

Redes neuronales recurrentes RNN Long-Short Term Memory (LSTM) Unidad Recurrente Cerrada (GRU)

**Cuando importa el pasado (El contexto)...
Física Computacional II**

Ph.D. Santiago Echeverri Arteaga

Red Neuronal Recurrente

1. Muy usada en reconocimiento de texto, traducción, escritura, análisis sentimentalismo y series de tiempo
2. Dos salidas:
 - A. Predicción
 - B. Estado: Recuento del pasado hasta ese punto
3. Necesidad de:
 1. Manejar secuencias de datos de longitud variable
 2. Parámetros compartidos
 3. Hacer seguimiento a largo término
 4. Mantener información sobre los datos de entrada.
4. Kan

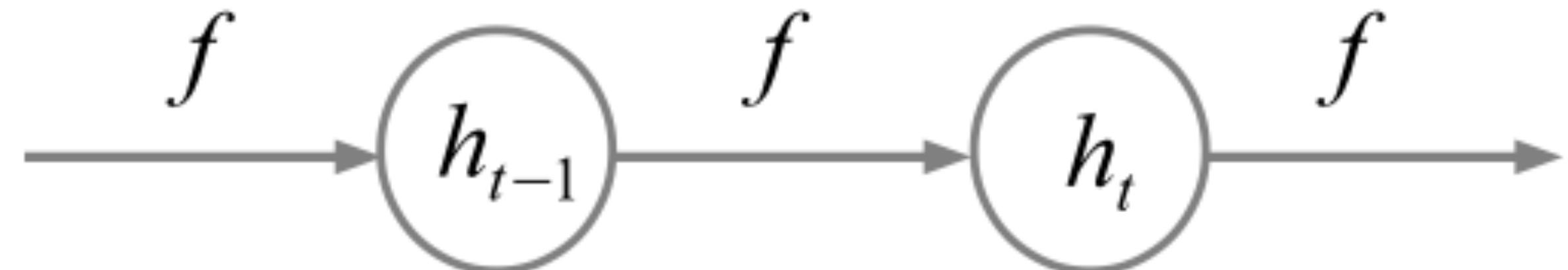
State System

Considere el sistema dinámico cuyo estado h_t en el tiempo t se actualiza como

$$h_t = f(h_{t-1}; \theta) \quad t \geq 1$$

donde la **función de transición** $f: \mathbb{R}^k \times \mathbb{R}^m \rightarrow \mathbb{R}^k$ y $\theta \in \mathbb{R}^m$ es un vector de parámetros independiente de t .

Se consideran los h_t random variables y por tanto generadoras de un álgebra $\mathcal{H}_t = \mathcal{G}(h_t)$ i.e. Cada estado h_t contiene información \mathcal{H}_t sobre el sistema dinámico



¿Cómo surgen aquí los gradientes desvanecidos?

Asuma que f es diferenciable y satisface la desigualdad $\|\partial f/\partial h\| < \lambda < 1$, luego por el Teorema del valor central

$$\|f(h; \theta) - f(h'; \theta)\| < \|\partial_h f\| \|h - h'\| < \lambda \|h - h'\|$$

Ahora, tómense dos estados cualesquiera en del espacio de configuración $\omega, \omega' \in \Omega$, se tiene que

$$\|h_t(\omega) - h_t(\omega')\| = \|f(h_{t-1}(\omega); \theta) - f(h_{t-1}(\omega'); \theta)\| = \lambda \|h_{t-1}(\omega) - h_{t-1}(\omega')\| < \lambda^2 \|h_{t-2}(\omega) - h_{t-2}(\omega')\| < \dots < \lambda^n \|h_0(\omega) - h_0(\omega')\|$$

Que por el teorema del emparedado en el límite $t \rightarrow \infty$ se tiene que

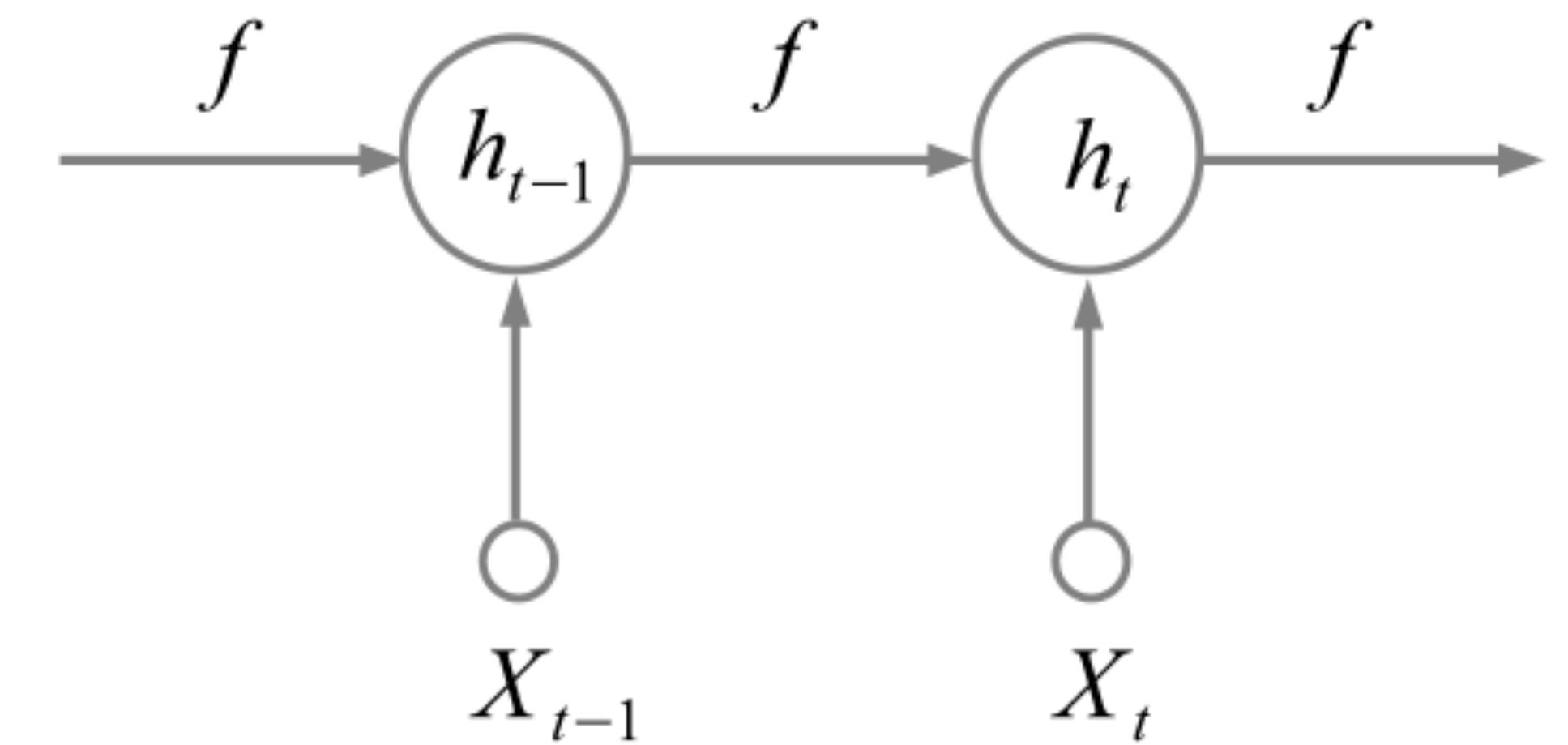
$$\lim_{t \rightarrow \infty} \|h_t(\omega) - h_t(\omega')\| = 0$$

Lo que implica que $\lim_{t \rightarrow \infty} h_t = c$ por lo que $f(c; \theta) = c$ es decir, c es el punto fijo de f

La consecuencia de esto es que la información obtenida por \mathcal{H}_t disminuye conforme t aumenta. El sistema está perdiendo la información sobre el pasado (Vanishing gradients)

¿Solución?

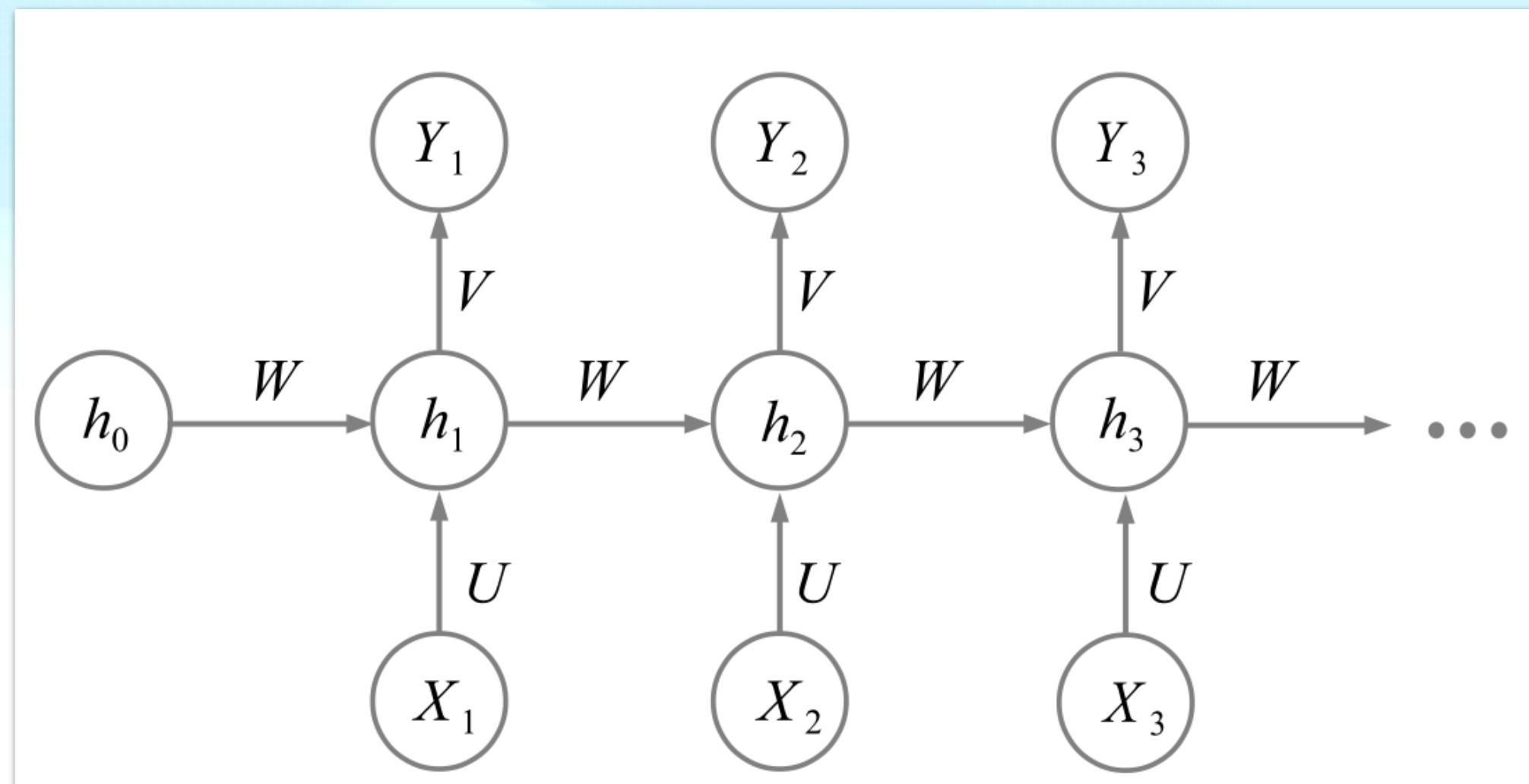
Se debe considerar información adicional que va siendo insertada al sistema.
Para esto considérese un proceso estocástico $X_t \quad t > 0$ tal que
$$h_t = f(h_{t-1}, X_t; \theta)$$



Red Neuronal Recurrente

Las redes neuronales recurrentes se pueden entender de dos maneras:

