REDES RECURRENTES

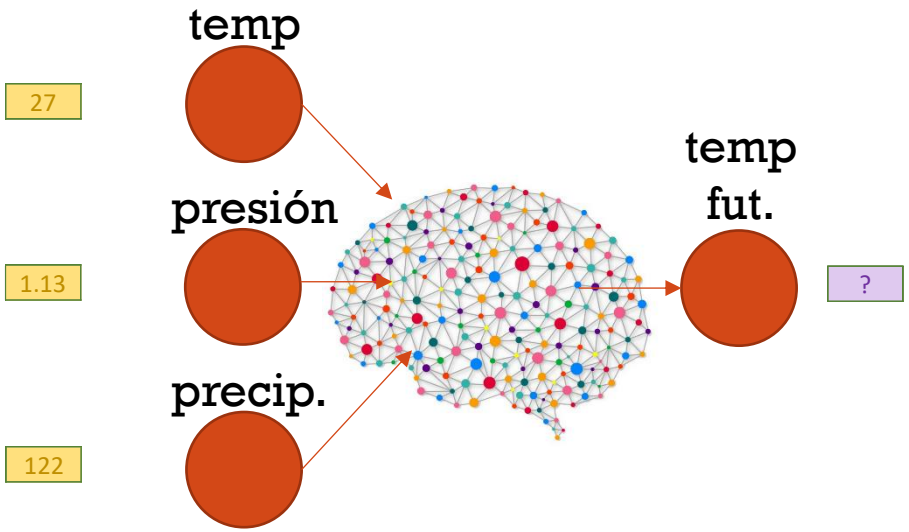
Profesor: Daniel Osorio

SECUENCIAS EN REDES NEURONALES CON MEMORIA

- Presentar cada dato secuencial uno a uno, en pasos diferentes ordenados

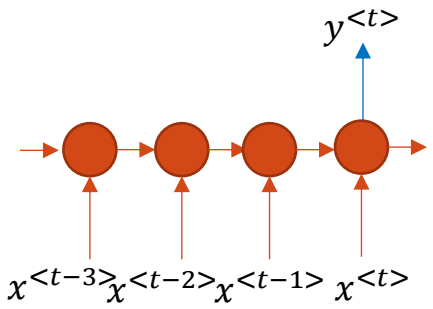
	<t-3>	<t-2>	<t-1>	<t>	<t+1>	<t+2>
temp	21	22	23	25	26	27
presión	1.01	0.98	1.06	1.08	1.11	1.13
precip.	95	102	99	112	118	122

- Cada batch incluirá instancias de secuencias del mismo largo

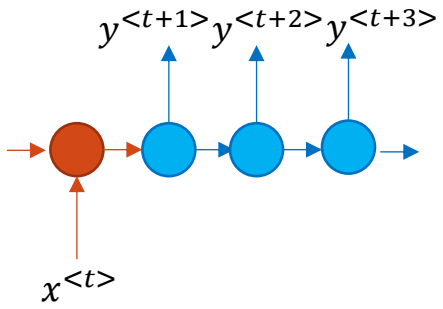


Modelo que recibe una secuencia de 4 pasos de tiempo para 3 variables de entrada y produce un pronóstico futuro

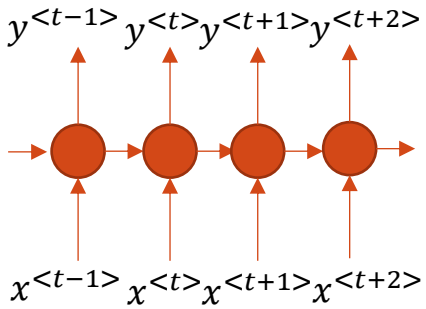
REDES RECURRENTES



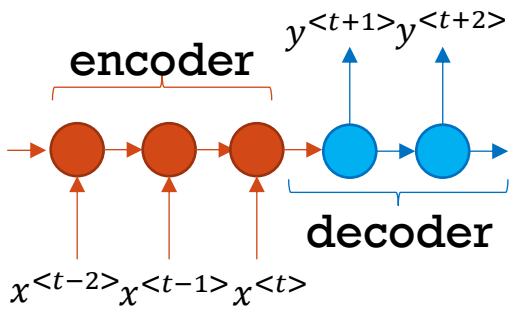
Secuencia a valor



Valor a secuencia (generación)



Secuencia a secuencia (mismo largo)



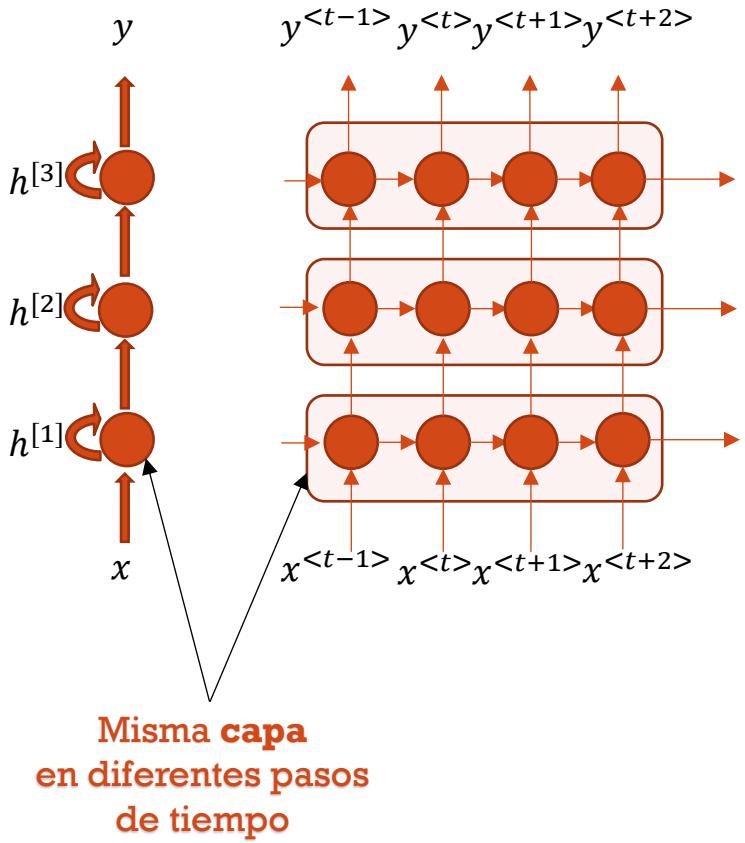
Secuencia a secuencia (largos diferentes, con delay)

