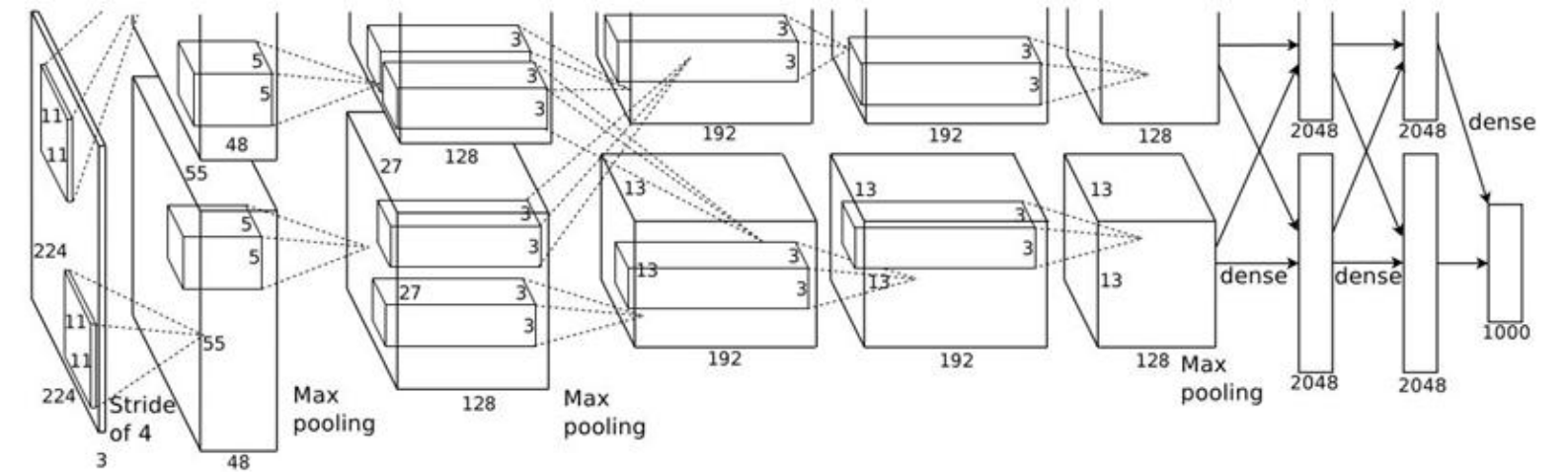
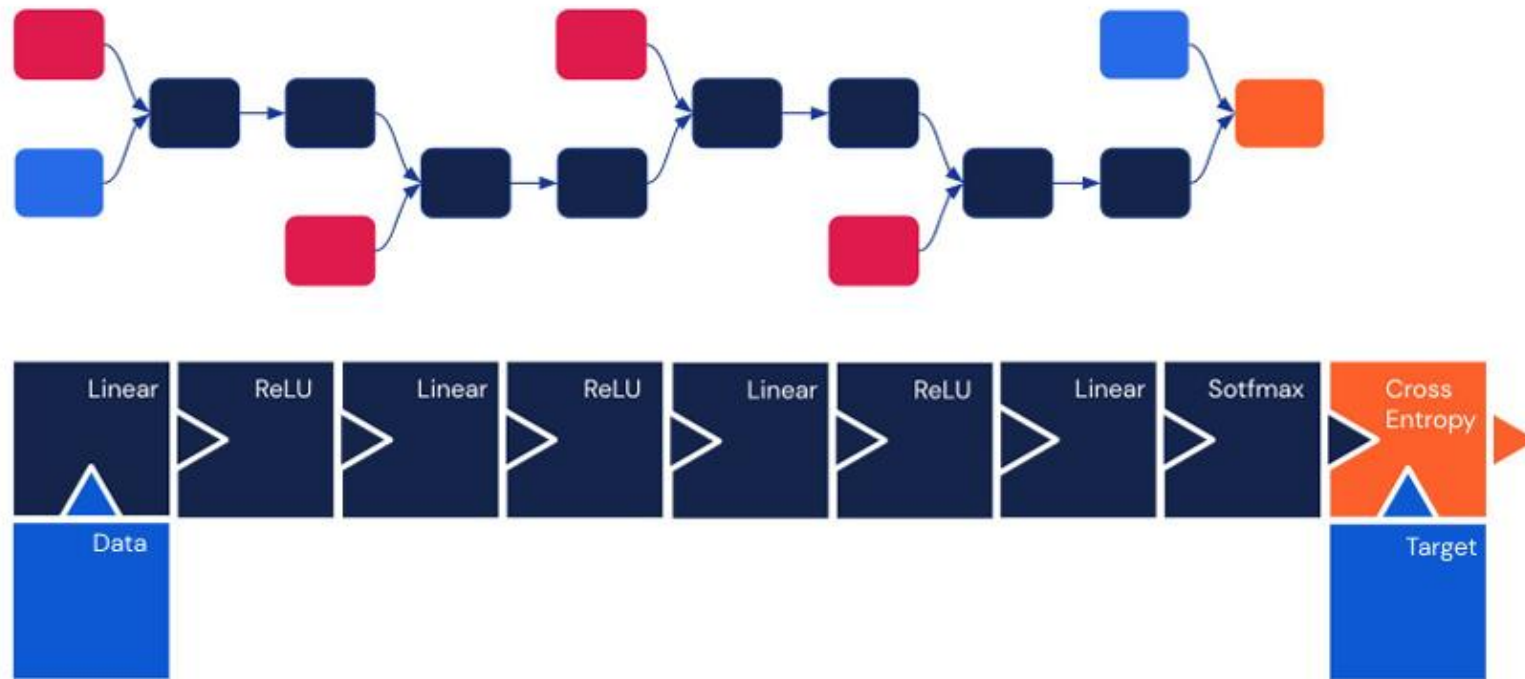


Sesión 1.2

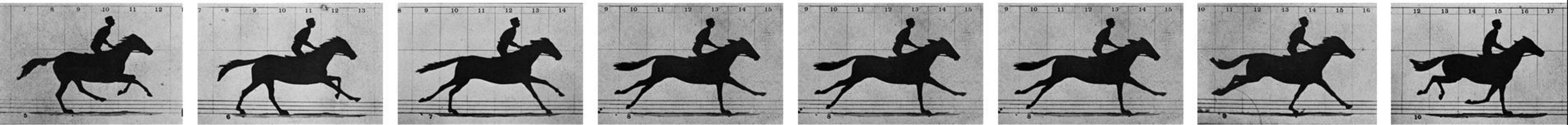
Redes recurrentes

LSTM, GRU

Deep Models



Secue*ncias*



Colecciones de datos donde:

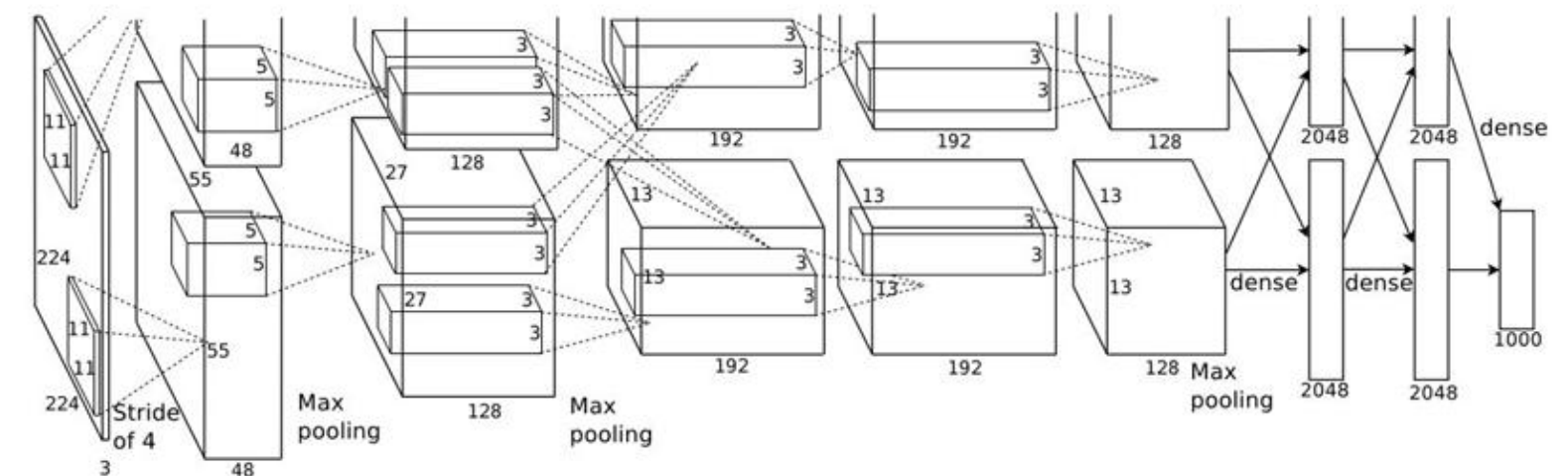
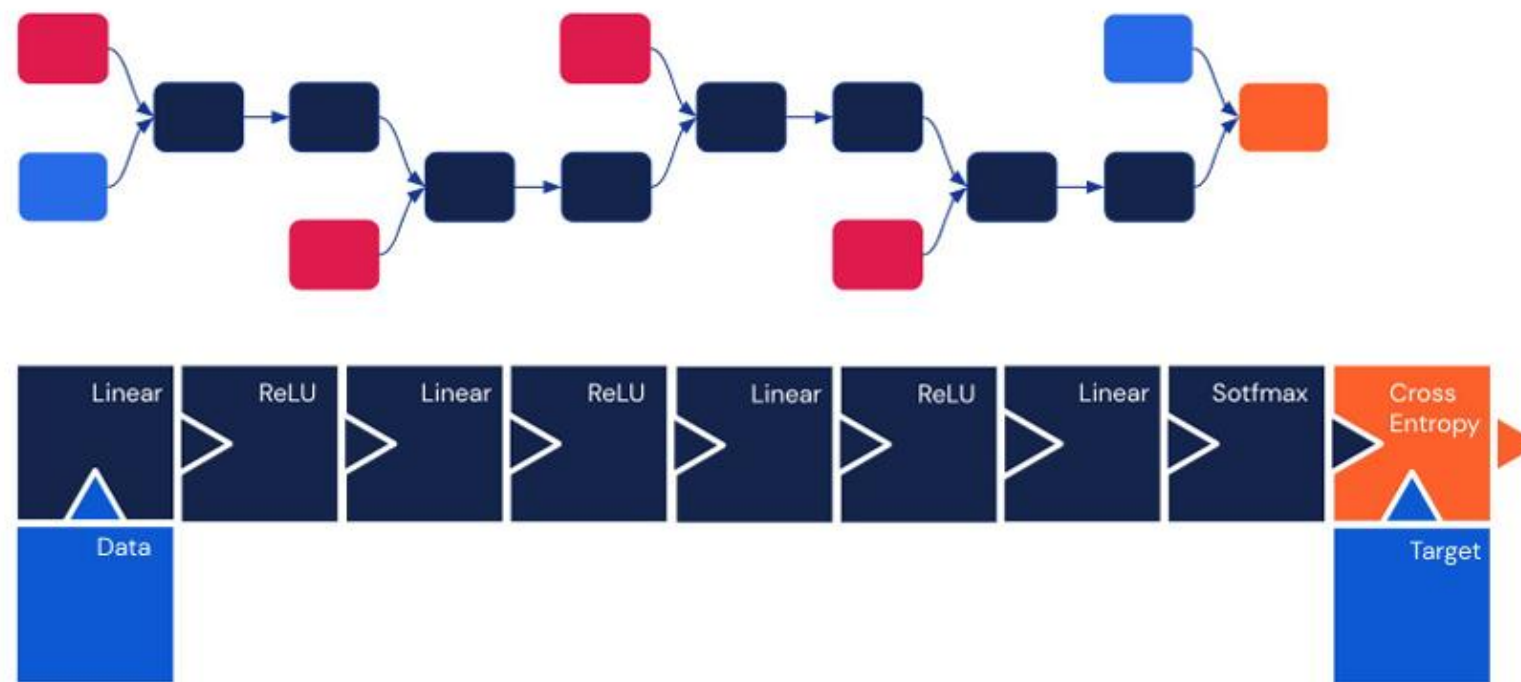
- Los elementos pueden **repetirse**
- El **orden** importa
- De longitud **variable** (potencialmente infinita)



Modelar *Secuencias*

- Los elementos pueden **repetirse**
- El **orden** importa
- De longitud **variable** (potencialmente infinita)

Los modelos previos no funcionan bien con datos secuenciales



Secue*ncias*

"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

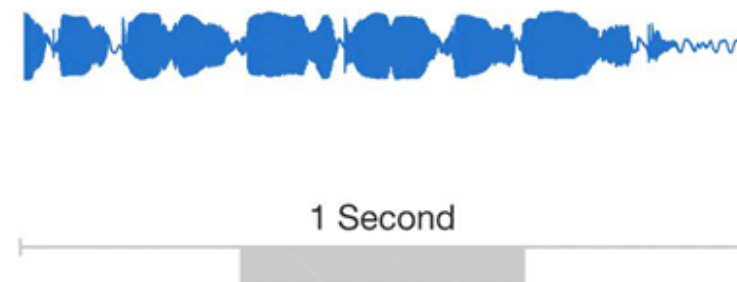
Palabras, letras



Secue*ncias*

“Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!”

