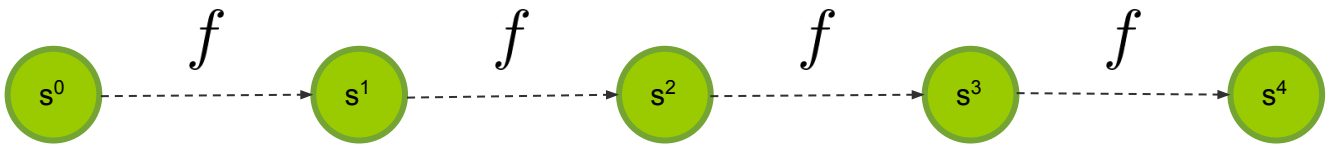
Clase 7

- Redes Recurrentes tradicionales (RNN)
- Problemas con las RNNs
- Modelos más complejos
    - LSTM
    - GRU

Redes Recurrentes (RNNs)

- Inspiradas en sistemas dinámicos

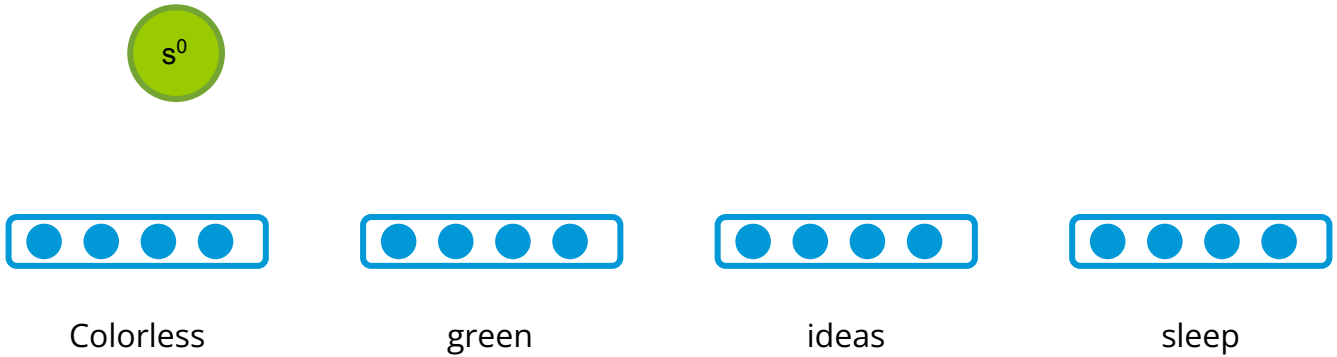
$$s^t = f(s^{t-1})$$



Redes Recurrentes (RNNs)

- Idea: Usar palabras como inputs y llevar un estado interno

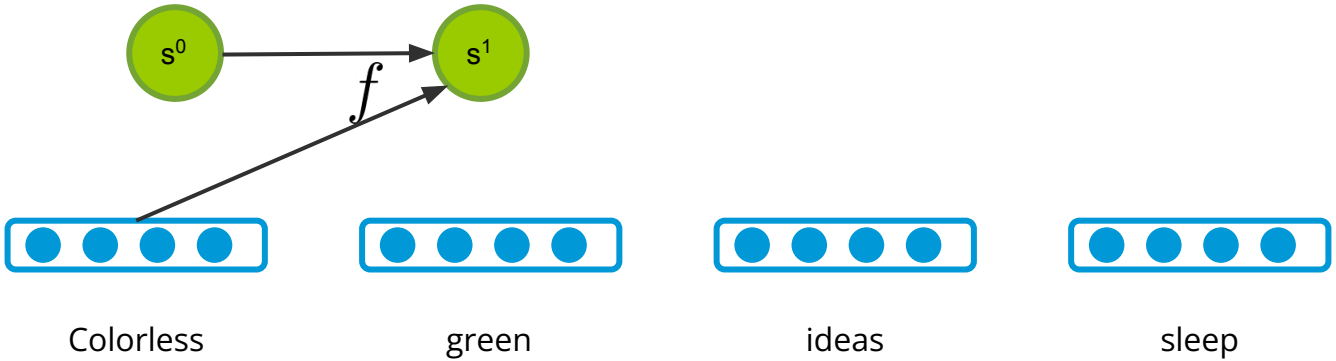
$$s^t = f(x^t, s^{t-1})$$



Redes Recurrentes (RNNs)

- Idea: Usar palabras como inputs y llevar un estado interno

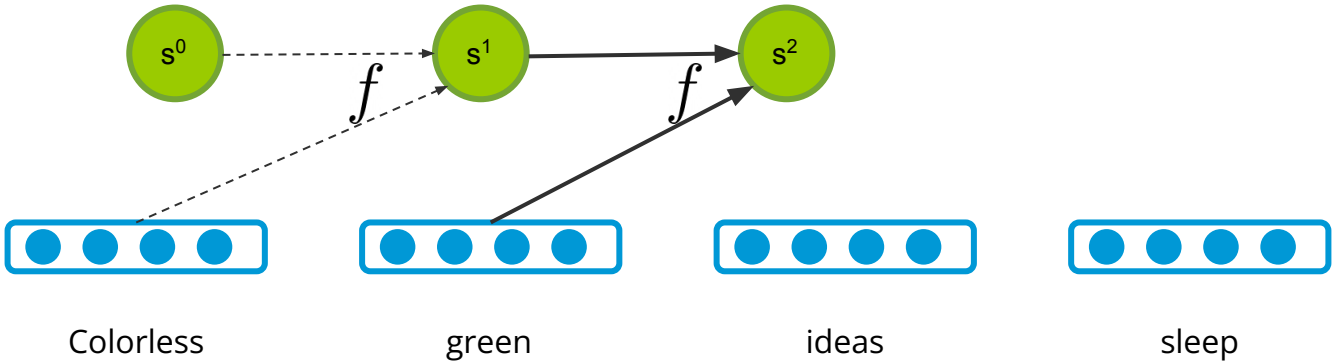
$$s^t = f(x^t, s^{t-1})$$



Redes Recurrentes (RNNs)