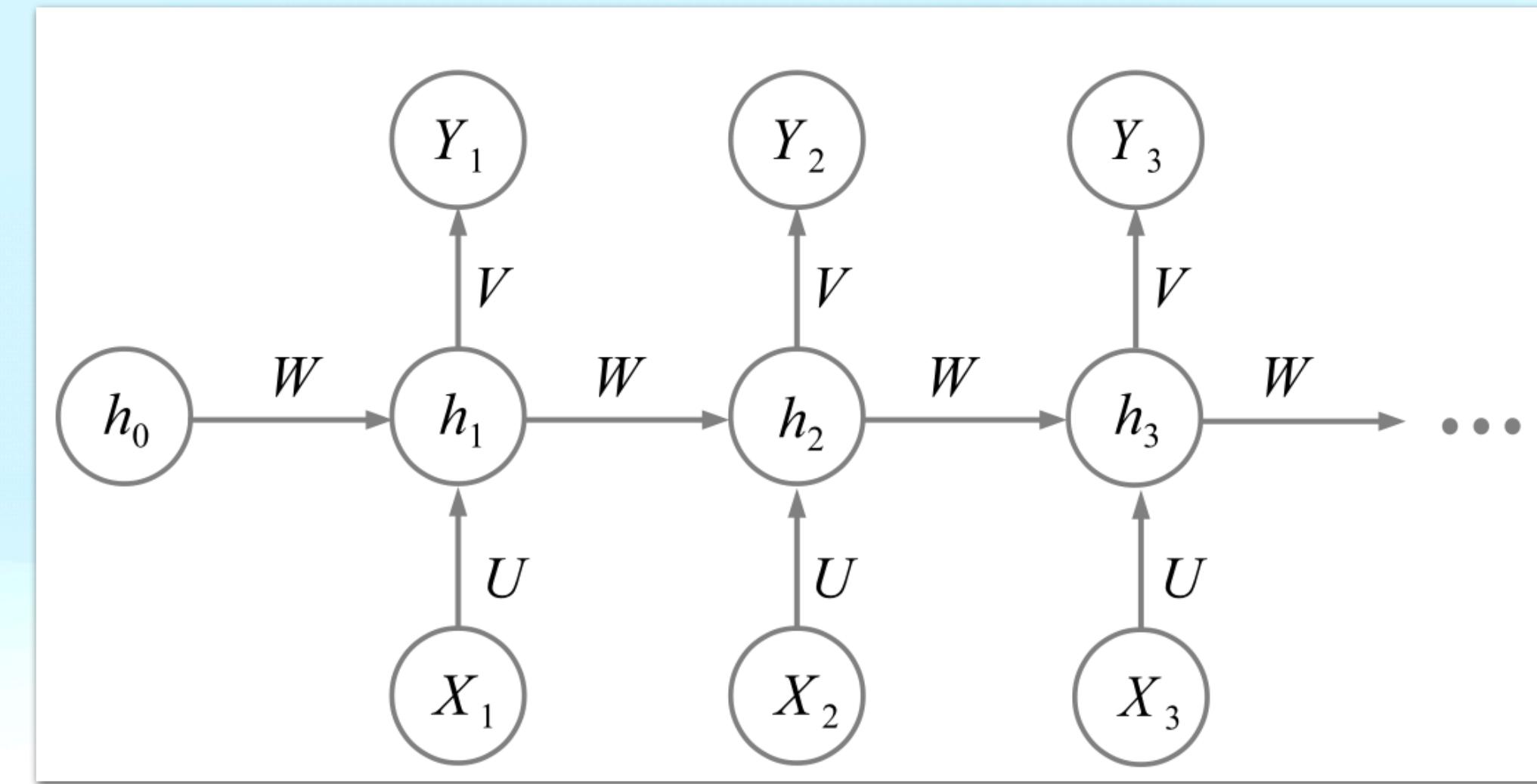
1. Un State System que en cada estado (excepto el inicial h_0) se obtiene una salida Y_t . Aquí el estado h_t se llama estado oculto y depende del estado previo h_{t-1} y del input presente X_t , el cual provee una salida Y_t .
2. Se puede pensar también una RNN como una red feedforward con input layer X_t , hidden layer h_t y output layer Y_t ($X_t \rightarrow h_t \rightarrow Y_t$). En donde la capa oculta h_t se conecta con otra capa oculta h_{t+1}



Red Neuronal Recurrente

3. Las ecuaciones estándar para la forward propagation en una RNN son

$$h_t = f_1(Wh_{t-1} + UX_t + b)$$
 y
$$Y_t = f_2(Vh_t + c)$$
4. La función f_1, f_2 son funciones de activación. f_2 puede ser la identidad.
5. La matriz W es la que determina la transferencia entre hidden states, la matriz U la transferencia entre el input y el hidden state, la matriz V la transferencia a la salida y b, c dos vectores de bias
6. $U_{s \times r}, W_{s \times s}, V_{t \times s}$
 1. s :dimensión del estado oculto
 2. r :dimensión del input
 3. t :dimensión del vector de salida
7. A veces se entrena solo la salida final y se ignoran las intermedias
8. BPTT: Backpropagation Through Time. Sensible a la longitud de la secuencia debido al vanishing gradients
9. En NLP se suele usar una Embedding layer previo al RNN para detectar palabras similares y codificarlas numéricamente



Deep Learning Architectures, A Mathematical Approach

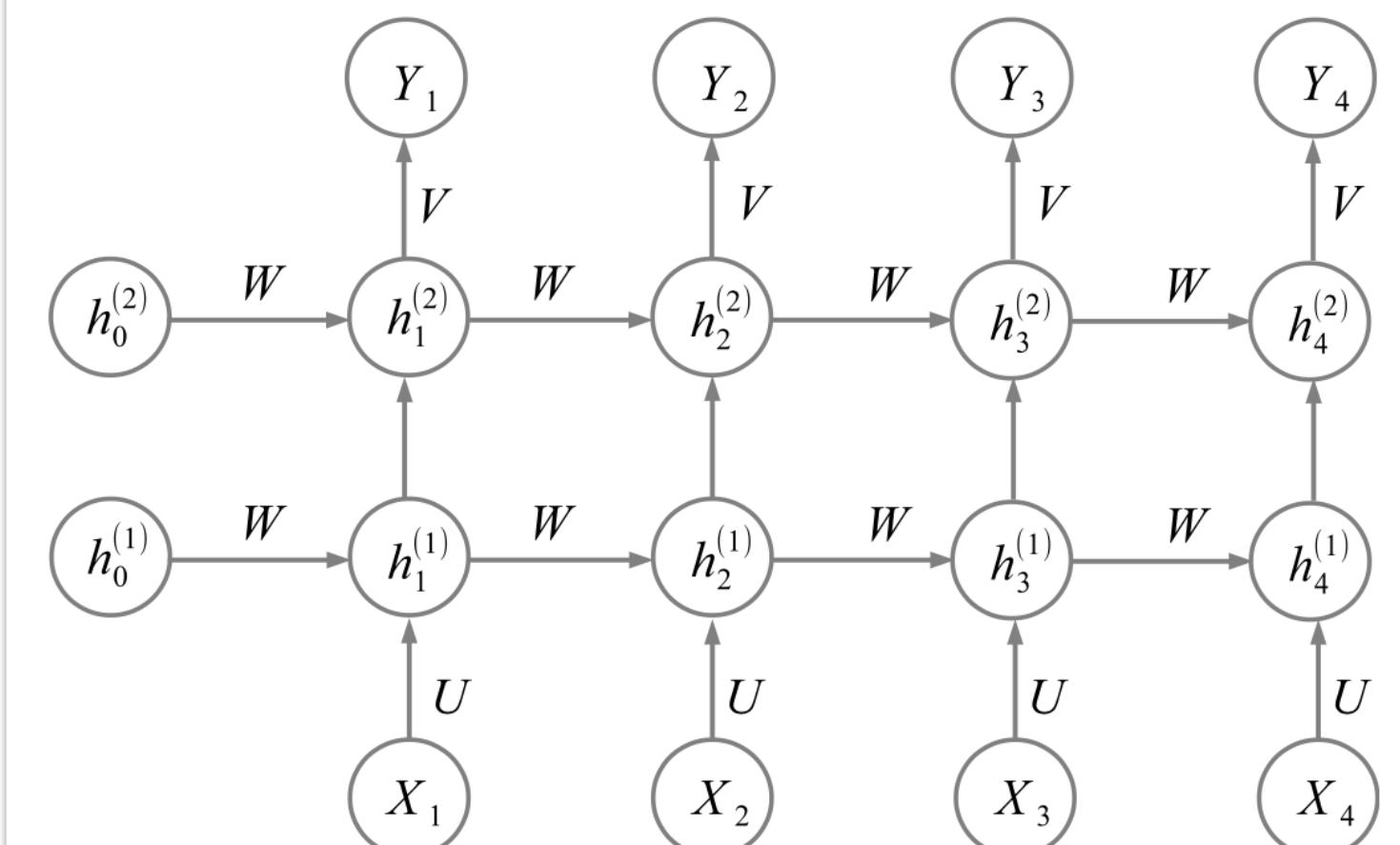


Figure 17.4: A deep RNN with $T = 4$ and $L = 2$.

Usos y problemas de una RNN

- **Usos:** Pronósticos, evolución de sistemas, reconocimiento de texto, predicción de donde va a existir un fallo con datos de sensores, secuenciación genómica
- **Contras:** Dificulta aprender la información del inicio de la secuencia, debe mantener todos los estados antiguos cuando pueden ser necesarios solo algunos

When you don't use LSTM
for a long sequence RNN



Long-Short Term Memory (LSTM)

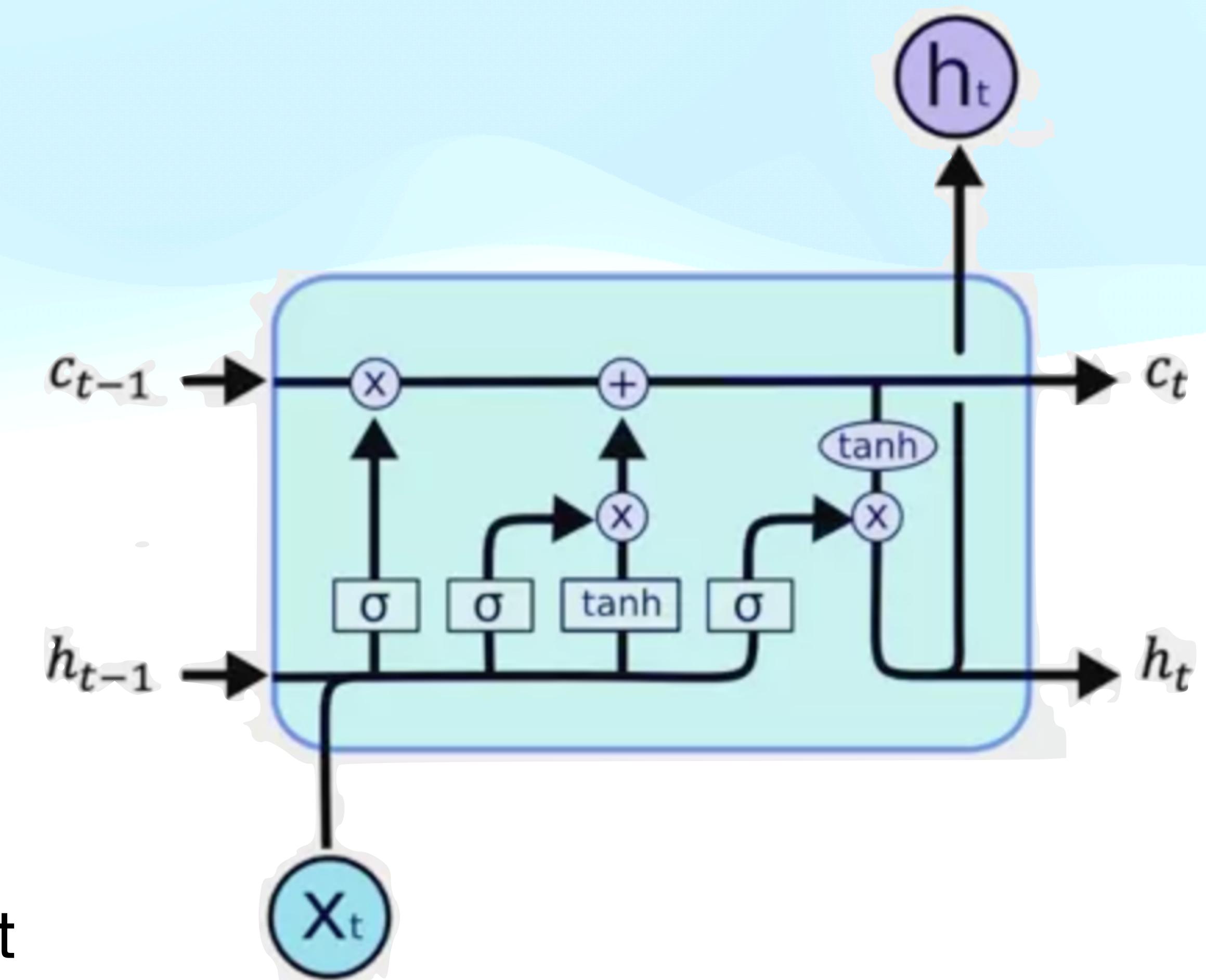
La idea es vieja (1977) pero el poder computacional es nuevo

