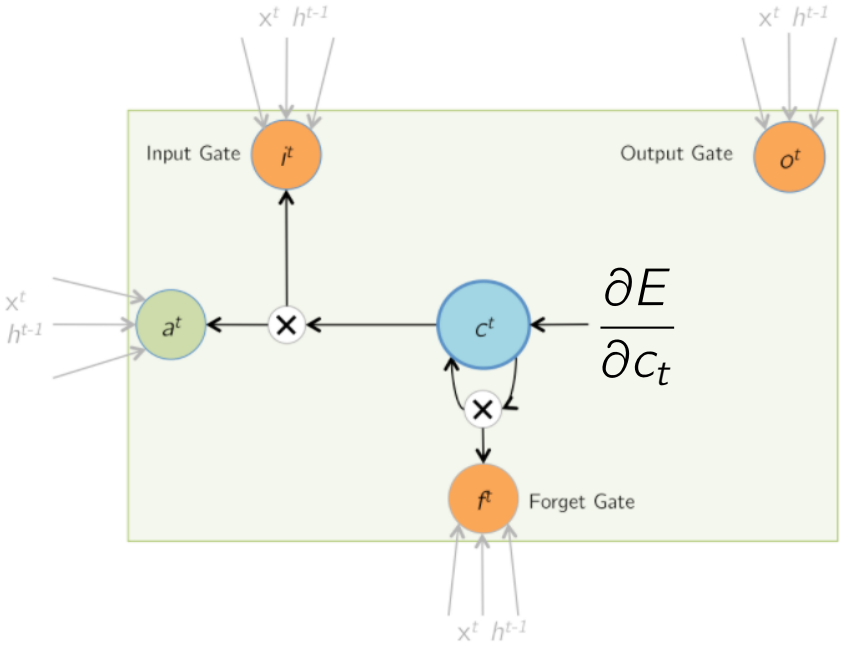
2 actualización del estado

$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$

$$\frac{\partial E}{\partial i_t[j]} = \frac{\partial E}{\partial c_t[j]} \frac{\partial c_t[j]}{\partial i_t[j]}$$

$$\frac{\partial c_t[j]}{\partial i_t[j]} = a_t[j] \Rightarrow \frac{\partial E}{\partial i_t} = \frac{\partial E}{\partial c_t} \odot a_t$$

LSTM - Aprendizaje $\partial E/\partial f_t$



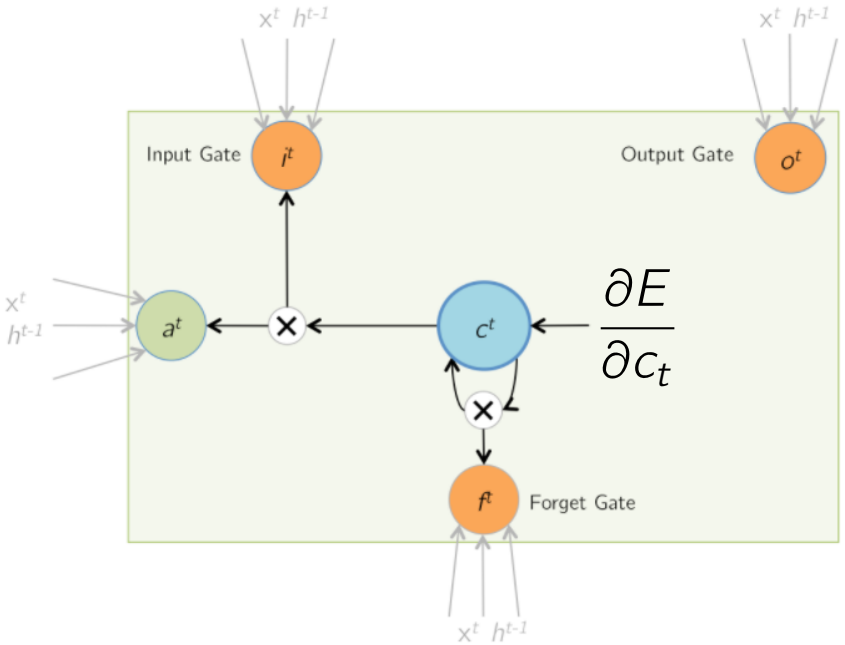
2 actualización del estado

$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$

$$\frac{\partial E}{\partial f_t[j]} = \frac{\partial E}{\partial c_t[j]} \frac{\partial c_t[j]}{\partial f_t[j]}$$

$$\frac{\partial c_t[j]}{\partial f_t[j]} = c_{t-1}[j] \Rightarrow \frac{\partial E}{\partial f_t} = \frac{\partial E}{\partial c_t} \odot c_{t-1}$$