- Idea: Usar palabras como inputs y llevar un estado interno

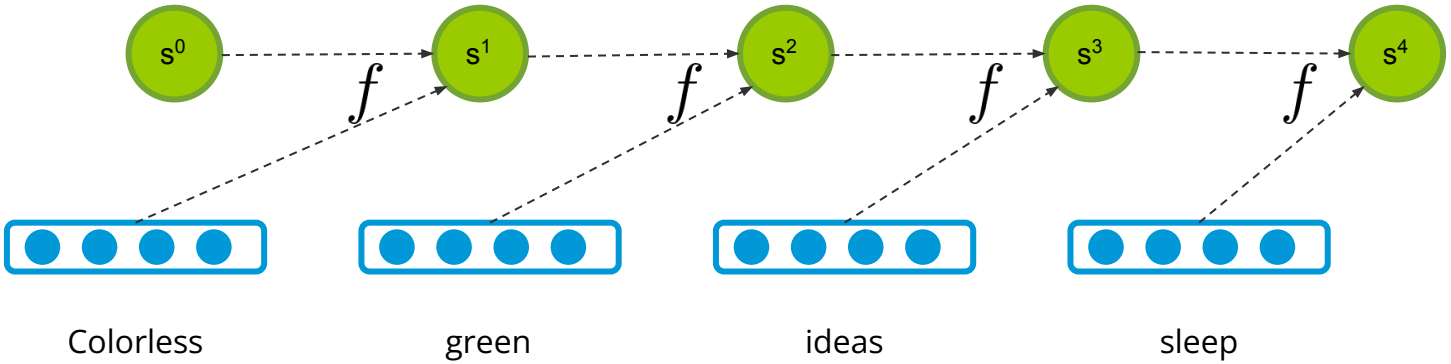
$$s^t = f(x^t, s^{t-1})$$



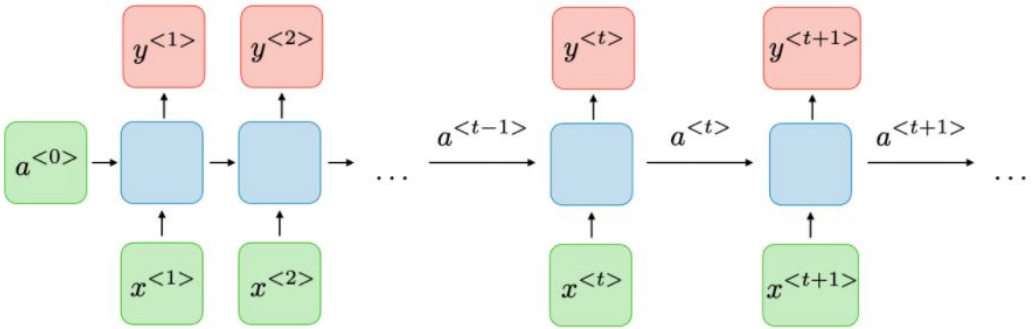
Redes Recurrentes (RNNs)

- Idea: Usar palabras como inputs y llevar un estado interno

$$s^t = f(x^t, s^{t-1})$$



Redes Recurrentes (RNNs)

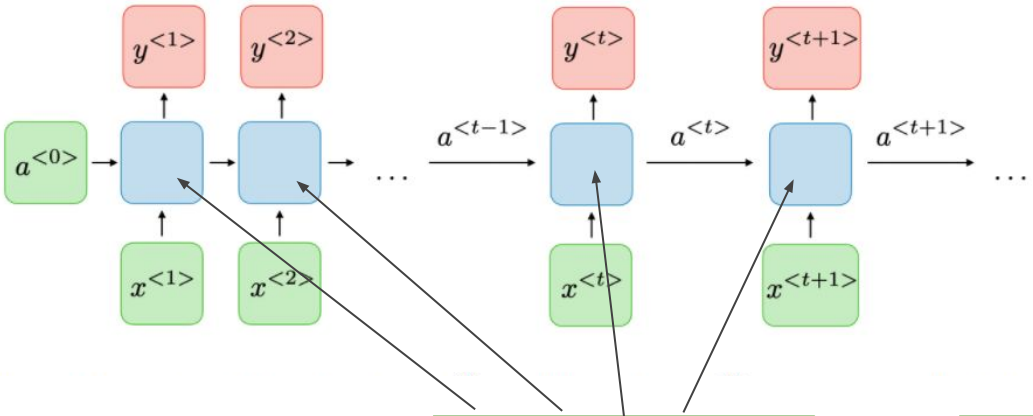


For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

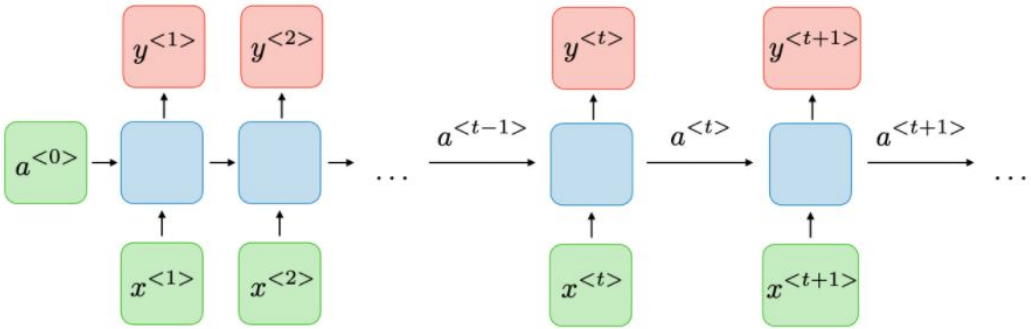
Redes Recurrentes (RNNs)



Contienen pesos de la red y se conocen como cells

Los pesos son los mismos para todos los inputs de la secuencia

Redes Recurrentes (RNNs)

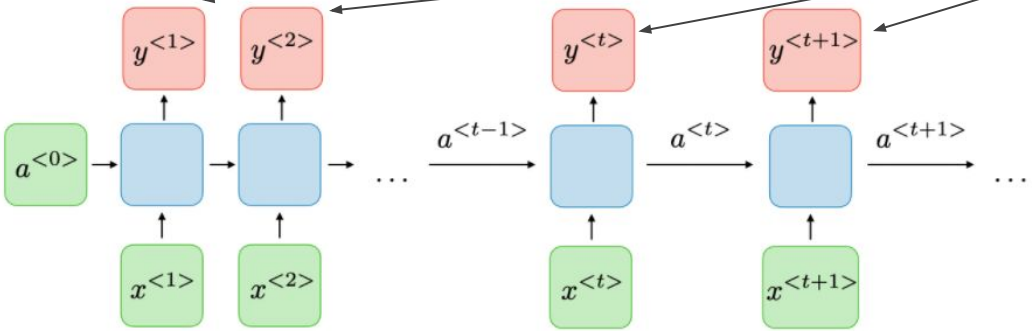


Xt es cada una de las palabras en la secuencia

Las palabras se procesan una a la vez

Redes Recurrentes (RNNs)