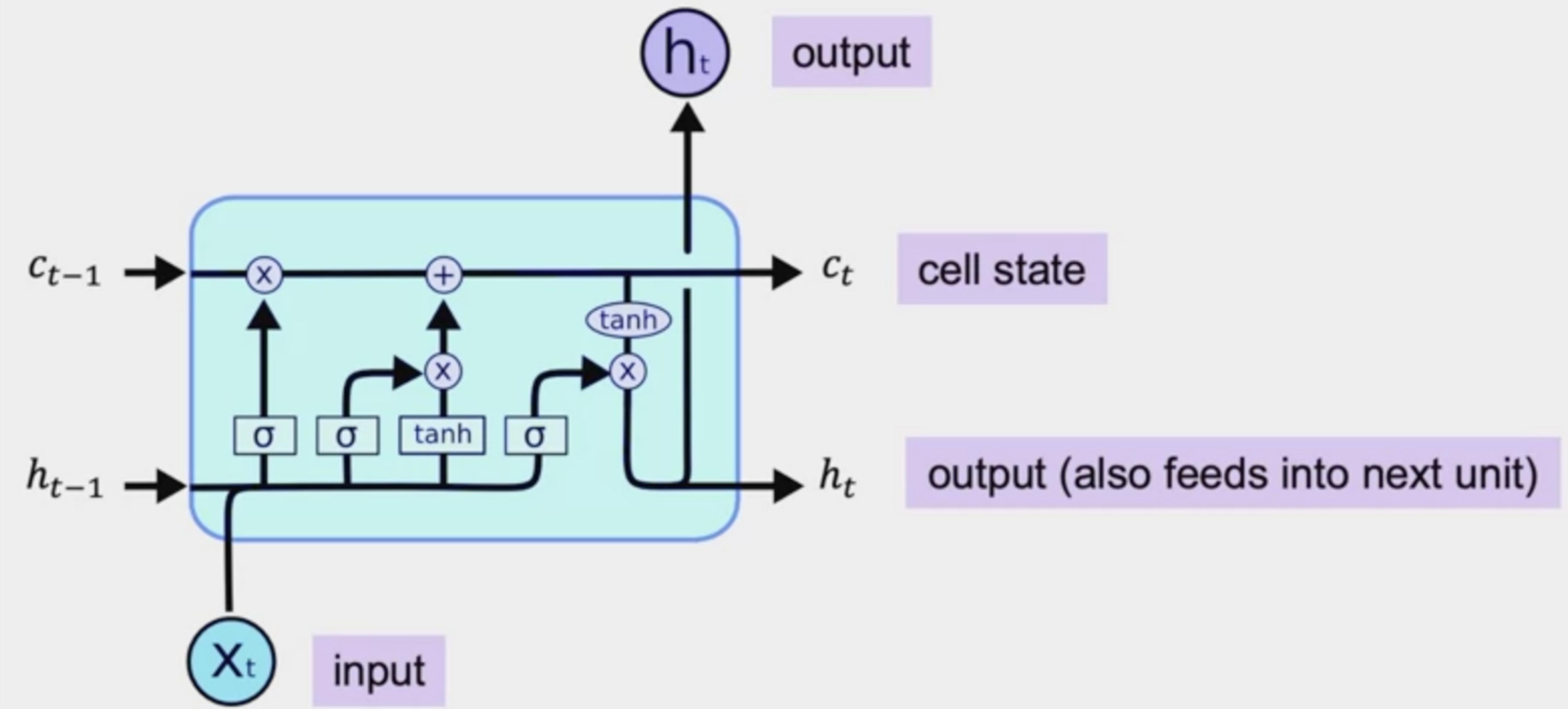
- RNN tiende a disminuir la influencia de los primeros términos. Para solventar esto se añade una unidad de memoria explícita o Gate Unit, las cuales controlan que tanto se quedan los eventos en memoria. Para ello se introduce el estado de la celda c_t

Input Gate: Si tiene cierto valor, causa que el item sea guardado en memoria

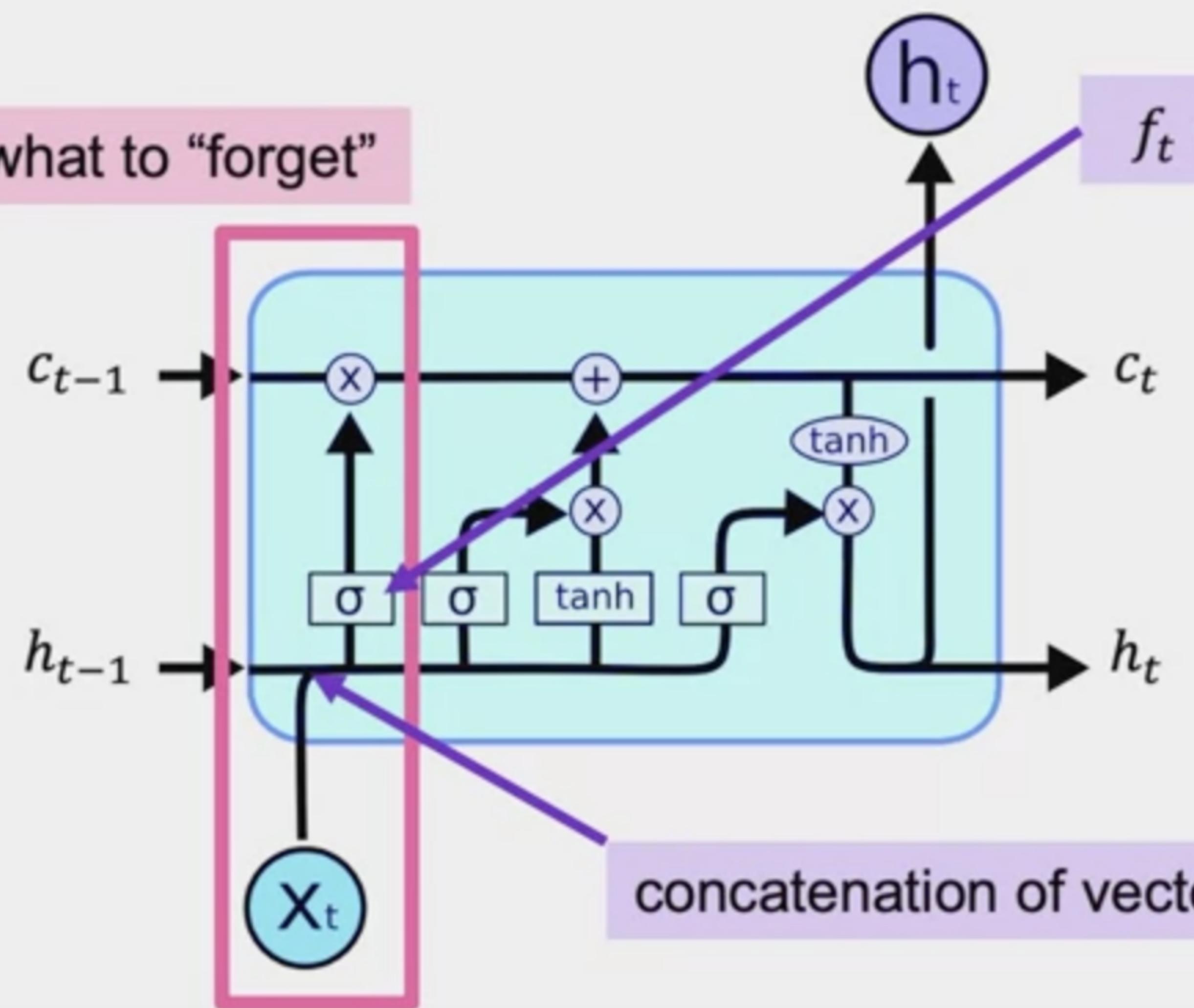
Forget Gate: Si tiene cierto valor, el item se remueve de memoria

Output Gate: Si tiene cierto valor, la hidden unit continua a la red



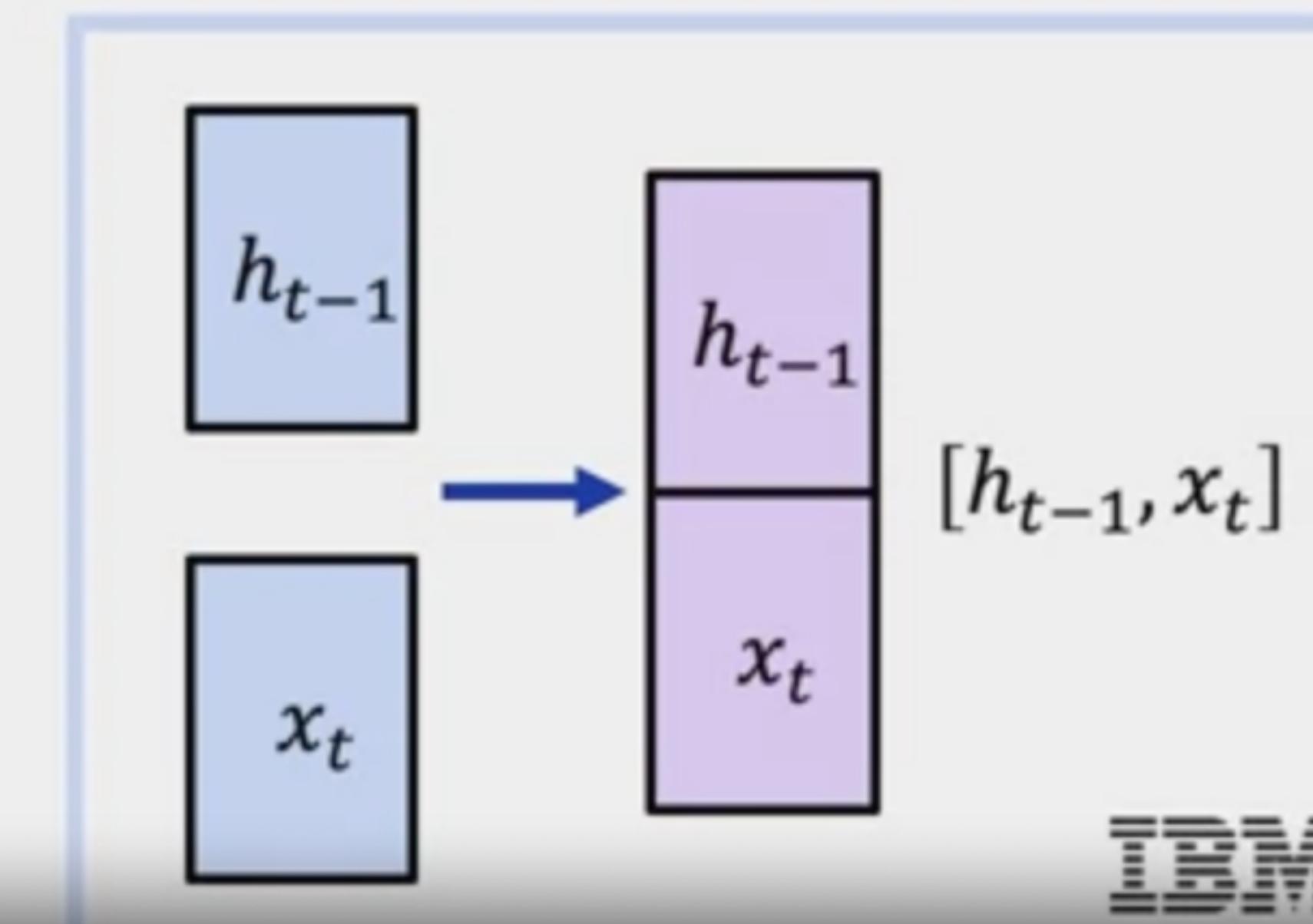


Decide what to “forget”



based on previous output
and current input.