LSTM - Aprendizaje $\partial E / \partial a_t$



2 actualización del estado

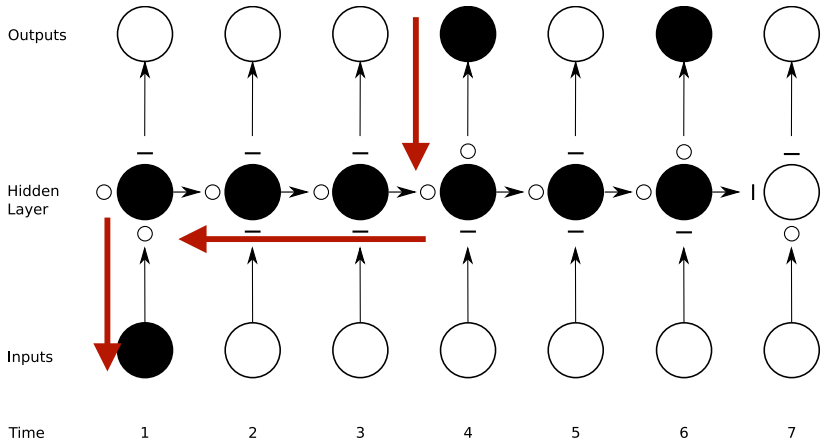
$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$

si la compuerta de entrada es 1, la señal de error se propaga perfectamente hacia la neurona verde que era la única en la arquitectura RNN tradicional (sin compuertas).

$$\frac{\partial E}{\partial a_t[j]} = \frac{\partial E}{\partial c_t[j]} \frac{\partial c_t[j]}{\partial a_t[j]}$$

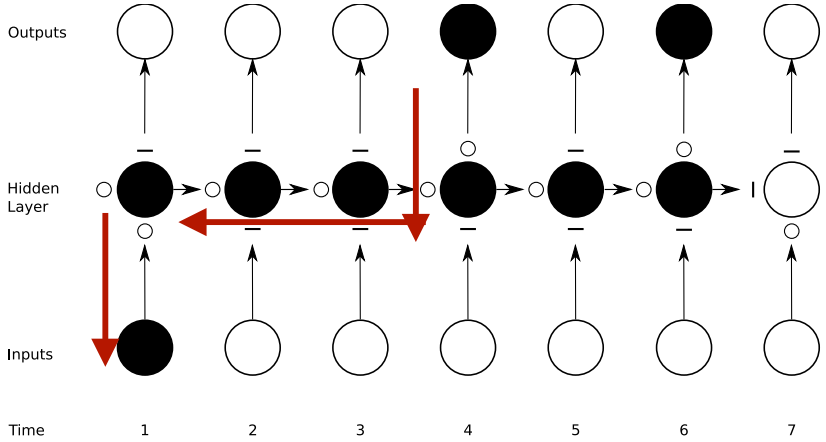
$$\frac{\partial c_t[j]}{\partial a_t[j]} = i_t[j] \Rightarrow \frac{\partial E}{\partial f_t} = \frac{\partial E}{\partial c_t} \odot i_t$$

LSTM - Aprendizaje



La red puede aprender a sincronizar sus compuertas no sólo para propagar selectivamente una información hacia el futuro durante el forward-pass sino que **también puede aprender a propagar selectivamente una señal de error hacia el pasado en el backward-pass.**

LSTM y Gradientes Desvanecientes



Antes:

$$\frac{\partial E_4}{\partial U} = \sum_{i=1}^{4} \frac{\partial h_i}{\partial U} \frac{\partial E_4}{\partial h_i}$$

$$= \sum_{i=1}^{4} \frac{\partial h_i}{\partial U} \frac{\partial h_4}{\partial h_i} \frac{\partial E_4}{\partial h_4}$$

