

Tema 7: Recurrent Neural Networks

Recordatorio laboratorio

- ▶ Es la semana que viene
- ▶ No es obligatoria la asistencia
- ▶ Podéis entregarlo de forma individual o en grupo
- ▶ Podéis hacer el laboratorio con un grupo y luego entregarlo de forma individual, con ese grupo o con otro grupo

Recurrent Neural Networks

- ▶ Recurrent Neural Networks
 - ▶ Modelos del lenguaje con RNNs
 - ▶ Arquitecturas LSTM y GRU
- Semana 10
- Semana 11

Recurrent Neural Networks

- ▶ Tratan problemas sobre **secuencias**.
- ▶ Gran importancia en áreas como el procesamiento del lenguaje natural, traducción automática, *speech recognition*...

Sistemas Cognitivos Artificiales

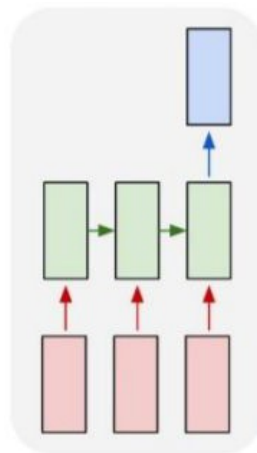
Roberto Casado Vara

Tema 7.1: Recurrent Neural Networks

Universidad Internacional de La Rioja

Recurrent Neural Networks

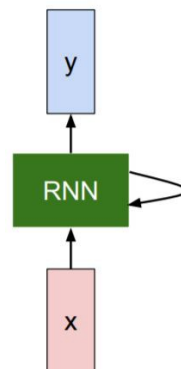
- ▶ Hasta ahora, hemos visto dos tipos de redes neuronales: feed-forward y CNNs. En ambas, tenemos un input de tamaño fijo (por ejemplo, una imagen) y una única salida.
- ▶ En muchos problemas de machine learning nos gustaría tener más flexibilidad, con arquitecturas que tengan una **longitud variable** de inputs y de outputs.
- ▶ Por ejemplo: representación **bag-of-words** de un texto vs **secuencia de palabras**.



Fuente de la imagen: Stanford CS231N

Recurrent Neural Networks

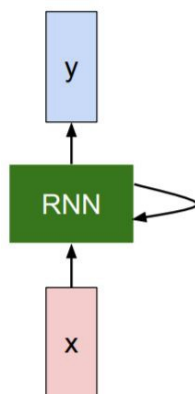
- ▶ Las **Recurrent Neural Networks (RNNs)**, o redes neuronales recurrentes, son un tipo de red neuronal capaz de trabajar con información secuencial.
- ▶ Mantienen un estado interno o **hidden state**, que podemos considerar como una memoria de lo visto hasta el momento.
- ▶ Aplican una fórmula recurrente sobre una secuencia de entrada de manera que, en cada paso de la secuencia, se depende del valor de input x de ese momento y del hidden state anterior h .
- ▶ Cada paso en la secuencia se conoce como ***time step***.



Fuente de la imagen: Stanford CS231N

Recurrent Neural Networks

- ▶ La potencia de las RNNs radica en su capacidad de **modelar relaciones temporales** entre elementos de la secuencia a través del estado interno de la red.
- ▶ Esto explica por qué este tipo de arquitectura ha tenido gran éxito en problemas de **lenguaje natural**, donde el contexto y el orden son determinantes.
- ▶ Las redes recurrentes son muy versátiles y pueden aplicarse a varios tipos de problema según queramos secuencias de entrada y/o de salida.



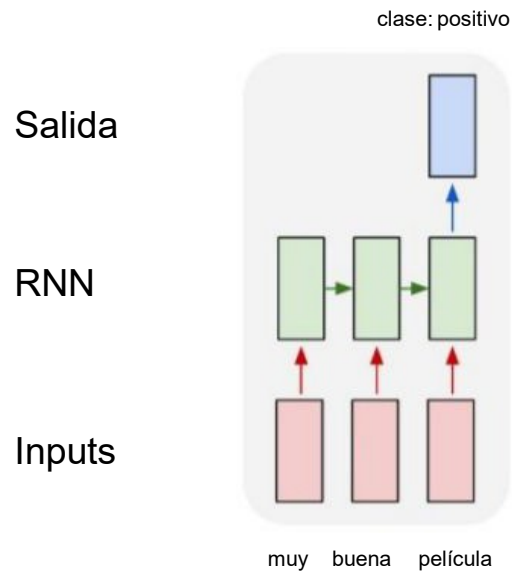
Fuente de la imagen: Stanford CS231N

Secuencia a única salida

Input: Secuencia

Output: Único output al final de la secuencia.

Ejemplo: Análisis de sentimiento de una oración o texto



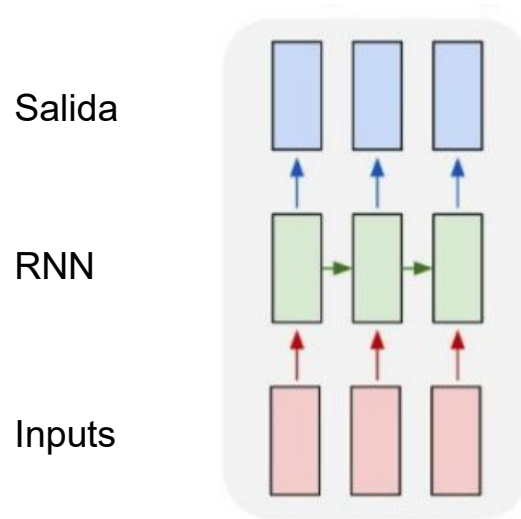
Fuente de la imagen: Stanford CS231N

Secuencia a secuencia

Input: Secuencia

Output: Secuencia con un valor de salida por cada elemento de entrada.

Ejemplo: Clasificación de imágenes frame a frame en un vídeo.



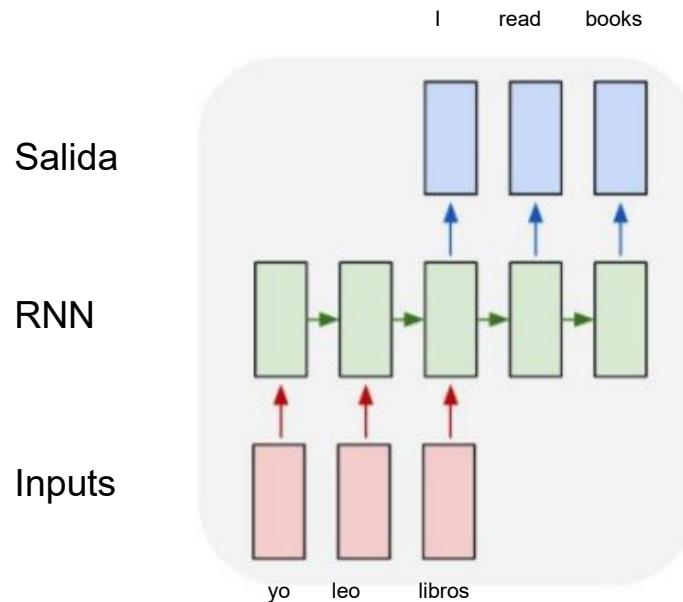
Fuente de la imagen: Stanford CS231N

Secuencia a secuencia (seq2seq)

Input: Secuencia

Output: Secuencia. Se espera a leer toda la secuencia de entrada antes de generar la secuencia de salida.