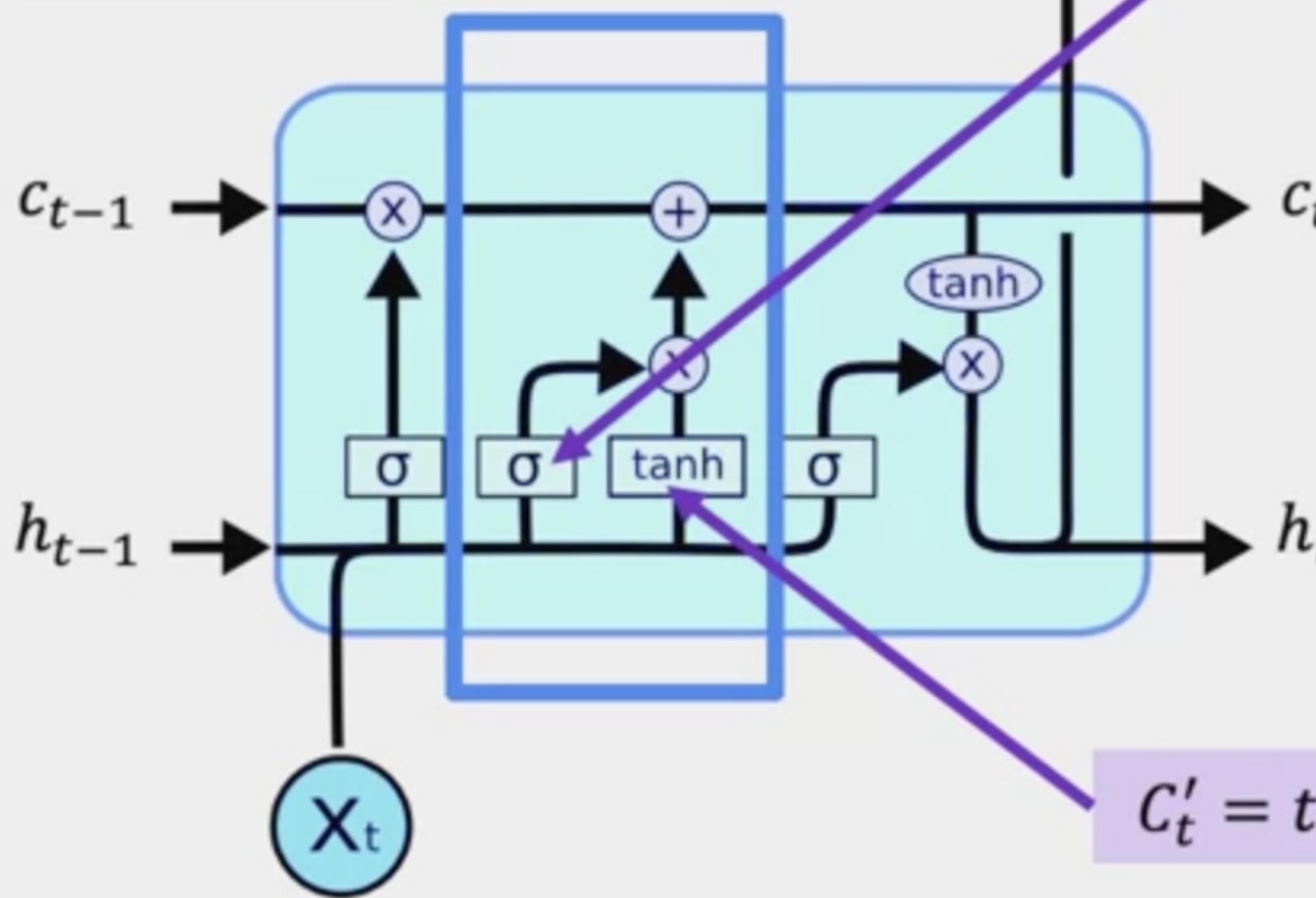
concatenation of vectors



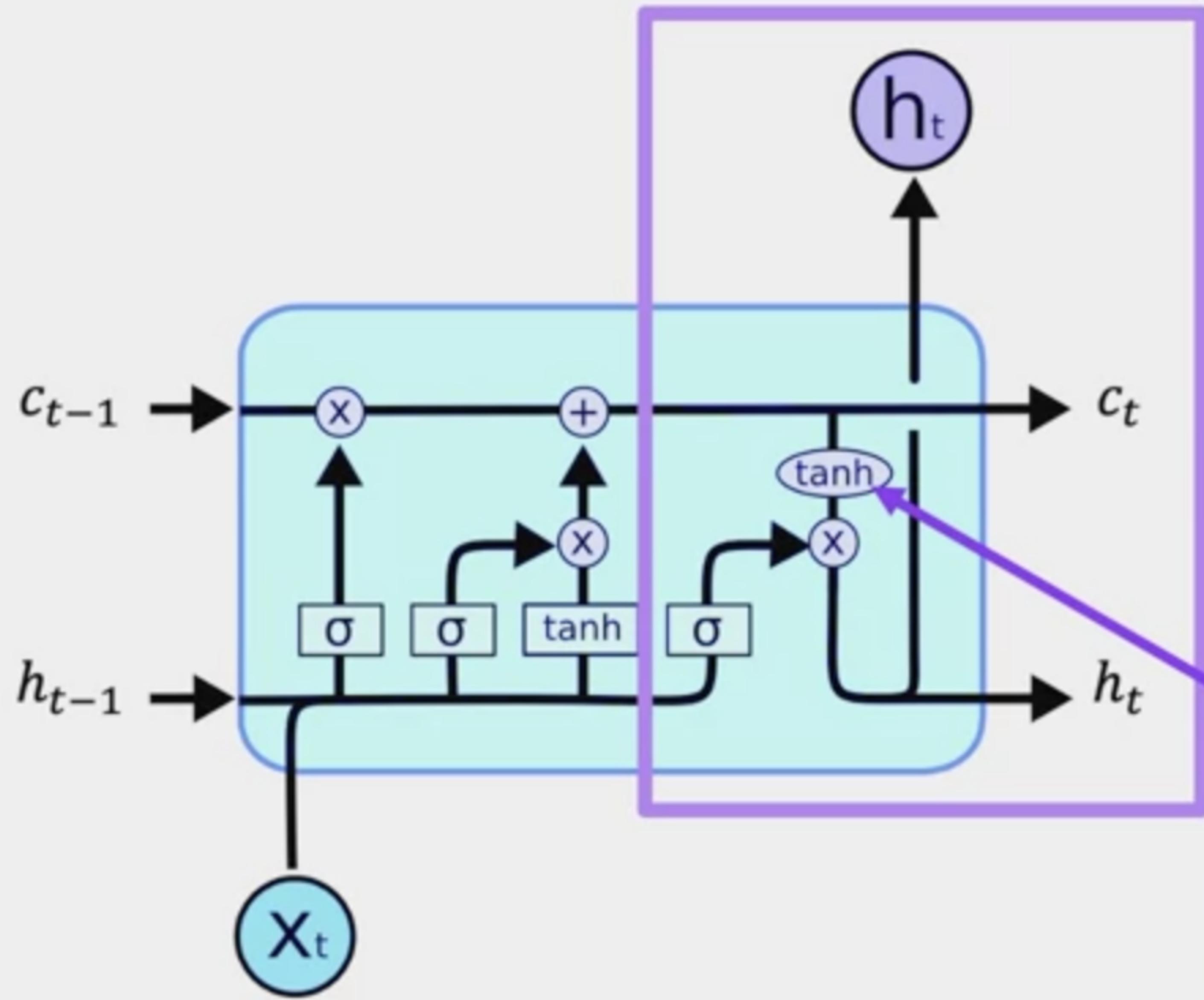
IBM

Add in “new” information

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$



$$C'_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

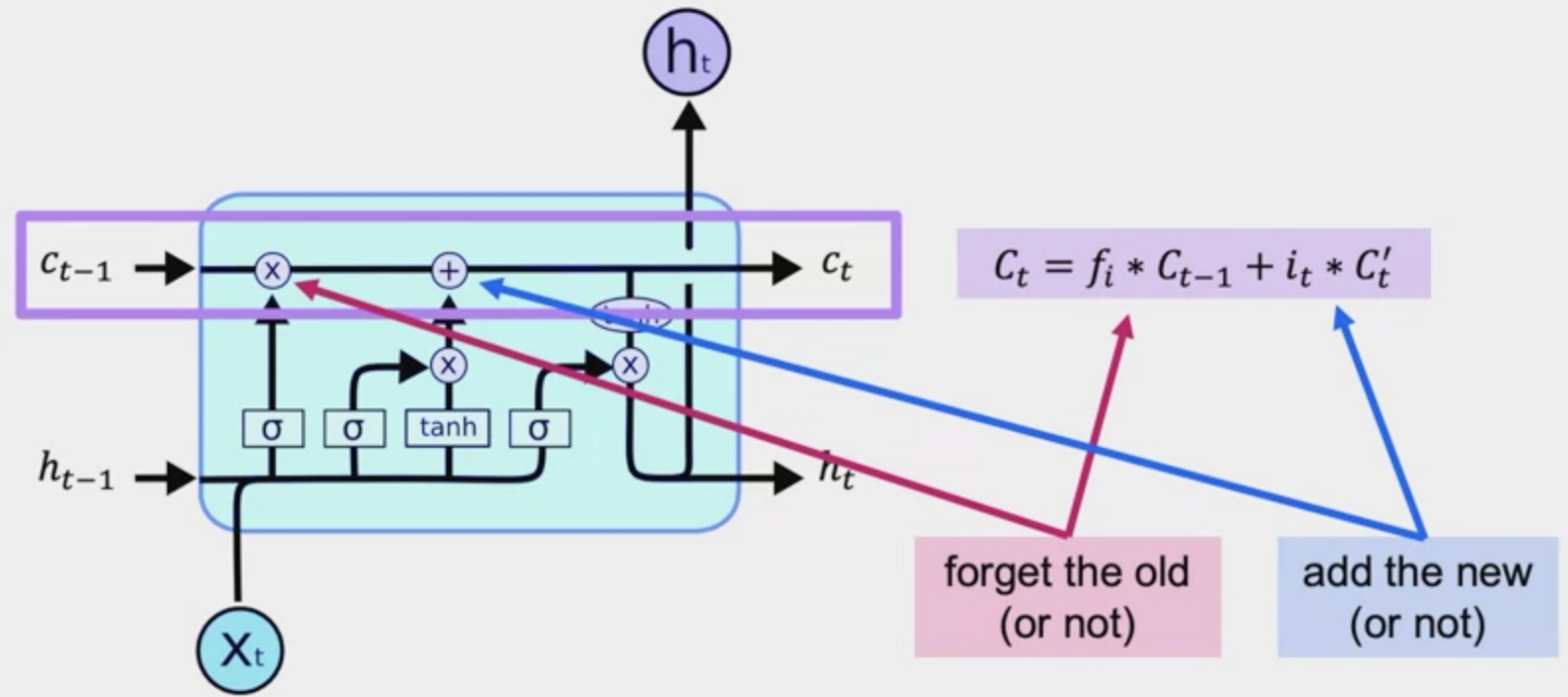


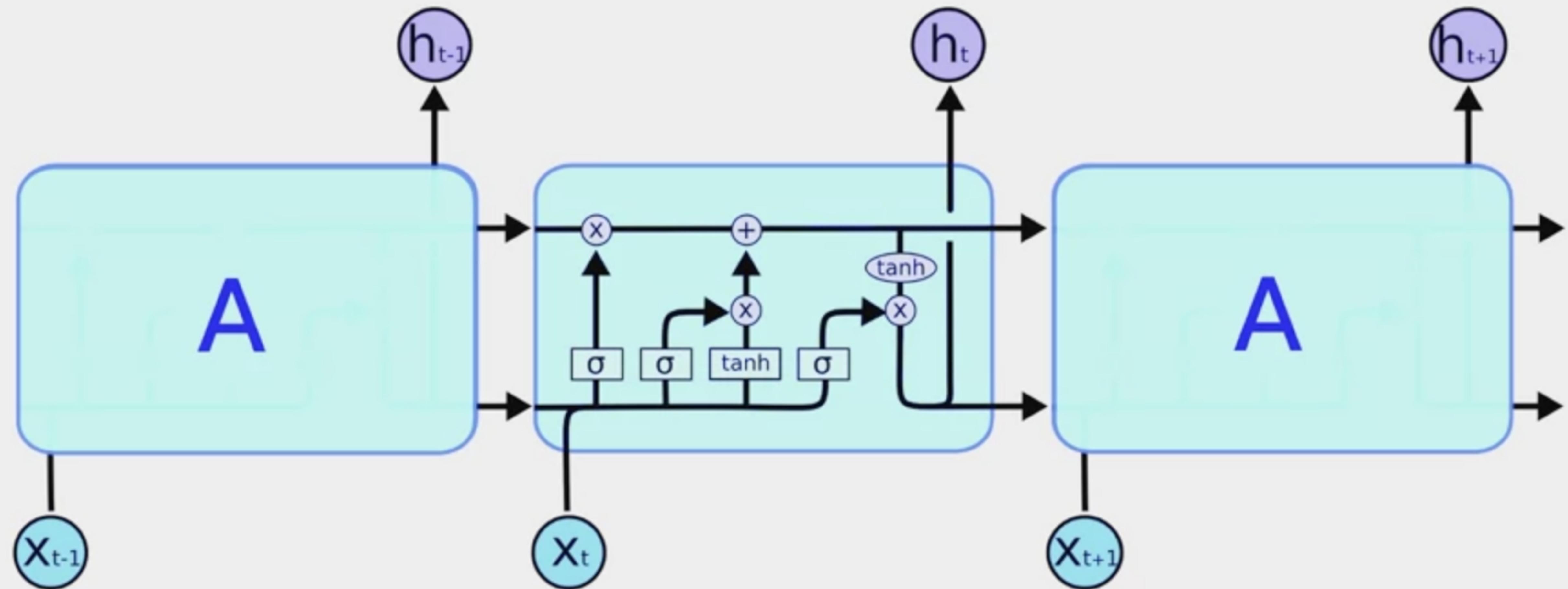
Final stage computes the output

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

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

Note: No weights here





Long-Short Term Memory (LSTM)

La idea es vieja (1977) pero el poder computacional es nuevo

- La celda se actualiza en dos momentos, \otimes forget gate y \oplus añade nueva información