Palabras, letras



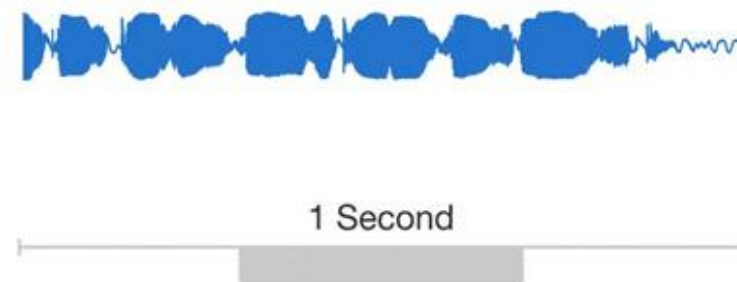
Habla



Secue*ncias*

"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

Palabras, letras



Habla



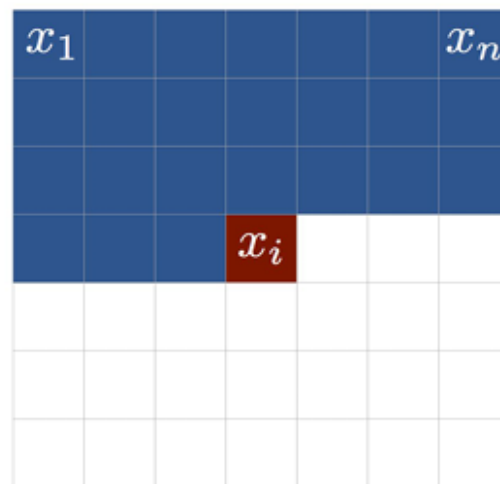
Video



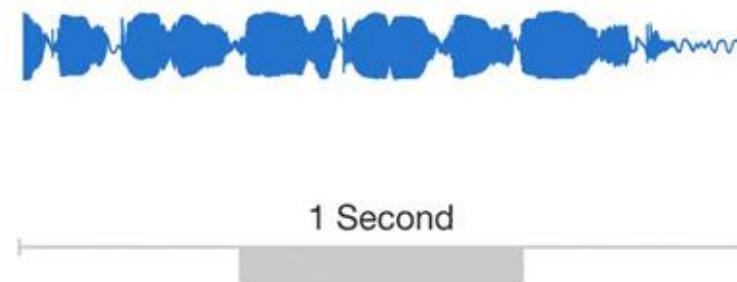
Secue*ncias*

"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

Palabras, letras



Imágenes



Habla



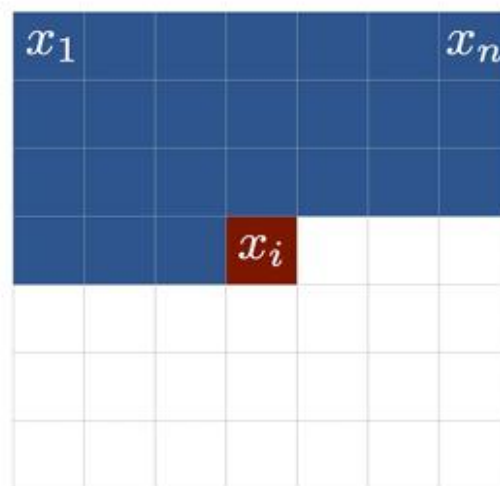
Video



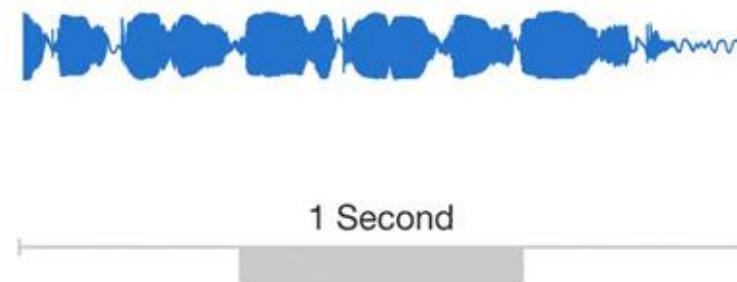
Secuencias

"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

Palabras, letras



Imágenes



Habla



Video

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def forward_backward_prop(w, T):
5     hs = [0.5]
6     for _ in range(T):
7         hs.append(np.tanh(w*hs[-1]))
8
9     dh = 1
10    for t in range(T):
11        dh = (1-hs[-1-t]**2) * w * dh
12
13    return hs[-1], dh
14
15 T = 10 # sequence length
16 wlim = 4 #limit of interval over weights w
17
18 results = []
19 ws = np.linspace(-wlim, wlim, 1000)
20 for w in ws:
21     results.append(forward_backward_prop(w, T))
22
23 plt.plot(ws, [r[0] for r in results], label='RNN state')
24 plt.plot(ws, [r[1] for r in results], label='Gradients')
  
```

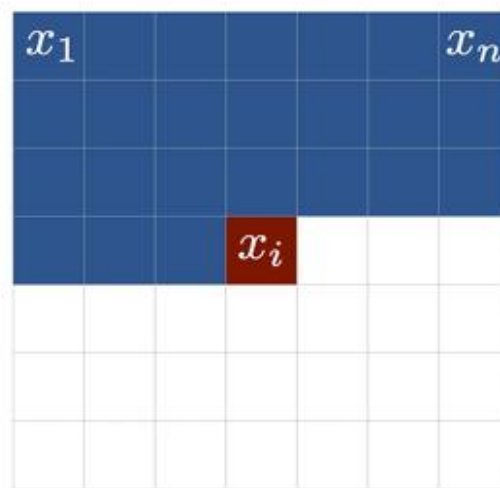
Programas



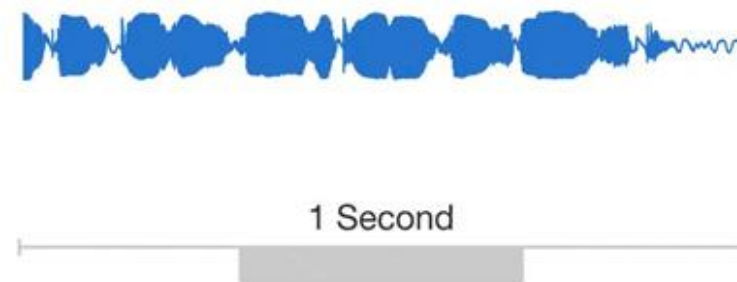
Secuencias

"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

Palabras, letras



Imágenes



Habla

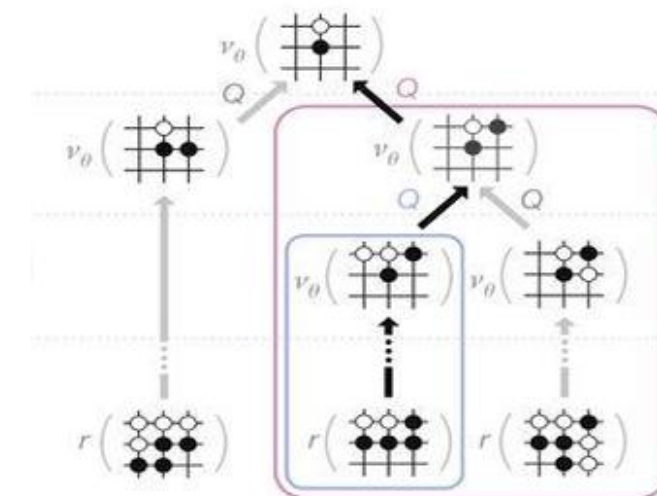
```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def forward_backward_prop(w, T):
5     hs = [0.5]
6     for _ in range(T):
7         hs.append(np.tanh(w*hs[-1]))
8
9     dh = 1
10    for t in range(T):
11        dh = (1-hs[-1-t]**2) * w * dh
12