Ejemplo: Machine translation.



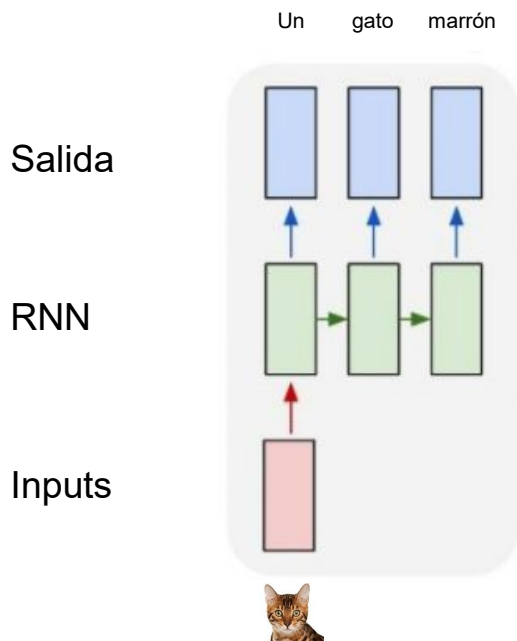
Fuente de la imagen: Stanford CS231N

Input único a secuencia

Input: Elemento único.

Output: Secuencia.

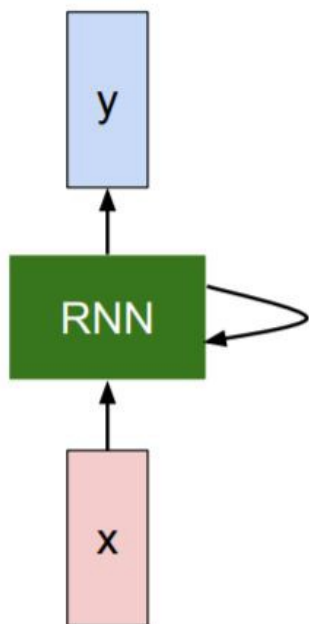
Ejemplo: Image captioning.



Fuente de la imagen: Stanford CS231N

Formulación de RNNs

- ▶ **RNN**: Toma una secuencia de valores de entrada y, de manera recurrente, aplica una transformación a partir de cada **valor de entrada** y del **hidden state** que posee la red en ese momento. Obteniendo:
 - Nuevo estado interno
 - (Opcional) Nuevo valor de salida



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

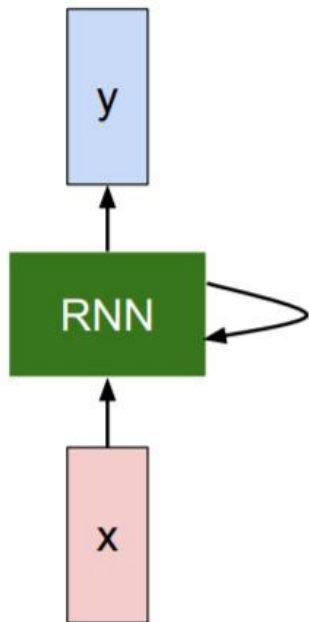
Función de recurrencia de una RNN

- h_t - (vector) hidden state en tiempo t . Su dimensión es un hiperparámetro.
- f_W - transformación no lineal con W los parámetros o weights de la RNN.
- x_t - (vector) input en tiempo t .

Fuente de la imagen: Stanford CS231N

Vanilla RNN

- ▶ **RNN:** Toma una secuencia de valores de entrada y, de manera recurrente, aplica una transformación a partir de cada valor de entrada y del **hidden state** que posee la red en ese momento. Obteniendo:
 - Nuevo estado interno
 - (Opcional) Nuevo valor de salida



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

Ecuaciones para una Vanilla RNN (RNN sencilla):

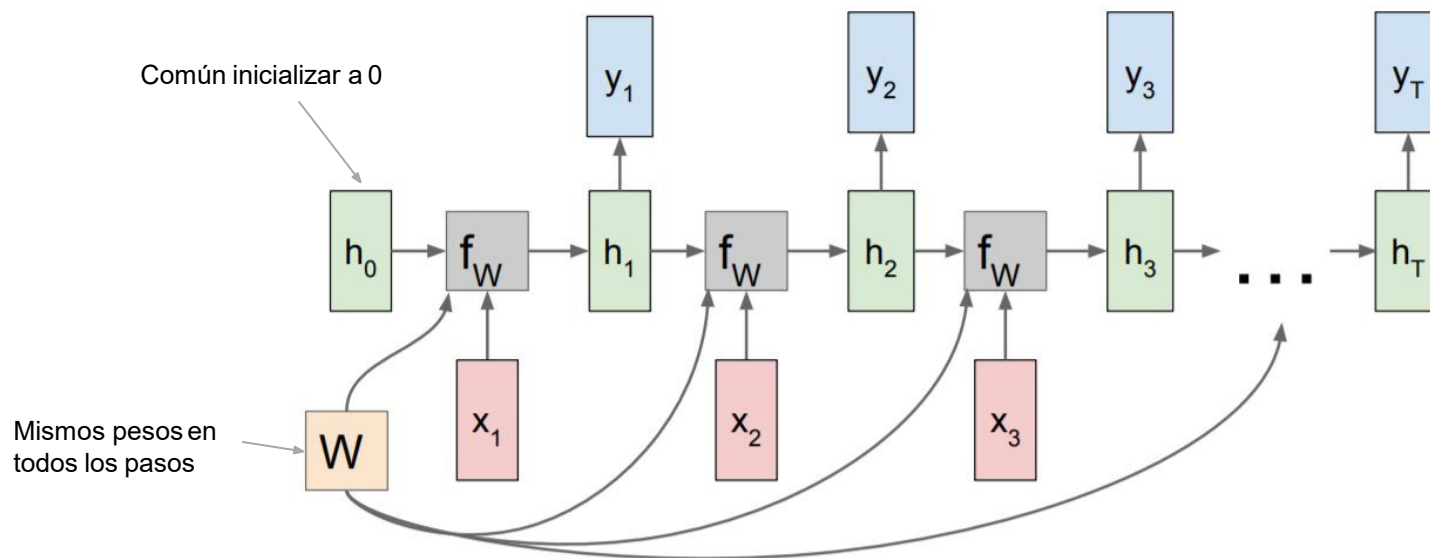
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