Yt es el output en cada tiempo t - para cada input

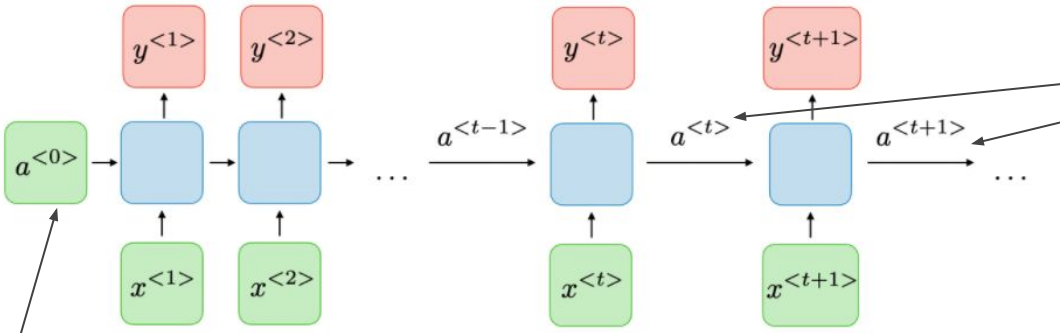


For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

Redes Recurrentes (RNNs)



At es el estado oculto en cada tiempo t

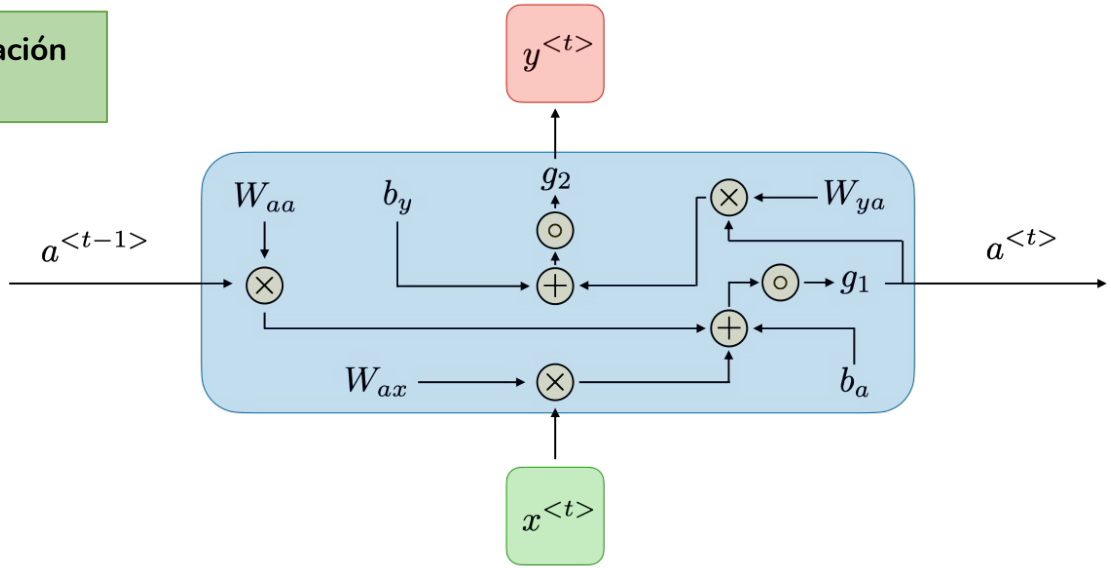
Ao (estado inicial) se inicializa normalmente en cero - “Memoria vacía”

El objetivo de la red es acumular información relevante en estos estados ocultos

Se puede considerar un “encoding” de la información vista hasta el momento

Redes Recurrentes (RNNs)

RNN cell, representación compacta



$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Redes Recurrentes (RNNs)

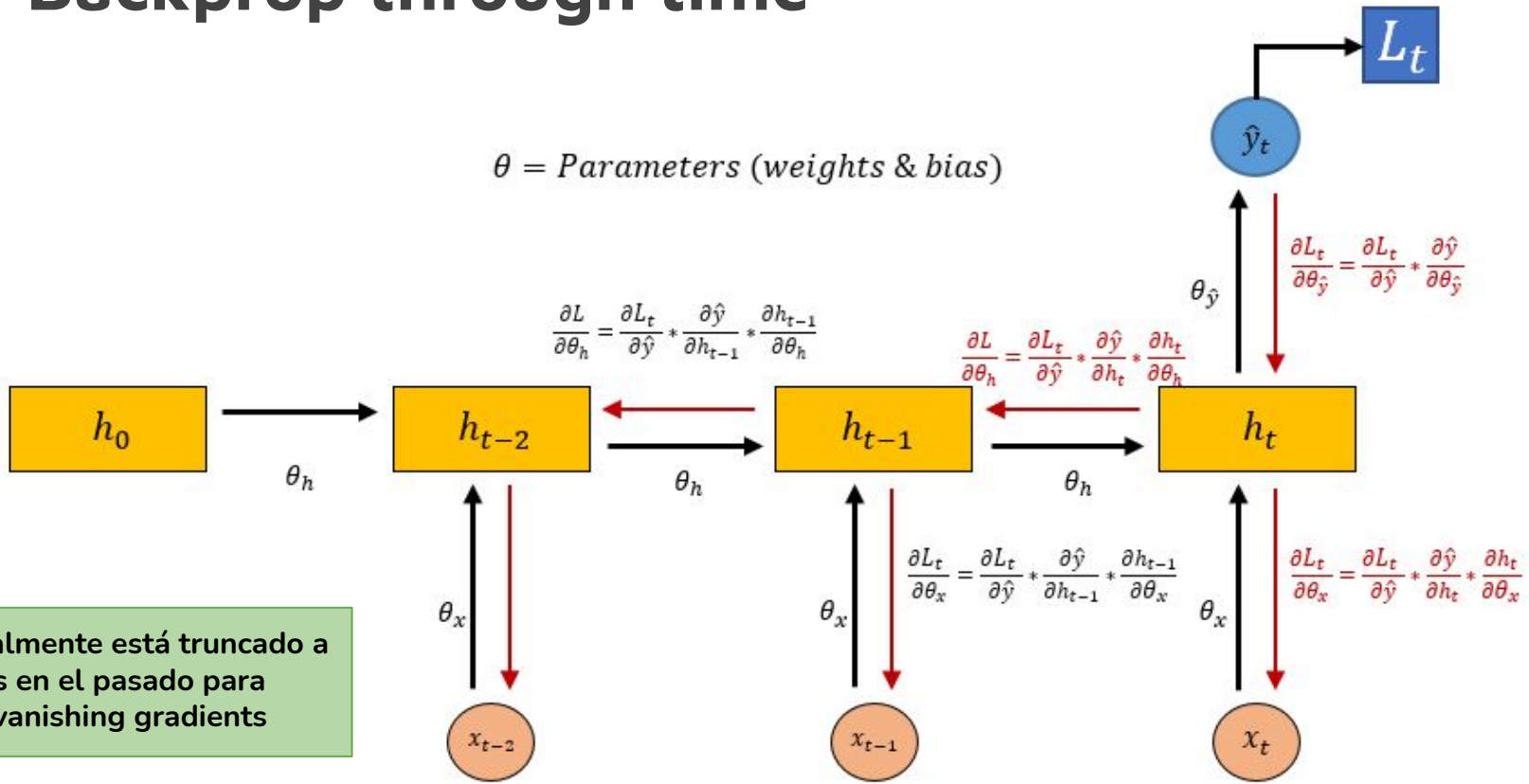
- El costo para una red recurrente se define como la **suma** de los costos en **cada paso T**