Diferentes arquitecturas para diferentes tipos de problemas:

- Series de tiempo
- Videos
- Traducción de textos
- Chatbots
- Creación automática (texto, audio)
- Reconocimiento de habla
- Descripción de imágenes
- Reconocimiento de género musical
- ADN
- …

REDES RECURRENTES

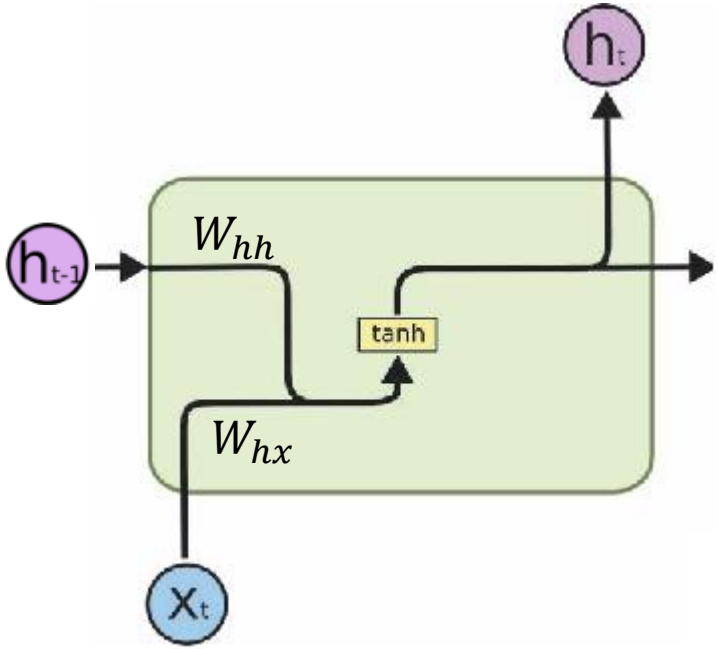
- Un modelo recurrente esta compuesto por capas recurrentes y por una capa de salida densa. La capa tiene neuronas conectadas consigo mismas y con las neuronas de la capa anterior.
- En un paso t, cada capa l recurrente:
  - Recibe los **inputs** del paso actual que vienen de la capa anterior, que notaremos $h^{[l-1]<t>}$, ($x^{<t>}$ en el caso de la primera capa)
  - Recibe los **estados** escondidas anteriores de la misma capa, que notaremos $h^{[l]<t-1>}$, que sirven como **memoria** de la red.
  - Produce y conserva como **activación** el estado actual $h^{[l]<t>}$, conectada con la misma neurona en el siguiente paso, y como **salida**
- La capa densa final produce una **salida** $y^{<t>}$ por cada paso de tiempo como predicción, y funciona como cualquier capa densa, con una matriz de pesos W_{yh} y un vector de sesgos b_y.
- Los parámetros de cada capa recurrente están compuestos por:
  - W_{hx}: modifica inputs del paso actual, para calcular los estados escondidos actuales $h^{<t>}$.
  - W_{hh}: modifica los estados $h^{<t-1>}$ anteriores, para calcular las activaciones escondidas actuales $h^{<t>}$.
  - b_h: vector de sesgos usado en el cálculo de h. Se puede descomponer en un sesgo de entrada b_{xh} y un sesgo de estados b_{hh}, tal que $b_h = b_{xh} + b_{hh}$



REDES RECURRENTES – SIMPLE RNN

SimpleRNN: Se trata de un modelo de RNN en donde $y^{<t>} = h^{<t>}$:

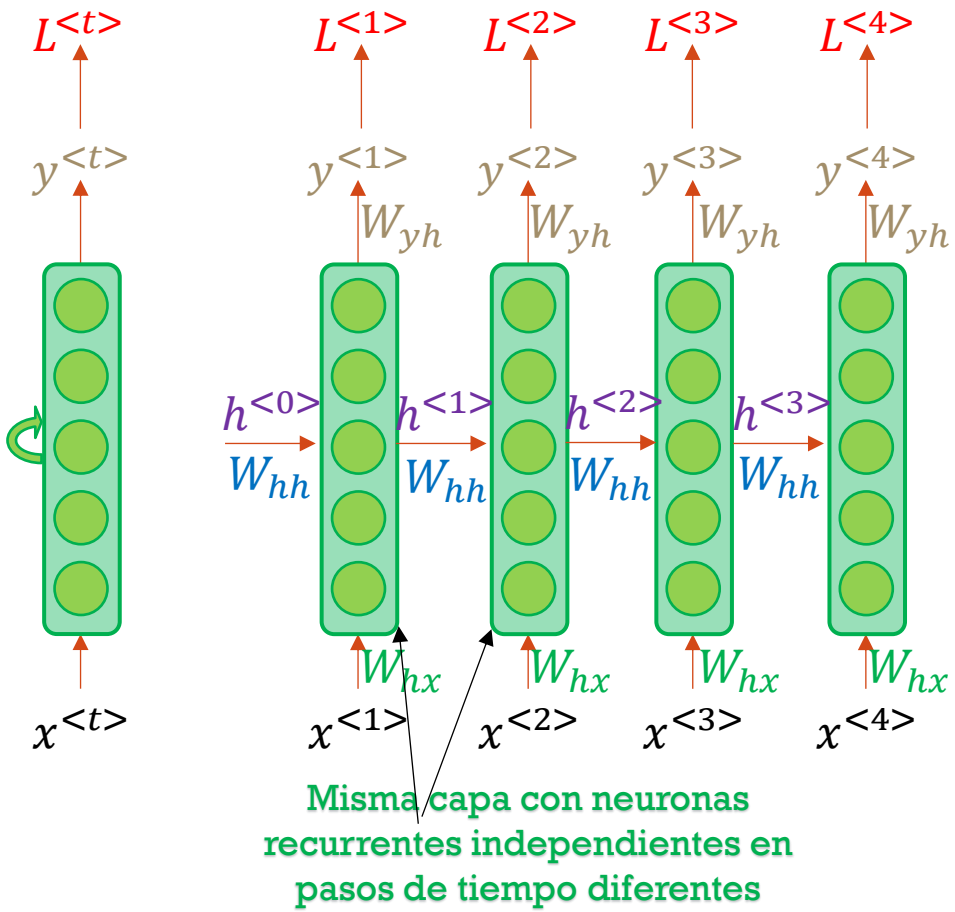
- $h^{<t>} = g(W_{hx} * x^{<t>} + W_{hh} * h^{<t-1>} + b_h)$
- Parámetros:
  - W_{hx}: matriz de pesos aplicada a los inputs
  - W_{hh}: matriz de pesos aplicada a las activaciones escondidas anteriores
  - b_h: sesgo
  - g: usualmente es **tanh**