W_{hh} , W_{xh} , W_{hy} matrices de parámetros

Fuente de la imagen: Stanford CS231N

“Desenrollando” RNNs



Función de recurrencia:

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

Ecuaciones para una Vanilla RNN:

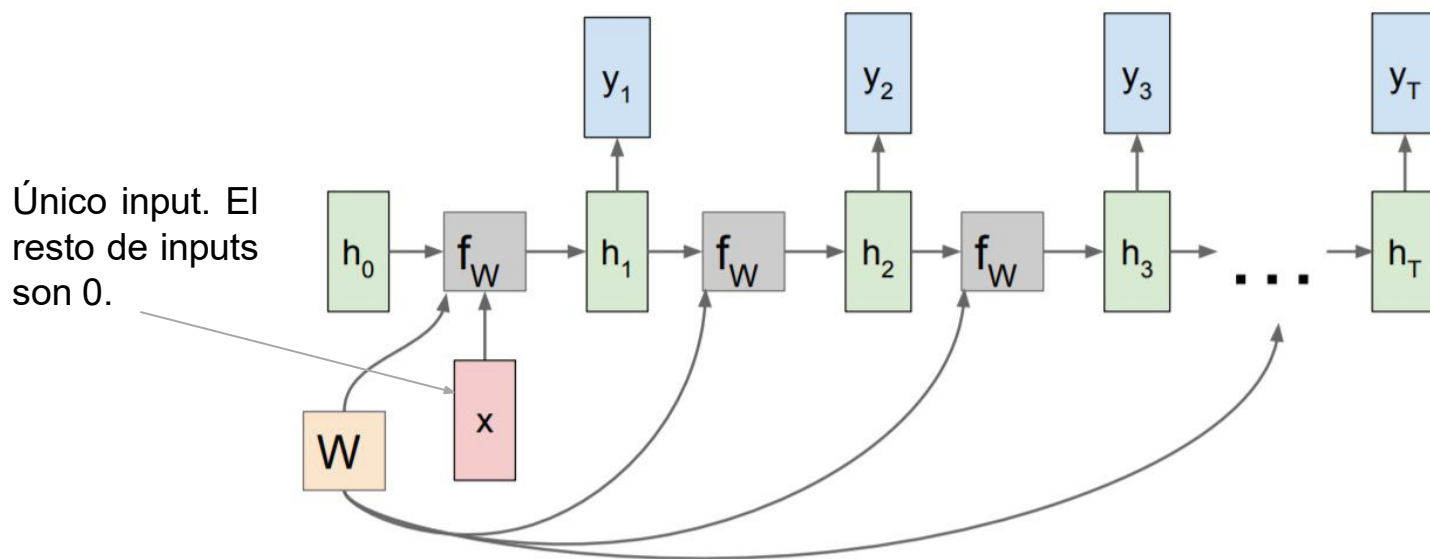
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

W_{hh}, W_{xh}, W_{hy} matrices de parámetros

Fuente de la imagen: Stanford CS231N

“Desenrollando” RNNs



Función de recurrencia:

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

Ecuaciones para una Vanilla RNN:

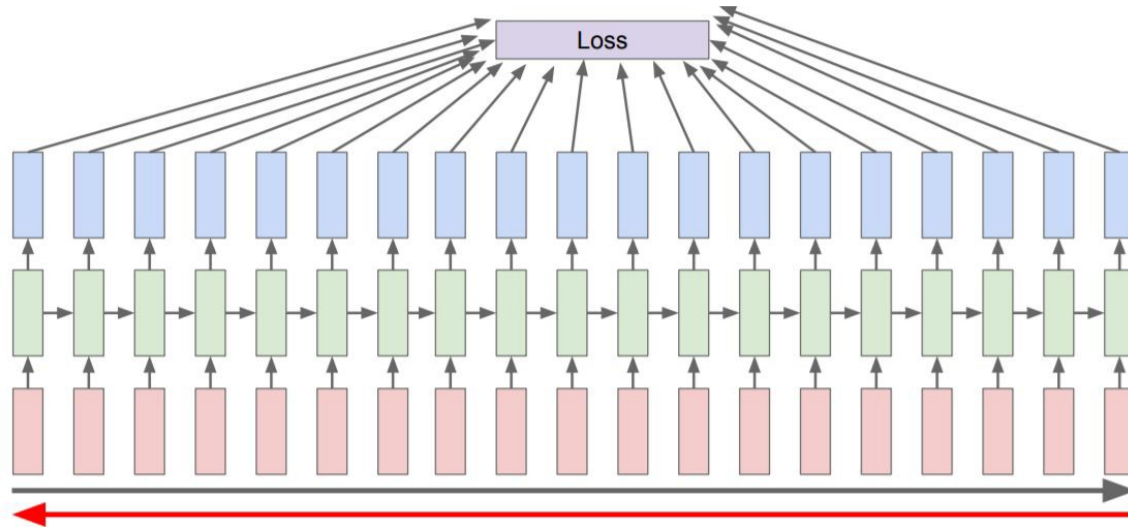
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