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

- Hacemos backpropagation para cada punto en el tiempo (cada elemento de la secuencia)
  - **Backpropagation through time**

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}^{(T)}}{\partial W}\Big|_{(t)}$$

Backprop through time



Generalmente está truncado a X steps en el pasado para evitar vanishing gradients

RNNs: Ventajas

- Podemos procesar inputs de cualquier largo
- La complejidad del modelo y la cantidad de pesos no crece con el tamaño del input
- Compartimos pesos a través del tiempo
- Podemos usar información contextual que observamos en el pasado
    - Almacenada en el estado de la red

RNNs: Desventajas

- Procesamiento secuencial de los inputs
  - No es paralelizable
  - No hace buen uso de optimizaciones de GPU
- Con inputs muy largos podemos olvidar información importante presente muchos pasos en el pasado

RNNs: Problemas

- Gradientes
  - Exploding gradients: gradientes que crecen exponencialmente.
  - Vanishing gradients: gradientes que disminuyen a valores extremadamente cercanos a 0.

- Entrenamiento
  - RNNs vanilla (tradicionales - sin ninguna modificación) tienden a no lograr aprender dependencias temporales muy largas.

RNNs: Problemas - Soluciones directas

- **Gradient clipping**: definir un tope para el valor máximo que pueden tomar los gradientes
- Identity initialization: Manera de inicializar los pesos de la red, presentado en: https://arxiv.org/pdf/1504.00941.pdf.
- ReLU como activación: Las relu sufren en **menor medida** de grandes variaciones en los gradientes

Variantes de RNNs

- Con el objetivo de **mejorar la “memoria” de las redes recurrentes**, varios modelos fueron propuestos

- Estos modelos alteran la arquitectura (y comportamiento) de las celdas en los modelos recurrentes tradicionales

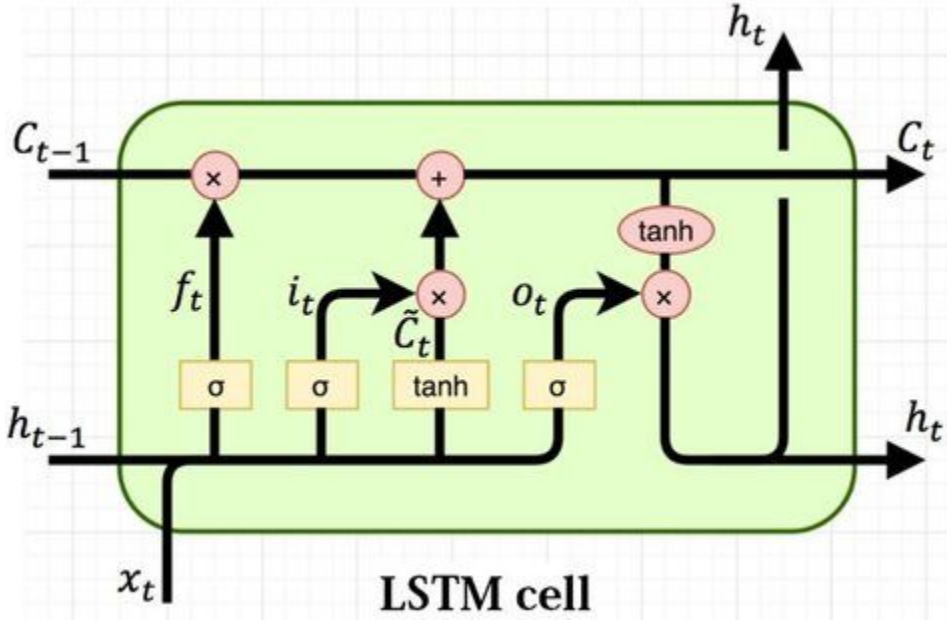
- Estado del arte:
  - **LSTM**
  - **GRU**

LSTM: Long Short-Term Memory

- Varias matrices de pesos como compuertas lógicas que controlan qué pasa con los inputs y la memoria
  - Update gate
    - Controla cómo y cuánto actualizar la memoria en este momento
  - Input gate
    - Qué tan importante es el input actual
  - Forget gate
    - Olvidar algo de información
  - Output gate
    - Cuánta información es importante para la salida