BACKPROPACATION THROUGH TIME

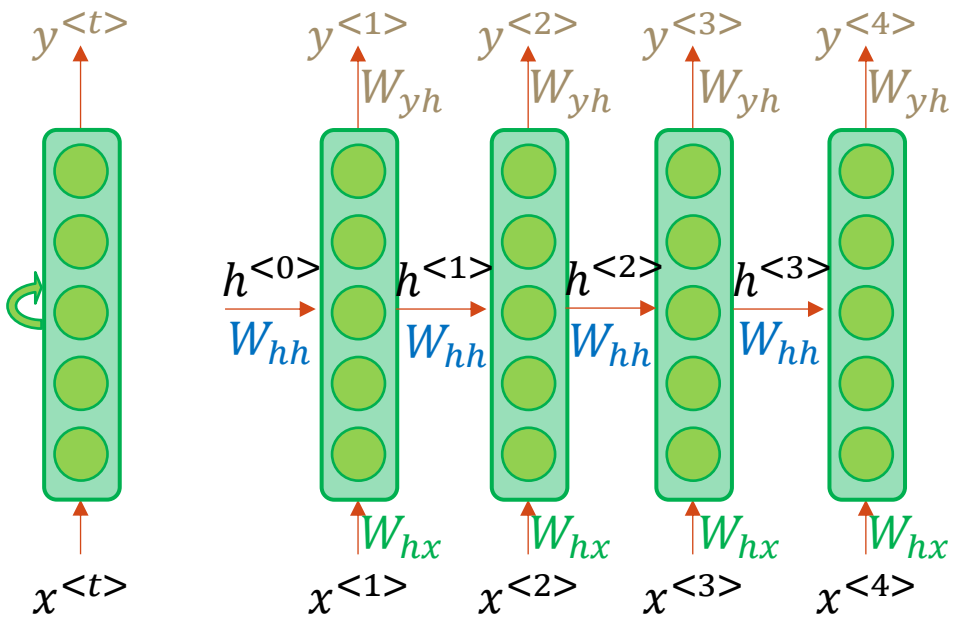
Profesor: Daniel Osorio

BACKPROPAGATION THROUGH TIME



- Cada capa de neuronas recurrentes:
  - Procesa la secuencia de entrada completa antes de entregar las salidas a la siguiente capa
  - Puede retornar la secuencia de salidas $y^{<1>}, y^{<2>}, \ldots, y^{<T>}$ procesada, o solo la última salida $y^{<T>}$.
- Parámetros (**independientes de los pasos de tiempo, reutilizados**):
  - W_{hx}: considera la entrada $x^{<t>}$ actual para calcular $h^{<t>}$
  - W_{hh}: considera la entrada $h^{<t-1>}$ anterior para calcular $h^{<t>}$
  - W_{yh}: considera la entrada $h^{<t>}$ actual para calcular $y^{<t>}$ (i.e. densa).
- Flujo de propagación temporal:
  - $h^{<t>} = g_1(W_{hh} * h^{<t-1>} + W_{hx} * x^{<t>} + b_h)$
  - Estado: tanh sobretodo
- Flujo de salida:
  - $y^{<t>} = g_2(W_{yh} * h^{<t>} + b_y)$
  - Salida: sigmoide o softmax (clasificación), o lineal (regresión)
- Se calcula el Loss $L^{<t>}$ por cada paso de tiempo y se agrega dependiendo de la tarea

BACKPROPAGATION THROUGH TIME



$$h^{<0>} = \vec{0}$$

$$h^{<t>} = g_1\left(\underbrace{\begin{bmatrix} W_{hh} & W_{hx} \end{bmatrix}}_{W_h} * \begin{bmatrix} h^{<t-1>} \\ x^{<t>} \end{bmatrix} + b_h \right)$$