W_{hh} , W_{xh} , W_{hy} matrices de parámetros

Fuente de la imagen: Stanford CS231N

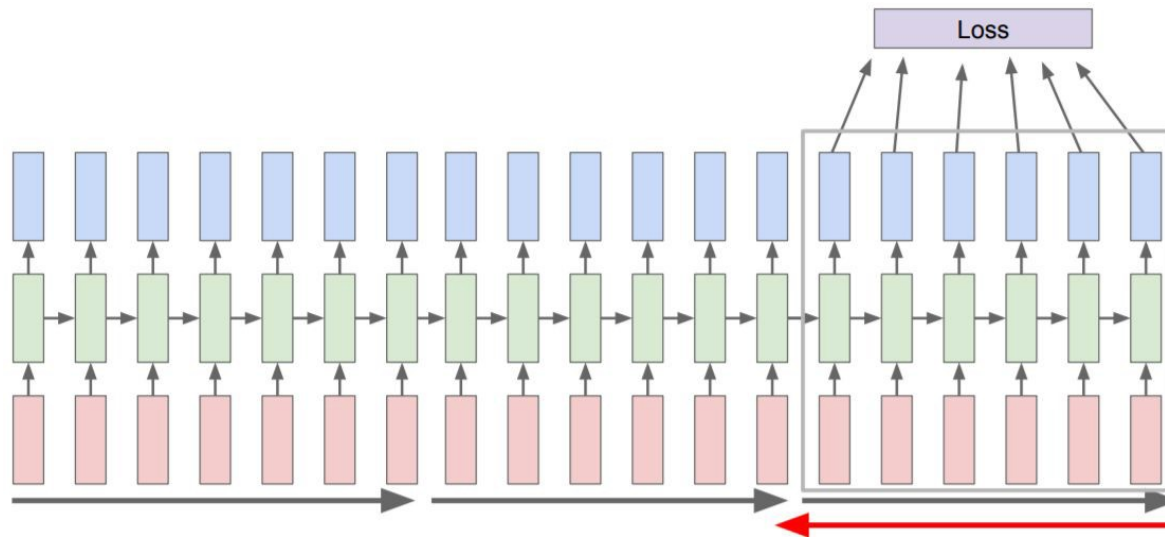
Backpropagation through time



- ▶ Es necesario “desenrollar” el grafo de computación a lo largo del tiempo y aplicar backpropagation a partir de todos los outputs que hemos obtenido.
- ▶ **Backpropagation through time (BPTT):**
 - Se hace forward pass a lo largo de toda la secuencia para calcular la loss a partir de todos los outputs.
 - Se hace backward pass a lo largo de toda la secuencia.

Fuente de la imagen: Stanford CS231N

Backpropagation through time



- ▶ BPTT es muy costoso para secuencias largas.
- ▶ En la práctica, se utiliza **Truncated Backpropagation through time**
- ▶ La secuencia se parte en trozos y se hace backpropagation en un pequeño número de steps, trozo a trozo.

Fuente de la imagen: Stanford CS231N

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 7.2: Modelos del lenguaje con RNNs

Universidad Internacional de La Rioja

Modelos del lenguaje

- ▶ Los **modelos del lenguaje**, o *language models*, son una aplicación de las redes recurrentes donde éstas han conseguido muy buenos resultados.
- ▶ Un modelo del lenguaje es un modelo que asigna una **probabilidad** a una secuencia de palabras (o de caracteres).


$$p(w_1, w_2, \dots, w_T)$$

- ▶ Estos modelos son muy útiles. Por ejemplo, en traducción automática se puede tener una serie de traducciones candidatas, con lo que podemos valernos de la probabilidad según el modelo para elegir la mejor traducción.

$$p(\text{"la casa es pequeña"}) > p(\text{"pequeña la casa es"})$$

Modelos del lenguaje



recurrent neural networks| 

recurrent neural networks
recurrent neural networks **in r**
recurrent neural networks **as weighted language recognizers**
recurrent neural networks **for language understanding**
recurrent neural networks **tutorial**
recurrent neural networks **quiz**
recurrent neural networks **time series**
recurrent neural networks **for emotion recognition in video**
recurrent neural networks **tensorflow**
recurrent neural networks **design and applications**

[Learn more](#)

Report inappropriate predictions

Modelos del lenguaje tradicionales

- ▶ Normalmente se calculan las probabilidades condicionando la probabilidad de que cada palabra aparezca después de las palabras anteriores. Se asume independencia y se aproxima a sólo unas pocas palabras anteriores:

$$P(w_1, \dots, w_m) = \prod_{i=1}^m P(w_i | w_1, \dots, w_{i-1}) \approx \prod_{i=1}^m P(w_i | w_{i-(n-1)}, \dots, w_{i-1})$$

- ▶ Las probabilidades se estiman contando en el texto cuántas veces aparecen las palabras juntas.

$$p(w_2 | w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \quad p(w_3 | w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

- ▶ Este modelo sencillo es muy útil y funciona correctamente. Sin embargo, para alcanzar buenas calidades se hace computacionalmente muy costoso.

Modelos del lenguaje con RNNs

- ▶ Un modelo del lenguaje puede tratarse de manera natural con recurrent neural networks.
- ▶ Podemos suministrar una secuencia de palabras a nuestra red recurrente, de manera que la red nos dé una distribución de probabilidad de la palabra siguiente.
- ▶ El **hidden state** de la RNN codifica en cierta medida el texto visto hasta ahora, por lo que la red es capaz de condicionar la probabilidad de la siguiente palabra a lo visto anteriormente.
- ▶ El input de cada time step puede ser un *word vector* o una representación *one-hot*.
- ▶ La salida en cada time step viene dada por una capa softmax con las probabilidades de que cada palabra del vocabulario sea la siguiente en la secuencia.

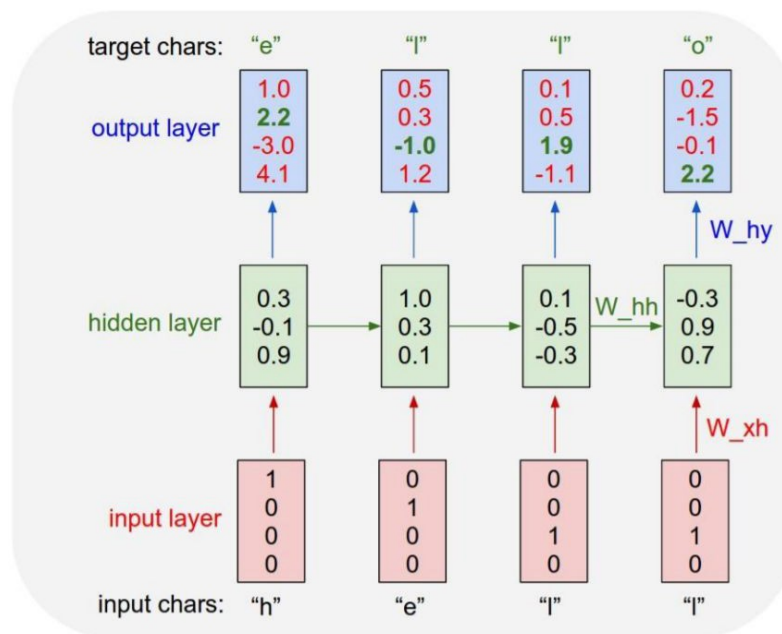
Modelos del lenguaje con RNNs

► **Ejemplo:** Modelo del lenguaje con caracteres.