13    return hs[-1], dh
14
15 T = 10 # sequence length
16 wlim = 4 #limit of interval over weights w
17
18 results = []
19 ws = np.linspace(-wlim, wlim, 1000)
20 for w in ws:
21     results.append(forward_backward_prop(w, T))
22
23 plt.plot(ws, [r[0] for r in results], label='RNN state')
24 plt.plot(ws, [r[1] for r in results], label='Gradients')
  
```

Programas

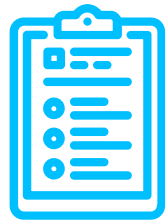


Video



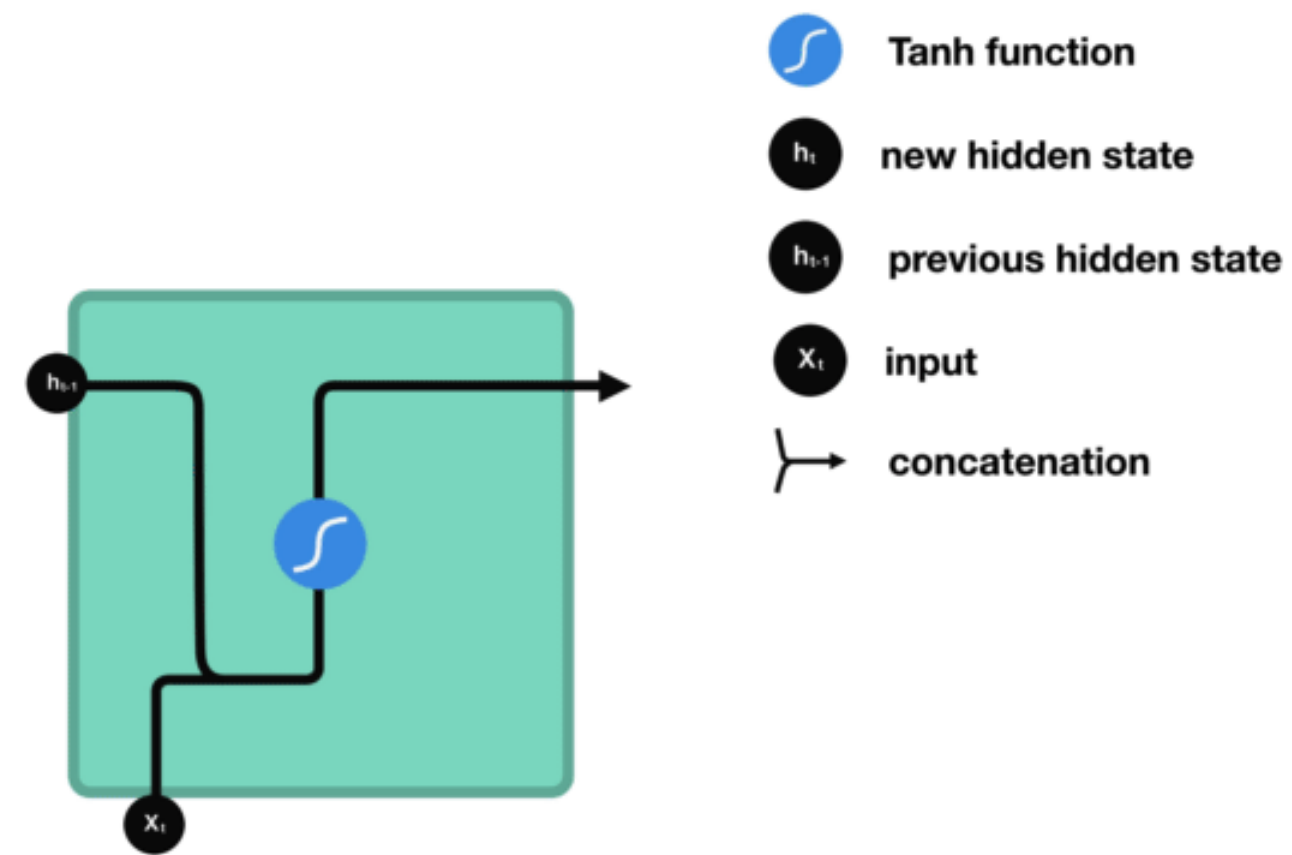
Toma de decisiones

1.

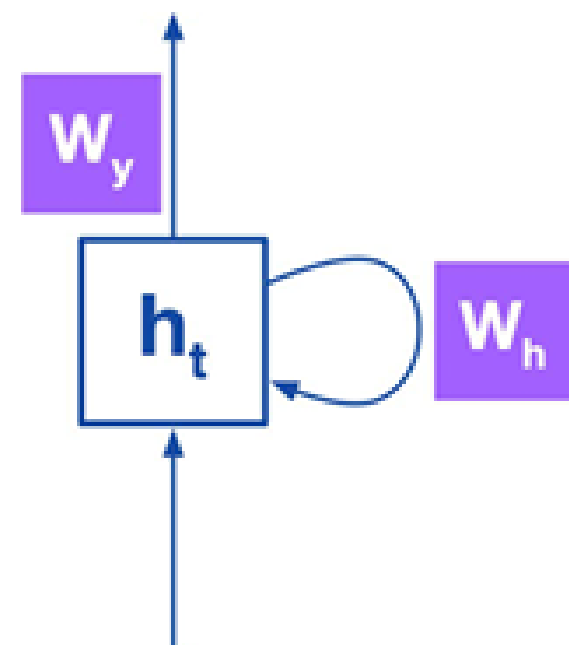


RNN *Tradicionales*

Arquitectura *de las RNN*



Recurrent Neural Network



Alguna función
con parámetro W_h

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

Nuevo estado Estado previo Nueva entrada

Alguna función
con parámetro W_y

$$y_t = f_{W_y}(h_t)$$

Salida actual

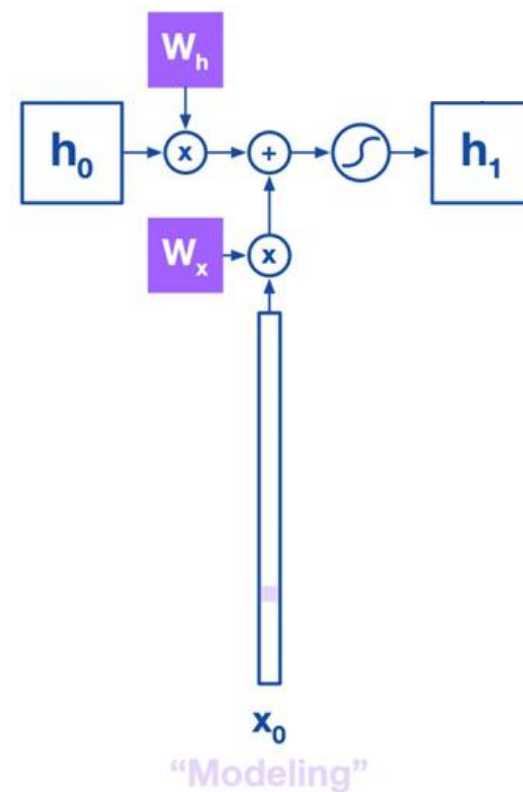
Nuevo estado



Recurrent Neural Network

Se recibe la primera palabra de entrada x_0 .

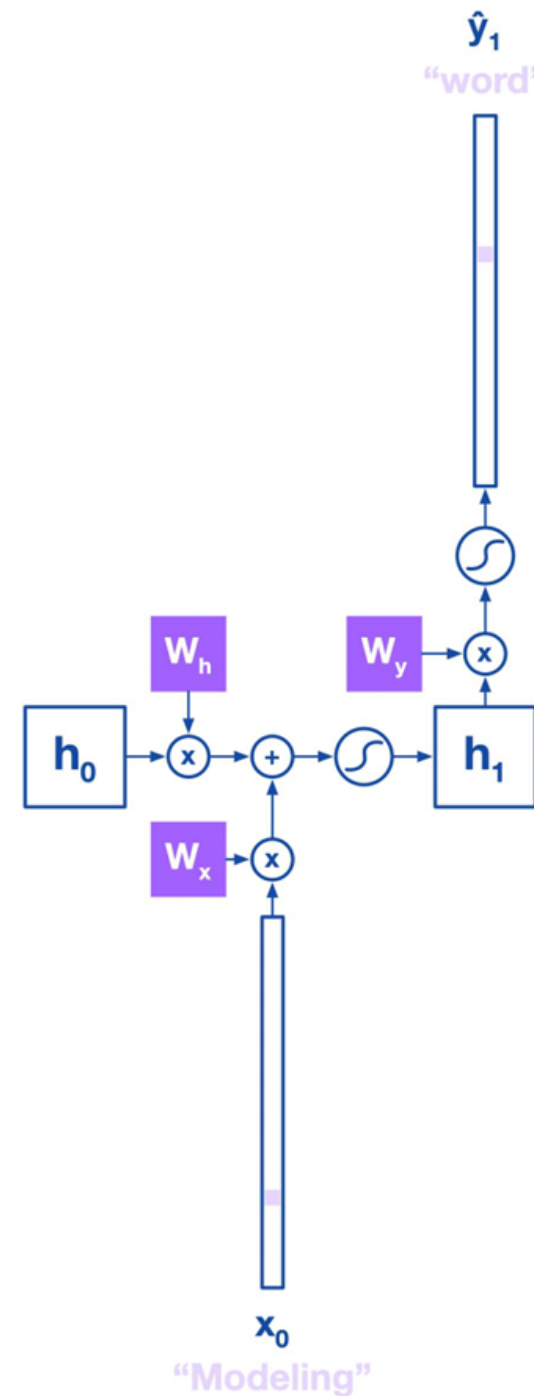
$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$



"Modeling"



Recurrent Neural Network

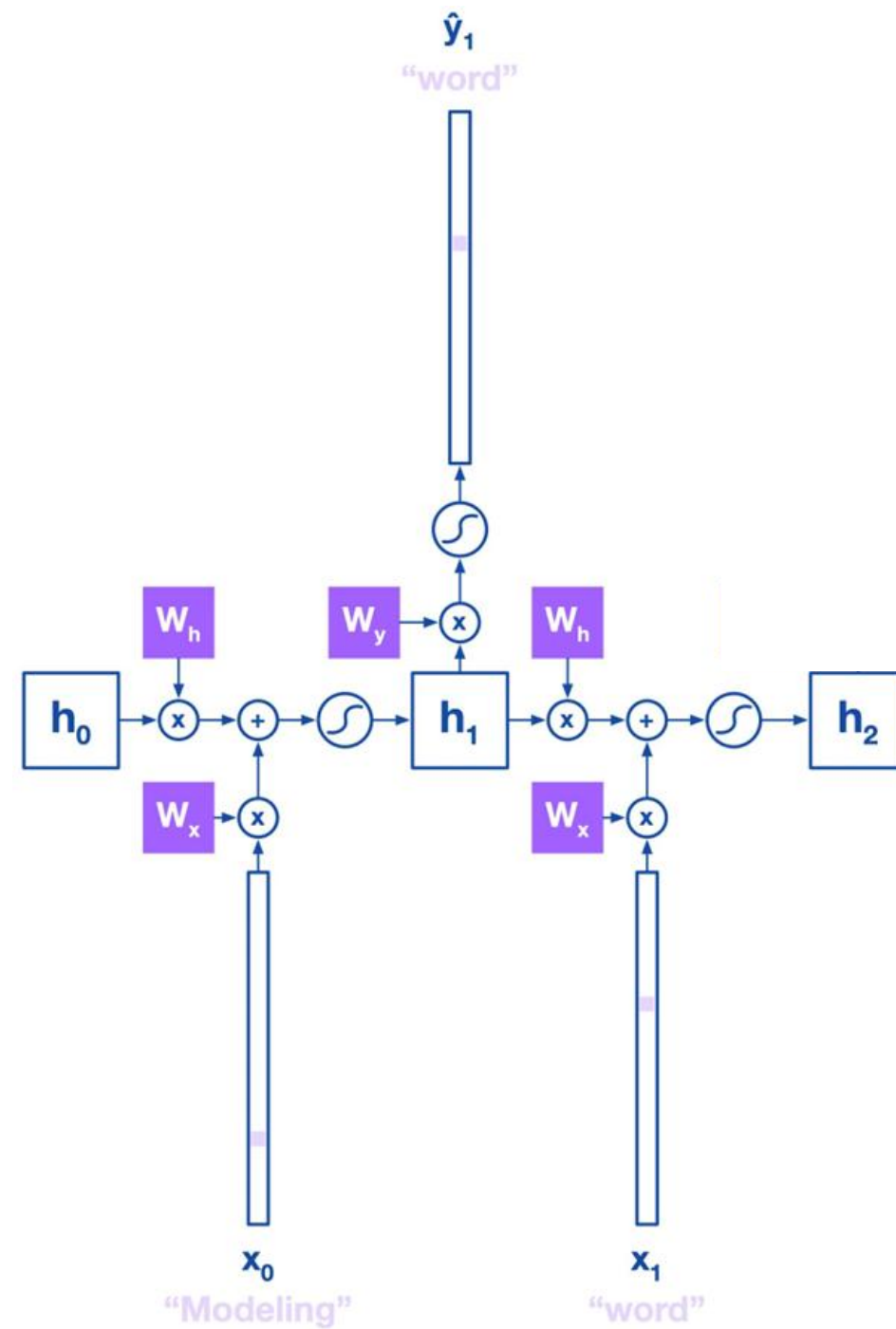


RNN predicen la salida y_1 (la siguiente palabra) a partir del estado h_1 .

$$y_t = \text{Softmax}(W_y h_t)$$

Softmax genera una distribución de probabilidad entre todas las palabras posibles.

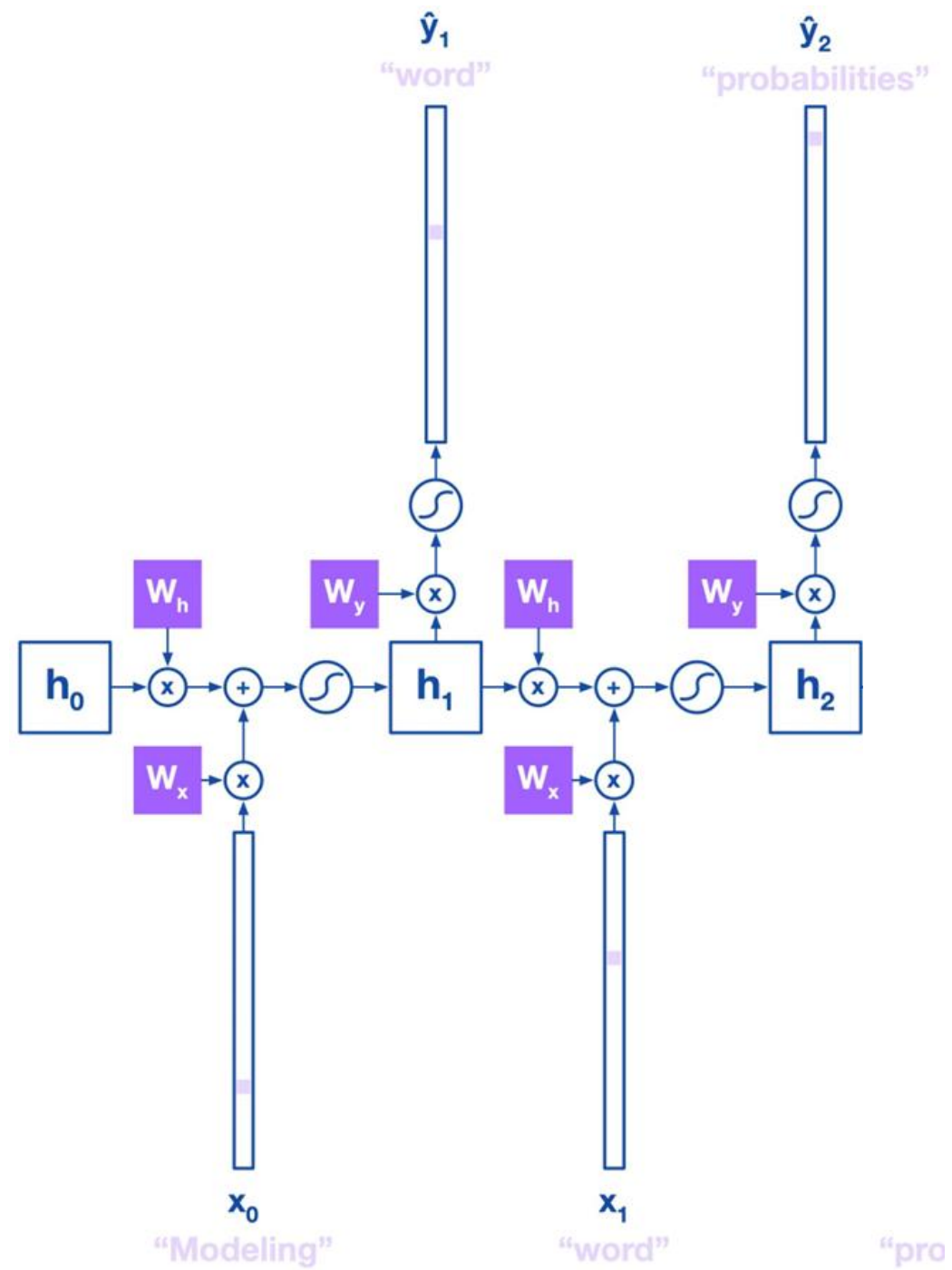
Recurrent Neural Network



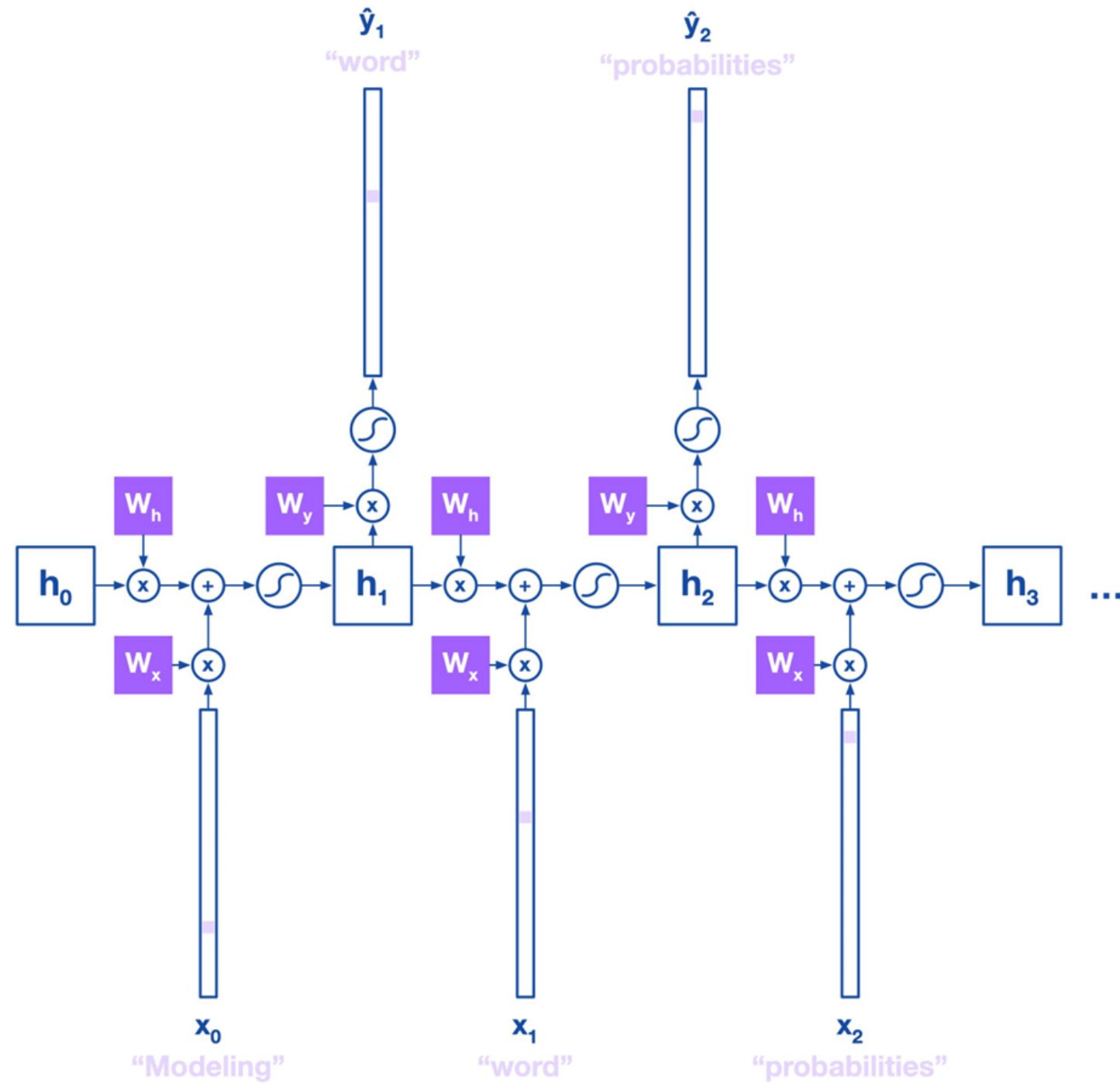
Siguiente palabra de la frase x_1 como entrada



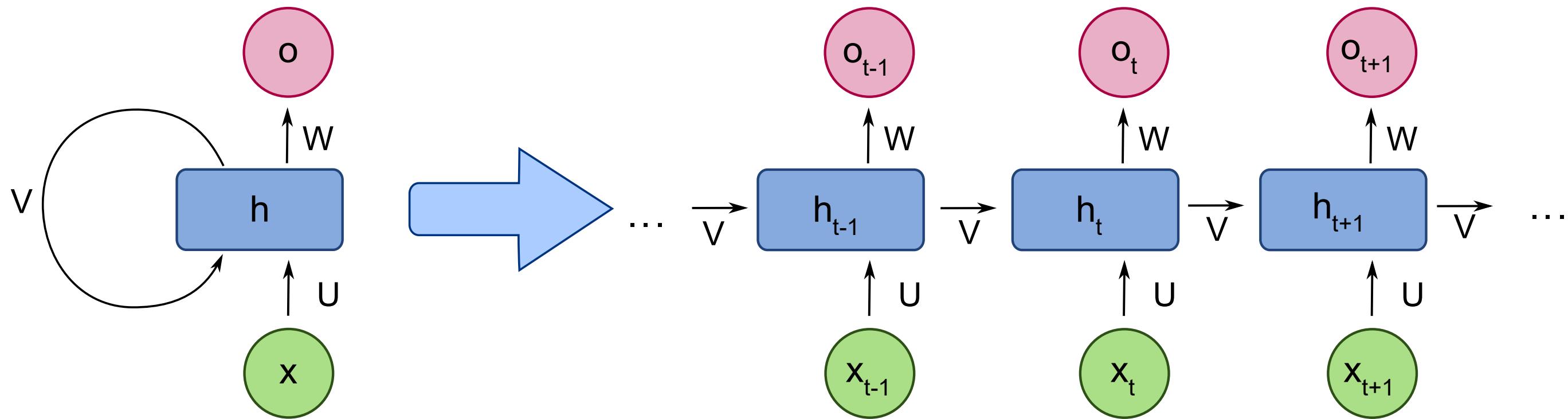
Recurrent Neural Network



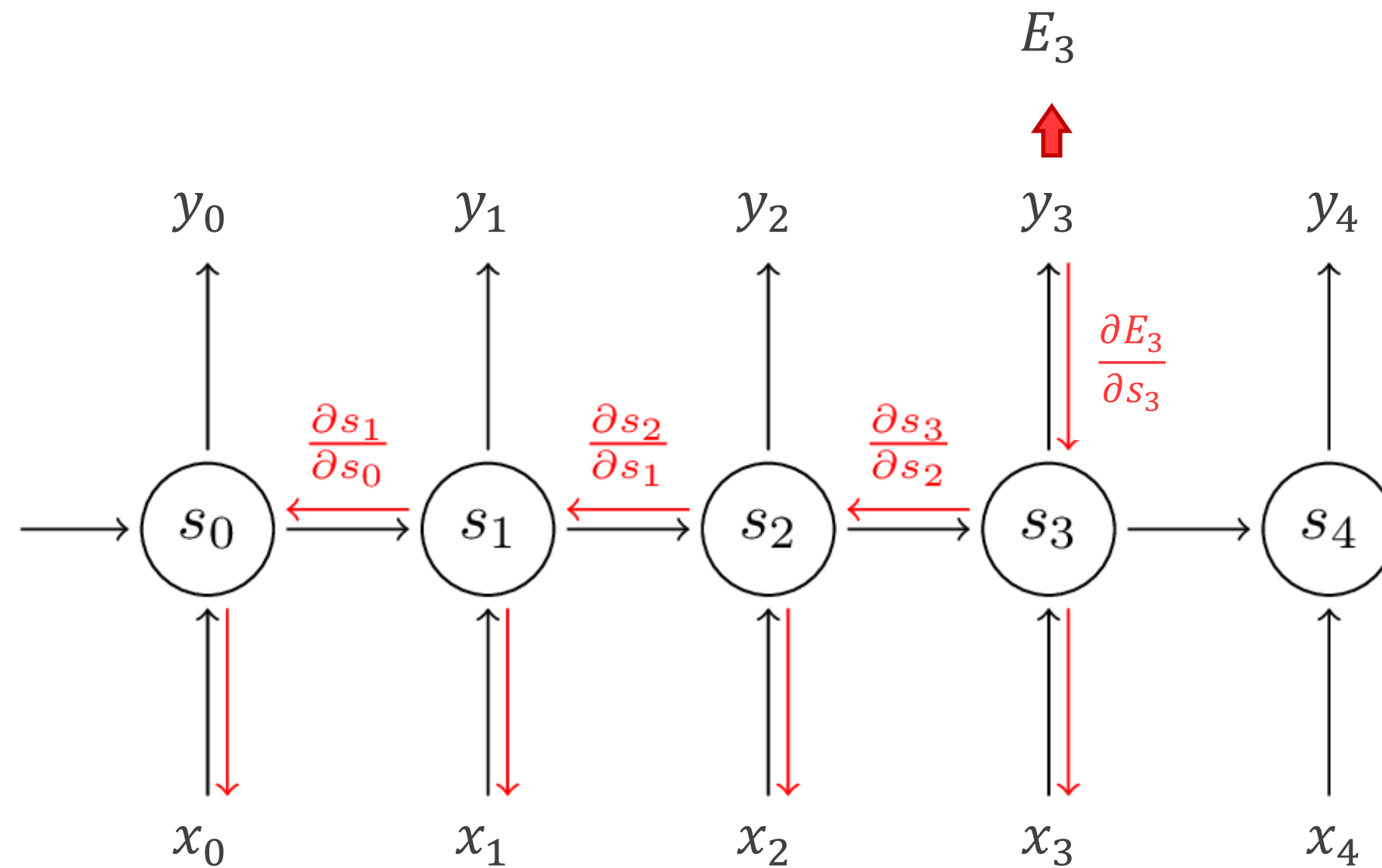
Recurrent Neural Network



Recurrent Neural Network



Vanilla RNN *Gradient Flow*

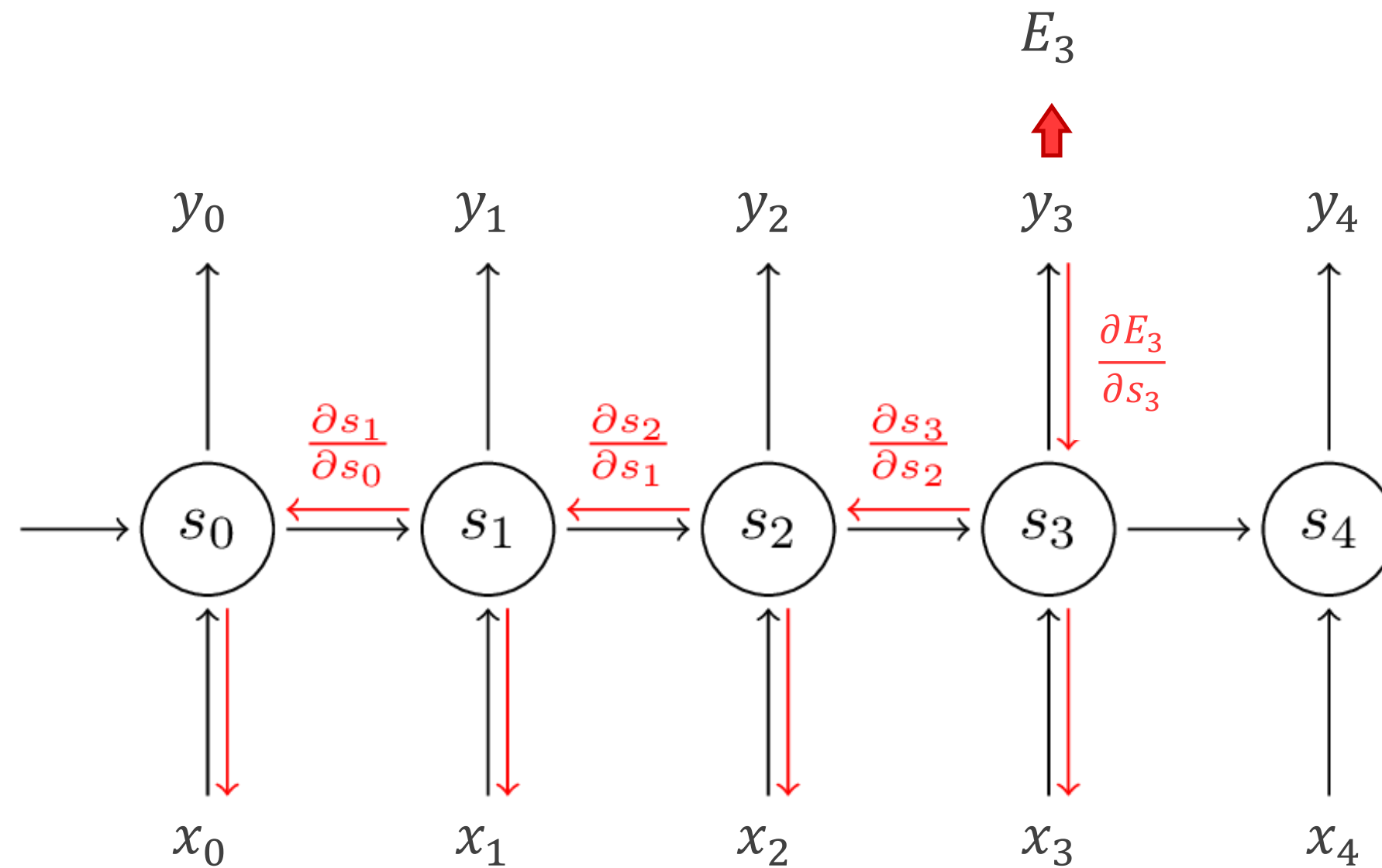