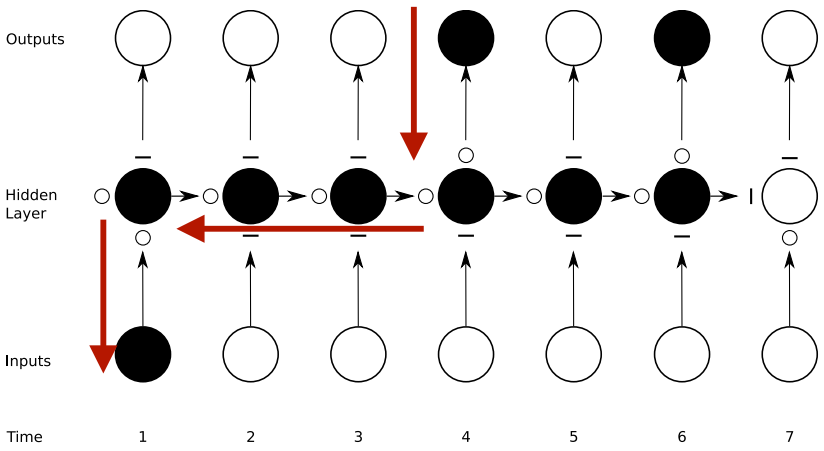
$$\frac{\partial h_4}{\partial h_1} = \prod_{j=1}^{3} \frac{\partial h_{j+1}}{\partial h_j}$$

$$\left\| \frac{\partial h_4}{\partial h_1} \right\| \approx \|U\|^3$$

La influencia de $\frac{\partial h_1}{\partial U} \approx x_1$ se desvanece

(imaginemos, por simplicidad, que todos los nodos son escalares)

LSTM y Gradientes Desvanecientes



Ahora

$$\frac{\partial E_4}{\partial U} = \sum_{i=1}^{4} \frac{\partial c_i}{\partial U} \frac{\partial E_4}{\partial c_i}$$

$$= \sum_{i=1}^{4} \frac{\partial c_i}{\partial U} \frac{\partial c_4}{\partial c_i} \frac{\partial E_4}{\partial c_4}$$

$$= \frac{\partial c_1}{\partial U} \frac{\partial c_4}{\partial c_1} \frac{\partial E_4}{\partial c_4}$$

Como

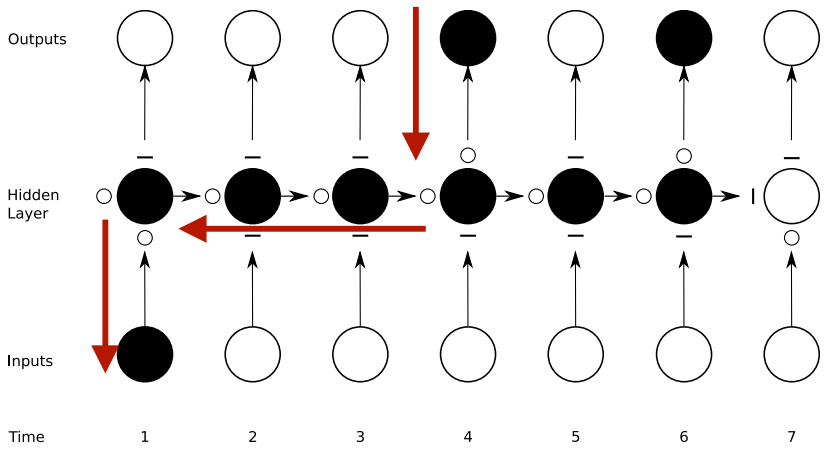
$$c_t = f_t \odot c_{t-1} + i_t \odot a_t$$

Obtenemos

$$\frac{\partial c_4}{\partial c_1} = \frac{\partial c_4}{\partial c_3} \frac{\partial c_3}{\partial c_2} \frac{\partial c_2}{\partial c_1}$$

$$= f_3 \cdot f_2 \cdot f_1$$

LSTM - Aprendizaje



$$\frac{\partial E_4}{\partial U} = \sum_{i=1}^{4} \frac{\partial c_i}{\partial U} \frac{\partial E_4}{\partial c_i}$$

$$= \sum_{i=1}^{4} \frac{\partial c_i}{\partial U} \frac{\partial c_4}{\partial c_i} \frac{\partial E_4}{\partial c_4}$$

$$= \frac{\partial c_1}{\partial U} \frac{\partial c_4}{\partial c_1} \frac{\partial E_4}{\partial c_4}$$

$$= \frac{\partial c_1}{\partial U} \frac{\partial E_4}{\partial c_4}$$

$$\frac{\partial E_4}{\partial U} = \frac{\partial a_1}{\partial U} \frac{\partial c_1}{\partial a_1} \frac{\partial E_4}{\partial c_4}$$