► Vocabulario de 4 caracteres: [h, e, l, o]

► **Input:** representaciones *one-hot*

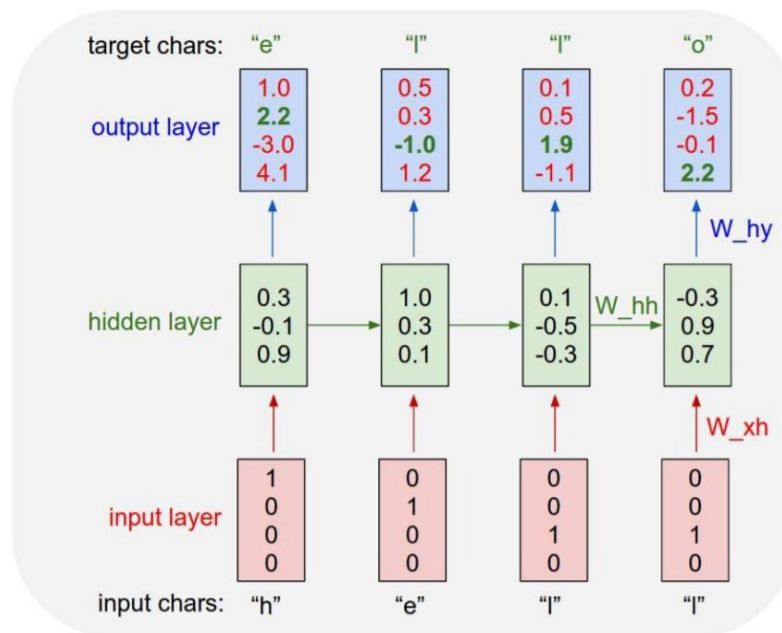
- h: [1, 0, 0, 0]
- e: [0, 1, 0, 0]
- l: [0, 0, 1, 0]
- o: [0, 0, 0, 1]



Fuente de la imagen: Stanford CS231N

Modelos del lenguaje con RNNs

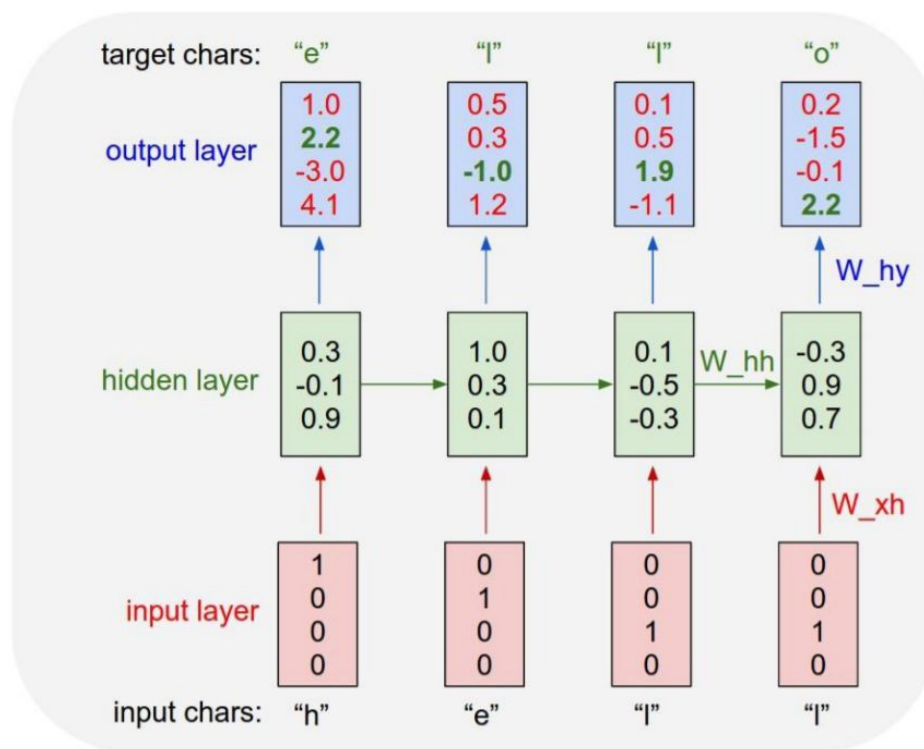
- ▶ **Ejemplo:** Modelo del lenguaje con caracteres.
- ▶ Vocabulario de 4 caracteres: [h, e, l, o]
- ▶ **Output:** softmax sobre los 4 posibles caracteres.



Fuente de la imagen: Stanford CS231N

Modelos del lenguaje con RNNs

- ▶ **Ejemplo:** Modelo del lenguaje con caracteres.
- ▶ Vocabulario de 4 caracteres: [h, e, l, o]
- ▶ **Entrenamiento:** secuencias de texto. Se intenta predecir el siguiente carácter.