- Utiliza dos estados: **cell state** y **hidden state** a través del tiempo

LSTM cell



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$

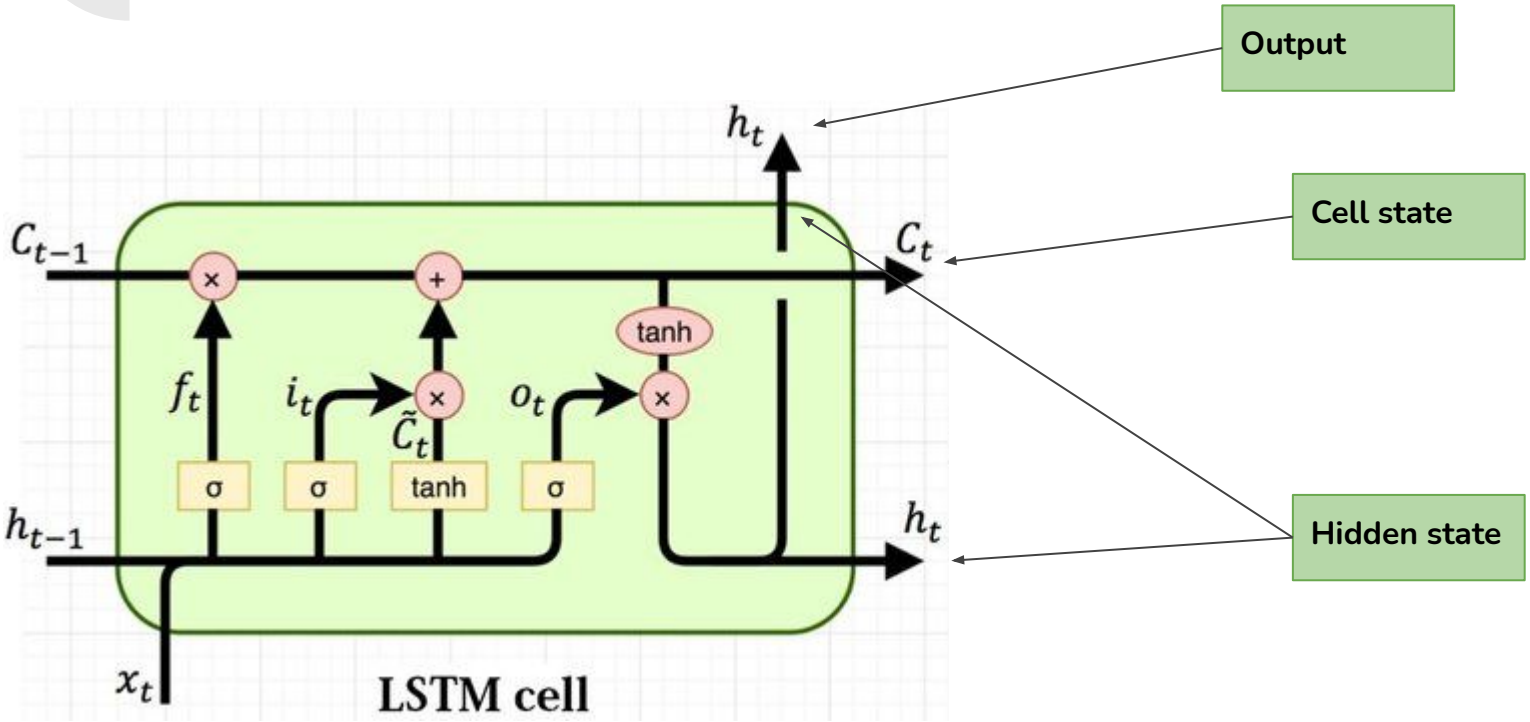
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$

$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$

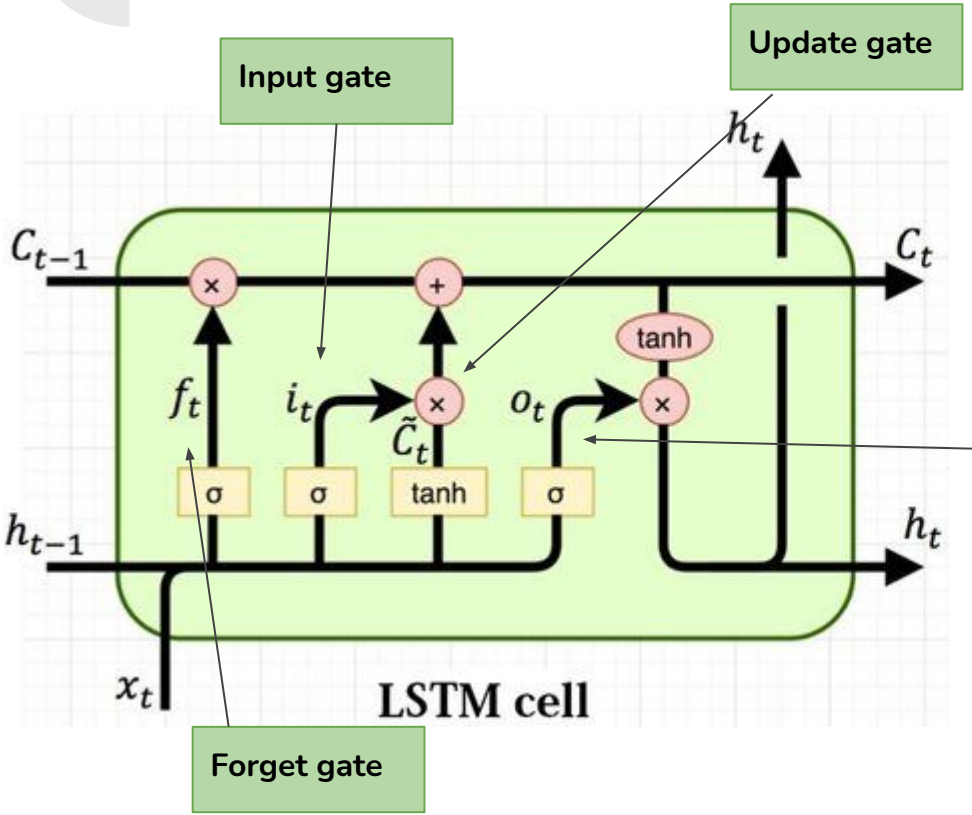
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$

$$h_t = \tanh(C_t) * o_t$$

LSTM cell


Output
Cell state
Hidden state
h_t
C_{t-1}
C_t
tanh
f_t
i_t
o_t
$\tilde{C}_t$
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
LSTM cell