#N capa L | #N capa L | #N capa L-1 | #N capa L-1 | #N capa L

$$y^{<t>} = g_2\left(\underbrace{W_{yh}}_{W_y} * h^{<t>} + b_y \right)$$

#N capa L | #N capa L | #N capa L | #N capa L

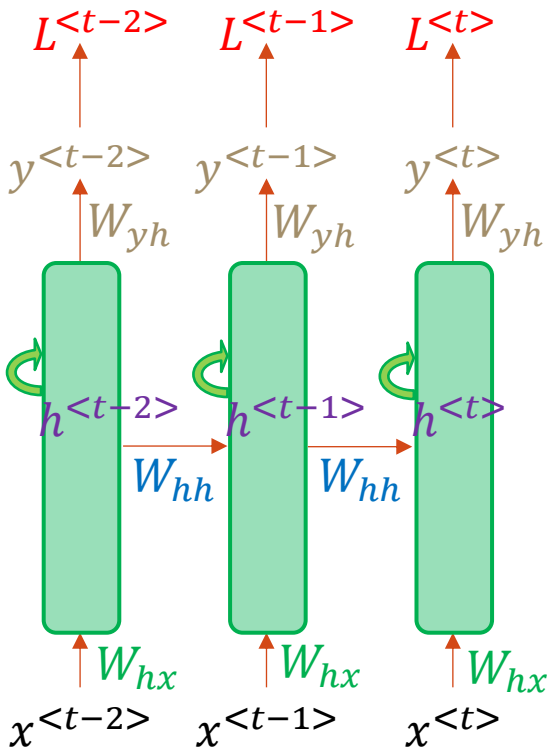
$$h^{[l]<t>} = g_1\left(\begin{bmatrix} whh^{[l]}_{11} & whh^{[l]}_{12} & \cdots & whh^{[l]}_{1k_l} \\ whh^{[l]}_{21} & whh^{[l]}_{22} & \cdots & whh^{[l]}_{2k_l} \\ \cdots & \cdots & \cdots & \cdots \\ whh^{[l]}_{k_l1} & whh^{[l]}_{k_l2} & \cdots & whh^{[l]}_{k_lk_l} \end{bmatrix} \begin{bmatrix} whx^{[l]}_{11} & whx^{[l]}_{12} & \cdots & whx^{[l]}_{1k_{l-1}} \\ whx^{[l]}_{21} & whx^{[l]}_{22} & \cdots & whx^{[l]}_{2k_{l-1}} \\ \cdots & \cdots & \cdots & \cdots \\ whx^{[l]}_{k_l1} & whx^{[l]}_{k_l2} & \cdots & whx^{[l]}_{k_lk_{l-1}} \end{bmatrix} \begin{bmatrix} h^{[l]<t-1>}_1 \\ h^{[l]<t-1>}_2 \\ \cdots \\ h^{[l]<t-1>}_k \\ x^{[l-1]<t>}_1 \\ x^{[l-1]<t>}_2 \\ \cdots \\ x^{[l-1]<t>}_{k_{l-1}} \end{bmatrix} + \begin{bmatrix} b^{[l]}_1 \\ b^{[l]}_2 \\ \cdots \\ b^{[l]}_{k_l} \end{bmatrix} \right)$$

Neuronas capa L | Neuronas capa L-1 | Neuronas de la capa L | #N de la capa L | #N de la capa L-1

BACKPROPAGATION THROUGH TIME

- Cada secuencia se trata en su totalidad antes de hacer el back propagation. Se calcula el loss $L^{<t>}(\hat{y}^{<t>}, y^{<t>})$ para cada paso de tiempo. El loss $L^{<t>}$ en el paso de tiempo t depende de las activaciones previas de todas las neuronas de las capas escondidas, de $h^{<1>}$ a $h^{<t>}$
- Dependiendo del tipo de tarea se puede agregar cada $L^{<t>}$, o considerar solo la última $L^{<T>}$.
- Se propagan los errores desde el último paso de tiempo hasta el primero, siguiendo el descenso de gradiente con las derivadas parciales de la perdida con respecto a los parámetros W_h, b_h, W_y, b_y → "Through time"
- Para los gradientes de W_y y b_y, se puede aplicar back propagation estático sincrónico tradicional.
- Para los gradientes de W_h y b_h, el cálculo es dinámico y asincrónico, consiste en sumar los gradientes de cada paso de tiempo.
- Todas las capas recurrentes reciben una secuencia como input, por lo que todas las capas recurrentes, excepto la última, envían la secuencia de outputs generados a la siguiente capa recurrente.

BACKPROPAGATION THROUGH TIME



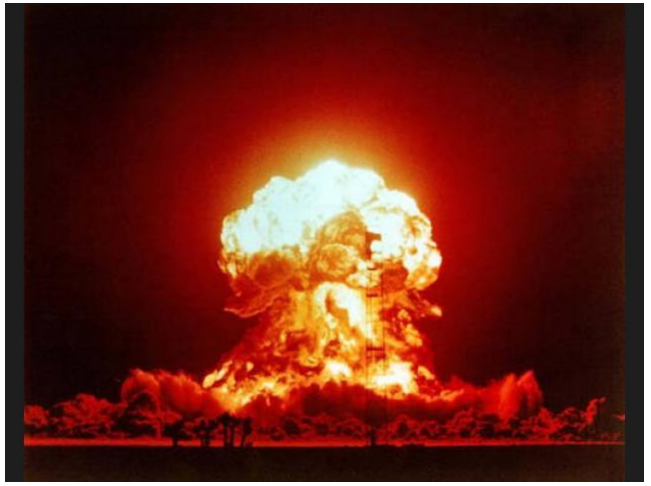
$$L = \sum_{t=1}^{T} L^{<t>}$$

$$\frac{\partial L^{<t>}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial L^{<t>}}{\partial y^{<t>}} \cdot \frac{\partial y^{<t>}}{\partial h^{<t>}} \cdot \left(\sum_{k=1}^{t} \frac{\partial h^{<t>}}{\partial h^{<k>}} \cdot \frac{\partial h^{<k>}}{\partial \boldsymbol{W}_{hh}} \right)$$

$$\frac{\partial h^{<t>}}{\partial h^{<k>}} = \prod_{i=k+1}^{t} \frac{\partial h^{<i>}}{\partial h^{<i-1>}}$$

Puede causar problemas de desvanecimiento o explosión del gradiente

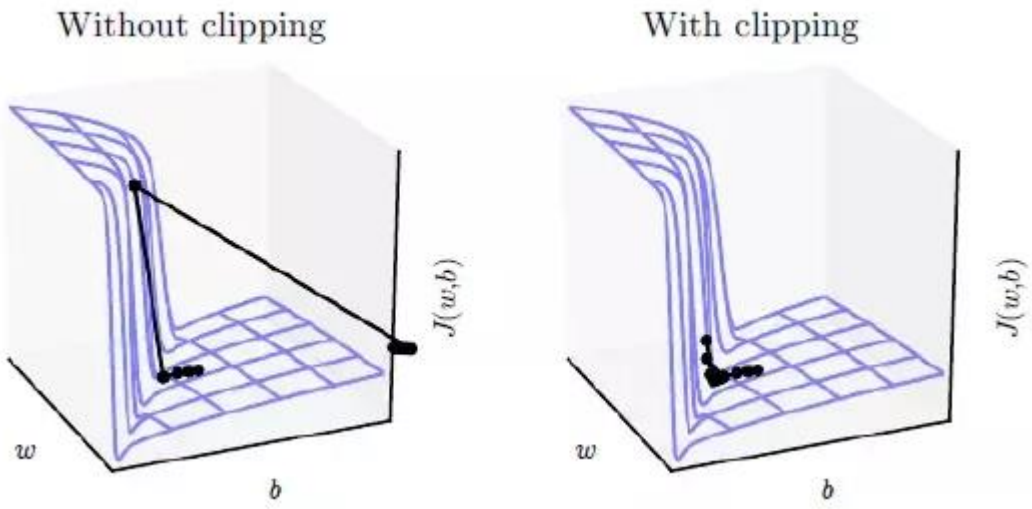
EXPLOSIÓN DE GRADIENTE



Gradient clipping:

- Las matrices de pesos son las mismas para cada secuencia, por lo que se reutilizan varias veces (multiplicaciones) al tratar cada paso de tiempo.
- Permite controlar la **explosión del gradiente** que ocurre cuando hay pendientes muy importantes (o muy pequeñas, respectivamente) en la superficie de optimización, situación común en las RNN
- **Limita la magnitud de la actualización** de los parámetros a partir del gradiente al propagar a través del tiempo durante el entrenamiento, definiendo un umbral

→ Estabiliza el aprendizaje de los modelos, pero se reduce la influencia de los primeros pasos de las secuencias

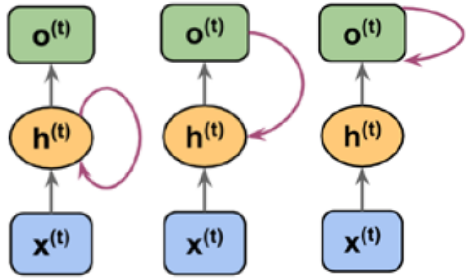


Ian Goodfellow et. al, MIT press, 2016

REDES RECURRENTES

Consideraciones de las RNN básicas

- Las secuencias sirven para representar las entradas y/o salidas de las RNNs
- Aplicaciones múltiples: forecasting, traducción, generación, …
- La arquitectura recurrente presentada es la más común (hidden to hidden) pero se pueden crear redes con otros tipos de recurrencia: output-to-hidden, output-to-output

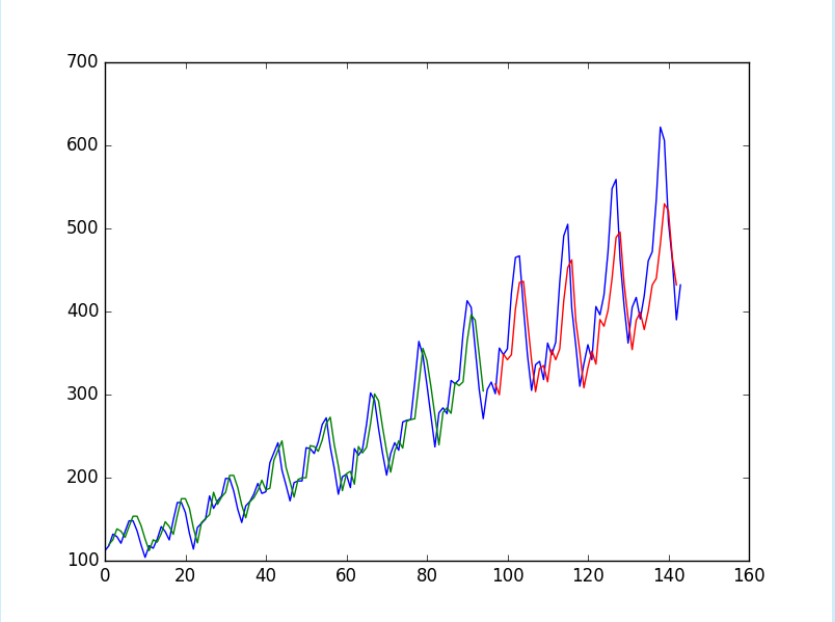


- Sufren de un posible **desvanecimiento** o **explosión del gradiente**:
  - Uso de **gradient clipping**
  - **TBPTT**: Truncated Back-Propagation through Time
  - Imposibilidad de capturar patrones y dependencias en las secuencias
  - Olvido de patrones de corto plazo en el largo plazo (memoria de corto plazo)

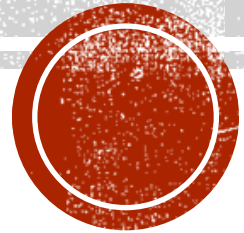
→ **Redes recurrentes con memoria de largo plazo (e.g. GRUs, LSTMs)**