Función de error:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$



Vanilla RNN *Gradient Flow*



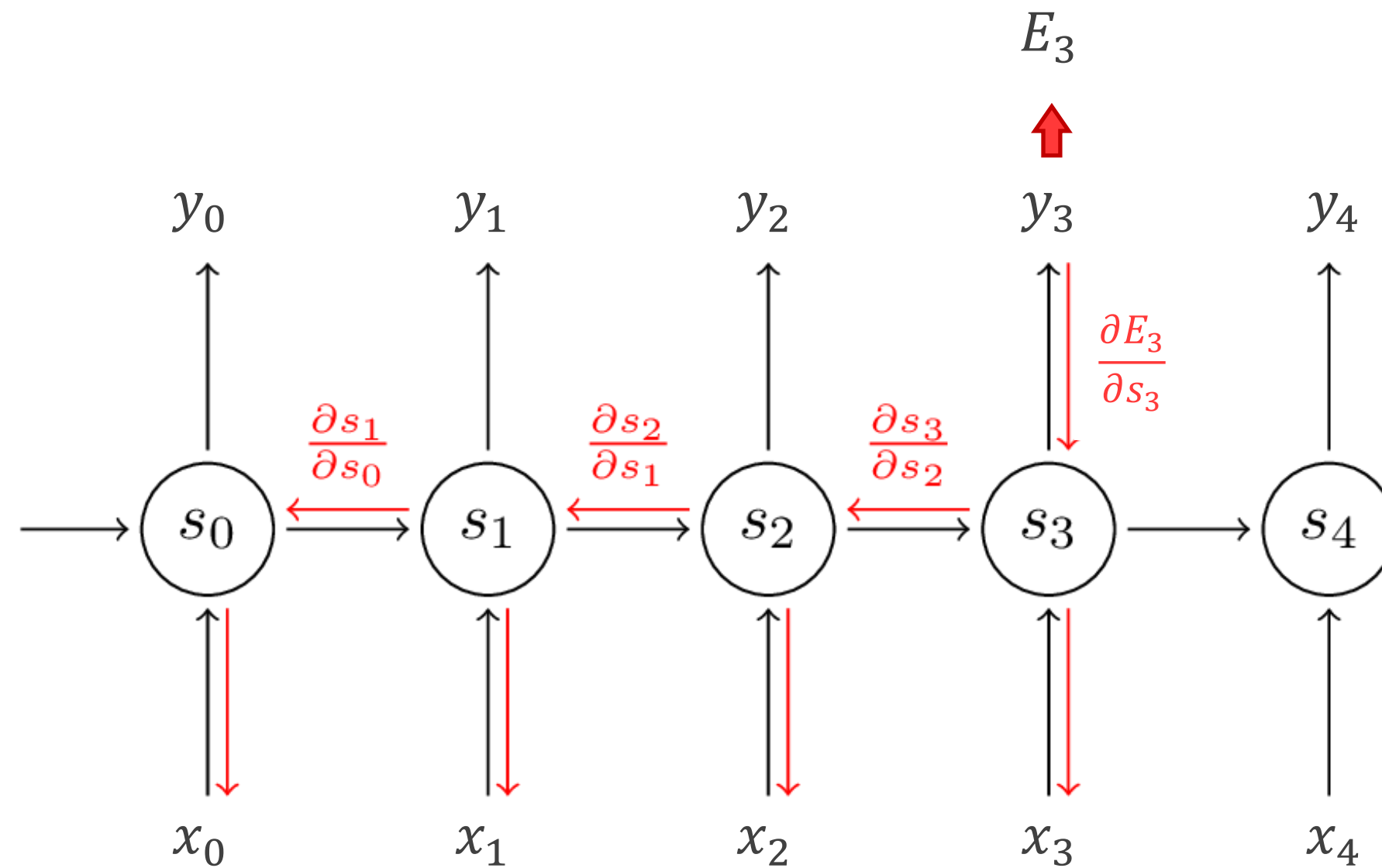
Función de error:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

Cálculo del gradiente:

$$\frac{\partial E}{\partial W} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial W}$$

Vanilla RNN *Gradient Flow*



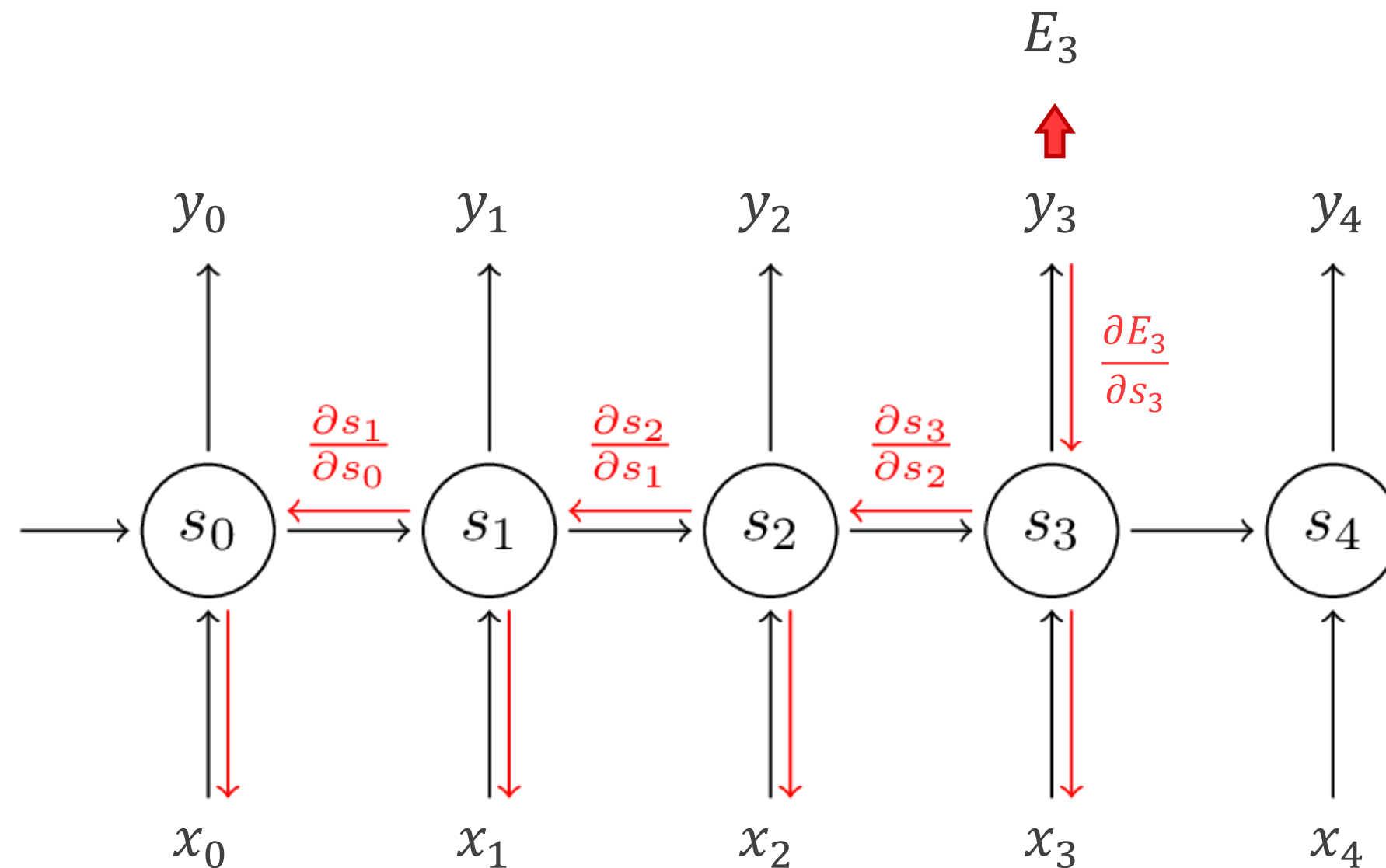
Función de error:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

Cálculo del gradiente:

$$\frac{\partial E}{\partial W} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial W} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial x_k}{\partial W} \right)$$

Vanilla RNN *Gradient Flow*



Función de error:

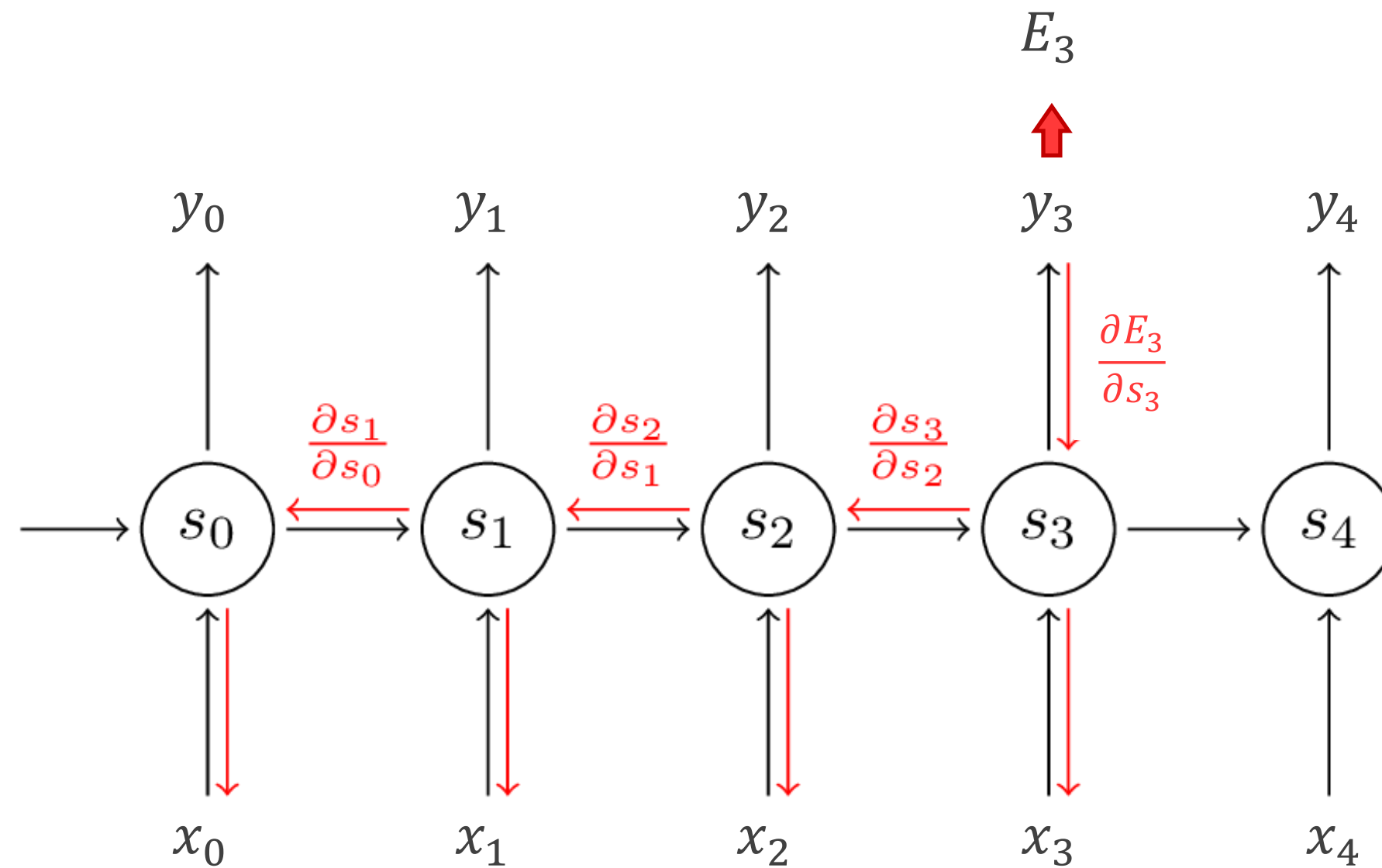
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

Cálculo del gradiente:

$$\frac{\partial E}{\partial W} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial W} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial x_k}{\partial W} \right)$$

$$\frac{\partial x_t}{\partial x_k} = \prod_{t \geq i > k} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{t \geq i > k} W^T \text{diag}(\sigma(x_{i-1}))$$

Vanilla RNN *Gradient Flow*



Función de error:

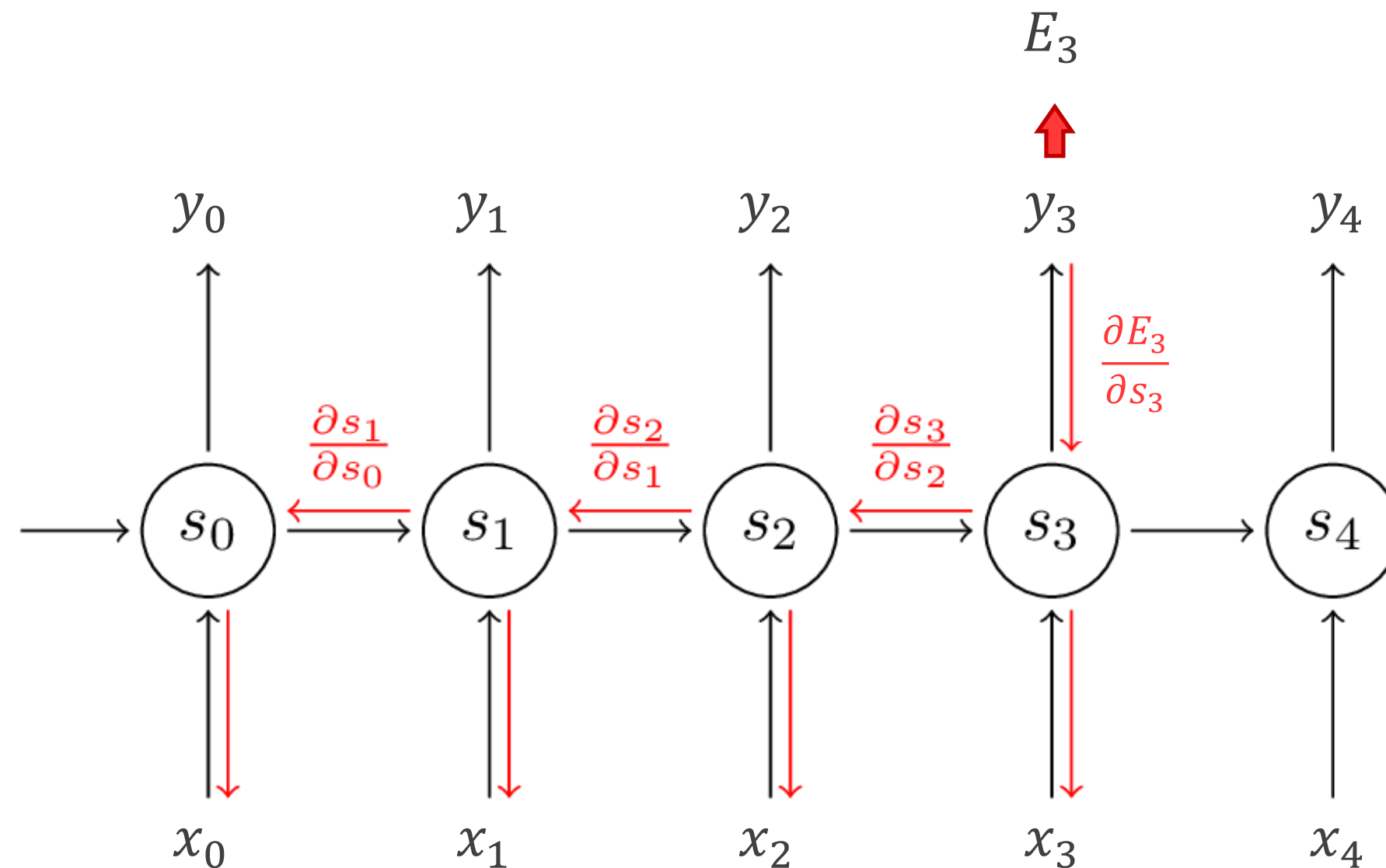
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

Cálculo del gradiente:

$$\frac{\partial E}{\partial W} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial W} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial x_k}{\partial W} \right)$$

$$\frac{\partial x_t}{\partial x_k} = \prod_{t \geq i > k} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{t \geq i > k} W^T \text{diag}(\sigma(x_{i-1}))$$