Forget Gate: $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ donde $[h_{t-1}, x_t]$ son los dos vectores concatenados

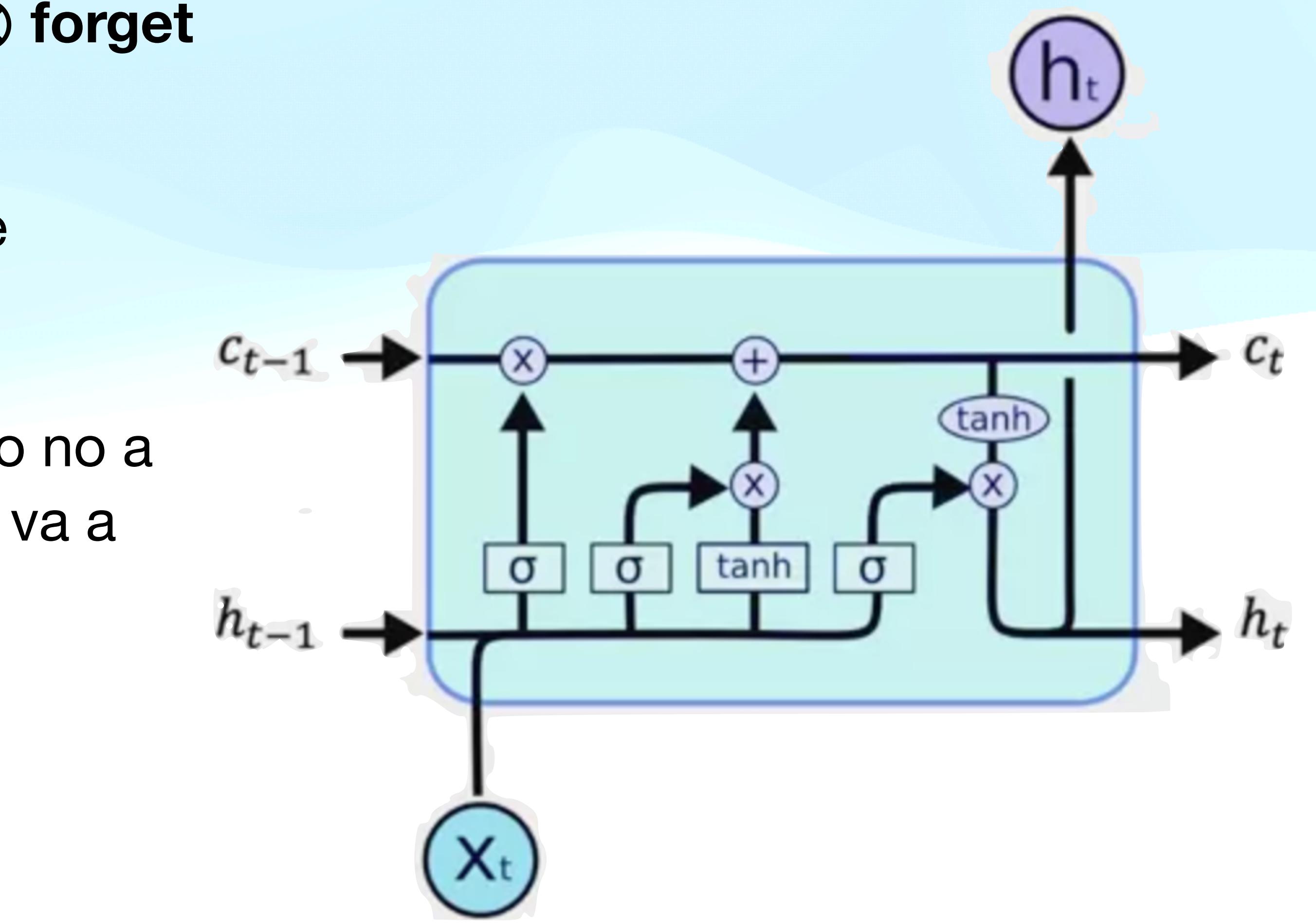
Input Gate: $\tanh()$ la información que se va o no a agregar y i_t la porción de información que se va a agregar.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$C'_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

Output Gate: $C_t = f_t \odot C_{t-1} + i_t \odot C'_t$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad h_t = o_t \tanh(C_t)$$

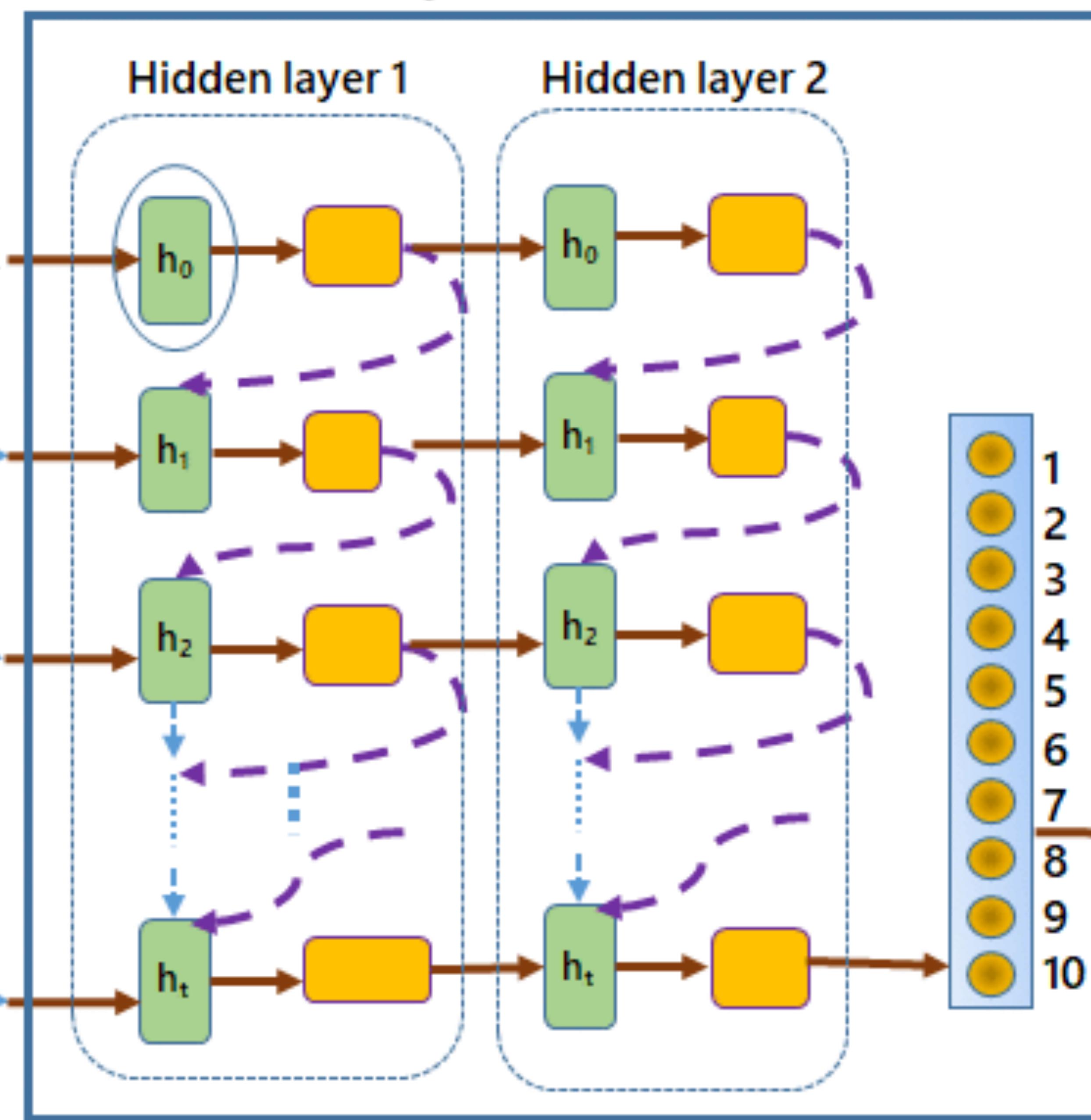
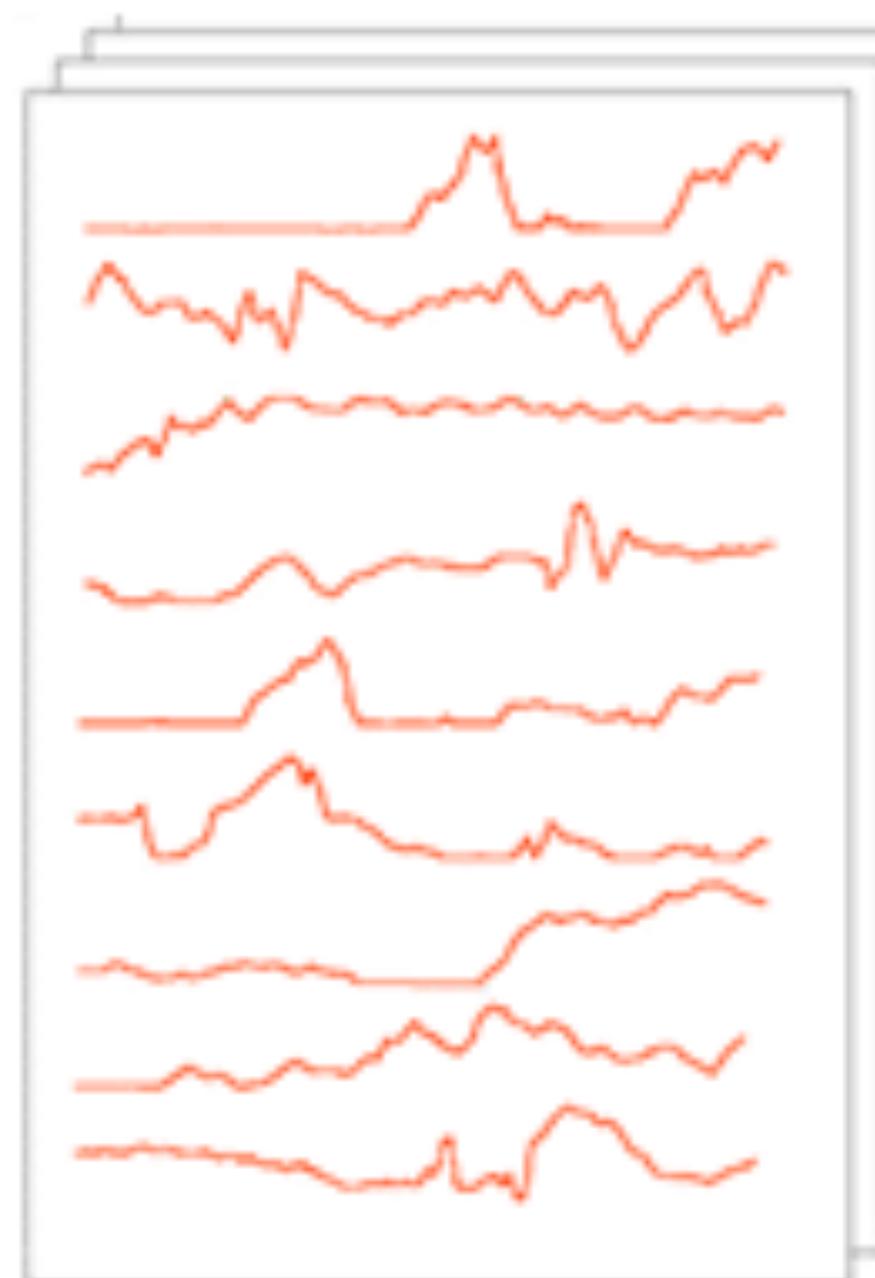