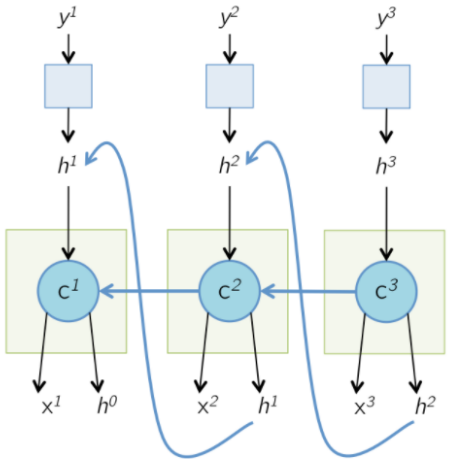
$$= x_1 \, i_1 \, \frac{\partial E_4}{\partial c_4}$$

$$= x_1 \frac{\partial E_4}{\partial c_4}$$

Los gradientes no se desvanencen!

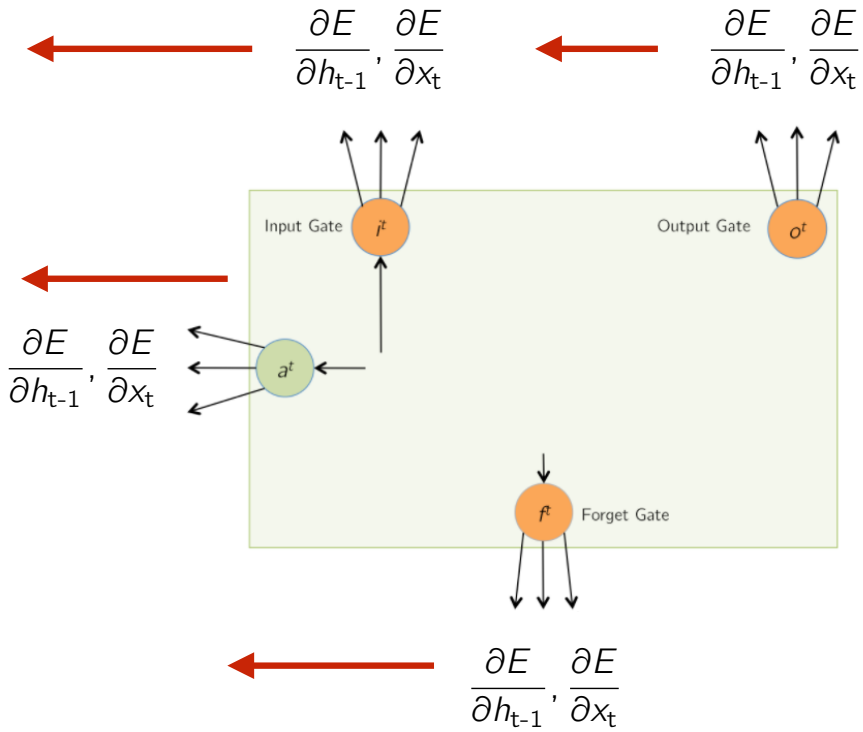
LSTM - Cierre de la Recursión

- Para completar la recurrencia característica de BP debemos propagar hacia atrás aquello que asumimos recibimos para hacer todos los cálculos.



backward pass

1 estado de compuertas



LSTM - Aprendizaje $\partial E/\partial h_{t-1}, \partial E/\partial x_t$

- Basta ver las ecuaciones ...

$$
\begin{aligned}
a_t &= \sigma_a \left(U x_t + W h_{t-1} \right) \\
i_t &= \sigma_i \left(U_i x_t + W_i h_{t-1} \right) \\
o_t &= \sigma_o \left(U_o x_t + W_o h_{t-1} \right) \\
f_t &= \sigma_f \left(U_f x_t + W_f h_{t-1} \right)
\end{aligned}
$$

- Por lo tanto:

$$
\frac{\partial E}{\partial h_{t-1}}[i] = \sum_j \frac{\partial E}{\partial a_t[j]} \frac{\partial a_t[j]}{\partial h_{t-1}[i]} + \sum_j \frac{\partial E}{\partial i_t[j]} \frac{\partial i_t[j]}{\partial h_{t-1}[i]} + \sum_j \frac{\partial E}{\partial o_t[j]} \frac{\partial o_t[j]}{\partial h_{t-1}[i]} + \sum_j \frac{\partial E}{\partial f_t[j]} \frac{\partial f_t[j]}{\partial h_{t-1}[i]}
$$

$$
\frac{\partial E}{\partial x_t}[i] = \sum_j \frac{\partial E}{\partial a_t[j]} \frac{\partial a_t[j]}{\partial x_t[i]} + \sum_j \frac{\partial E}{\partial i_t[j]} \frac{\partial i_t[j]}{\partial x_t[i]} + \sum_j \frac{\partial E}{\partial o_t[j]} \frac{\partial o_t[j]}{\partial x_t[i]} + \sum_j \frac{\partial E}{\partial f_t[j]} \frac{\partial f_t[j]}{\partial x_t[i]}
$$