Fuente de la imagen: Stanford CS231N

Modelos del lenguaje con RNNs

Input layer dim: 4

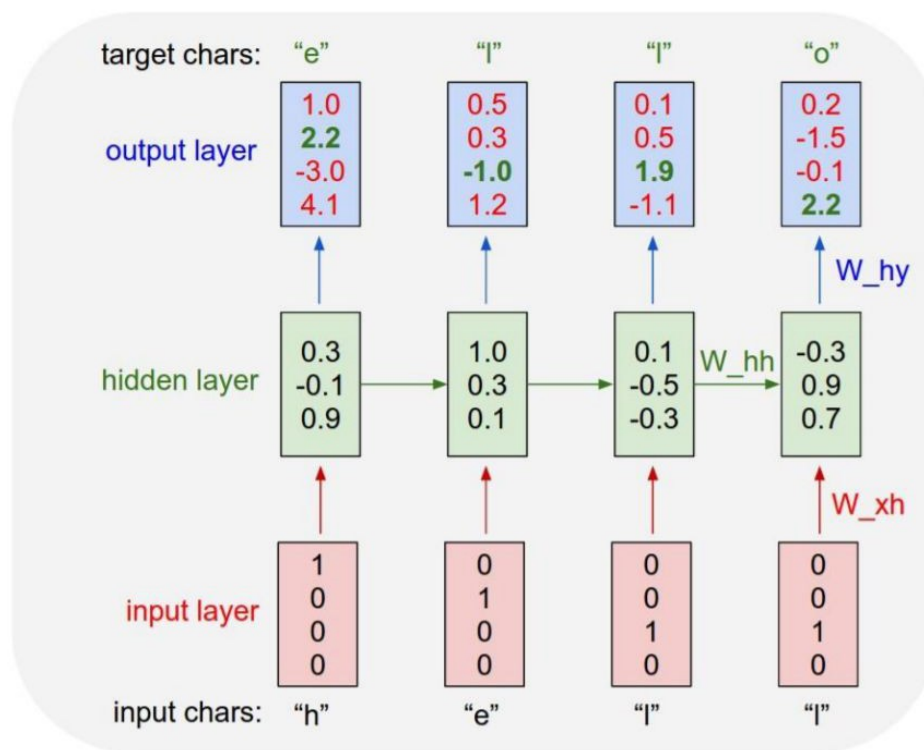
Hidden layer dim: 3

Output layer dim: 4

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

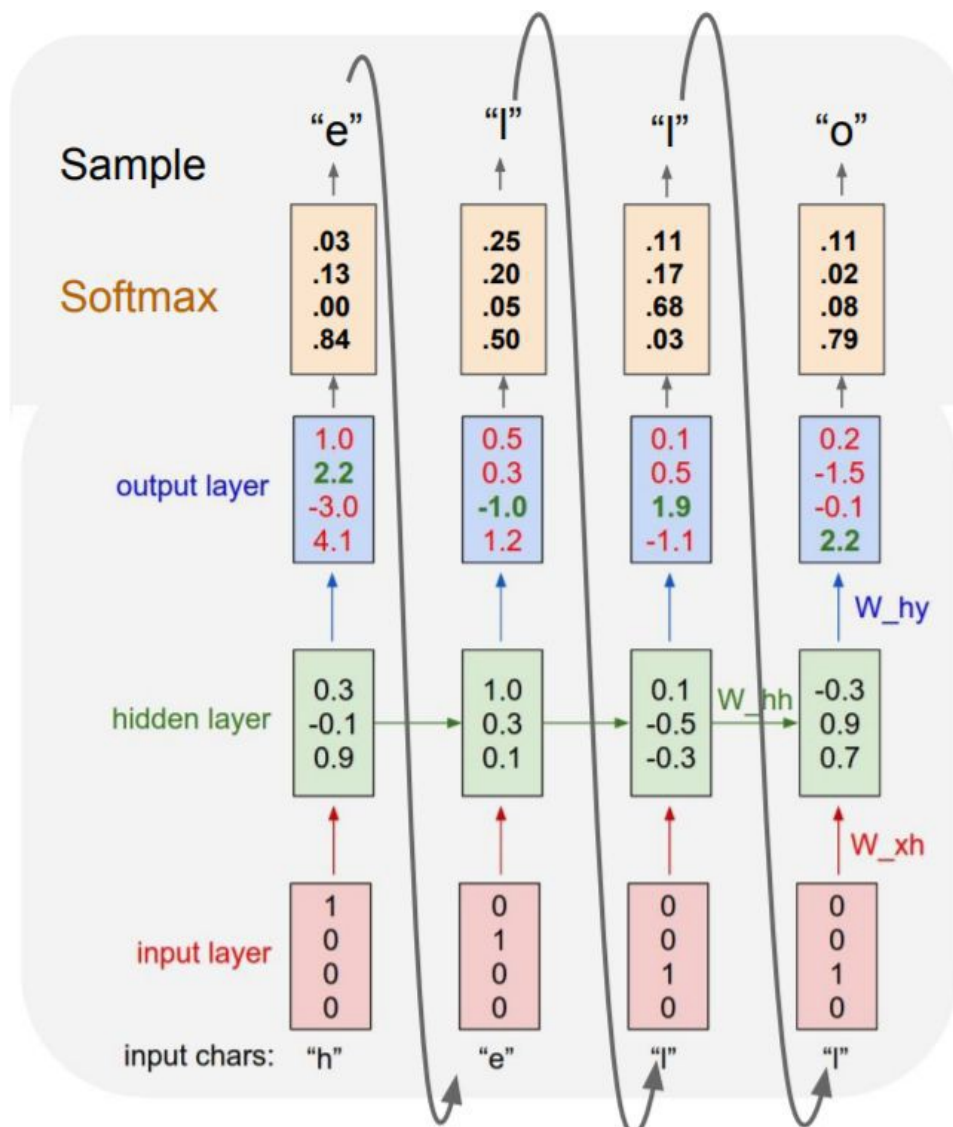
$$y_t = W_{hy}h_t$$

W_{hh} , W_{xh} , W_{hy} matrices de parámetros



Fuente de la imagen: Stanford CS231N

Inferencia: modelo generativo



Podemos utilizar el modelo del lenguaje como un **modelo generativo** y utilizarlo para generar texto.

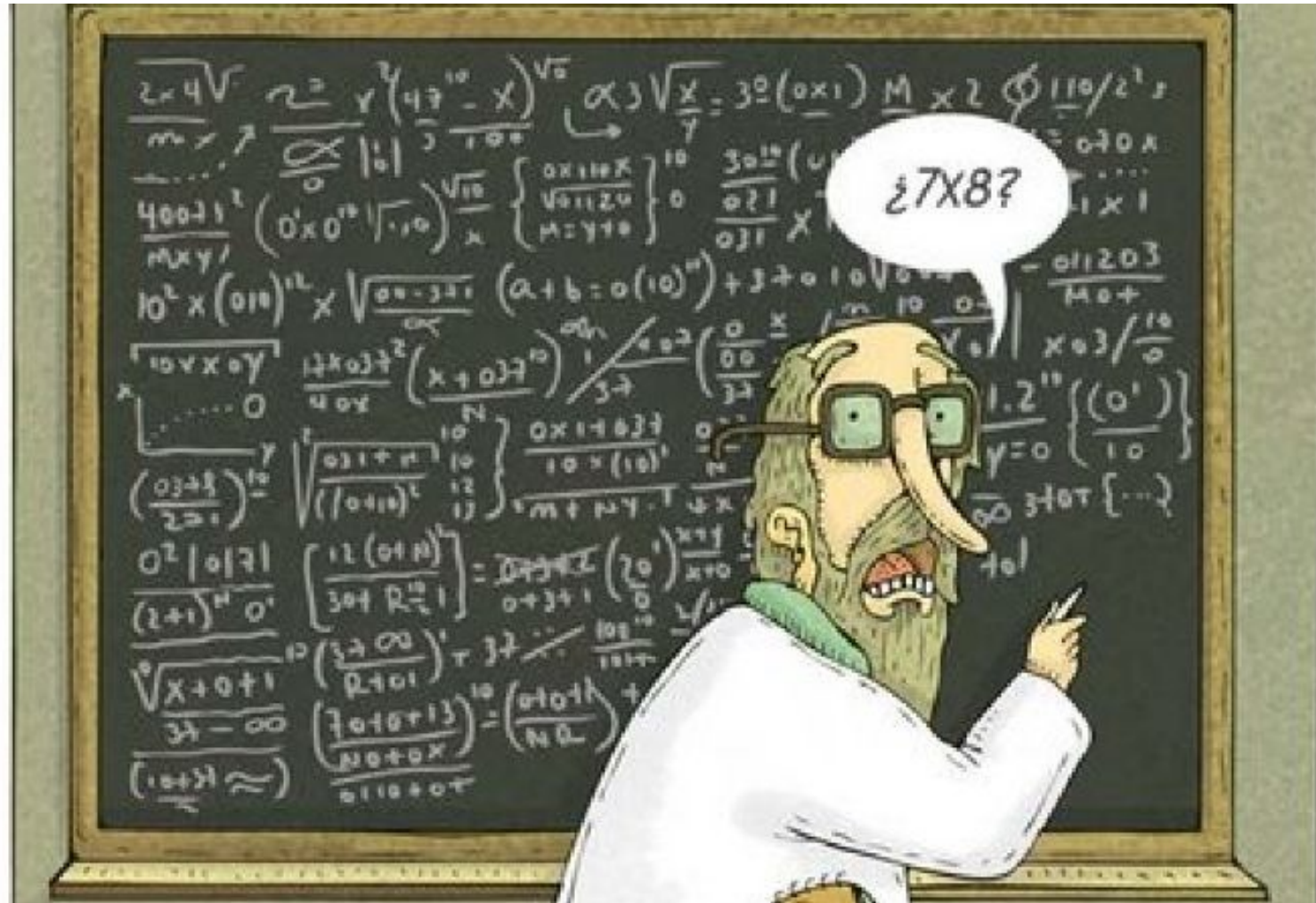
A partir de las probabilidades de la capa softmax, **muestreamos el siguiente carácter** a utilizar.