Vanishing Gradients



Función de error:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

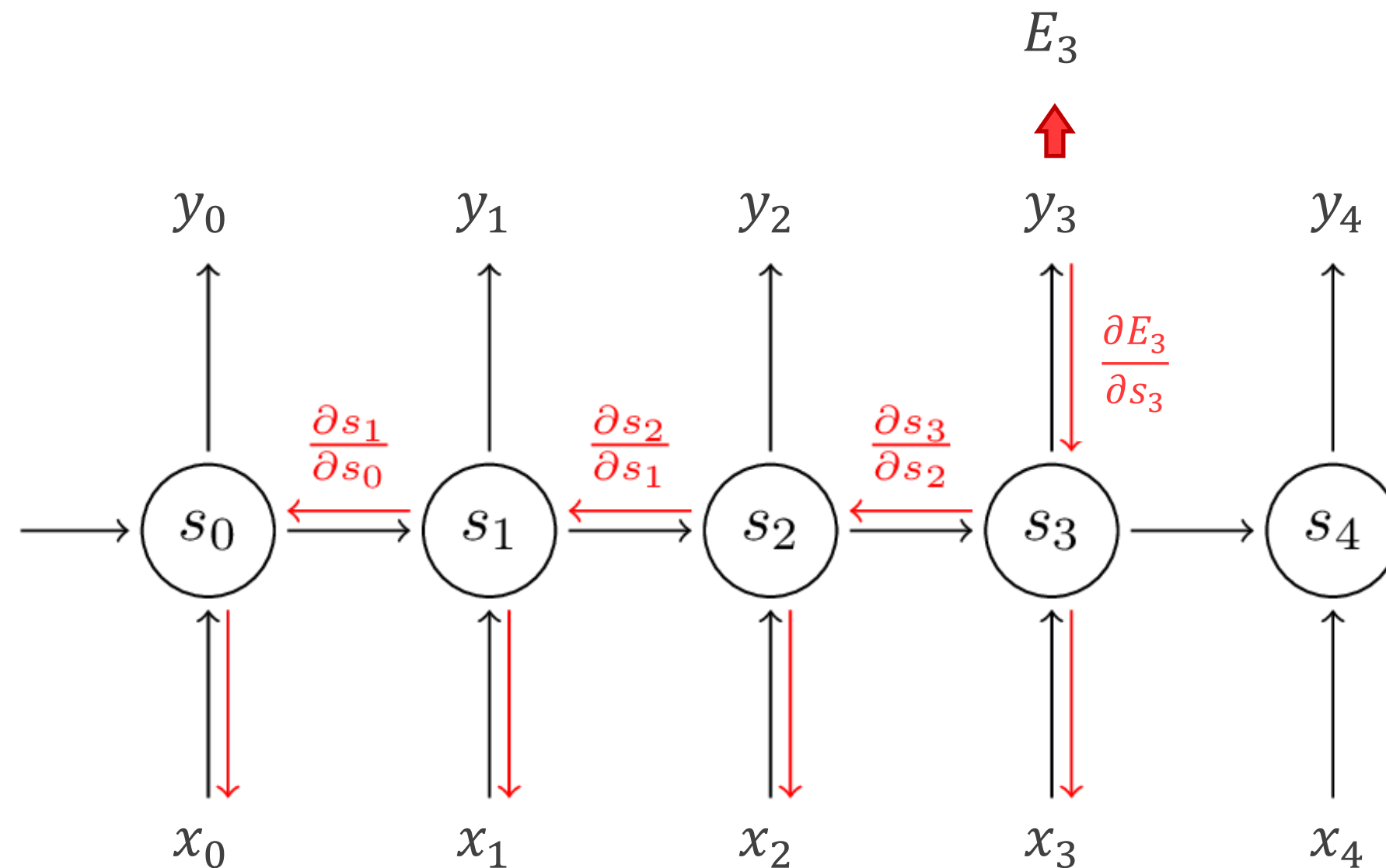
Cálculo del gradiente:

$$\frac{\partial E}{\partial W} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial W} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial x_k}{\partial W} \right)$$

$$\frac{\partial x_t}{\partial x_k} = \prod_{t \geq i > k} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{t \geq i > k} W^T \text{diag}(\sigma(x_{i-1}))$$

Si $\|W^T\| < 1$: El gradiente se desvanece.

Exploding Gradients



Función de error:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

Cálculo del gradiente:

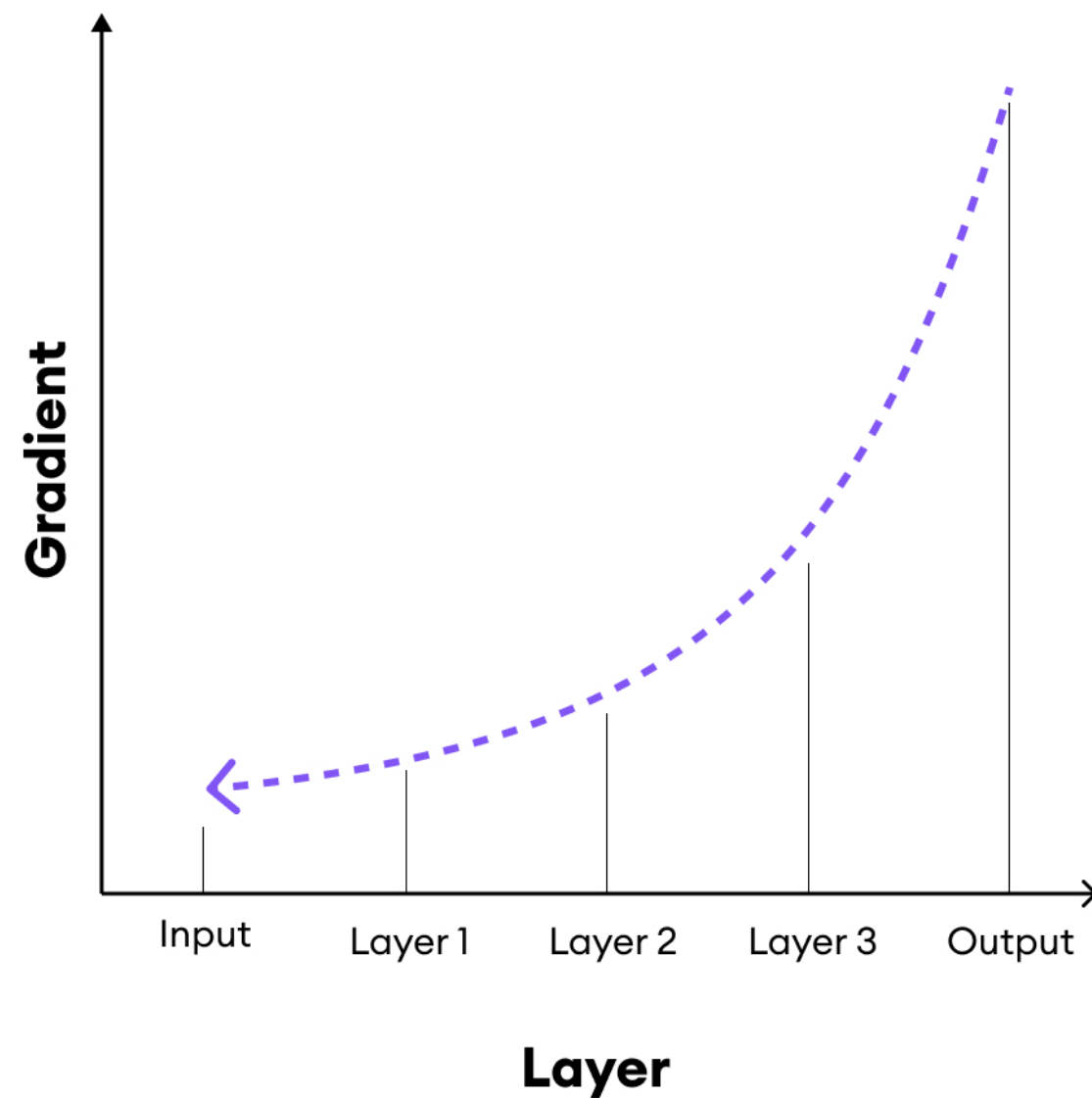
$$\frac{\partial E}{\partial W} = \sum_{1 \leq t \leq T} \frac{\partial E_t}{\partial W} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial x_t} \frac{\partial x_t}{\partial x_k} \frac{\partial x_k}{\partial W} \right)$$

$$\frac{\partial x_t}{\partial x_k} = \prod_{t \geq i > k} \frac{\partial x_i}{\partial x_{i-1}} = \prod_{t \geq i > k} W^T \text{diag}(\sigma(x_{i-1}))$$

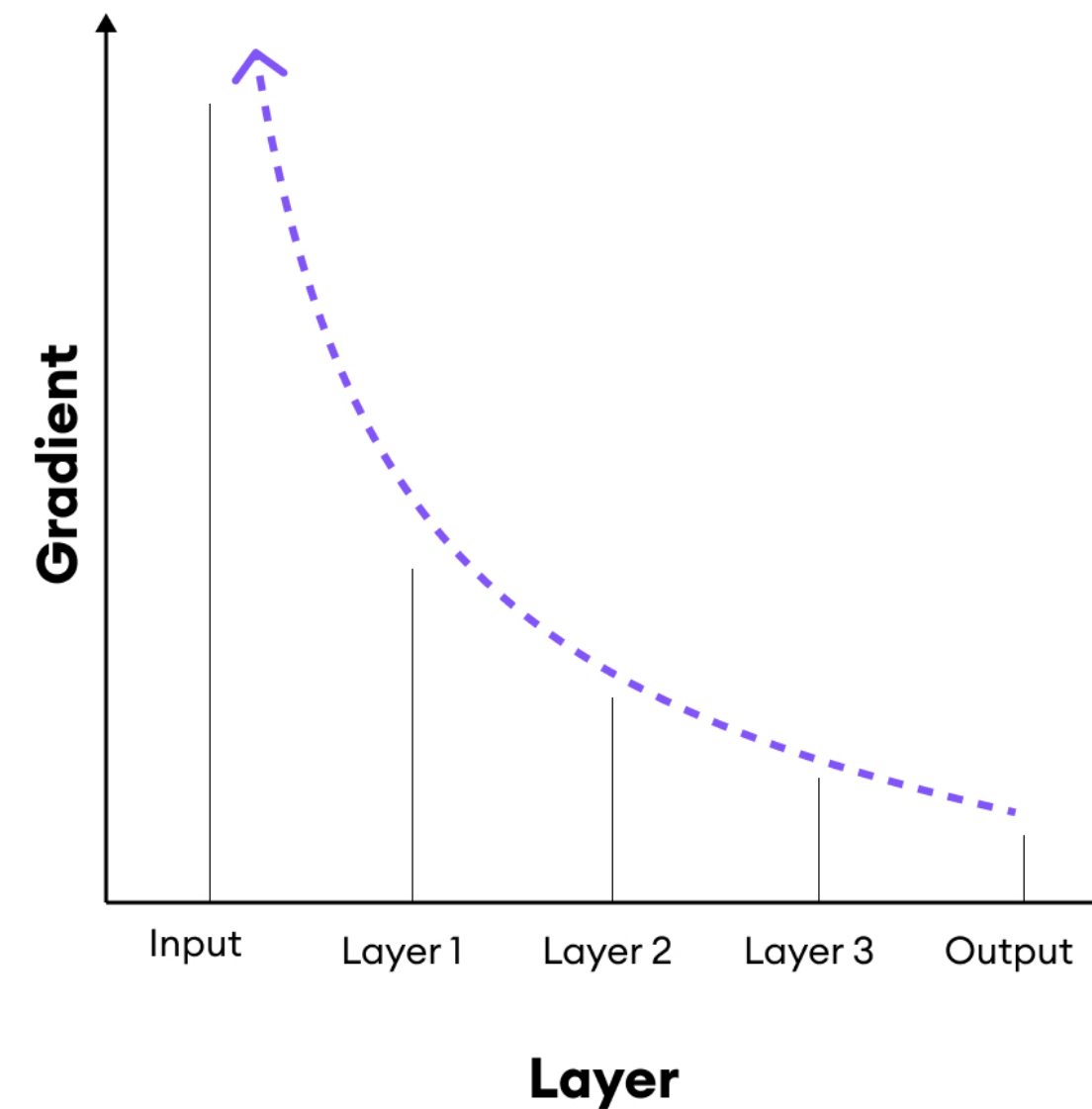
Si $\|W^T\| > 1$: El gradiente crece exponencialmente

Vanilla RNN *Gradient Flow*

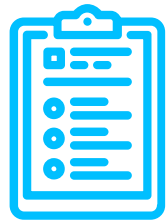
Vanishing Gradient



Exploding Gradient

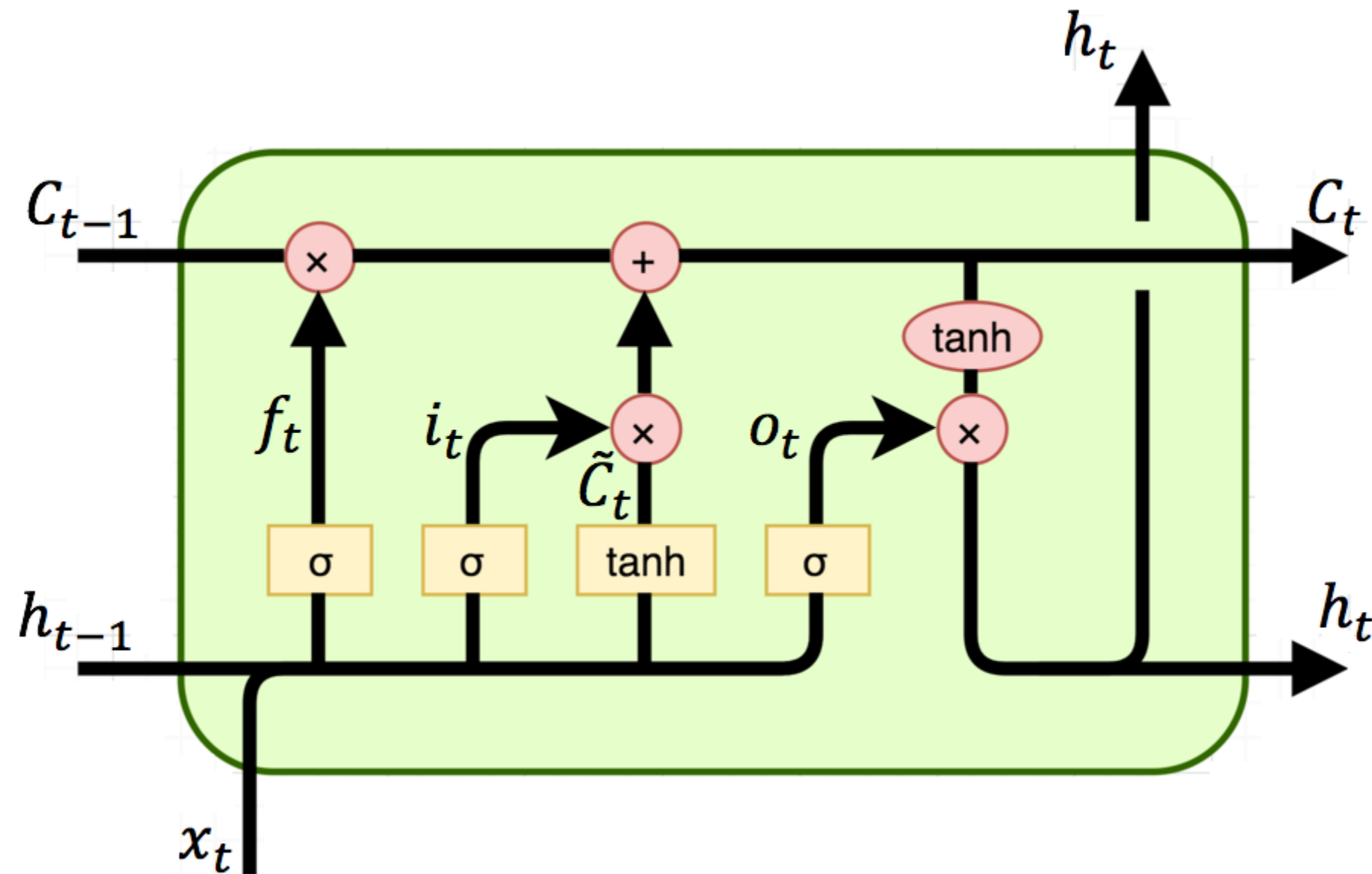


2.



LS*TM*

Long short-term *memory* (LSTM)



$$f_t = \sigma(\omega_f \cdot [h_{t-1}, x_t] + b_f)$$

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

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

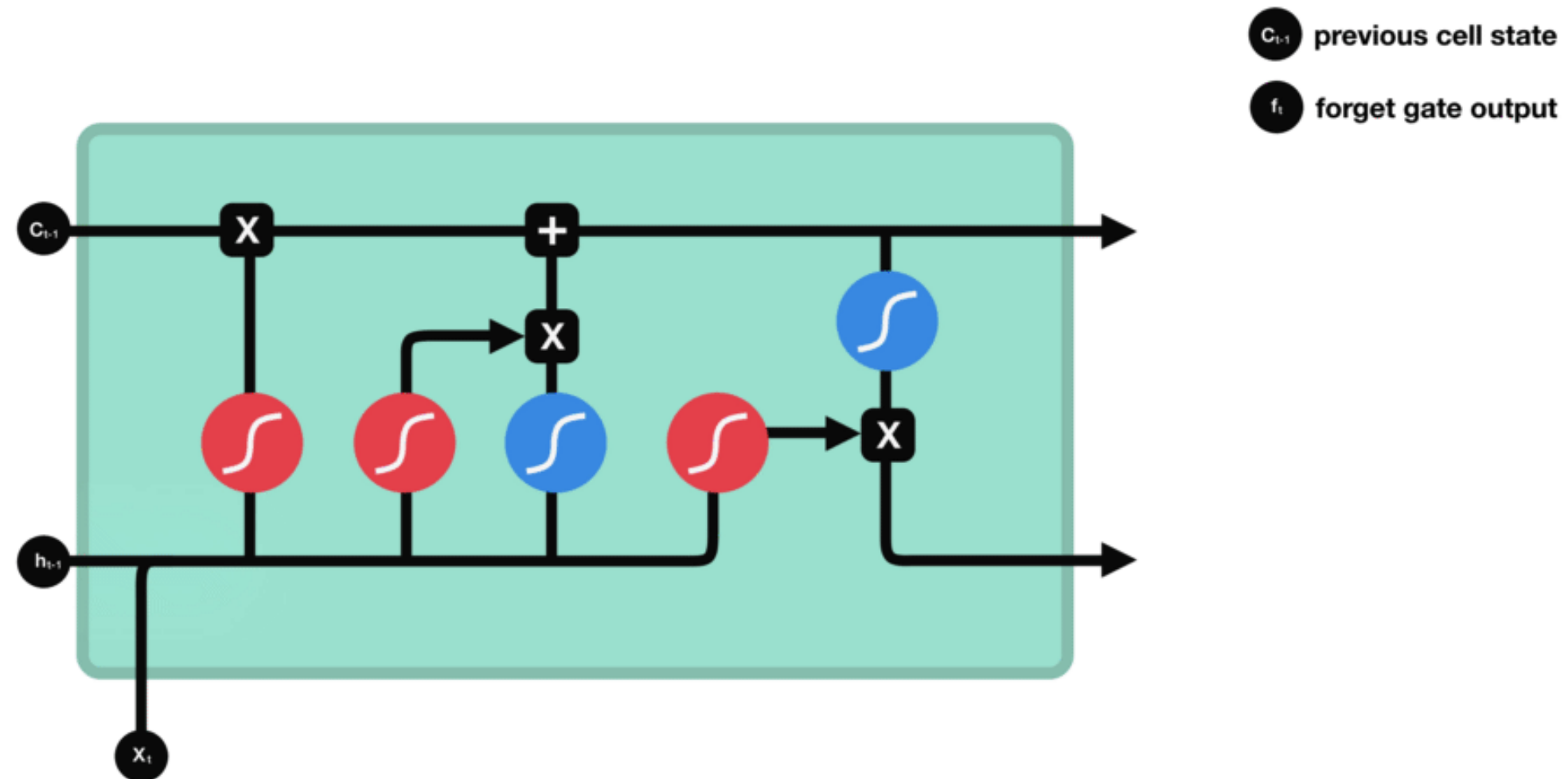
$$\tilde{C}_t = \tanh(\omega_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