LSTM cell



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

LSTM cell

$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$ Input gate

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$ Forget gate

$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$ Output gate

$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$ Update gate

$$h_t = \tanh(C_t) * o_t$$

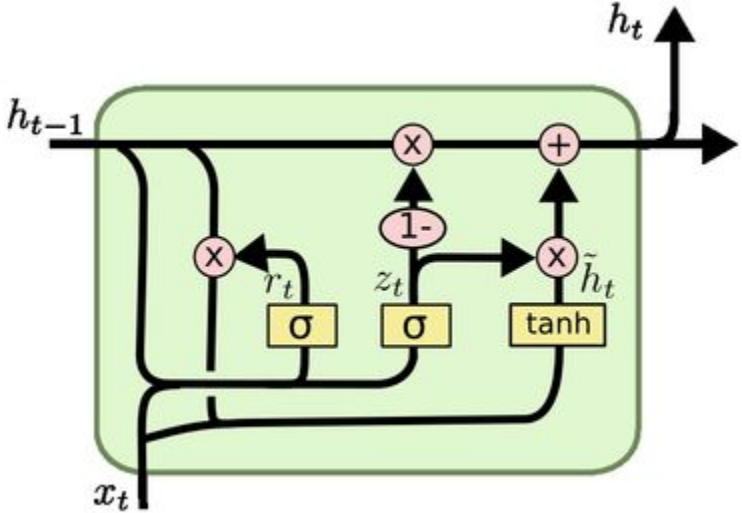
Cada Ui y Wi representa una matriz de pesos *distinta*, que se utiliza para procesar toda la secuencia.

GRU: Gated Recurrent Unit

- **Simplifica la celda de LSTM**
  - Cambiando las “compuertas”
  - Esto simplifica la matemática
  - **Facilita el entrenamiento**
  - **Reduce el número de pesos**

- Utiliza
  - Input gate -> cambia a Reset gate (controla cuánta importancia tiene el input actual)
  - Update gate

GRU cell



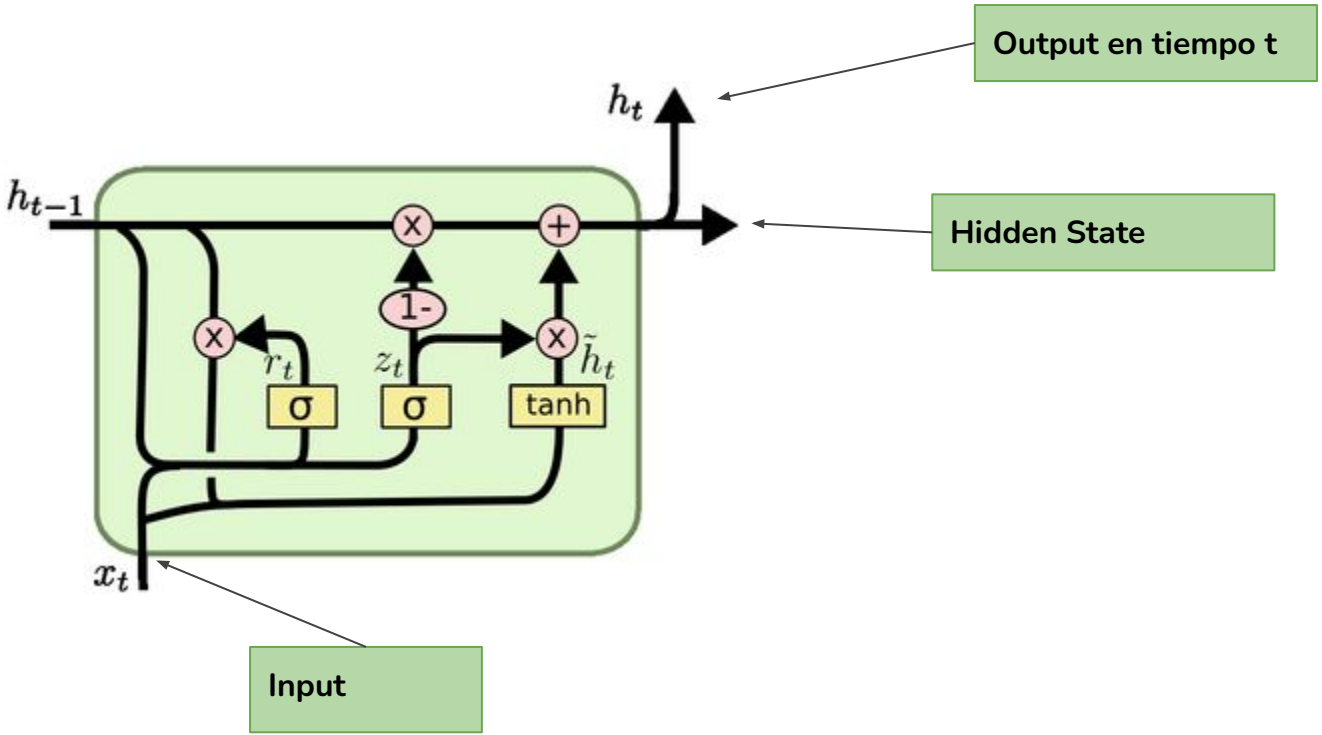
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

GRU cell


Output en tiempo t
h_t
h_{t-1}
Hidden State
x
+
1-
x
x
r_t
z_t
$\tilde{h}_t$
σ
σ
tanh
x_t
Input

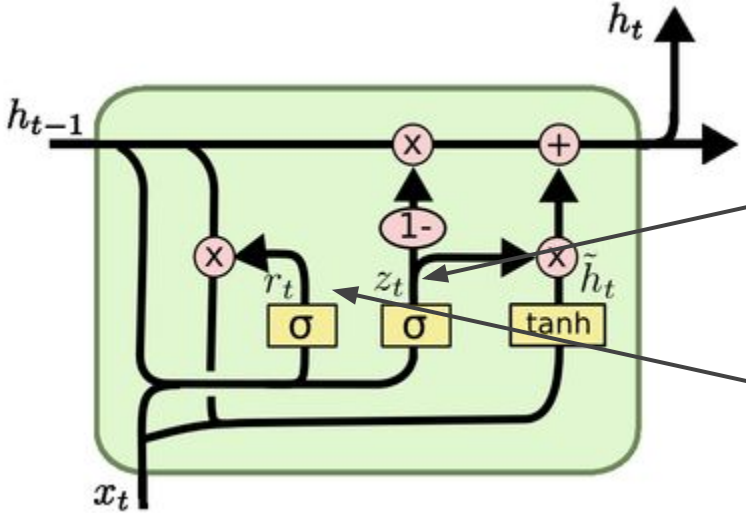
GRU cell

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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



Update gate

Reset gate (Input gate)