LSTM - Aprendizaje $\partial E/\partial h_{t-1}, \partial E/\partial x_t$

$$\frac{\partial E}{\partial h_{t-1}}[i] = \sum_j \frac{\partial E}{\partial a_t[j]} \frac{\partial a_t[j]}{\partial h_{t-1}[i]} + \sum_j \frac{\partial E}{\partial i_t[j]} \frac{\partial i_t[j]}{\partial h_{t-1}[i]} + \sum_j \frac{\partial E}{\partial o_t[j]} \frac{\partial o_t[j]}{\partial h_{t-1}[i]} + \sum_j \frac{\partial E}{\partial f_t[j]} \frac{\partial f_t[j]}{\partial h_{t-1}[i]}$$

$$\frac{\partial E}{\partial x_t}[i] = \sum_j \frac{\partial E}{\partial a_t[j]} \frac{\partial a_t[j]}{\partial x_t[i]} + \sum_j \frac{\partial E}{\partial i_t[j]} \frac{\partial i_t[j]}{\partial x_t[i]} + \sum_j \frac{\partial E}{\partial o_t[j]} \frac{\partial o_t[j]}{\partial x_t[i]} + \sum_j \frac{\partial E}{\partial f_t[j]} \frac{\partial f_t[j]}{\partial x_t[i]}$$

- Vectorialmente:

$$\frac{\partial E}{\partial h_{t-1}}[i] = \frac{\partial E}{\partial a_t} \cdot \frac{\partial a_t}{\partial h_{t-1}[i]} + \frac{\partial E}{\partial i_t} \cdot \frac{\partial i_t}{\partial h_{t-1}[i]} + \frac{\partial E}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_{t-1}[i]} + \frac{\partial E}{\partial f_t} \cdot \frac{\partial f_t}{\partial h_{t-1}[i]}$$

$$\frac{\partial E}{\partial x_t}[i] = \frac{\partial E}{\partial a_t} \cdot \frac{\partial a_t}{\partial x_t[i]} + \frac{\partial E}{\partial i_t} \cdot \frac{\partial i_t}{\partial x_t[i]} + \frac{\partial E}{\partial o_t} \cdot \frac{\partial o_t}{\partial x_t[i]} + \frac{\partial E}{\partial f_t} \cdot \frac{\partial f_t}{\partial x_t[i]}$$

LSTM - Aprendizaje $\partial E/\partial h_{t-1}, \partial E/\partial x_t$

- Para terminar, falta ver que:

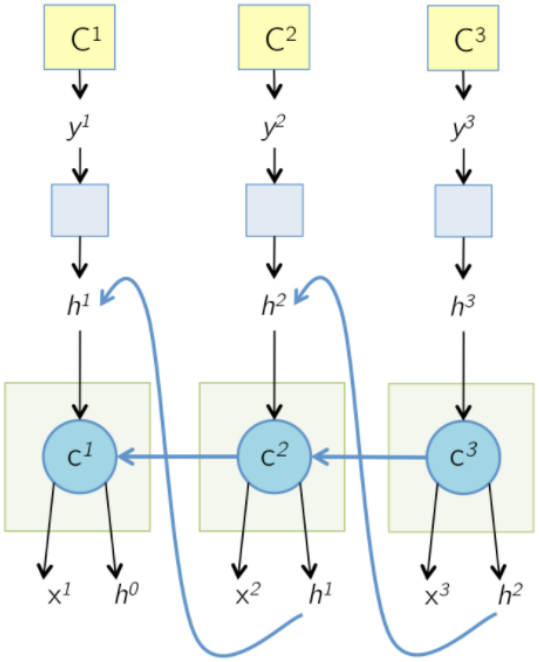
$$\frac{\partial a_t[j]}{\partial x_t[i]} = \sigma_a'(\cdot)W[j,i] \qquad \frac{\partial a_t[j]}{\partial h_{t-1}[i]} = \sigma_a'(\cdot)U[j,i]$$
$$\frac{\partial i_t[j]}{\partial x_t[i]} = \sigma_i'(\cdot)W_i[j,i] \qquad \frac{\partial i_t[j]}{\partial h_{t-1}[i]} = \sigma_i'(\cdot)U_i[j,i]$$
$$\frac{\partial o_t[j]}{\partial x_t[i]} = \sigma_o'(\cdot)W_o[j,i] \qquad \frac{\partial o_t[j]}{\partial h_{t-1}[i]} = \sigma_o'(\cdot)U_o[j,i]$$
$$\frac{\partial f_t[j]}{\partial x_t[i]} = \sigma_f'(\cdot)W_f[j,i] \qquad \frac{\partial f_t[j]}{\partial h_{t-1}[i]} = \sigma_f'(\cdot)U_f[j,i]$$