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

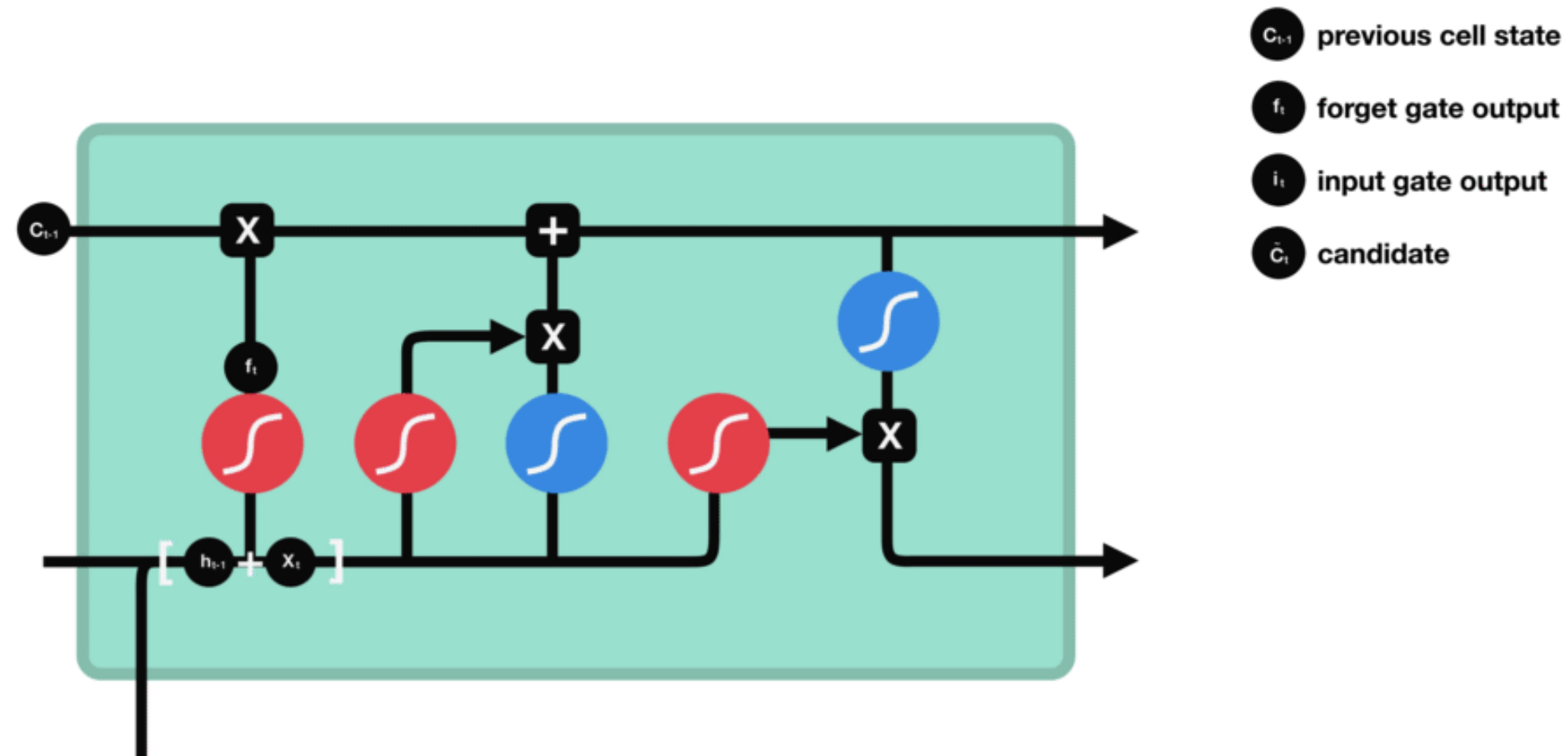
Long short-term *memory* (LSTM)

Forget gate



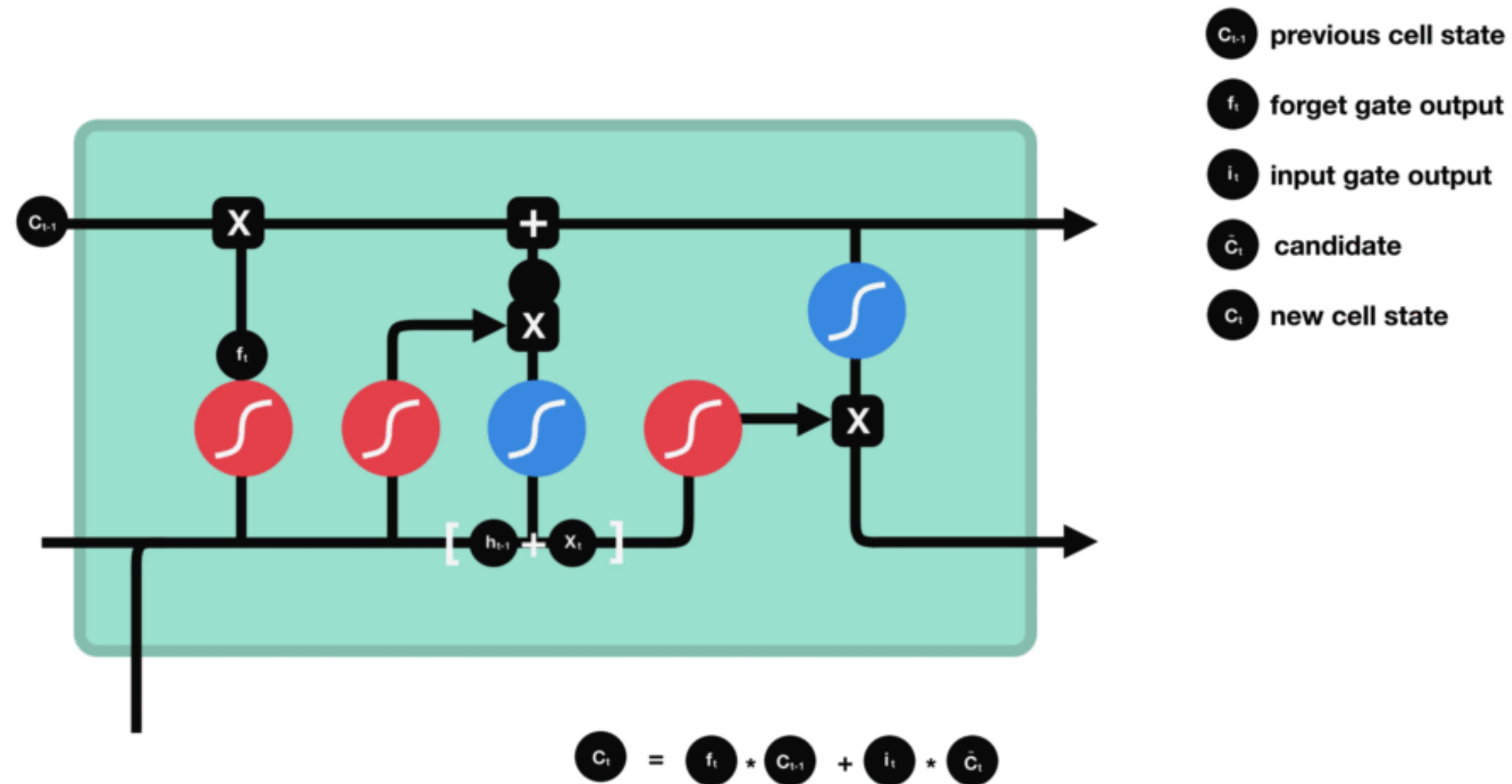
Long short-term *memory (LSTM)*

Input gate



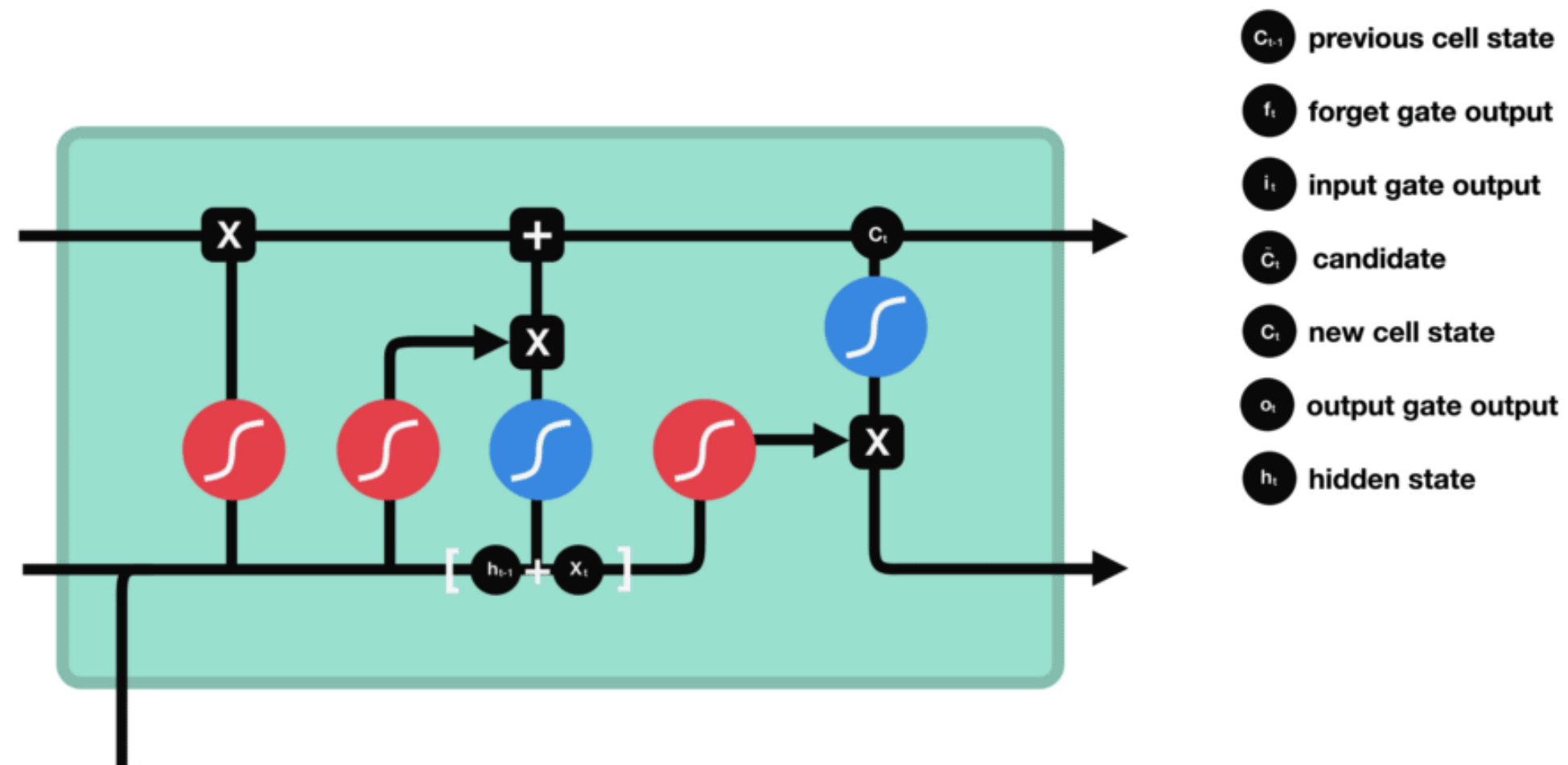
Long short-term *memory* (LSTM)

Cell state

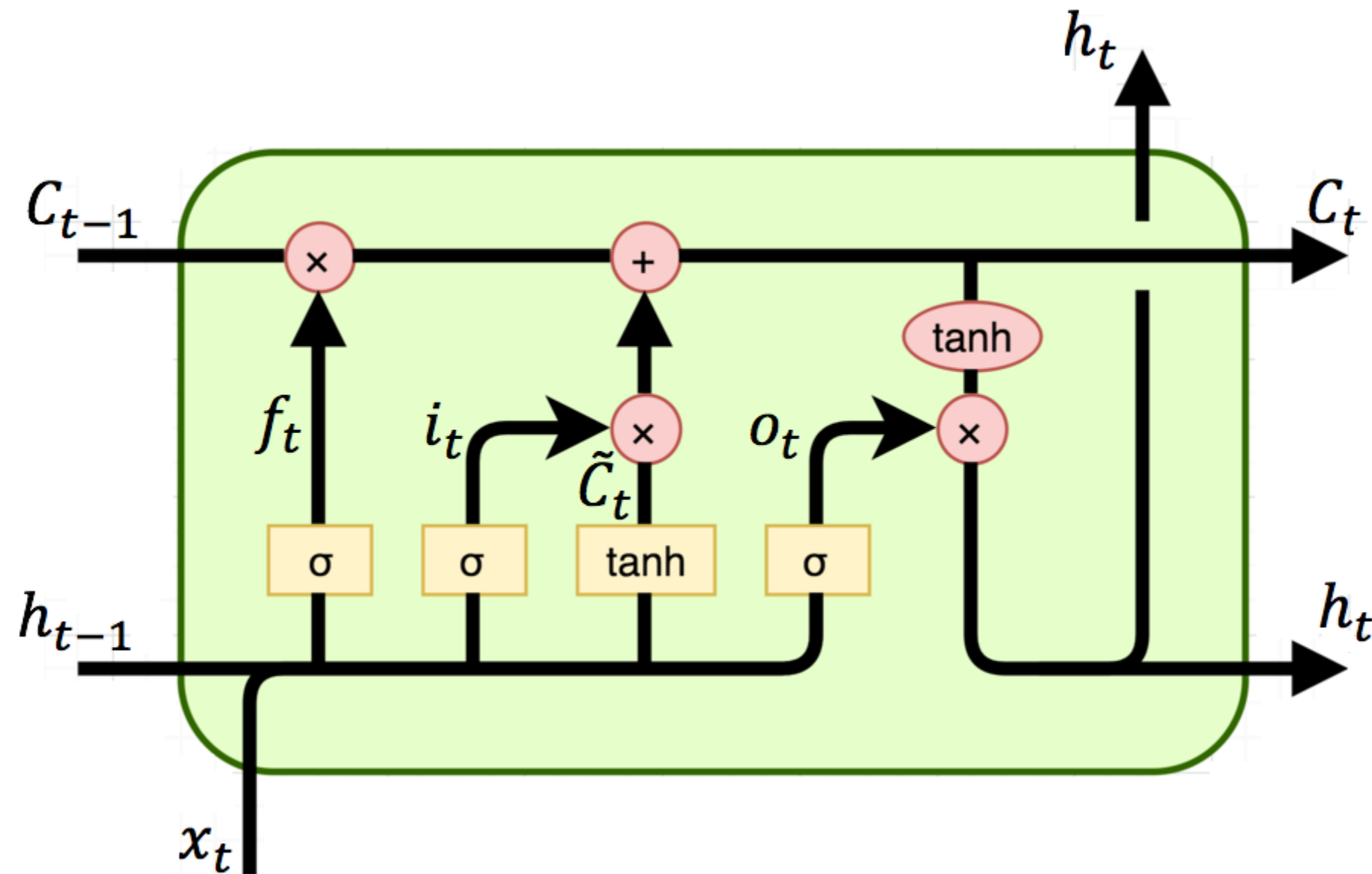


Long short-term *memory (LSTM)*

Output gate



Long short-term *memory* (LSTM)



$$f_t = \sigma(\omega_f \cdot [h_{t-1}, x_t] + b_f)$$

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

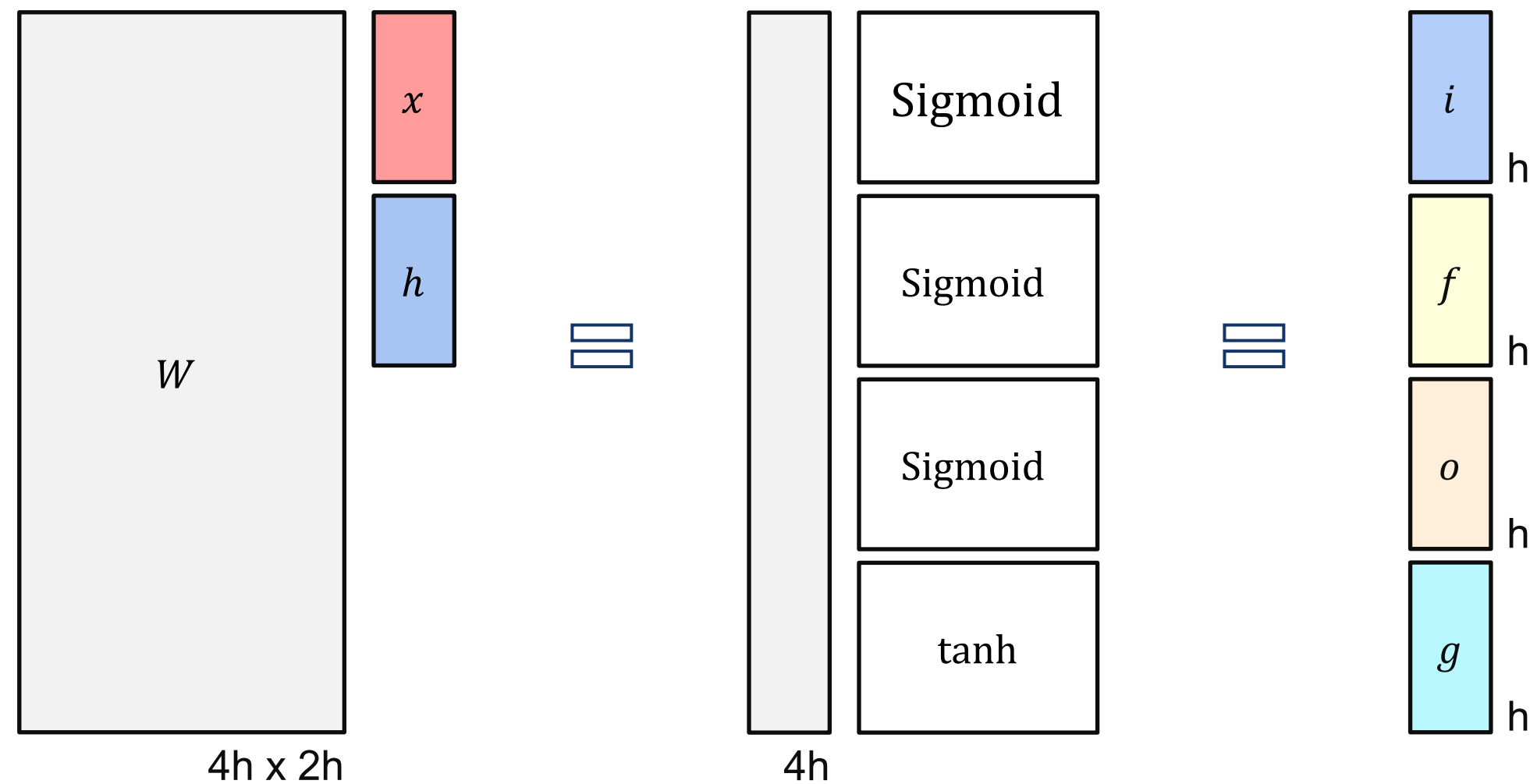
$$o_t = \sigma(\omega_o \cdot [h_{t-1}, x_t] + b_o)$$

$$\tilde{C}_t = \tanh(\omega_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

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

Long short-term *memory* (LSTM)



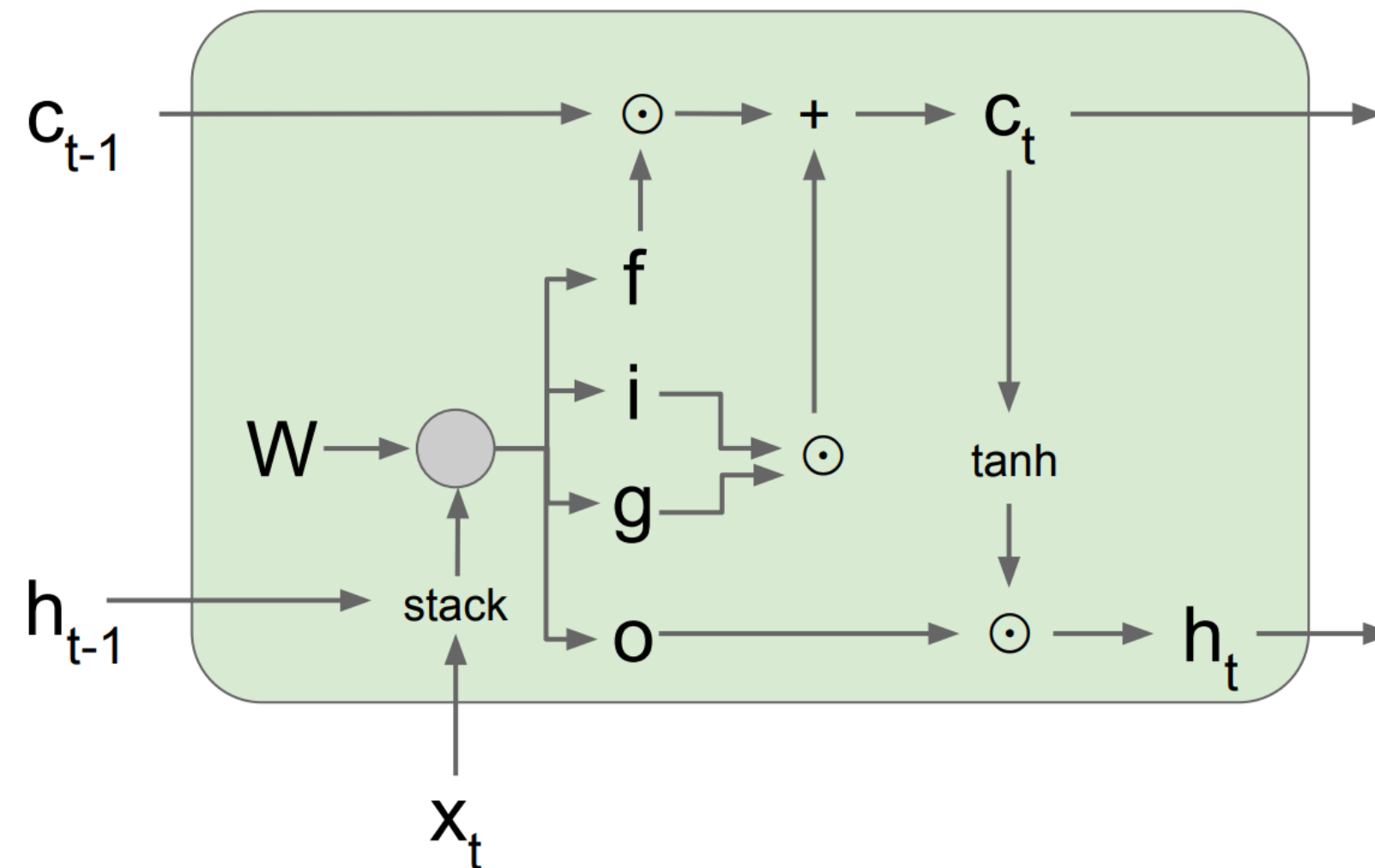
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

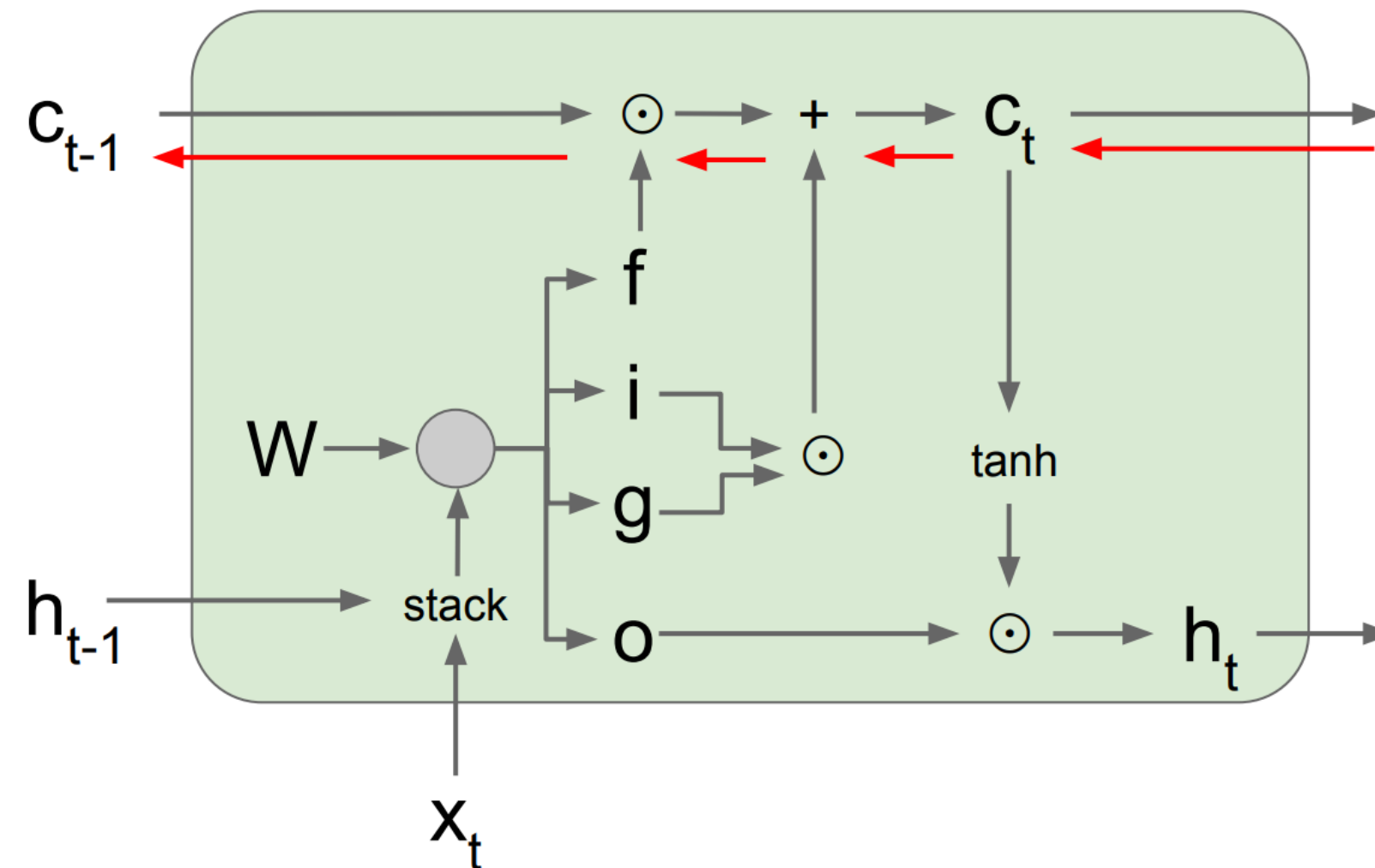
$$h_t = o_t \odot \tanh(C_t)$$



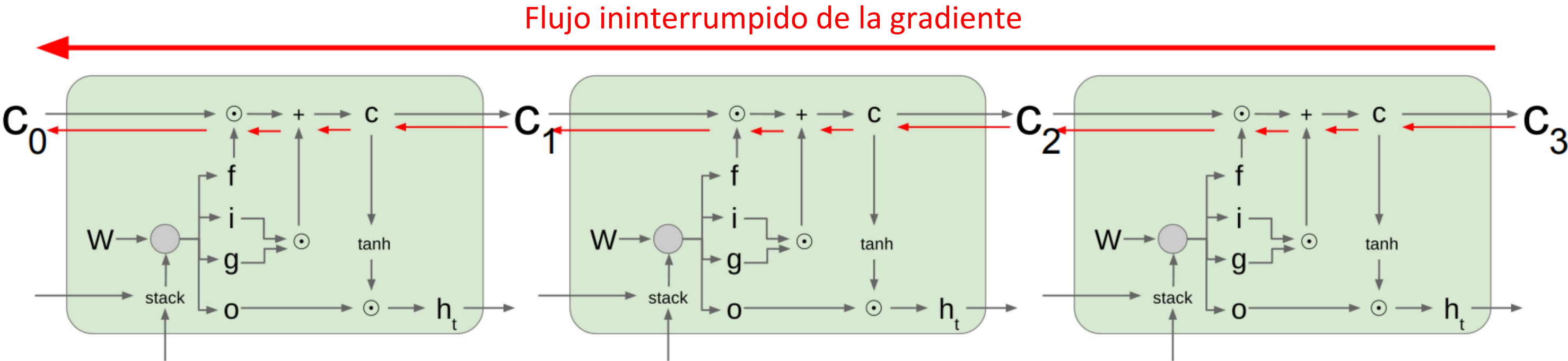
LSTM Gradient Flow



LSTM Gradient Flow

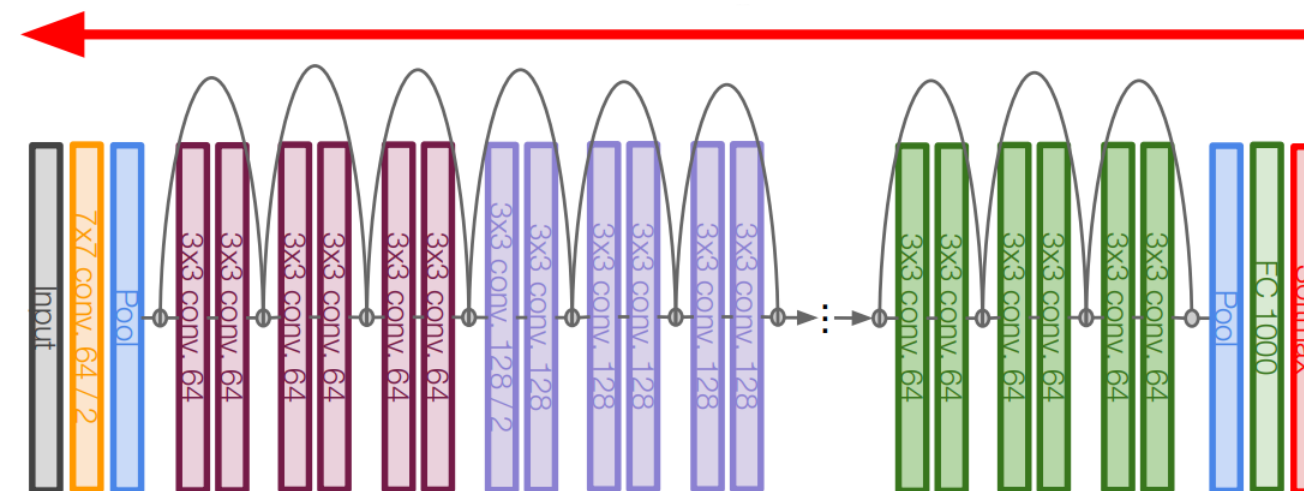
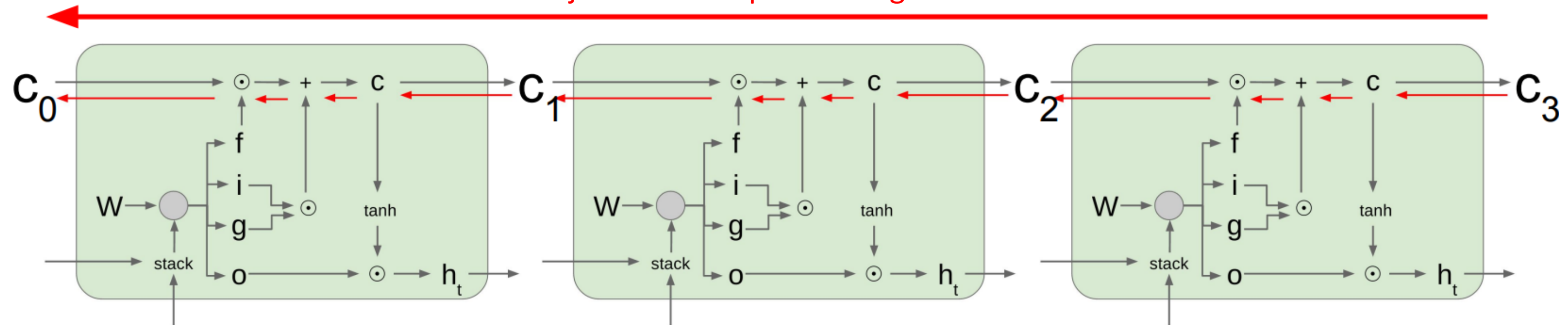


LSTM Gradient Flow



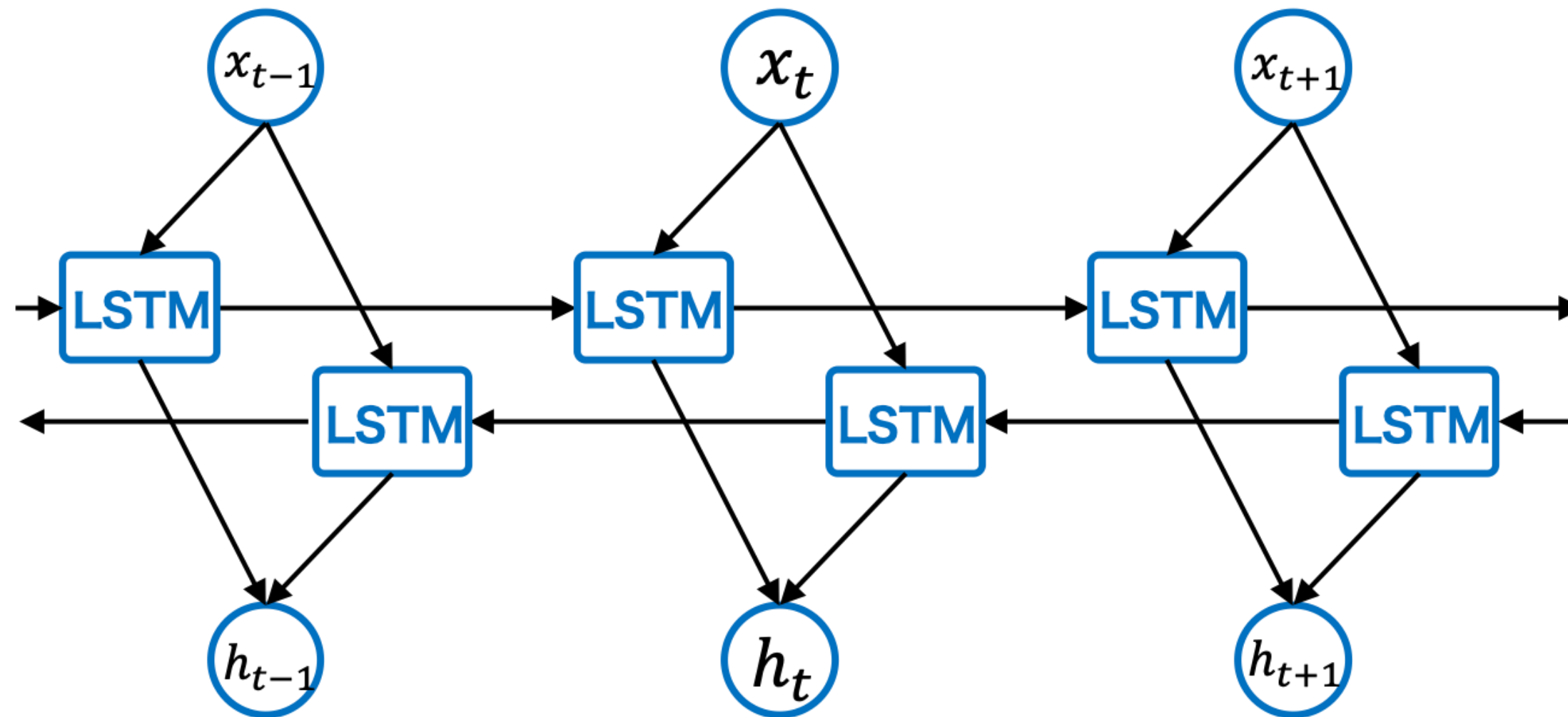
LSTM Gradient Flow

Flujo ininterrumpido de la gradiente

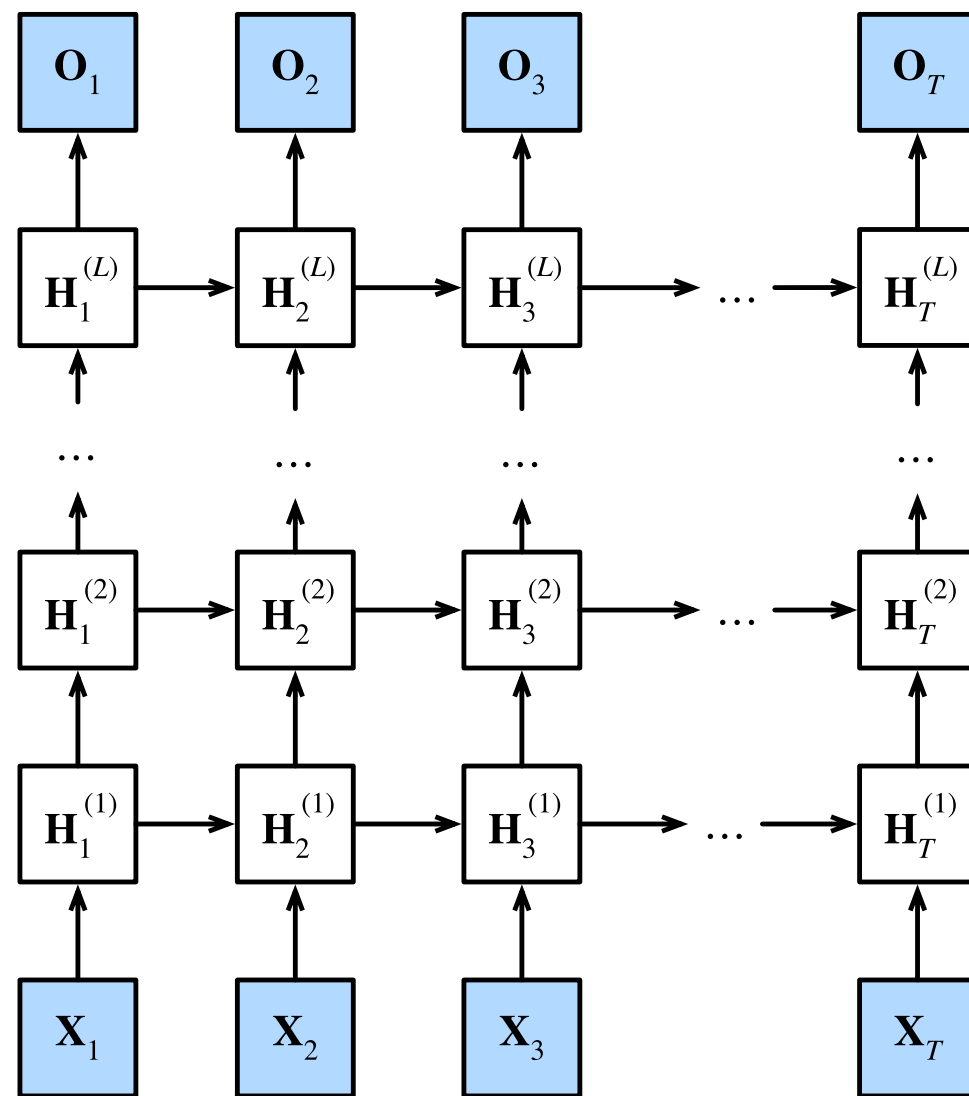


Similar a ResNet

Bidirectional *LSTM*



Deep *LSTM*

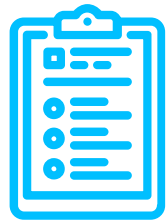


$$h_t^{(l)} = \phi_l \left(h_t^{(l-1)} W_{xh}^{(l)} + h_{t-1}^{(l)} W_{hh}^{(l)} + b_h^{(l)} \right)$$

$$O_t = h_t^{(L)} W_{hq} + b_q$$

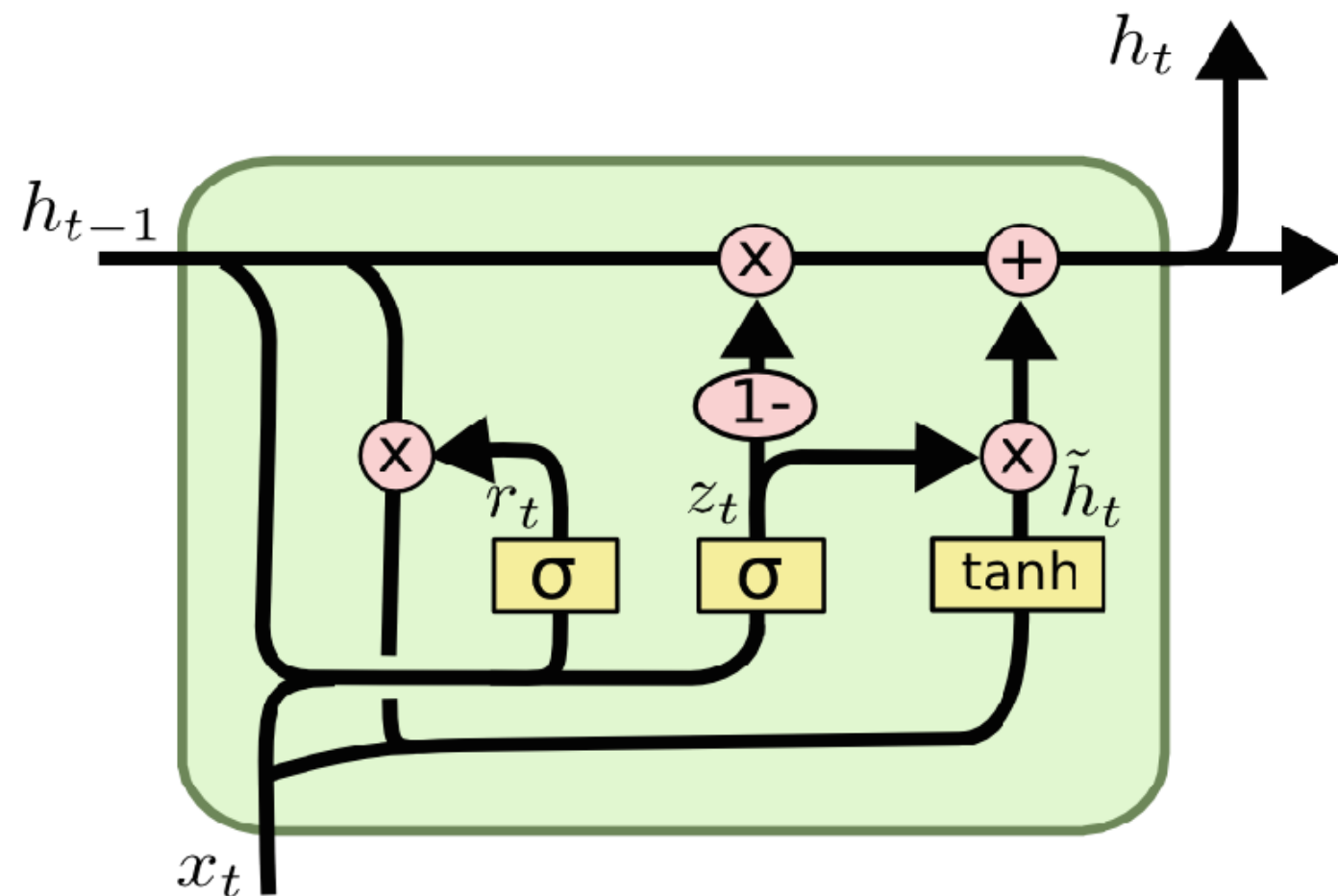


3.



GRU

GRU



Reset gate: $z_t = \sigma(\omega_z \cdot [h_{t-1}, x_t] + b_z)$