GRU cell

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$ Update gate

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$ Reset (input) gate

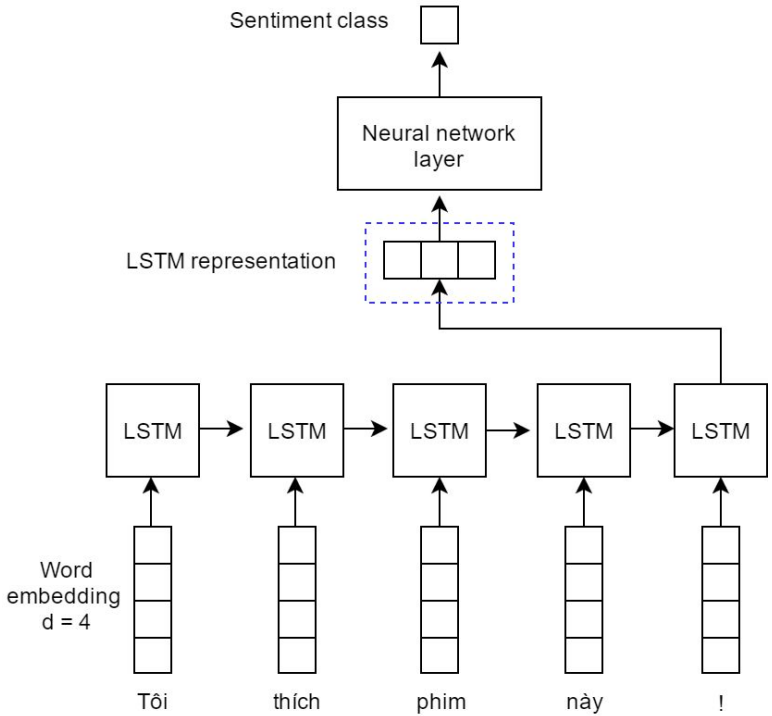
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$ Output en tiempo t

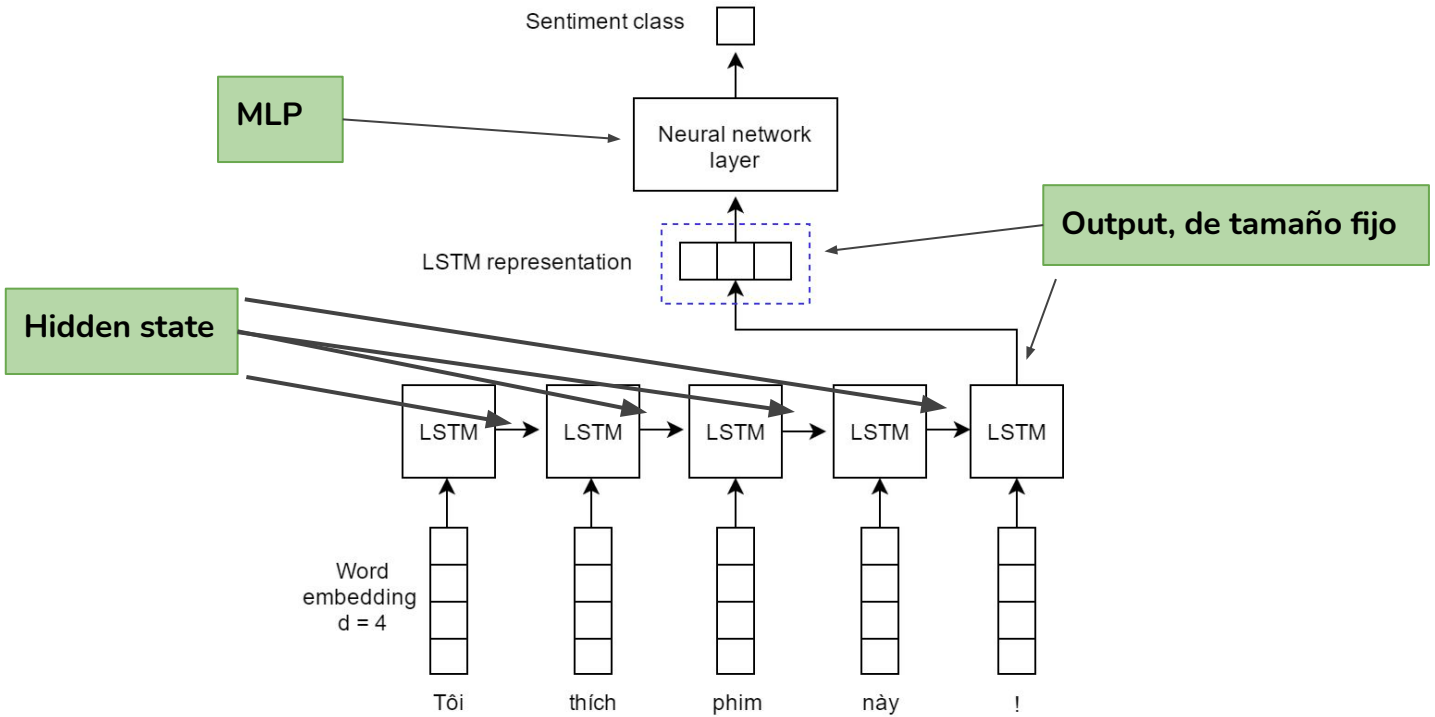
Ya no tenemos W y U como matrices de pesos, ahora solamente W.