FORECASTING CON RNN

Utilizar una red recurrente RNN para realizar forecasting univariado



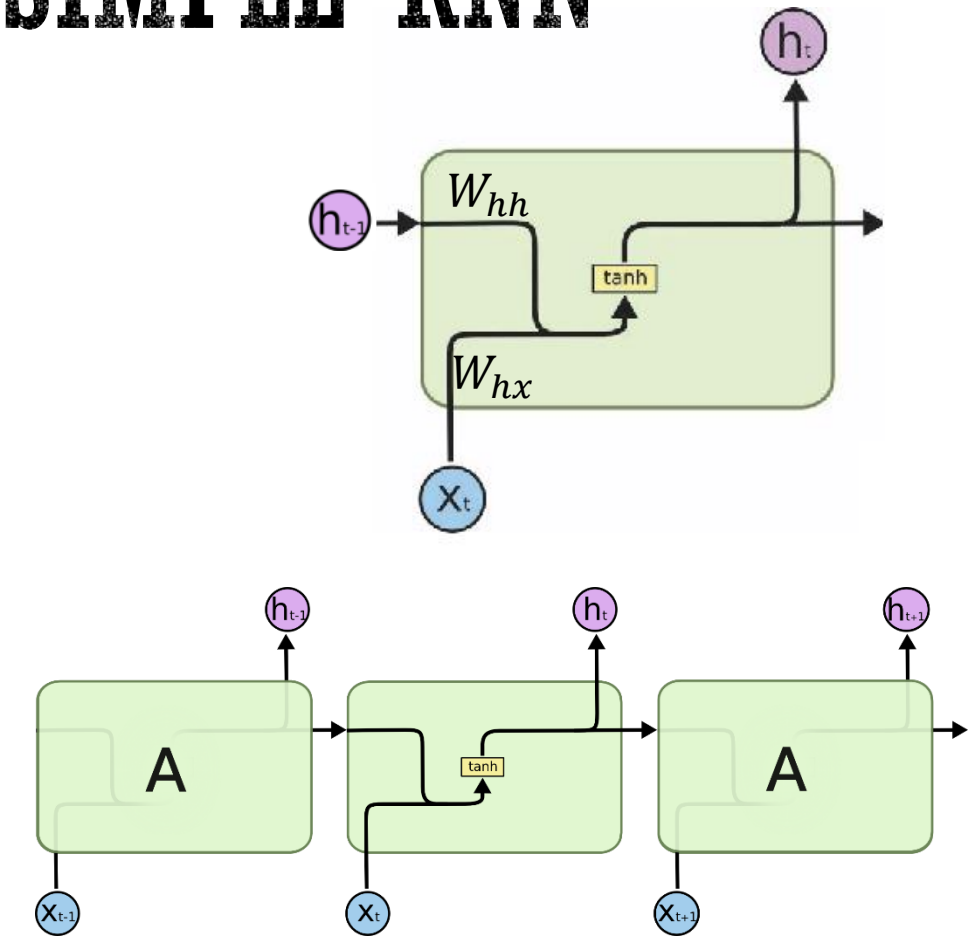
TIPOS DE REDES RECURRENTES



REDES RECURRENTES – SIMPLE RNN

SimpleRNN: Se trata de módulos recurrentes en donde $y^{<t>} = h^{<t>}$:

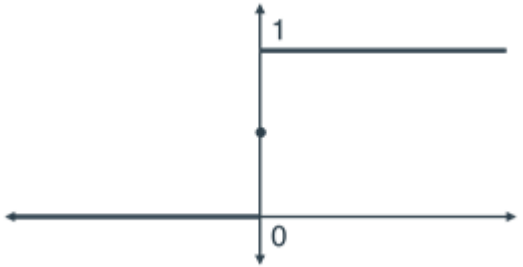
- $h^{<t>} = g(W_{hx} * x^{<t>} + W_{hh} * h^{<t-1>} + b_h)$
- Parámetros:
  - W_{hx}: matriz de pesos aplicada a los inputs
  - W_{hh}: matriz de pesos aplicada a las activaciones escondidas anteriores
  - b_h: sesgo
  - g: usualmente es tanh
- Las activaciones previas intervienen en la creación de las activaciones del siguiente paso, pero luego se pierden. No es posible memorizarlas a largo plazo.

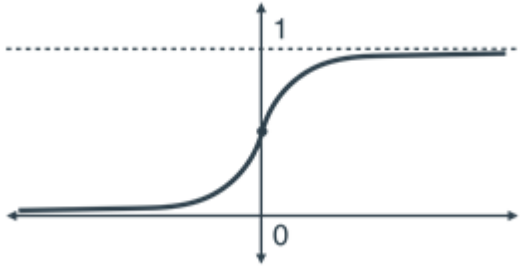


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

REDES RECURRENTES CON MEMORIA

- Las informaciones de estados anteriores lejanos influyen poco en las activaciones de estados futuros.
- Patrones de corto plazo encontrados son olvidados en el largo plazo.
- Secuencias largas implican desvanecimiento del gradiente en el entrenamiento
- Se define una célula o módulo (GRU o LSTM) con memoria, que se pueden apilar uno encima de otro.
- Utilización de **compuertas** (funciones sigmoides) para determinar si se toman o no decisiones de relevancia y conservación de informaciones pasadas y nuevas.

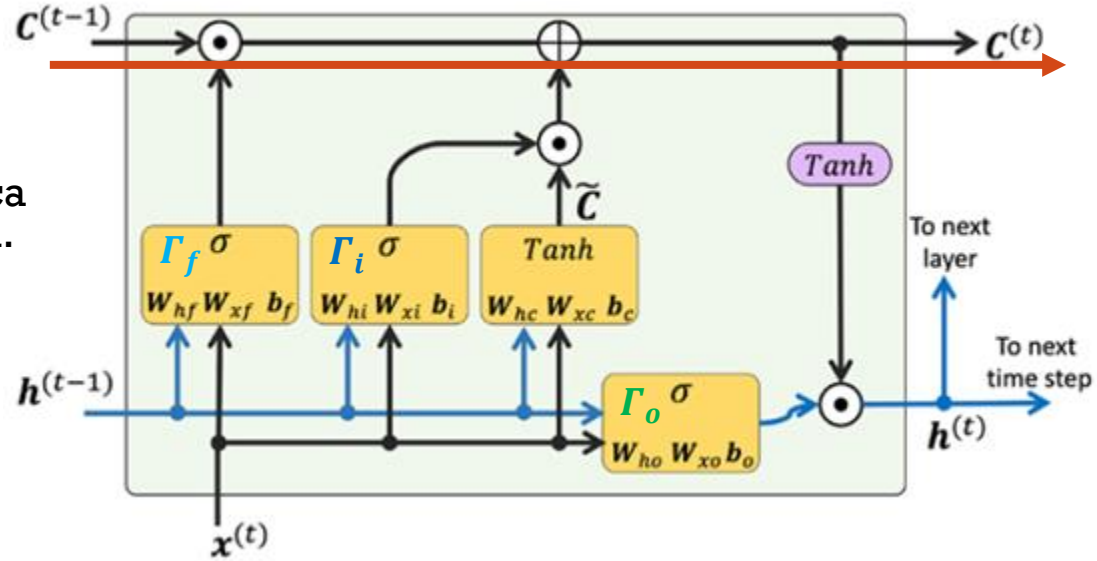




LSTM (LONG SHORT-TERM MEMORY)

LSTM: (Hochreiter & Schmidhuber, 1997):

- Células de memoria :
  - $h^{<t>}$: un estado escondido análogo al de los módulos SimpleRNN, que constituyen la salida de la célula, y que se comunican al siguiente paso de tiempo
  - $c^{<t>}$: un estado de célula (*cell state*), que solo se comunica al siguiente paso de tiempo, y que constituye la memoria. No tiene asociado una matriz de parámetros para controlarlo, de esta manera, no se incurre en problemas de desvanecimiento o explosión de gradiente, de tal manera que es una "autopista" de información.
  - Tres compuertas para controlar:
    - $\Gamma_f^{<t>}$: *forget gate*, influencia del estado de memoria anterior $c^{<t-1>}$ en su propia actualización ($c^{<t>}$)
    - $\Gamma_i^{<t>}$: *input gate*, influencia del nuevo estado de memoria propuesto $\tilde{c}^{<t>}$ en la actualización de la memoria $c^{<t>}$
    - $\Gamma_o^{<t>}$: *output gate*, intensidad del nuevo estado de memoria en la salida $h^{<t>}$