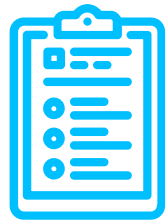
Update gate: $r_t = \sigma(\omega_r \cdot [h_{t-1}, x_t] + b_r)$

$$\tilde{h}_t = \tanh(\omega_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

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

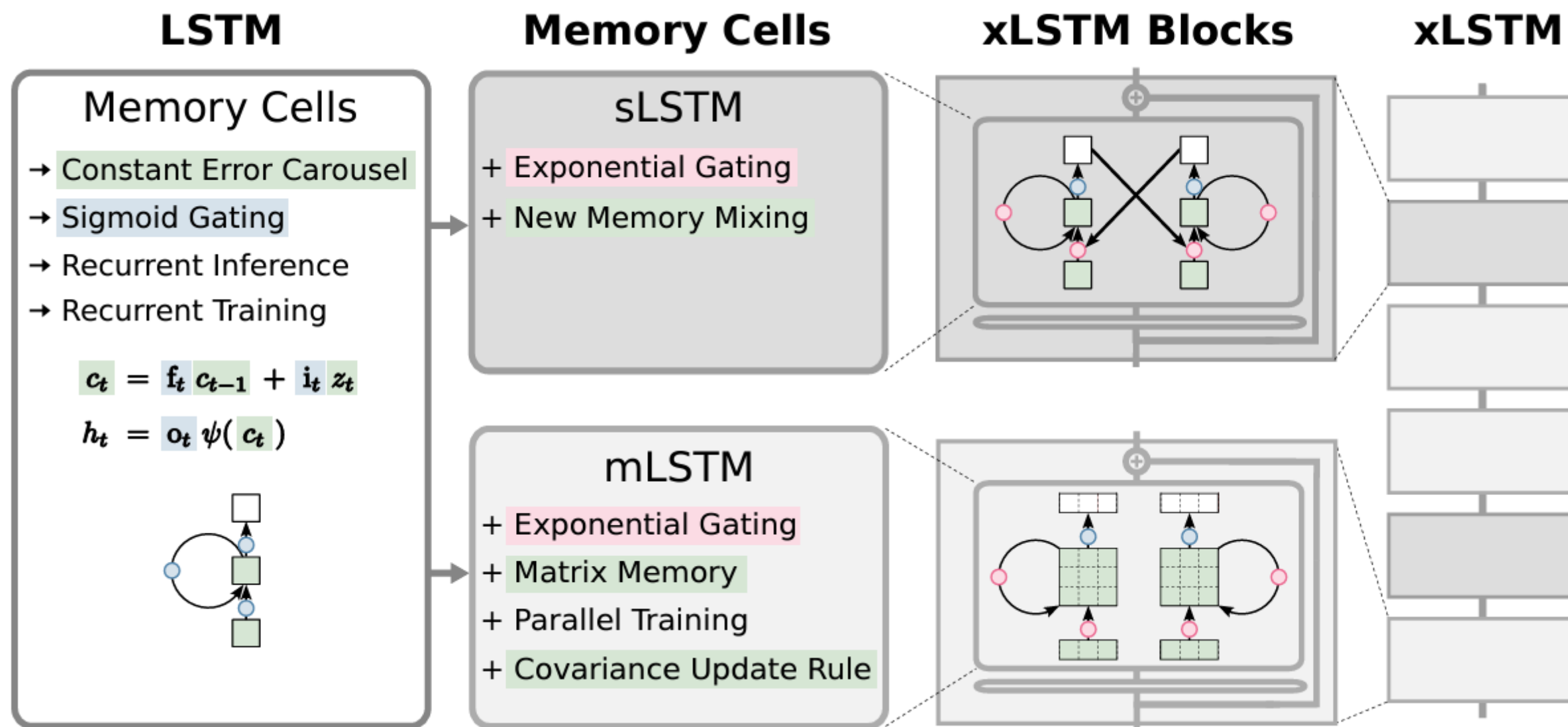


4.



xLSTM

xLSTM

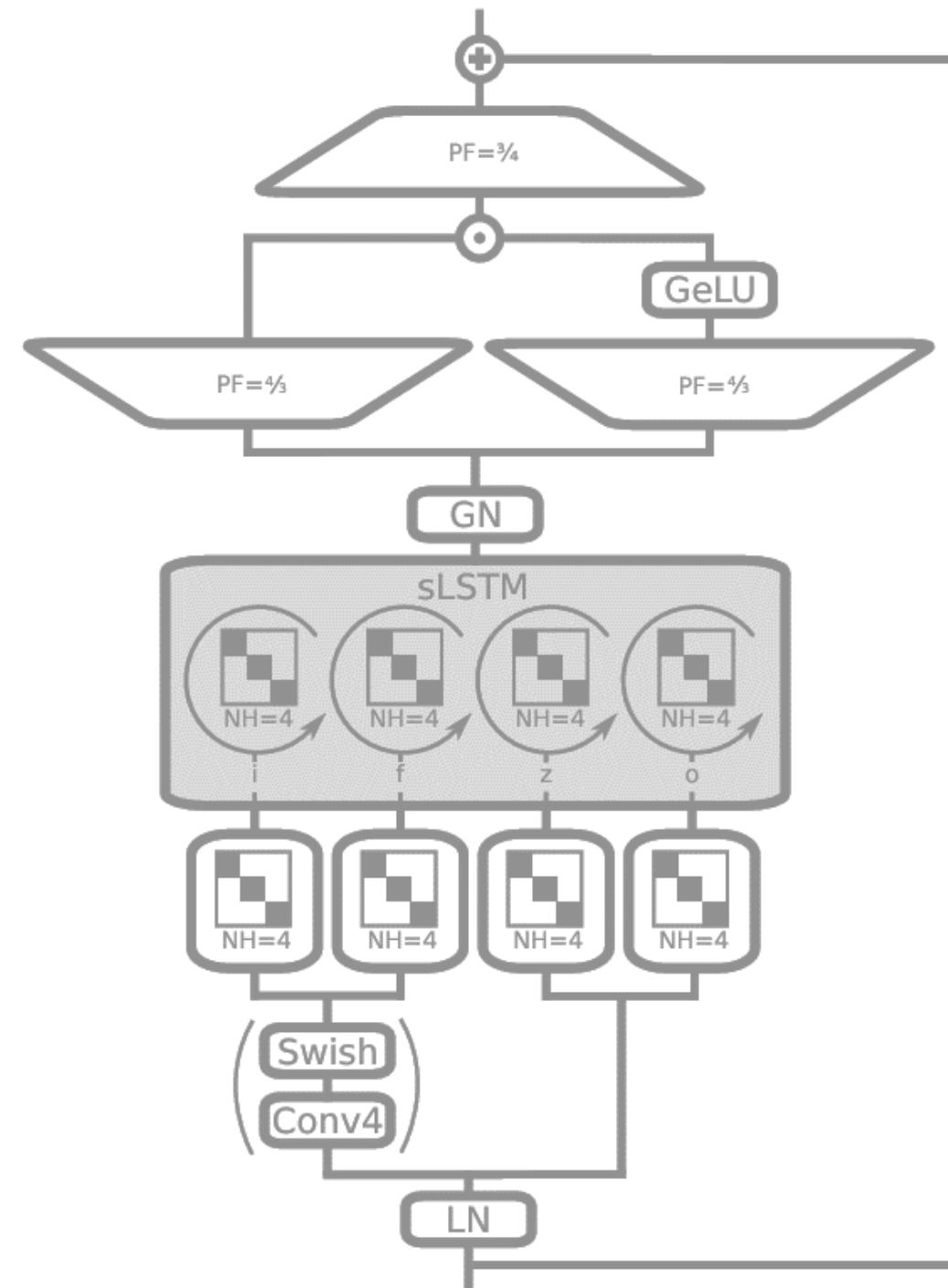


SLSTM

$c_t = f_t c_{t-1} + i_t z_t$	cell state	(8)
$n_t = f_t n_{t-1} + i_t$	normalizer state	(9)
$h_t = o_t \tilde{h}_t$	hidden state	(10)
$z_t = \varphi(\tilde{z}_t)$	cell input	(11)
$i_t = \exp(\tilde{i}_t)$	input gate	(12)
$f_t = \sigma(\tilde{f}_t) \text{ OR } \exp(\tilde{f}_t)$	forget gate	(13)
$o_t = \sigma(\tilde{o}_t)$	output gate	(14)
$\tilde{h}_t = c_t / n_t$		
$\tilde{z}_t = \mathbf{w}_z^\top \mathbf{x}_t + r_z h_{t-1} + b_z$		
$\tilde{i}_t = \mathbf{w}_i^\top \mathbf{x}_t + r_i h_{t-1} + b_i$		
$\tilde{f}_t = \mathbf{w}_f^\top \mathbf{x}_t + r_f h_{t-1} + b_f$		
$\tilde{o}_t = \mathbf{w}_o^\top \mathbf{x}_t + r_o h_{t-1} + b_o$		



sLSTM



mLSTM

$$C_t = f_t C_{t-1} + i_t v_t k_t^\top$$

cell state (19)

$$n_t = f_t n_{t-1} + i_t k_t$$

normalizer state (20)

$$h_t = o_t \odot \tilde{h}_t, \quad \tilde{h}_t = C_t q_t / \max \left\{ \left| n_t^\top q_t \right|, 1 \right\}$$

hidden state (21)

$$q_t = W_q x_t + b_q$$

query input (22)

$$k_t = \frac{1}{\sqrt{d}} W_k x_t + b_k$$

key input (23)

$$v_t = W_v x_t + b_v$$

value input (24)

$$i_t = \exp(\tilde{i}_t), \quad \tilde{i}_t = w_i^\top x_t + b_i$$

input gate (25)

$$f_t = \sigma(\tilde{f}_t) \text{ OR } \exp(\tilde{f}_t), \quad \tilde{f}_t = w_f^\top x_t + b_f$$

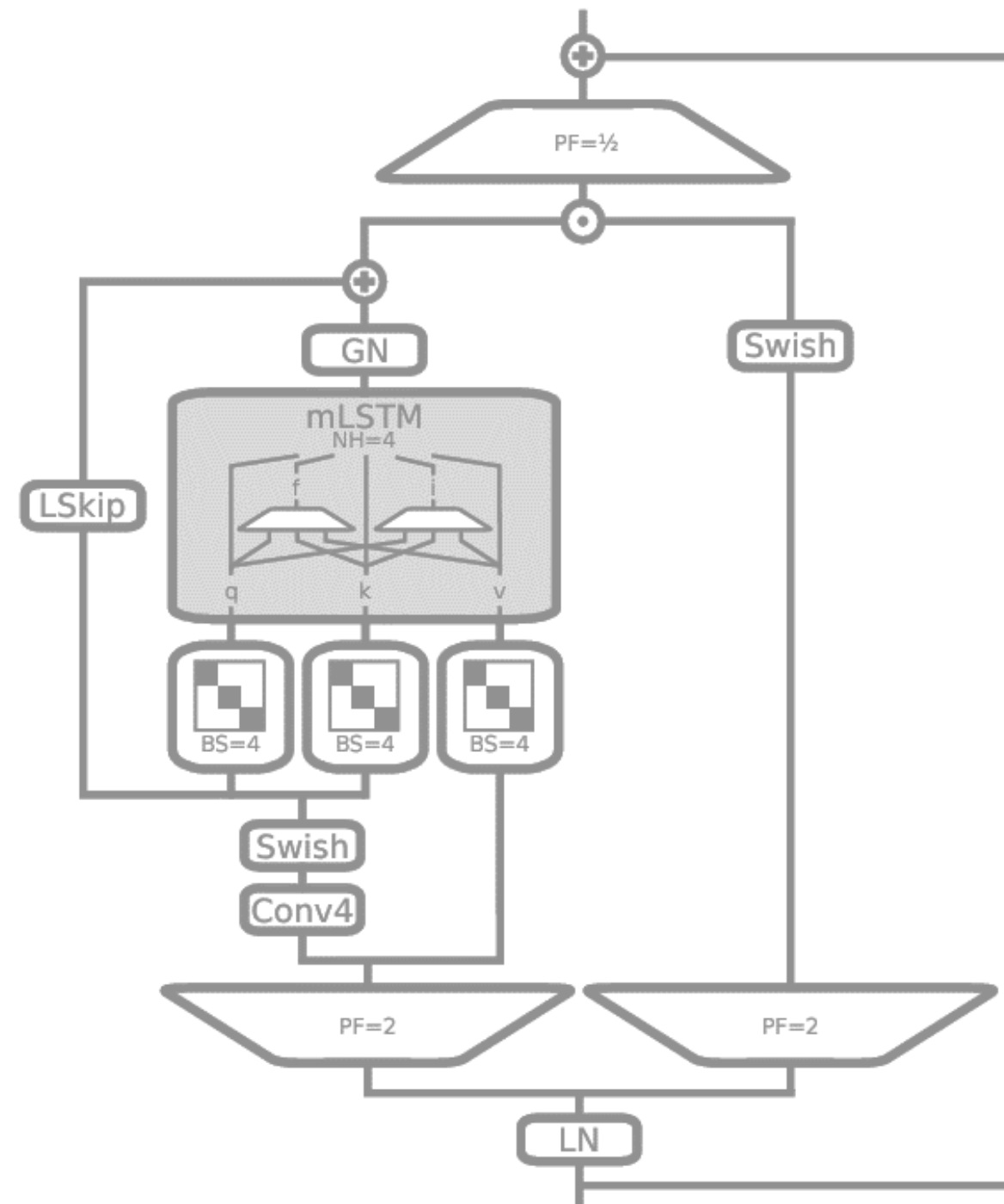
forget gate (26)