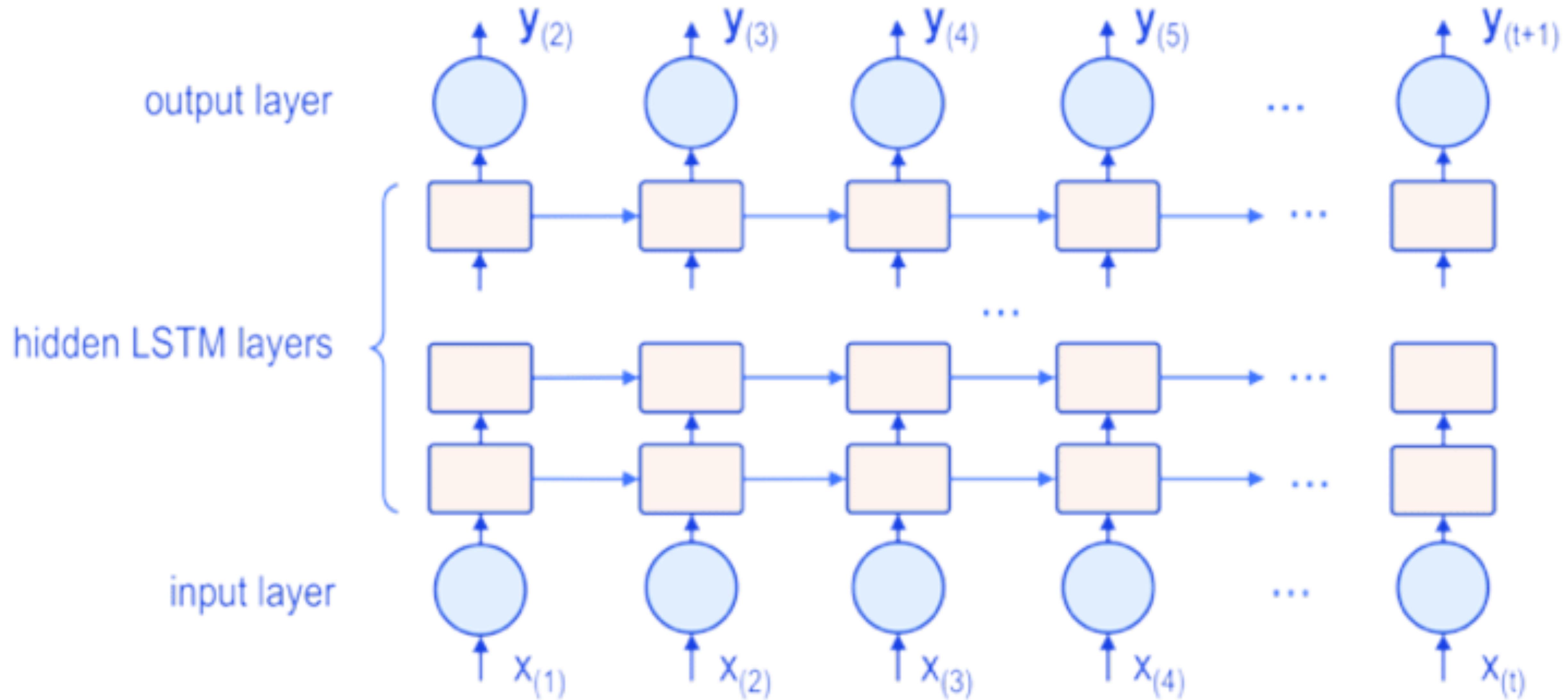
\odot Producto de Hadamard (Elemento a elemento)