LSTM (LONG SHORT-TERM MEMORY)

LSTM: (Hochreiter & Schmidhuber (1997)):

- Se tienen 2 valores de memoria/estado a comunicar:
  - $\tilde{c}^{<t>}$: valor candidato de memoria de largo alcance, solo se pasa al siguiente paso de tiempo
  - Se combina $c^{<t-1>}$ con $\tilde{c}^{<t>}$, para obtener $c^{<t>}$, usando las compuertas $\Gamma_i^{<t>}$ y $\Gamma_f^{<t>}$
  - $h^{<t>}$: activación local de la neurona, se pasa al siguiente paso de tiempo y a otras neuronas de capas posteriores, controlada por $\Gamma_o^{<t>}$
- Se tienen compuertas de actualización con nueva información (input) $\Gamma_i^{<t>}$, pero también una compuerta de olvido (forget) $\Gamma_f^{<t>}$, lo que permite combinar ambos valores en la memoria
- Se incluye una compuerta de salida (output) $\Gamma_o^{<t>}$

$$\Gamma_f^{<t>} = \sigma(W_f * [h^{<t-1>}, x^{<t>}] + b_f)$$

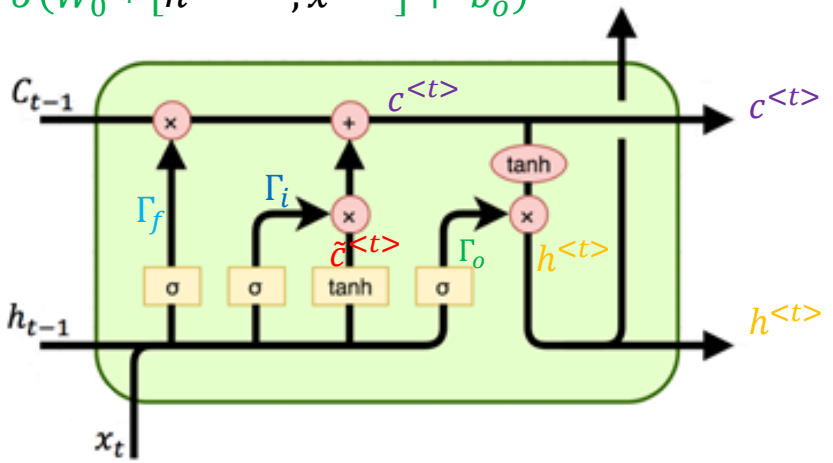
$$\Gamma_i^{<t>} = \sigma(W_i * [h^{<t-1>}, x^{<t>}] + b_i)$$

$$c^{<t>} = \Gamma_i^{<t>} * \tilde{c}^{<t>} + \Gamma_f^{<t>} * c^{<t-1>}$$

$$\tilde{c}^{<t>} = tanh(W_{\tilde{c}} * [h^{<t-1>}, x^{<t>}] + b_{\tilde{c}})$$

$$h^{<t>} = \Gamma_o^{<t>} * \tanh(c^{<t>})$$

$$\Gamma_o^{<t>} = \sigma(W_0 * [h^{<t-1>}, x^{<t>}] + b_o)$$



GRU (GATED RECURRENT UNITS) SIMPLE

GRUs (Cho, et al. (2014)):

En una RNN simple, la memoria del estado anterior de cada celda $h^{<t-1>}$ se olvida, al calcular el nuevo valor $h^{<t>}$, pero eventualmente el anterior podría ser más importante que el nuevo .

→Proponer un nuevo valor candidato de la memoria $\tilde{h}^{<t>}$ para eventualmente remplazar el valor de anterior $h^{<t-1>}$, si se dan las condiciones.

→Definir qué tan relevante es la anterior activación $h^{<t-1>}$ en el cálculo de la nueva memoria, a través de una "compuerta de relevancia" $\Gamma_r{}^{<t>}$.

→La decisión de sobreescribir el valor anterior lo va a dar una "compuerta de actualización" (update): $\Gamma_u{}^{<t>}$

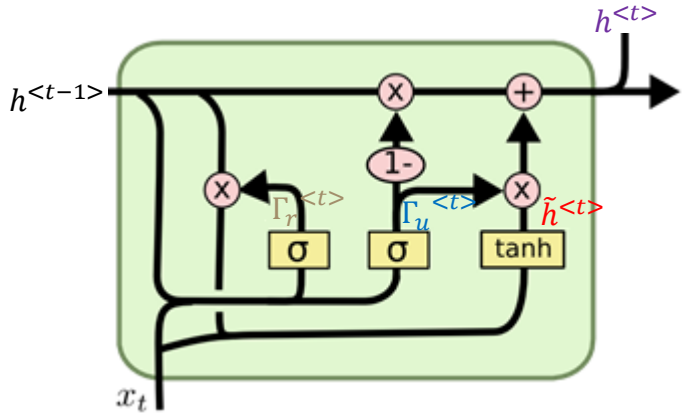
→Decidir el nuevo valor del estado de memoria $h^{<t>}$, considerando el estado candidato $\tilde{h}^{<t>}$ y el estado anterior $h^{<t-1>}$

$$\Gamma_r{}^{<t>} = \sigma(W_r * [h^{<t-1>}, x^{<t>}] + b_r)$$

$$\Gamma_u{}^{<t>} = \sigma(W_u * [\Gamma_r{}^{<t>} * h^{<t-1>}, x^{<t>}] + b_u)$$

$$h^{<t>} = \Gamma_u{}^{<t>} * \tilde{h}^{<t>} + (1 - \Gamma_u{}^{<t>}) * h^{<t-1>}$$

$$\tilde{h}^{<t>} = tanh(W_{\tilde{h}} * [h^{<t-1>}, x^{<t>}] + b_{\tilde{h}})$$



GRU VS LSTM

GRUs:

- $\tilde{h}^{<t>} = tanh(W_{\tilde{h}} * [h^{<t-1>}, x^{<t>}] + b_{\tilde{h}})$
- $\Gamma_r{}^{<t>} = \sigma(W_r * [h^{<t-1>}, x^{<t>}] + b_r)$
- $\Gamma_u{}^{<t>} = \sigma(W_u * [\Gamma_r{}^{<t>} * h^{<t-1>}, x^{<t>}] + b_u)$

- $h^{<t>} = \Gamma_u{}^{<t>} * \tilde{h}^{<t>} + (1 - \Gamma_u{}^{<t>}) * h^{<t-1>}$

LSTM:

- $\tilde{c}^{<t>} = tanh(W_{\tilde{c}} * [h^{<t-1>}, x^{<t>}] + b_{\tilde{c}})$

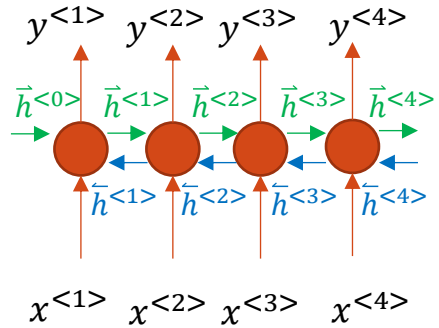
- $\Gamma_i{}^{<t>} = \sigma(W_i * [h^{<t-1>}, x^{<t>}] + b_i)$
- $\Gamma_f{}^{<t>} = \sigma(W_f * [h^{<t-1>}, x^{<t>}] + b_f)$
- $c^{<t>} = \Gamma_i{}^{<t>} * \tilde{c}^{<t>} + \Gamma_f{}^{<t>} * c^{<t-1>}$
- $\Gamma_o{}^{<t>} = \sigma(W_f * [h^{<t-1>}, x^{<t>}] + b_o)$
- $h^{<t>} = \Gamma_o{}^{<t>} * \tanh(c^{<t>})$

BIDIRECTIONAL

- Capas **bidireccionales** recurrentes: consideran el contexto
  - dado por los términos anteriores en la secuencia $\vec{h}^{<t>}$
  - dado por los términos posteriores en la secuencia $\overleftarrow{h}^{<t>}$
- La salida final depende de ambas direcciones

$$y^{<t>} = f_{act}\left(W_y[\vec{h}^{<t>}, \overleftarrow{h}^{<t>}] + b_y\right)$$

- Doble de parámetros
- Se necesita conocer la secuencia completa para poder aplicarlos (no servirían para reconocimiento de habla en tiempo real)
- Se pueden aplicar a RNN sencillos, GRUs y LSTMs, pero el concepto se extiende a los encoders de las arquitecturas de transformers (BERT: Bidirectional Encoder Representations from Transformers)



→ **memoria en doble sentido**

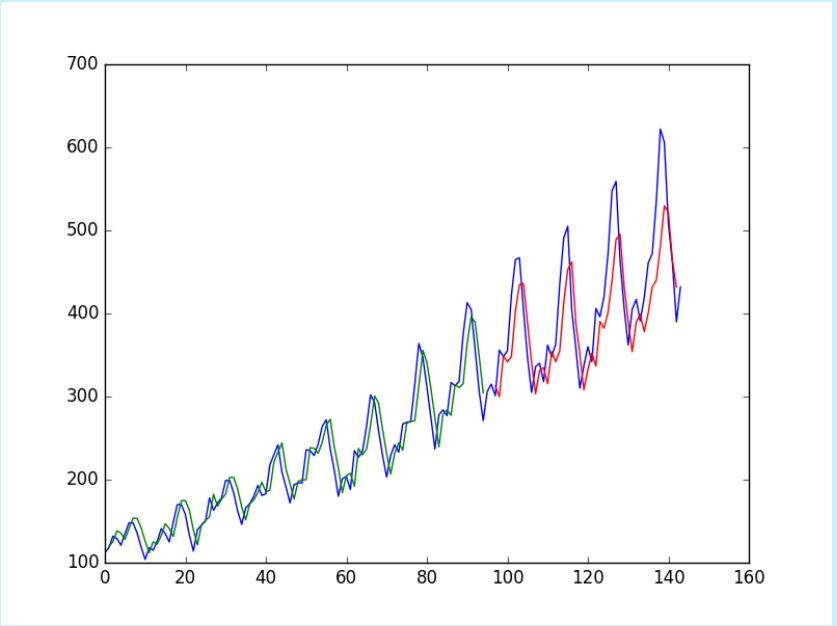
GRU VS LSTM

Consideración

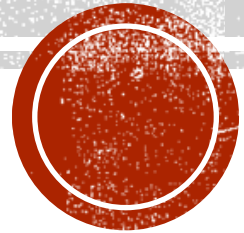
- Primero fueron las LSTMs (1997) que las GRUs (2014)
- Ninguno de los 2 es mejor que el otro,
  - Se han hecho estudios comparativos que no hay una mejor que la otra; en ciertos casos puede que GRU sea mejor que LSTM, en otros se puede encontrar el contrario.
  - Los GRUs son más simples y se escalan más fácilmente, tienen menos parámetros
  - Los LSTMs son más flexibles al tener 2 compuertas de memoria en vez de 1
  - Los LSTMs son más usados, al ser más antiguos
- La complejidad de redes con capas LSTM o GRU depende de la dimensión temporal (tamaño de las secuencias), por lo que no se empilan mas 2 o máximo 3 capas de estas.
- Un modelo de clasificación o regresión puede utilizar una primera recurrente, seguida de una parte final de redes fully connected

FORECASTING MULTIVARIADO CON RNN, LSTM Y GRU

Utilizar una red recurrente con módulos RNN, LSTM y GRU para realizar forecasting multivariado



GRACIAS



REFERENCIAS

- *Machine Learning with PyTorch and Scikit-Learn*, Sebastian Raschka, Yuxi Liu & Vahid Mirjalili, Packt, 2022
- *Dive into Deep Learning*, Aston Zhang, Zachary C. Lipton, Mu Li & Alexander J. Smola, https://d2l.ai/, 2022
- *Python Machine Learning (3rd ed.)*, Sebastian Raschka & Vahid Mirjalili, Packt, 2019
- *Neural Networks and Deep Learning,* Andrew Ng, Coursera, 2017
- *Deep Learning with Python*, Francois Chollet, Manning 2018
- *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, Aurélien Géron, 2017