Cada Wi es una matriz de pesos independiente y compartida a lo largo del tiempo

En la práctica: Clasificación

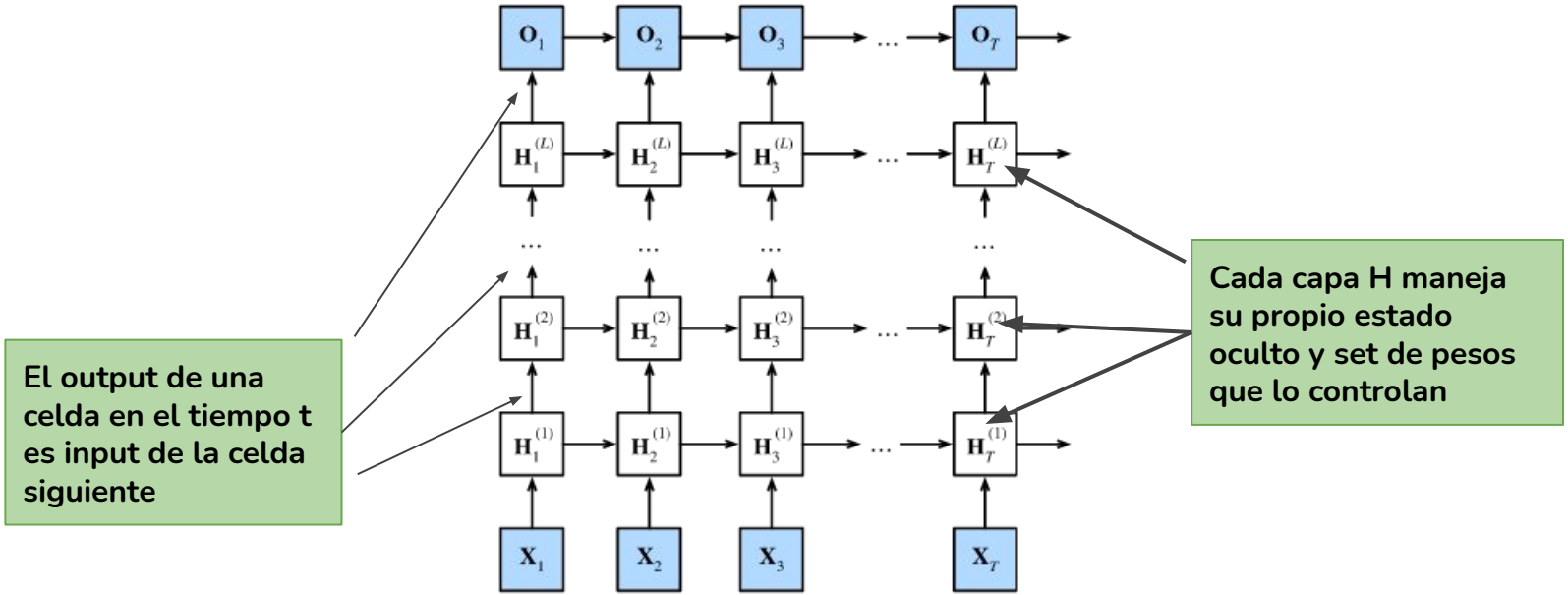


En la práctica: Clasificación

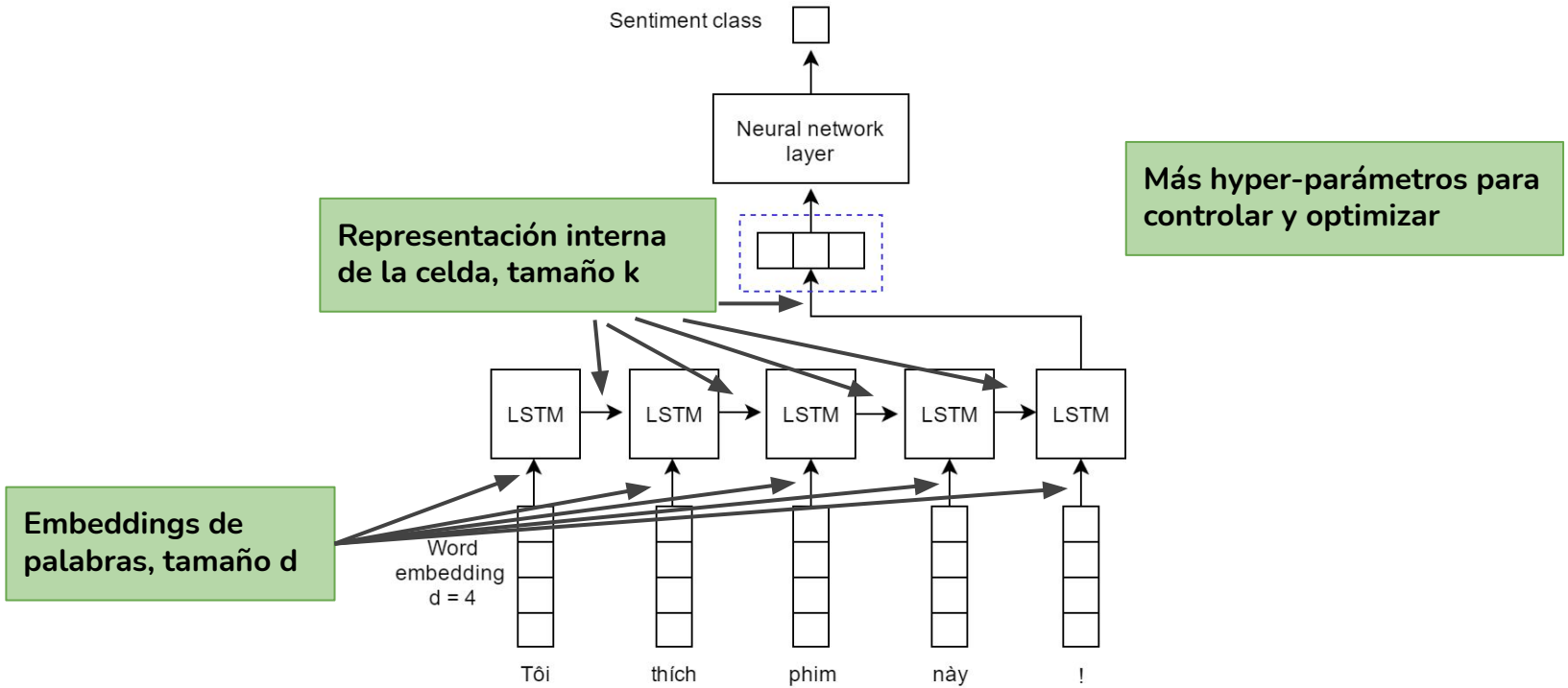


En la práctica

- Es fácil de ver que podemos combinar celdas “verticalmente” agrandando la capacidad de los modelos
  - Éste y otros trucos son clave en modelos que dominan en estado del arte



En la práctica: Tamaños



Embeddings

- Todas estas arquitecturas son capaces de **aprender embeddings** para cada problema (o usar pesos de embeddings pre-entrenados)

- En la mayoría de los frameworks existe una capa de "**Embedding**" que mapea de un dominio disperso (sparse) (ej. 500.000 índices de palabras) a un dominio denso (ej. vectores de 300 dimensiones)

- Por lo general usamos la salida de este tipo de capas como entrada a nuestras redes
    - Para evitar tener que convertir todo a tensores previamente

Referencias

- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
- https://pytorch.org/docs/stable/generated/torch.nn.RNN.html
- Ruseti, Stefan. (2019). Advanced Natural Language Processing Techniques for Question Answering and Writing Evaluation. 10.13140/RG.2.2.21901.69602.