$$a_t = \sigma_a\left(Ux_t + Wh_{t-1}\right)$$
$$i_t = \sigma_i\left(U_i x_t + W_i h_{t-1}\right)$$
$$o_t = \sigma_o\left(U_o x_t + W_o h_{t-1}\right)$$
$$f_t = \sigma_f\left(U_f x_t + W_f h_{t-1}\right)$$

LSTM - Aprendizaje

- Después de propagar las señales de error, la celda está lista para actualizar sus pesos:

backward pass



Como los pesos son compartidos en cada capa (tiempo) debemos sumar todos los updates t=T,T-1,….1

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \sigma'_a() \frac{\partial E}{\partial a_t} \cdot h_{t-1}^T$$

$$\frac{\partial E}{\partial U} = \sum_{t=1}^{T} \sigma'_a() \frac{\partial E}{\partial a_t} \cdot x_t^T$$

Análogamente con las otras matrices ….

LSTMs - Bidireccionales

- Para contribuir a mejorar su habilidad en el aprendizaje de dependencias de largo plazo, tanto una red de Elman como una LSTM (o cualquiera de sus variantes) pueden hacerse **bi-direccionales.**

IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 45, NO. 11, NOVEMBER 1997 2673

Bidirectional Recurrent Neural Networks

Mike Schuster and Kuldip K. Paliwal, *Member, IEEE*

***Abstract*— In the first part of this paper, a regular recurrent neural network (RNN) is extended to a bidirectional recurrent neural network (BRNN). The BRNN can be trained without the limitation of using input information just up to a preset future frame. This is accomplished by training it simultaneously in positive and negative time direction. Structure and training procedure of the proposed network are explained. In regression and classification experiments on artificial data, the proposed structure gives better results than other approaches. For real data, classification experiments for phonemes from the TIMIT database show the same tendency.**

that, at least theoretically, is able to use all available input information to predict a point in the output space.

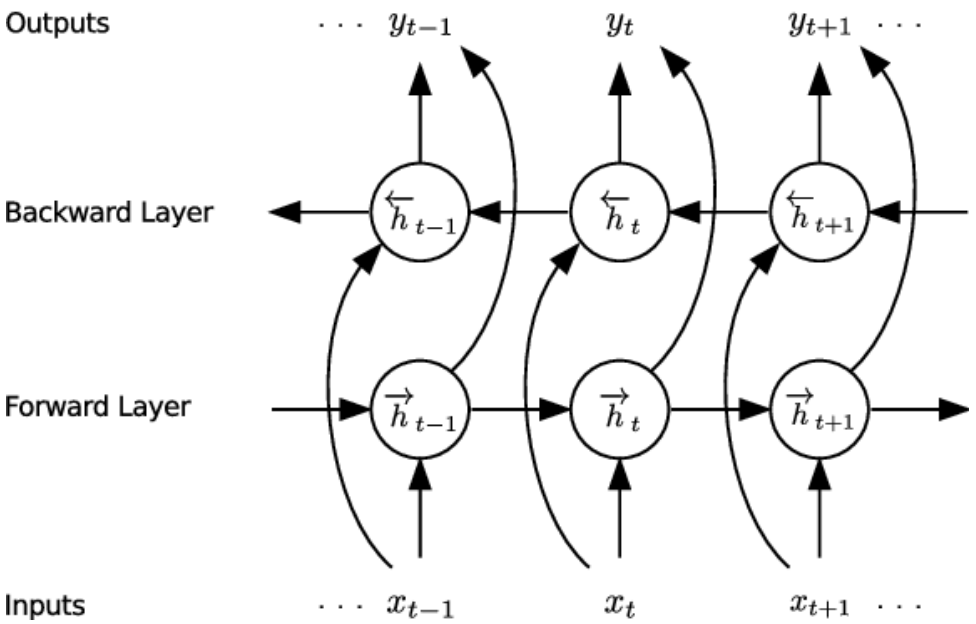
Many ANN structures have been proposed in the literature to deal with time varying patterns. Multilayer perceptrons (MLP's) have the limitation that they can only deal with static data patterns (i.e., input patterns of a predefined dimensionality), which requires definition of the size of the input window in advance. Waibel *et al*. [16] have pursued time delay neural networks (TDNN's), which have proven to be a useful

Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." *IEEE transactions on Signal Processing* 45.11 (1997): 2673-2681.

LSTMs - Bidireccionales

- Una capa bidireccional computa dos secuencias de estados ocultos, una forward y una backward:

$$\left(\overrightarrow{h_{1:T}}\right) = \overrightarrow{h}_1, \overrightarrow{h}_2, \ldots, \overrightarrow{h}_T \qquad \left(\overleftarrow{h_{1:T}}\right) = \overleftarrow{h}_1, \overleftarrow{h}_2, \ldots, \overleftarrow{h}_T$$



LSTMs - Bidireccionales

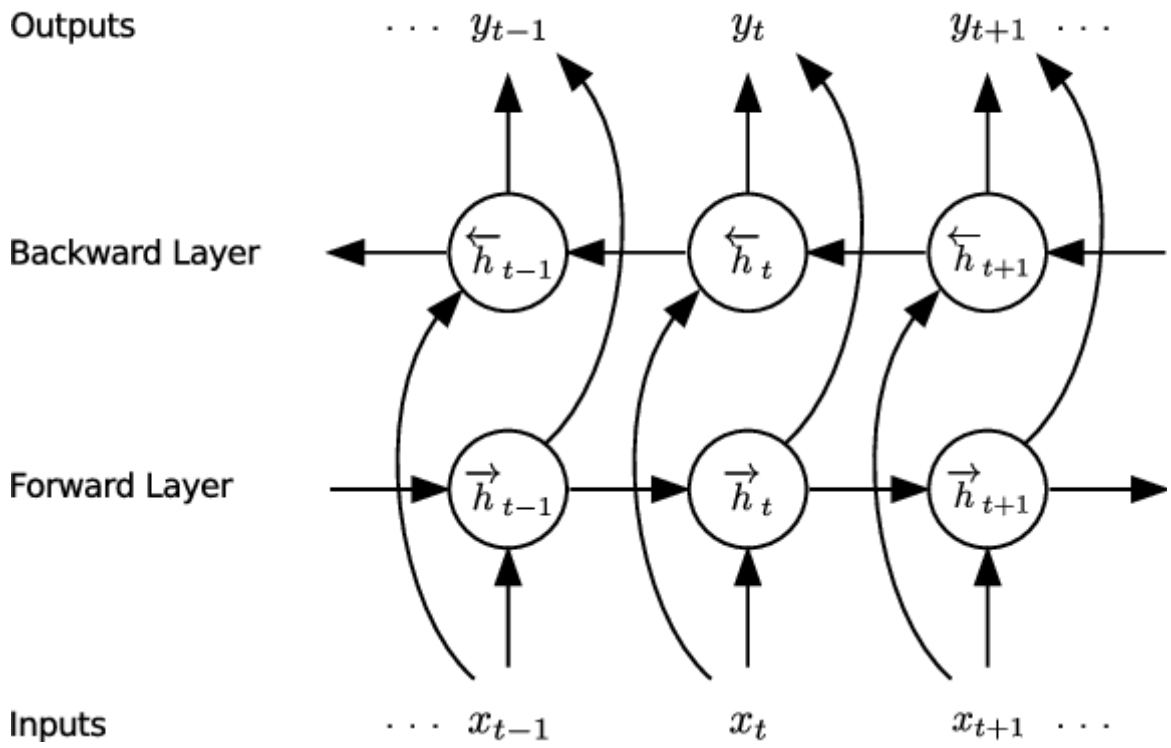
- Los estados ocultos forward se calculan leyendo la secuencia de izquierda a derecha, y los backward e calculan leyendo la secuencia de derecha a izquierda.