Hacer un muestreo aleatorio, en vez de sacar el carácter con la mayor probabilidad directamente, nos permite obtener **resultados diferentes** cada vez que usamos el modelo.

El carácter de output de cada time step se convierte en el input del siguiente.

Fuente de la imagen: Stanford CS231N

¿Dudas?



Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 7.3: Arquitecturas LSTM y GRU

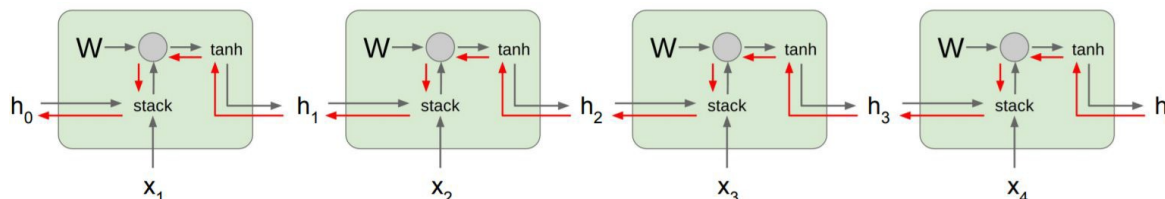
Que vamos a ver hoy

- ▶ Arquitecturas LSTM y GRU
- ▶ Práctica modelos LSTM para texto
- ▶ Práctica modelos LSTM para series temporales

Vanishing y exploding gradients

- ▶ Hemos visto redes recurrentes en su variante más sencilla, la **vanilla RNN**. Este tipo de RNNs tiene una serie de problemas que ha llevado a la creación de nuevas arquitecturas más efectivas y fáciles de entrenar.
- ▶ Dos problemas muy típicos en el entrenamiento de RNNs son los problemas de **exploding gradients** y **vanishing gradients**.
- ▶ **Vanishing gradients**: los gradientes “se desvanecen” y van a 0.
- ▶ **Exploding gradients**: los gradientes “explotan” y se hacen muy grandes.
- ▶ Vienen del hecho de tener que hacer backpropagation sobre una secuencia larga de elementos.
- ▶ Hacen que el entrenamiento se vuelva inestable o la red no entrene directamente.

Vanishing y exploding gradients



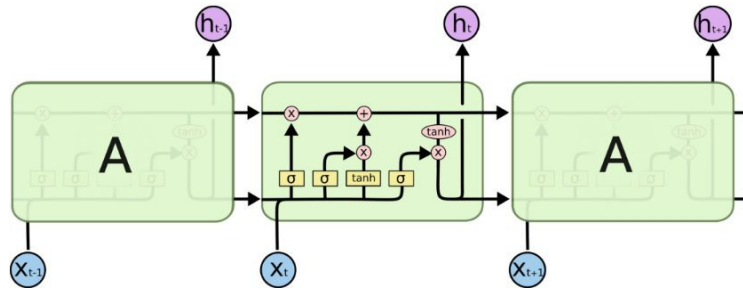
- ▶ El problema viene de la multiplicación continua por los elementos de la matriz W de pesos al hacer backpropagation a través de la secuencia.
- ▶ Intuitivamente y, de manera simplificada, el problema es similar a lo que pasa cuando multiplicamos un número por sí mismo en muchas ocasiones:
 - Si es mayor que 1, obtenemos valores cada vez más grandes: *exploding gradient*
 - Si está entre 0 y 1, obtenemos valores más cercanos a 0: *vanishing gradient*.
- ▶ Un truco práctico para solucionar *exploding gradients* (pero no *vanishing gradients*) es utilizar **gradient clipping**. Se mide la norma de los gradientes al hacer backpropagation en cada time step y, si es muy grande, se divide el gradiente por una constante.

Fuente de la imagen: Stanford CS231N

LSTM

- ▶ **LSTM: Long Short-Term Memory**. Datan de 1997.
- ▶ En la práctica, las redes recurrentes simples presentan problemas para aprender relaciones entre elementos en *time steps* muy separados, debido a factores como los *vanishing y exploding gradients*.
- ▶ Estos problemas hacen que parte del potencial de las redes recurrentes se pierda.
- ▶ Las **LSTM** se diseñaron para solucionar este problema de “memoria” de las **vanilla RNN**.
- ▶ Mantienen un estado interno adicional, el **cell state** (c_t), aparte del tradicional **hidden state** (h_t).

LSTM



Vanilla RNN

LSTM

$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

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

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

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

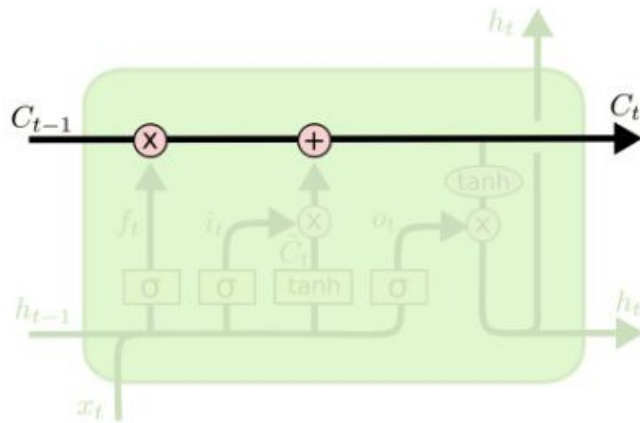
i, f, o son vectores con valores entre 0 y 1. Actúan de “puertas”

Cuatro vectores distintos que nos dan los valores de c_t y de h_t : i, f, o, g .

- f se conoce como **forget gate** y, al ser multiplicada por c_{t-1} , nos dice cuánto tenemos que olvidar del estado anterior.
- i se conoce como **input gate** y representa cuánto hemos de escribir en c_t a partir de lo visto en g (sin nombre en especial)
- c_t se obtiene como una mezcla de “cuánto recordamos del pasado” ($f \cdot c_{t-1}$) y “cuánta información nueva queremos añadir” ($i \cdot g$).
- o se conoce como **output gate** y nos dice cuánto de nuestro estado interno c_t hemos de revelar en el nuevo *hidden state*.