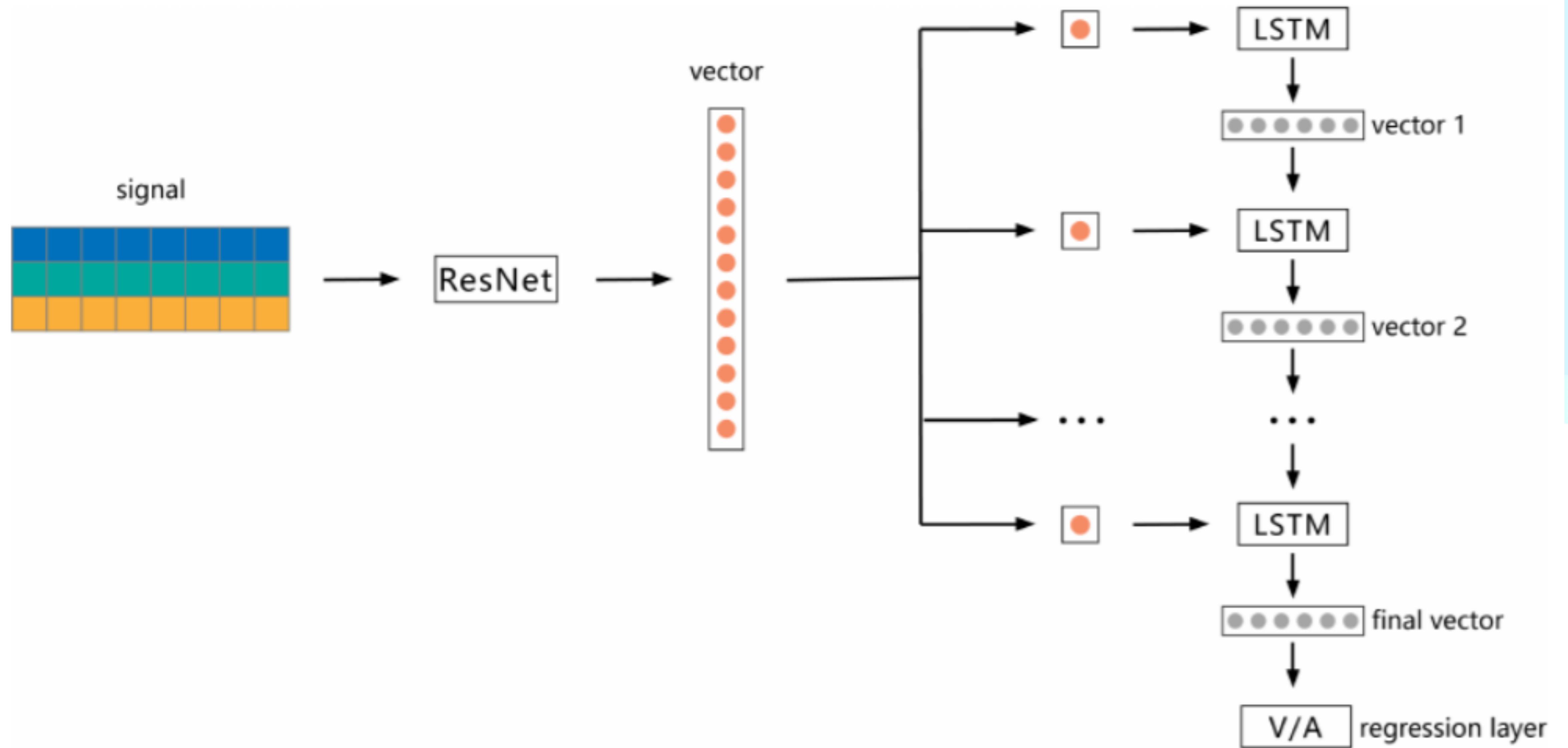
Deep LSTM

Input Data

$t_{\text{start}} \dots t_{\text{finish}}$

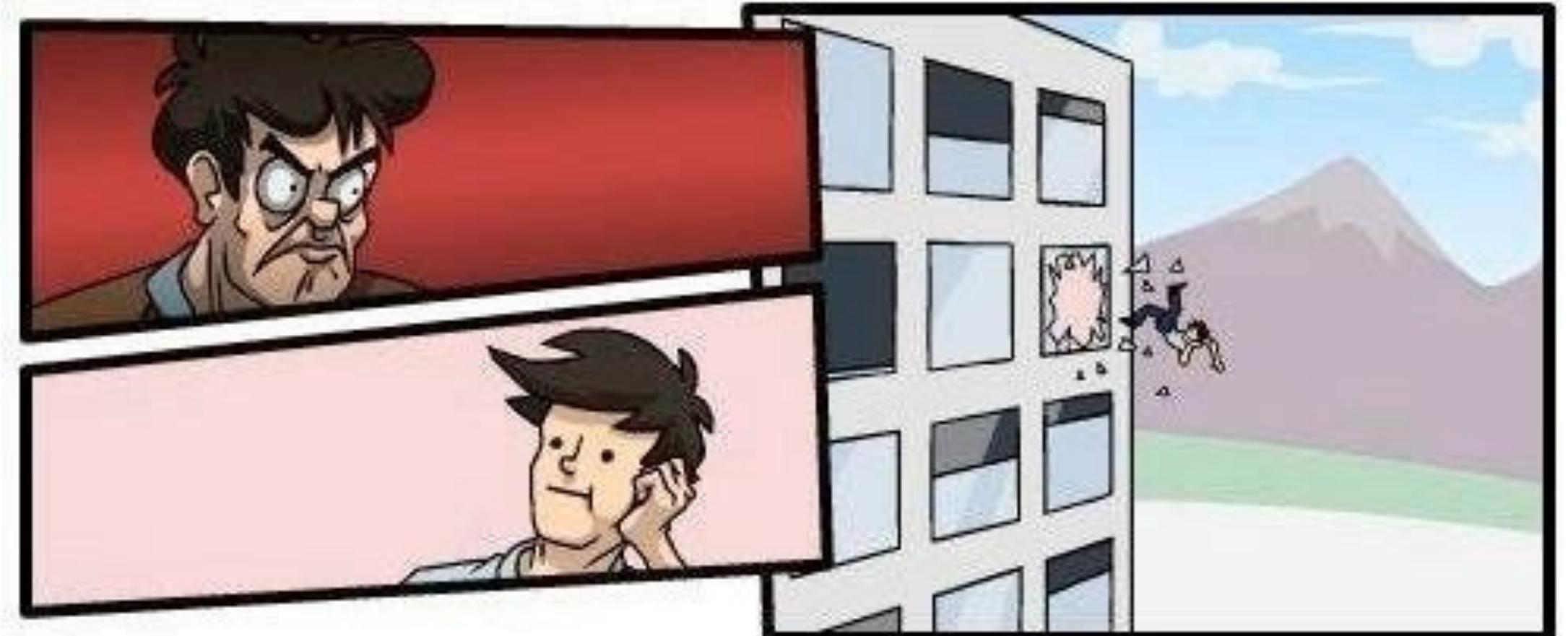
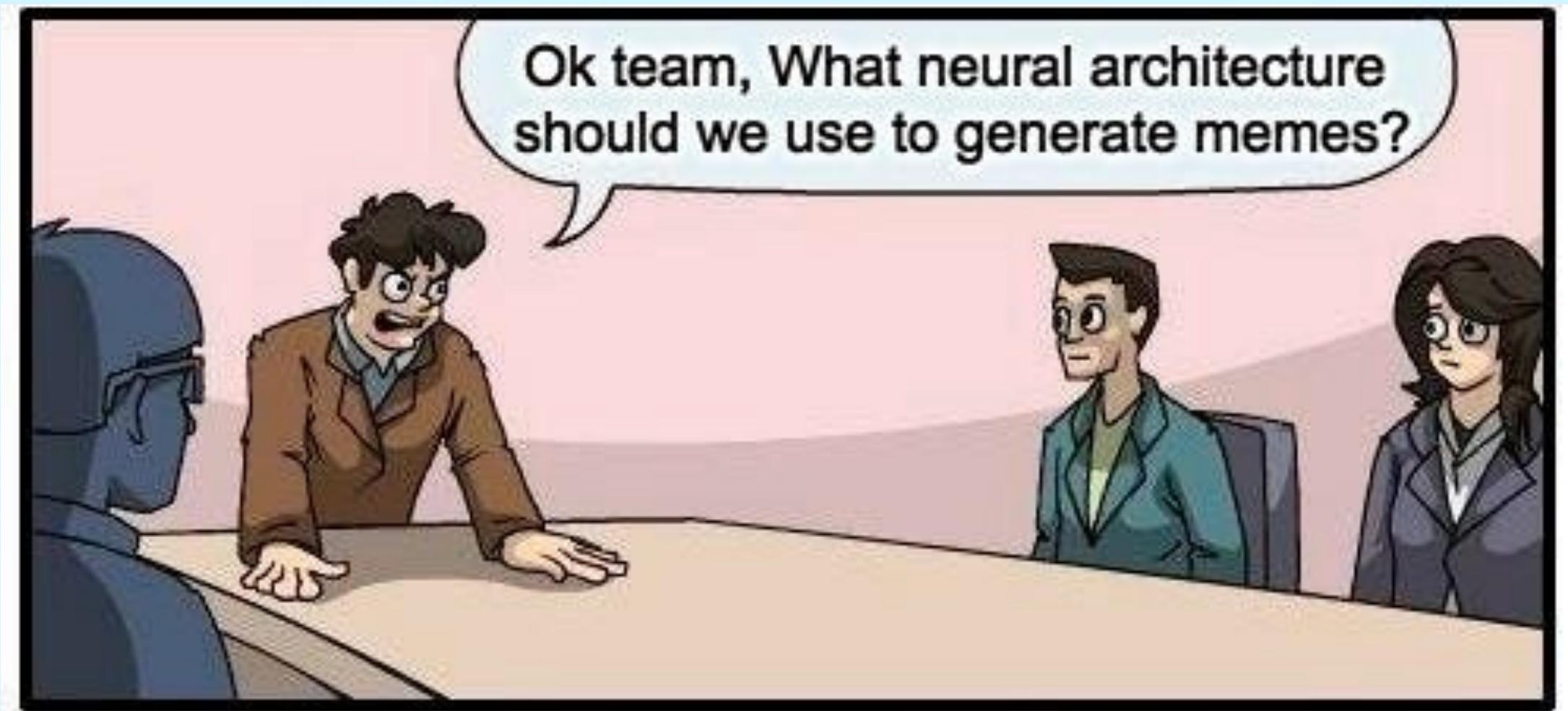






RESNET + LSTM

https://www.researchgate.net/publication/340715269_A_Systematic_Exploration_of_Deep_Neural_Networks_for_EDA-Based_Emotion_Recognition

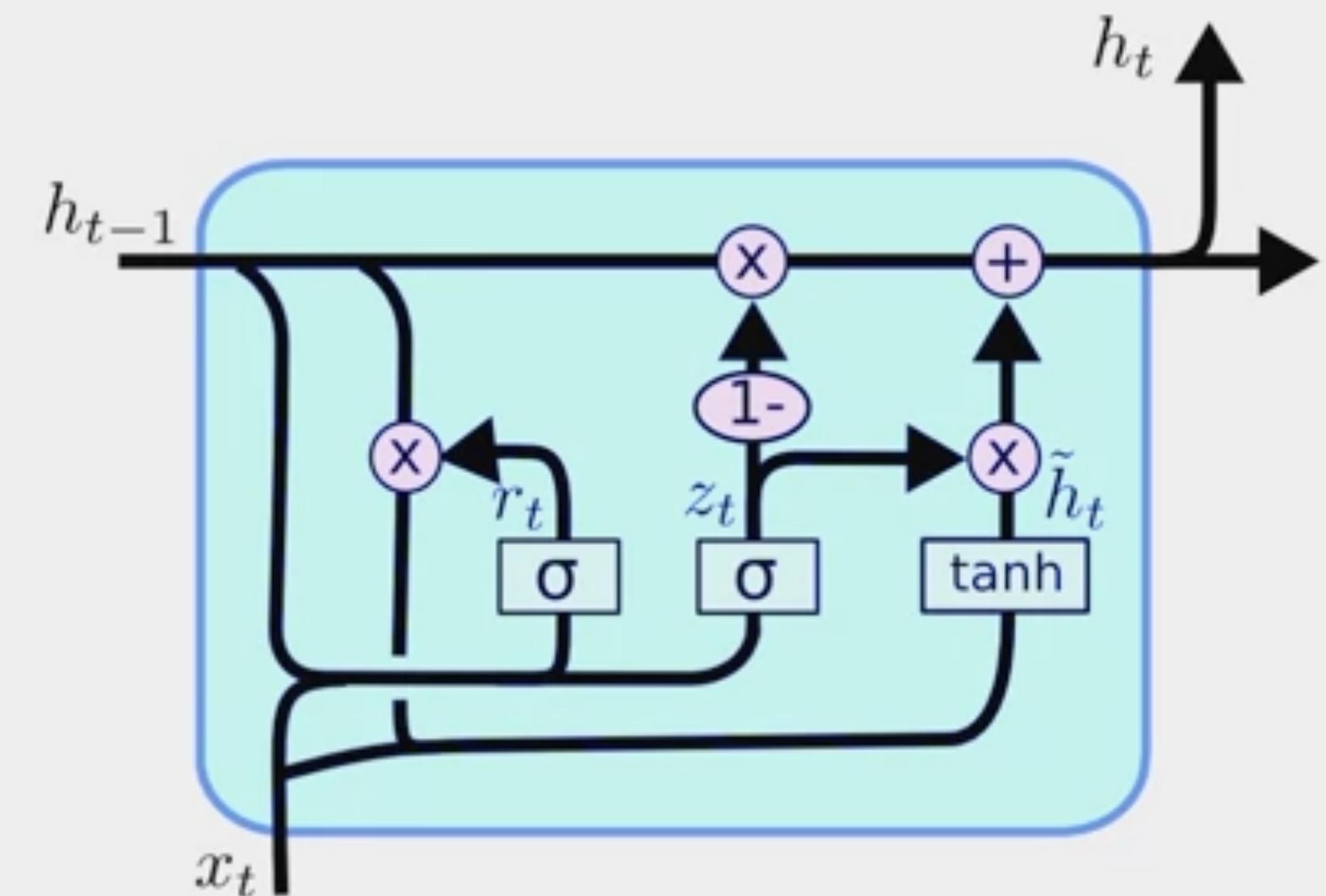




Unidad Recurrente Cerrada (GRU)

Simplificando las LSTM

- No hay vector de estado de celda c_t
- **Reset Gate:** $r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$
- **Update Gate:** $z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$
 $\tilde{h} = \tanh(W_h x_t + U_h(r_t \odot x_t) + b_h)$
 $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
- Rendimiento CASI tan bueno como LSTM pero con tiempos de entrenamiento mucho menores
- Falla cuando los patrones son muy complejos



**MY SEQ2SEQ IS TRAINING
SINCE 36 HOURS**

I SHOULD GET A GPU

imgflip.com

TRANSLATION

**QUESTION
ANSWERING**

Corporate needs you to find the differences
between this picture and this picture.

SEQ2SEQ

They're the same picture.

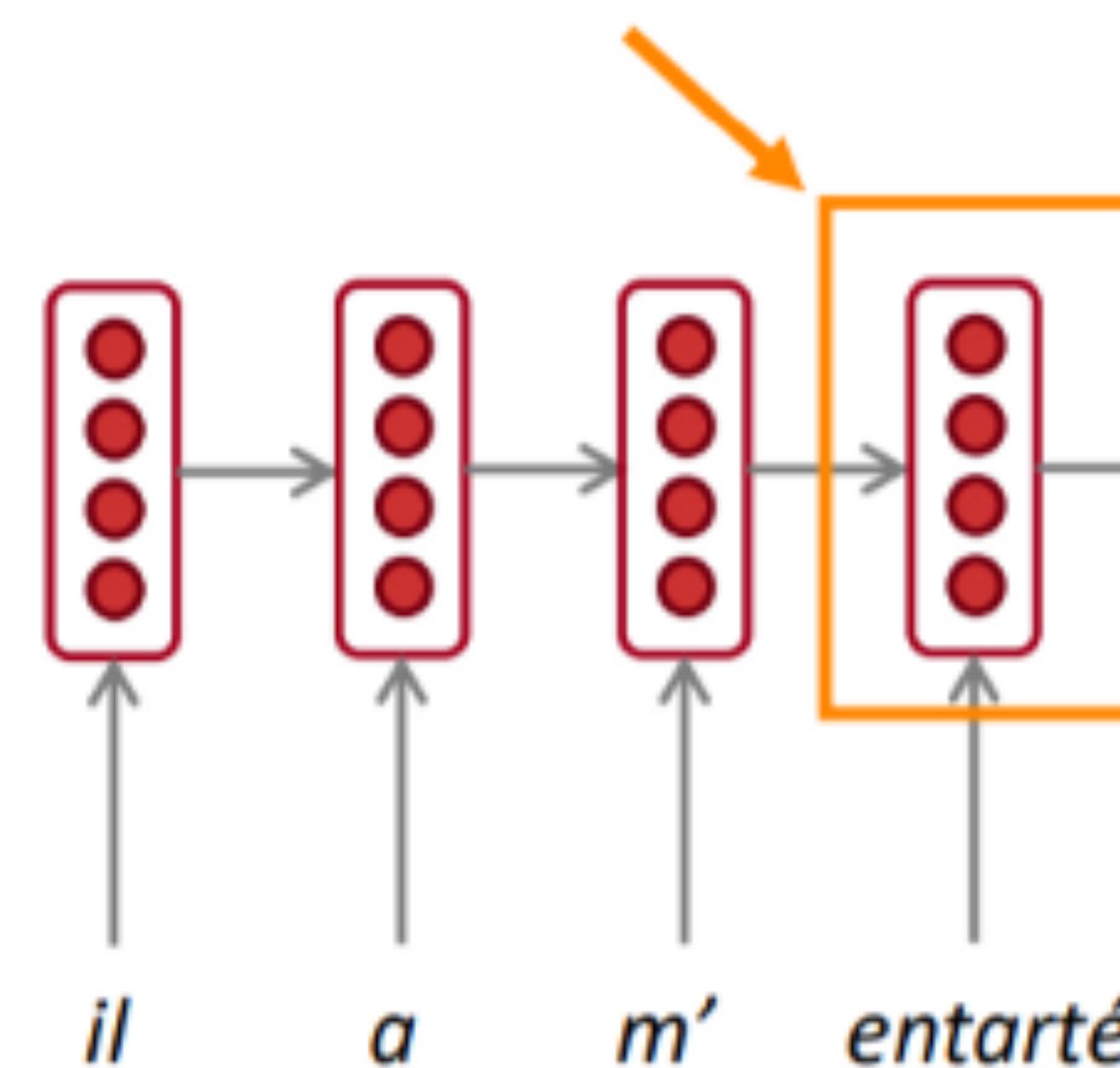
imgflip.com

The sequence-to-sequence model

Encoding of the source sentence.