- En cada time step, los estados ocultos se concatenan y se entregan a la capa de más arriba. Si esta capa es la capa de salida, tendríamos

$$h_t = \overrightarrow{h}_t \oplus \overleftarrow{h}_t = \left(\overrightarrow{h}_t^T, \overleftarrow{h}_t^T\right)^T$$

$$y = g_o(V h_t + b_o) = g_o(\overrightarrow{V}\overrightarrow{h}_t + \overleftarrow{V}\overleftarrow{h}_t + b_o)$$

LSTMs - Bidireccionales



SimpleRNN layer

`SimpleRNN` class

```
tf.keras.layers.SimpleRNN(
    units,
    activation="tanh",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
```

Fully-connected RNN where the output is to be fed back to input.

See the Keras RNN API guide for details about the usage of RNN API.

Arguments

- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use. Default: hyperbolic tangent (`tanh`). If you pass None, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **use_bias**: Boolean, (default `True`), whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. Default: `glorot_uniform`.

LSTM layer

LSTM class

```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
```

Long Short-Term Memory layer - Hochreiter 1997.

See the Keras RNN API guide for details about the usage of RNN API.

Based on available runtime hardware and constraints, this layer will choose different implementations (cuDNN-based or pure-TensorFlow) to maximize the performance. If a GPU is available and all the arguments to the layer meet the requirement of the CuDNN kernel (see below for details), the layer will use a fast cuDNN implementation.

The requirements to use the cuDNN implementation are:

1. `activation` == `tanh`
2. `recurrent_activation` == `sigmoid`
3. `recurrent_dropout` == 0

Keras

GRU layer

GRU class

```
tf.keras.layers.GRU(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
```

Gated Recurrent Unit - Cho et al. 2014.

See the Keras RNN API guide for details about the usage of RNN API.

Based on available runtime hardware and constraints, this layer will choose different implementations (cuDNN-based or pure-TensorFlow) to maximize the performance. If a GPU is available and all the arguments to the layer meet the requirement of the CuDNN kernel (see below for details), the layer will use a fast cuDNN implementation.

The requirements to use the cuDNN implementation are:

1. `activation` == `tanh`
2. `recurrent_activation` == `sigmoid`
3. `recurrent_dropout` == 0

Bidirectional layer

`Bidirectional` class

```
tf.keras.layers.Bidirectional(
    layer, merge_mode="concat", weights=None, backward_layer=None, **kwargs
)
```

Bidirectional wrapper for RNNs.

Arguments

- **layer**: `keras.layers.RNN` instance, such as `keras.layers.LSTM` or `keras.layers.GRU`. It could also be a `keras.layers.Layer` instance that meets the following criteria:
  1. Be a sequence-processing layer (accepts 3D+ inputs).
  2. Have a `go_backwards`, `return_sequences` and `return_state` attribute (with the same semantics as for the `RNN` class).
  3. Have an `input_spec` attribute.
  4. Implement serialization via `get_config()` and `from_config()`. Note that the recommended way to create new RNN layers is to write a custom RNN cell and use it with `keras.layers.RNN`, instead of subclassing `keras.layers.Layer` directly.
- **merge_mode**: Mode by which outputs of the forward and backward RNNs will be combined. One of {'sum', 'mul', 'concat', 'ave', None}. If None, the outputs will not be combined, they will be returned as a list. Default value is 'concat'.

Keras

ConvLSTM2D layer

`ConvLSTM2D` class

```
tf.keras.layers.ConvLSTM2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    activation="tanh",
    recurrent_activation="hard_sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
```

Convolutional LSTM.

It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of n integers, specifying the dimensions of the convolution window.
- **strides**: An integer or tuple/list of n integers, specifying the strides of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: One of `"valid"` or `"same"` (case-insensitive).

Entonces ...

- Una LSTM es una de las soluciones más populares para resolver el problema de las dependencias largas en redes recurrentes.
- La idea básica consiste en crear un carrusel de memoria que mantenga, sin alterar, elementos del pasado que se necesita liberar en el futuro. Este carrusel se protege del exterior mediante compuertas que permiten ser selectivos en lo que entra y lo que sale. La celda puede incorporar también una compuerta de olvido que se ha mostrado importante en la práctica.
- Una LSTM se puede entrenar con BPTT igual que una red de Elman, sin embargo sincronizando adecuadamente las compuertas, es posible evitar el problema de los gradientes desvanecientes.
- En la práctica la capacidad de una LSTM de aprender dependencias largas se puede mejorar implementando capas recurrentes bi-direccionales.