Fuente de la imagen: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTM



Vanilla RNN

$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

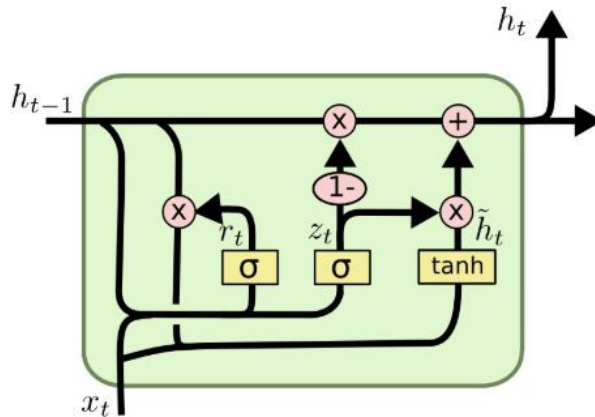
LSTM

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

- ▶ Las LSTM no sufren de *vanishing* o *exploding gradients*.
- ▶ La solución a estos problemas viene del estado interno c_t , que crea una especie de “autopista de la información” que ayuda durante backpropagation.

Fuente de la imagen: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

GRU



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

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

- ▶ **GRU - Gated Recurrent Unit.** Introducida en 2014.
- ▶ Tiene un sistema similar de *gates* al visto en la LSTM.
- ▶ Combina el cell state y el hidden state en un solo elemento.

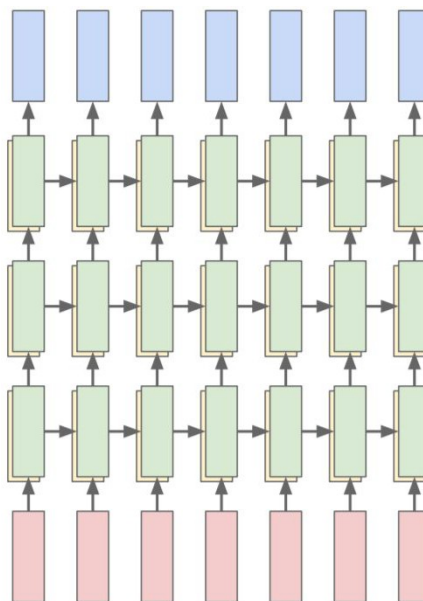
Fuente de la imagen: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Consideraciones prácticas

- ▶ Lo más común es utilizar directamente LSTMs o GRUs.
- ▶ Mucho mejor comportamiento en la práctica que Vanilla RNNs.

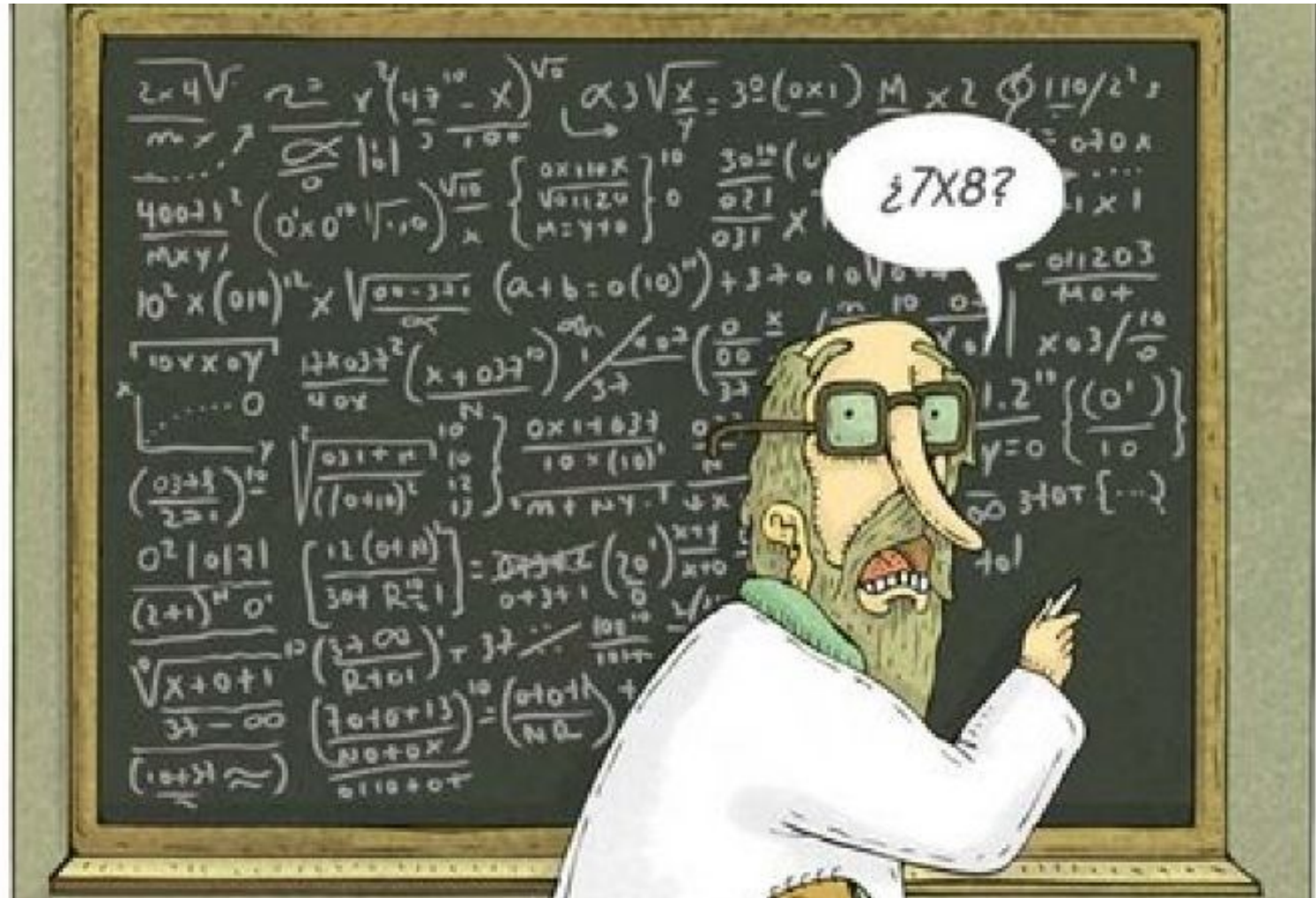
Consideraciones prácticas

- ▶ Es común apilar RNNs (**stacked RNNs**). La salida de una RNN se utiliza como la entrada secuencial de la RNN apilada encima de ella.
- ▶ También es común utilizar **RNNs bidireccionales** donde las secuencias de entrada se leen de atrás hacia adelante y de delante hacia atrás.



Fuente de la imagen: Stanford CS231N

¿Dudas?



UNIVERSIDAD
INTERNACIONAL
DE LA RIOJA

unir

www.unir.net