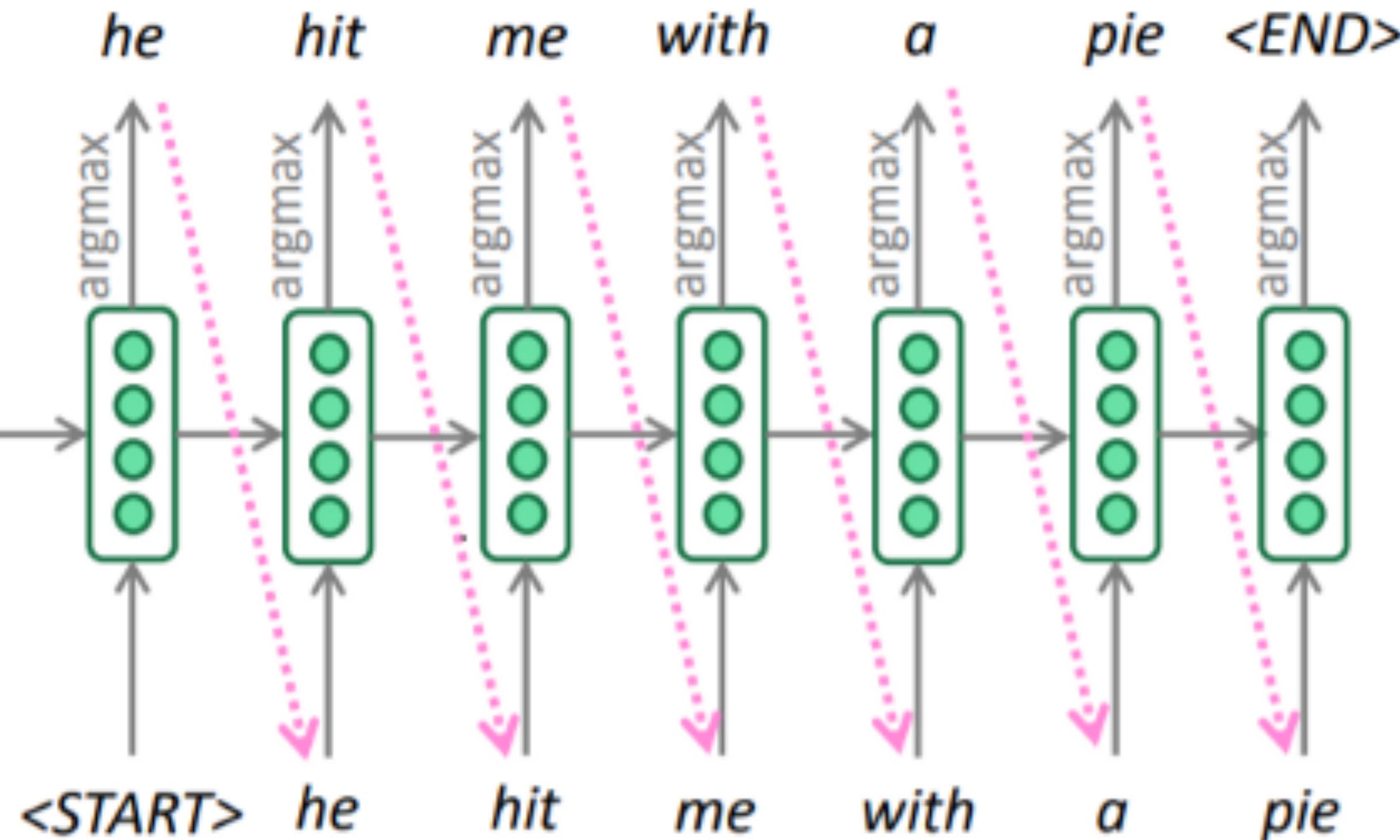
Provides initial hidden state
for Decoder RNN.

Encoder RNN



Source sentence (input)

Target sentence (output)

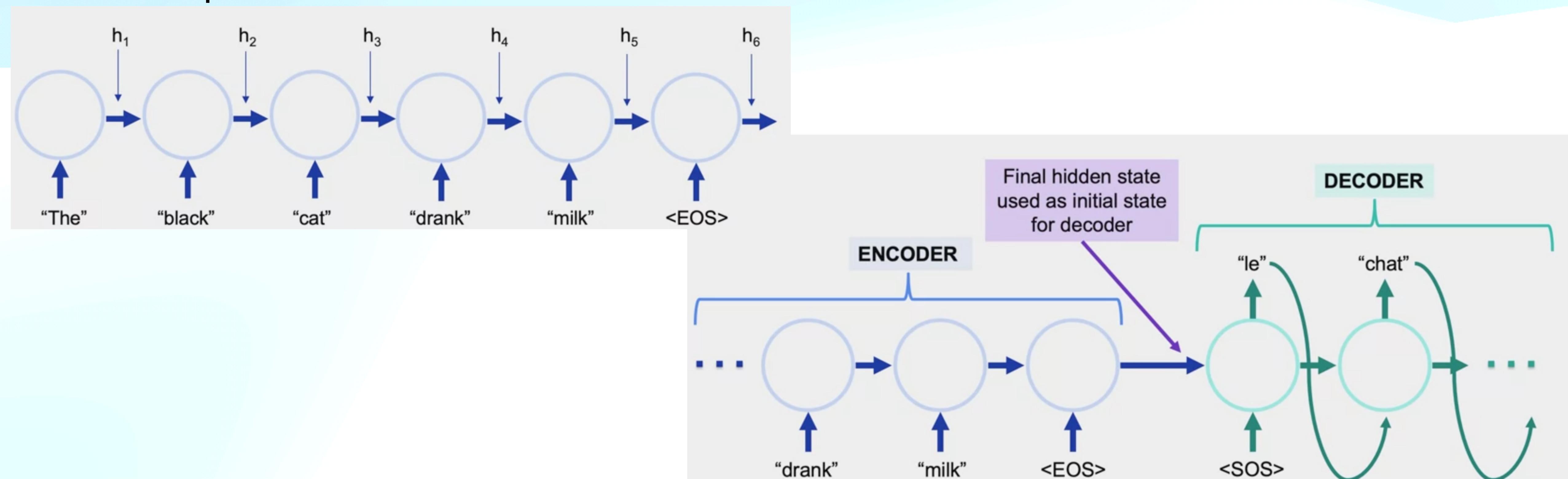


Decoder RNN

Seq2Seq

RNN Modelo Encoder-decoder

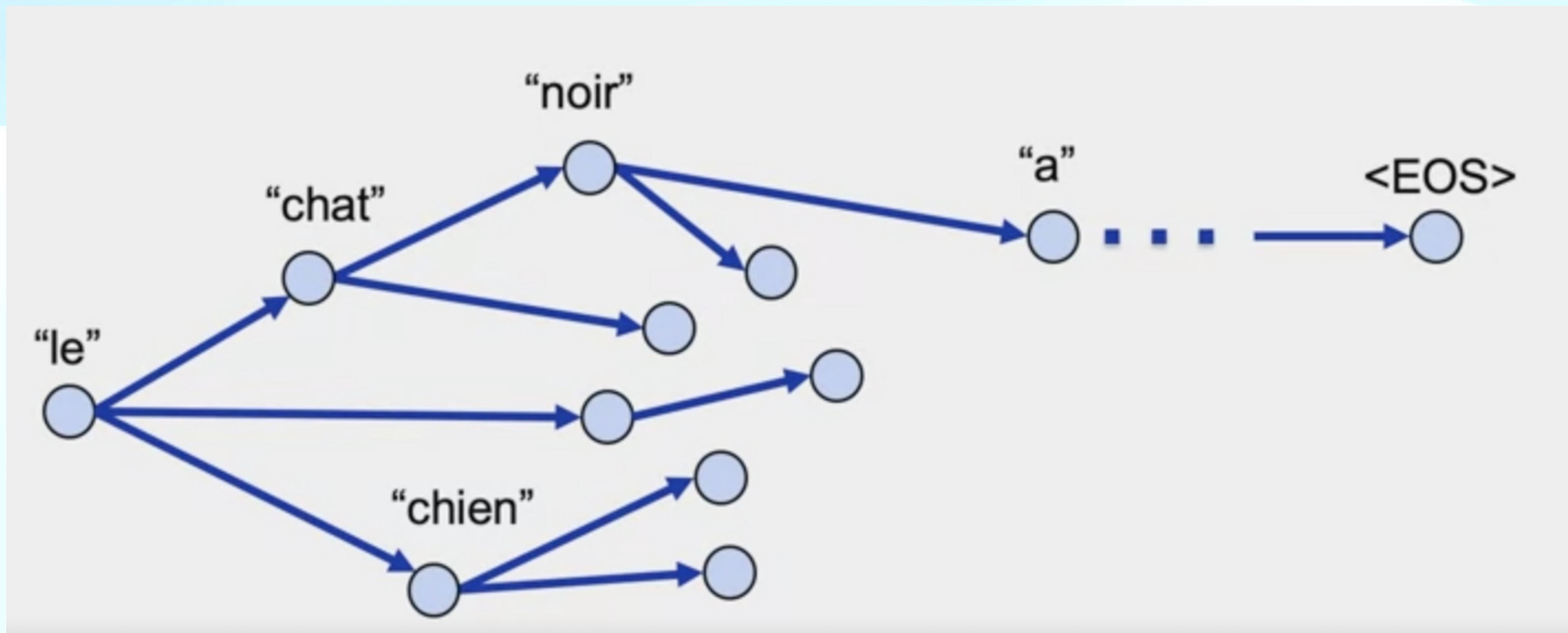
- Para predecir trayectorias (por eso es usado para predicción de texto). Por lo que si se equivoca en una predicción, la trayectoria será totalmente diferente a lo esperado 😞



Beam Search

Corrigiendo el problema del Seq2Seq

- Una solución a esto es el Seq2Seq con Beam Search, el cual produce múltiples trayectorias y evalúa cuál de ellas es la mejor (probabilísticamente)



Otro problema: Orden!

Al pasar un solo estado final el decoder no tiene forma de ordenar o reordenar las salidas

- Cuando se piensa en traducción el orden de las palabras cambia, por lo que se debe analizar cada una de las salidas y darle más peso a la salida del encoder que sea más similar (semánticamente) al estado del decoder en el que se está.
- $S(i, j)$ mide la similitud entre el estado i del encoder y el estado j del decoder
- $S(i, j)$ le da peso a las diferentes hidden states para dar una mejor predicción en el decoder