$$o_t = \sigma(\tilde{o}_t), \quad \tilde{o}_t = W_o x_t + b_o$$

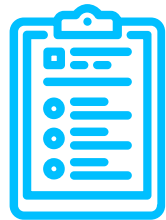
output gate (27)



mLSTM



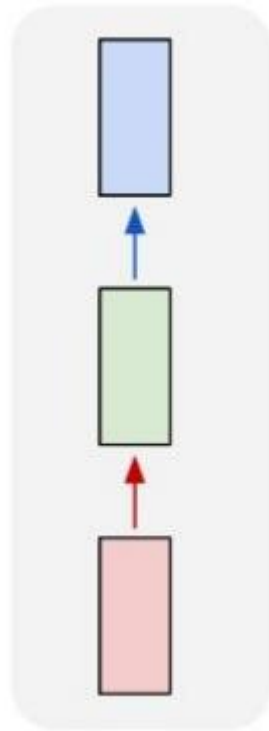
5.



Configuraciones

Configuraciones

one to one

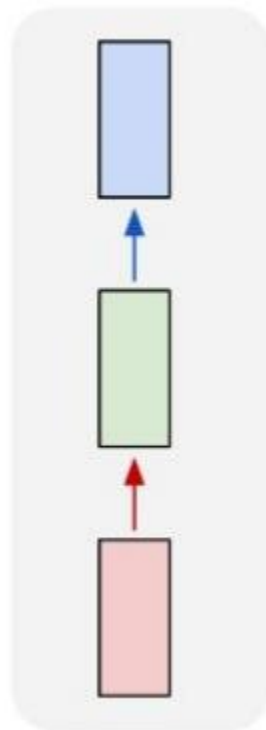


Ejemplo:
Clasificación
Regresión



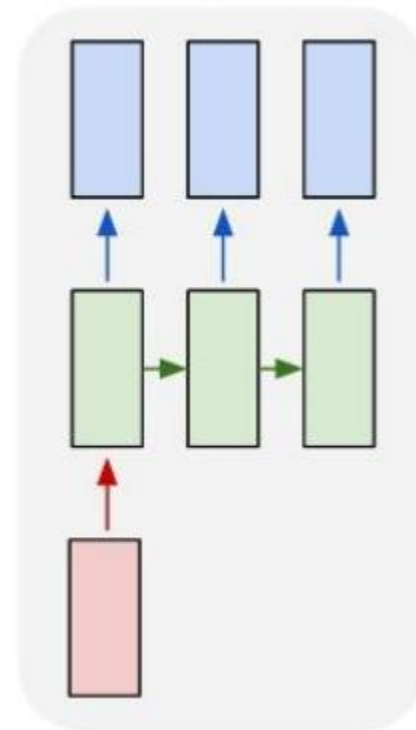
Configuraciones

one to one



Ejemplo:
 Clasificación
 Regresión

one to many

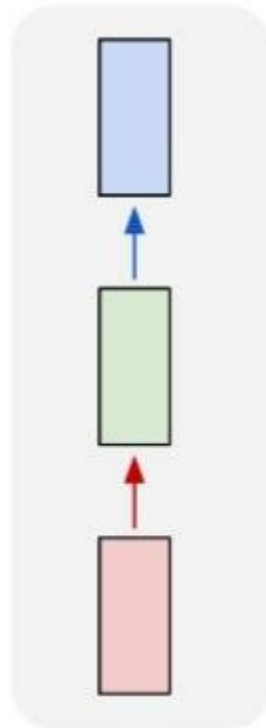


Ejemplo:
 Image caption
 Music generation



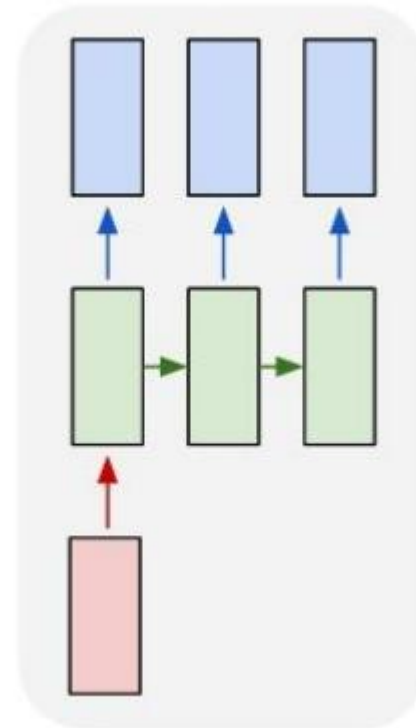
Configuraciones

one to one



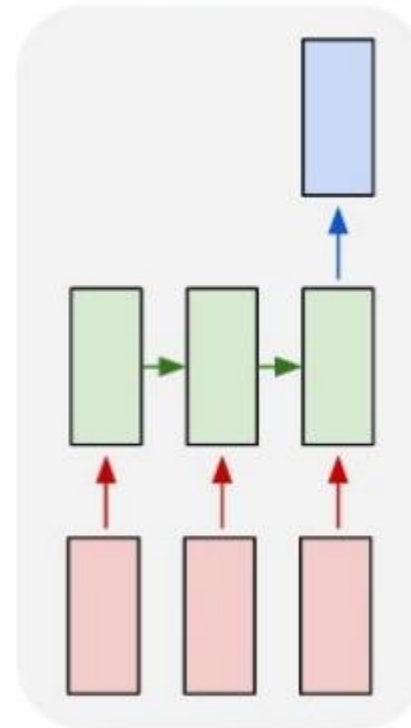
Ejemplo:
 Clasificación
 Regresión

one to many



Ejemplo:
 Image caption
 Music generation

many to one

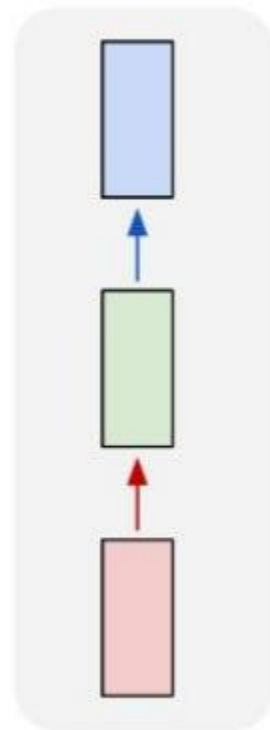


Ejemplo:
 clasificación de
 oraciones,
 respuesta a
 preguntas de
 opción múltiple



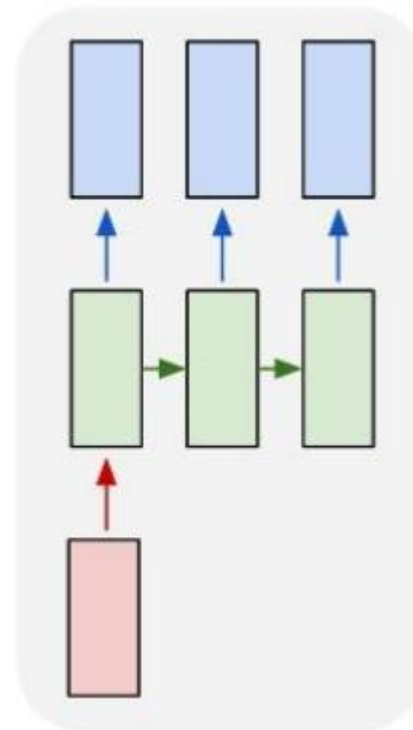
Configuraciones

one to one



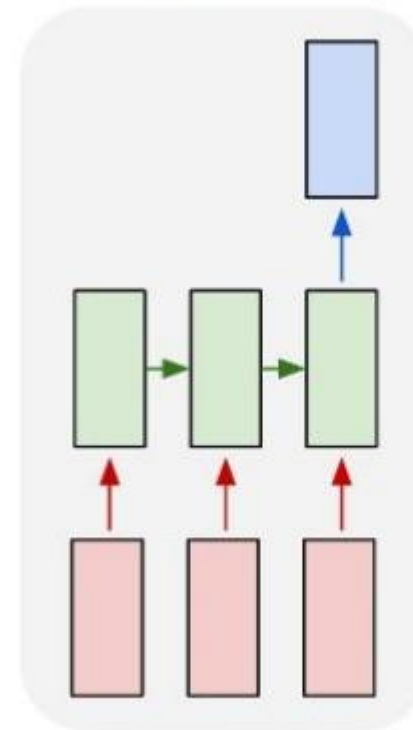
Ejemplo:
 Clasificación
 Regresión

one to many



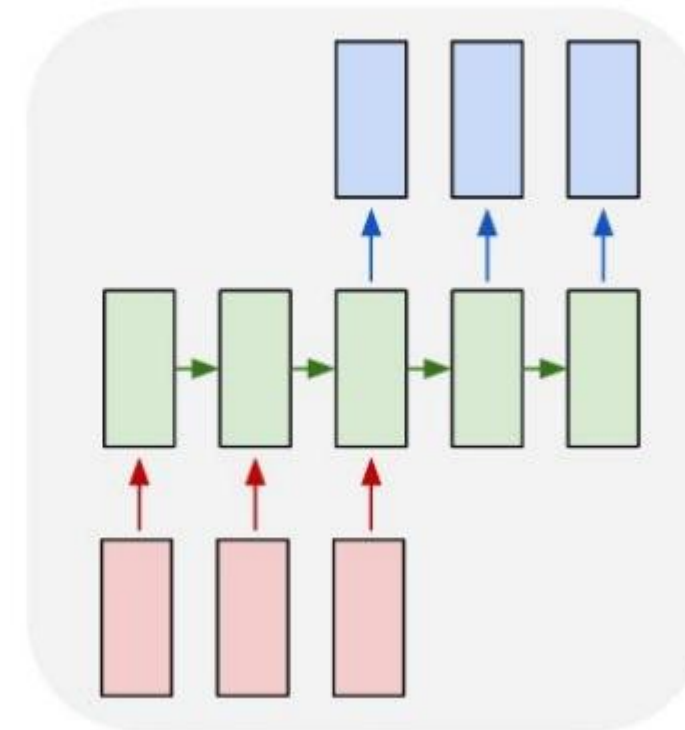
Ejemplo:
 Image caption
 Music generation

many to one



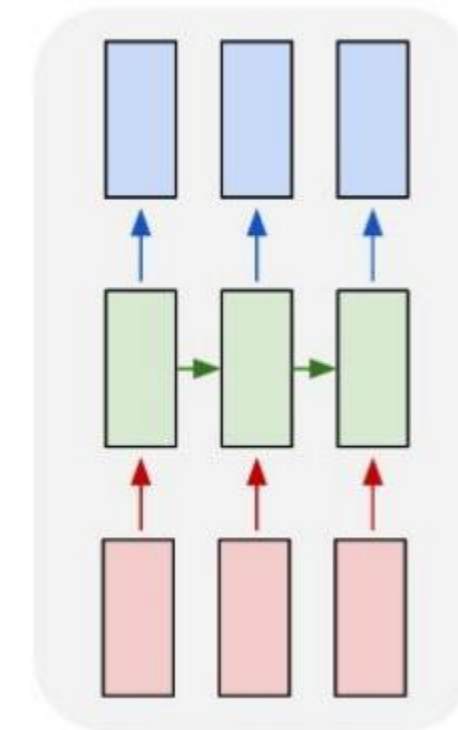
Ejemplo:
 clasificación de
 oraciones,
 respuesta a
 preguntas de
 opción múltiple

many to many

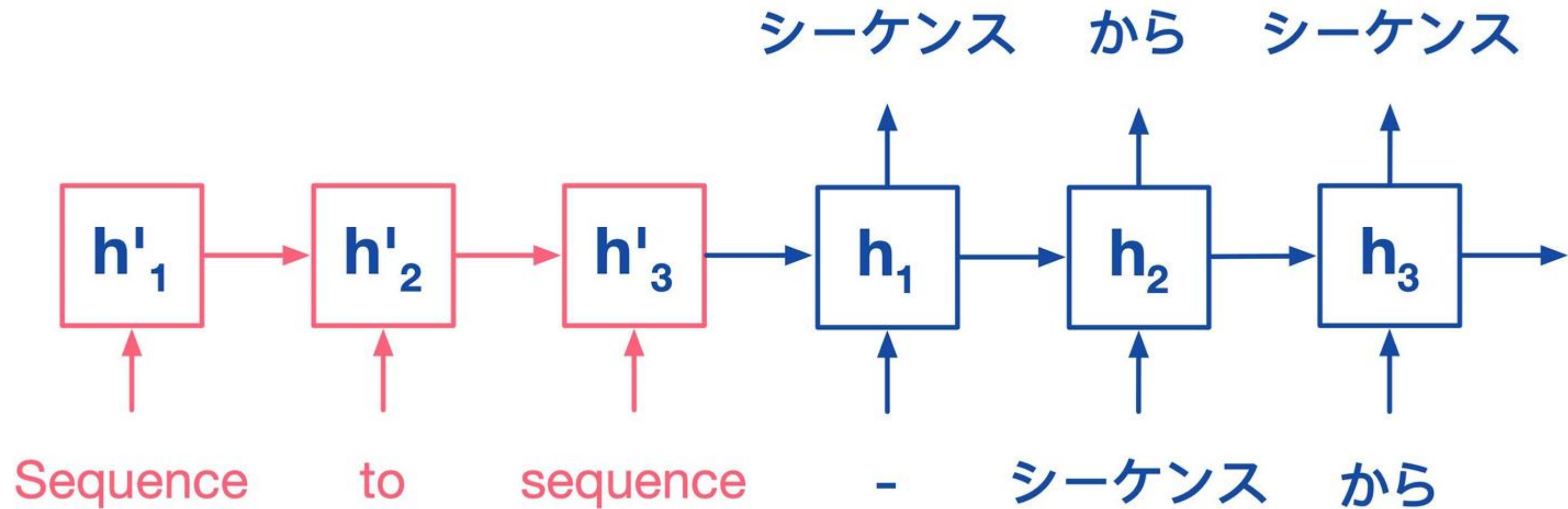


Ejemplo:
 machine translation,
 video classification,
 video captioning,

many to many



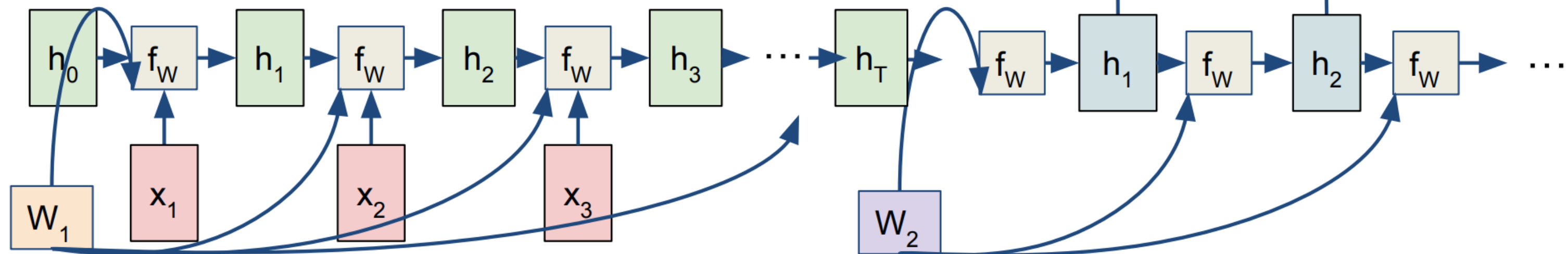
Sequence-to-*sequence models*



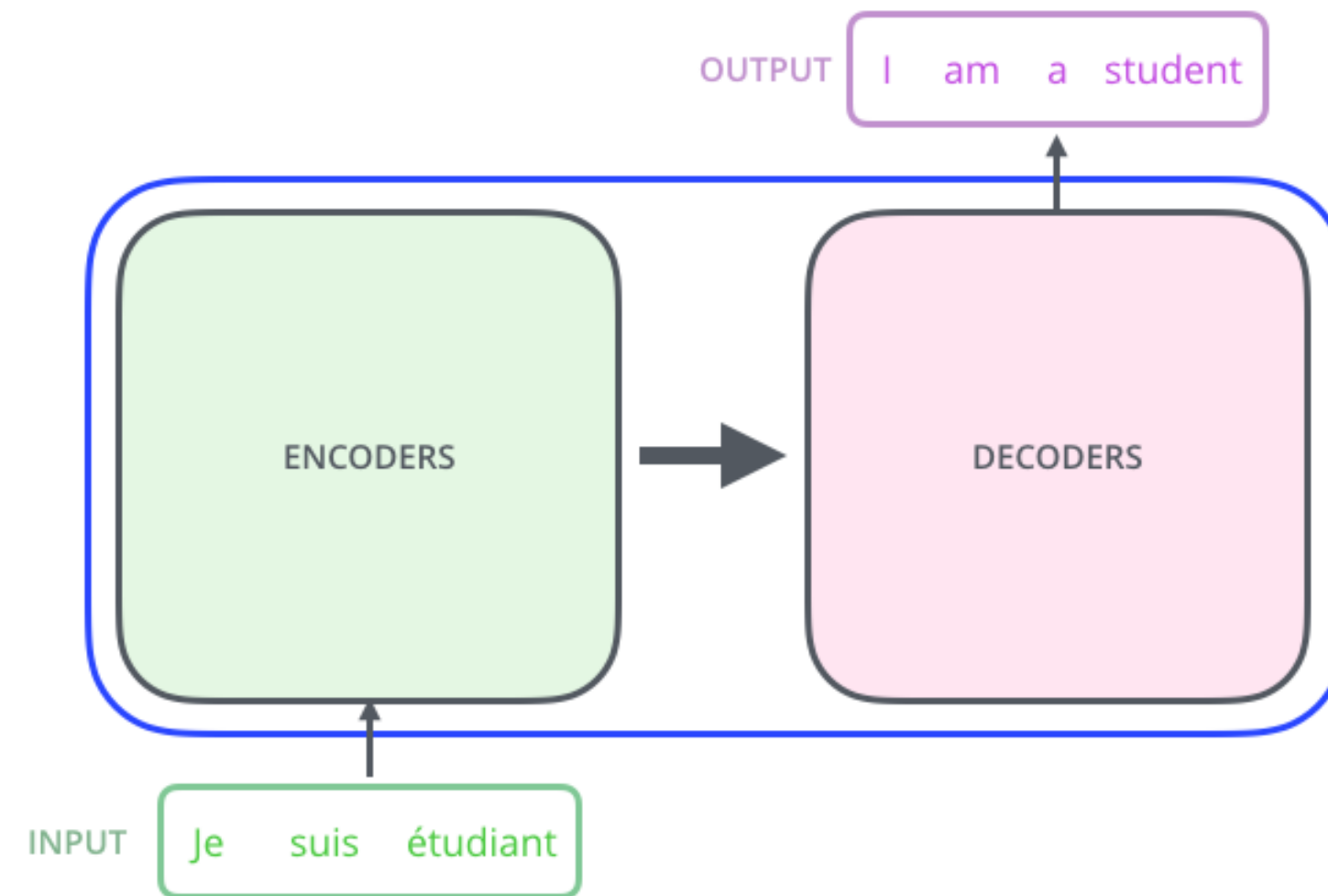
Sequence-to-sequence models

Many to one: Encode input sequence in a single vector

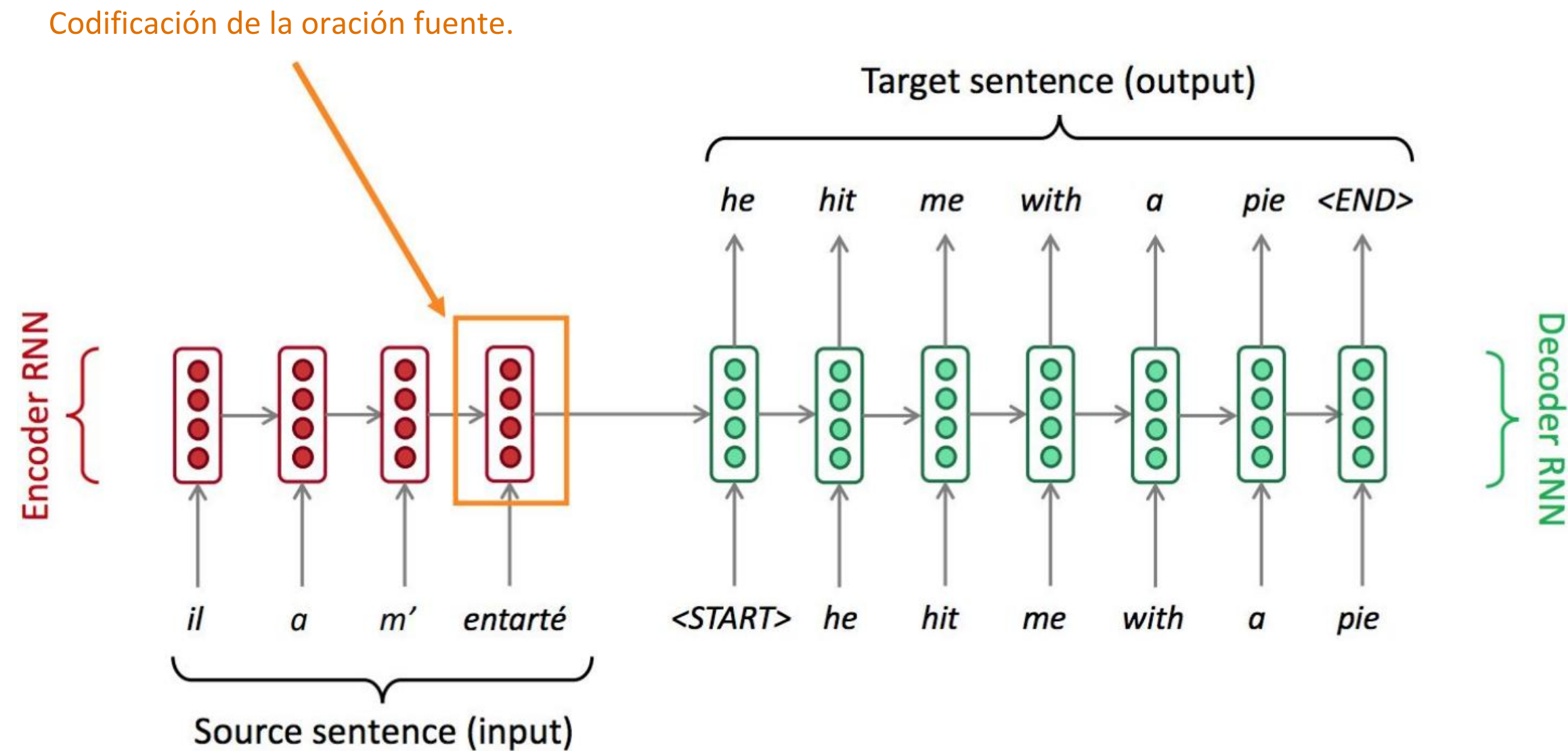
One to many: Produce output sequence from single input vector



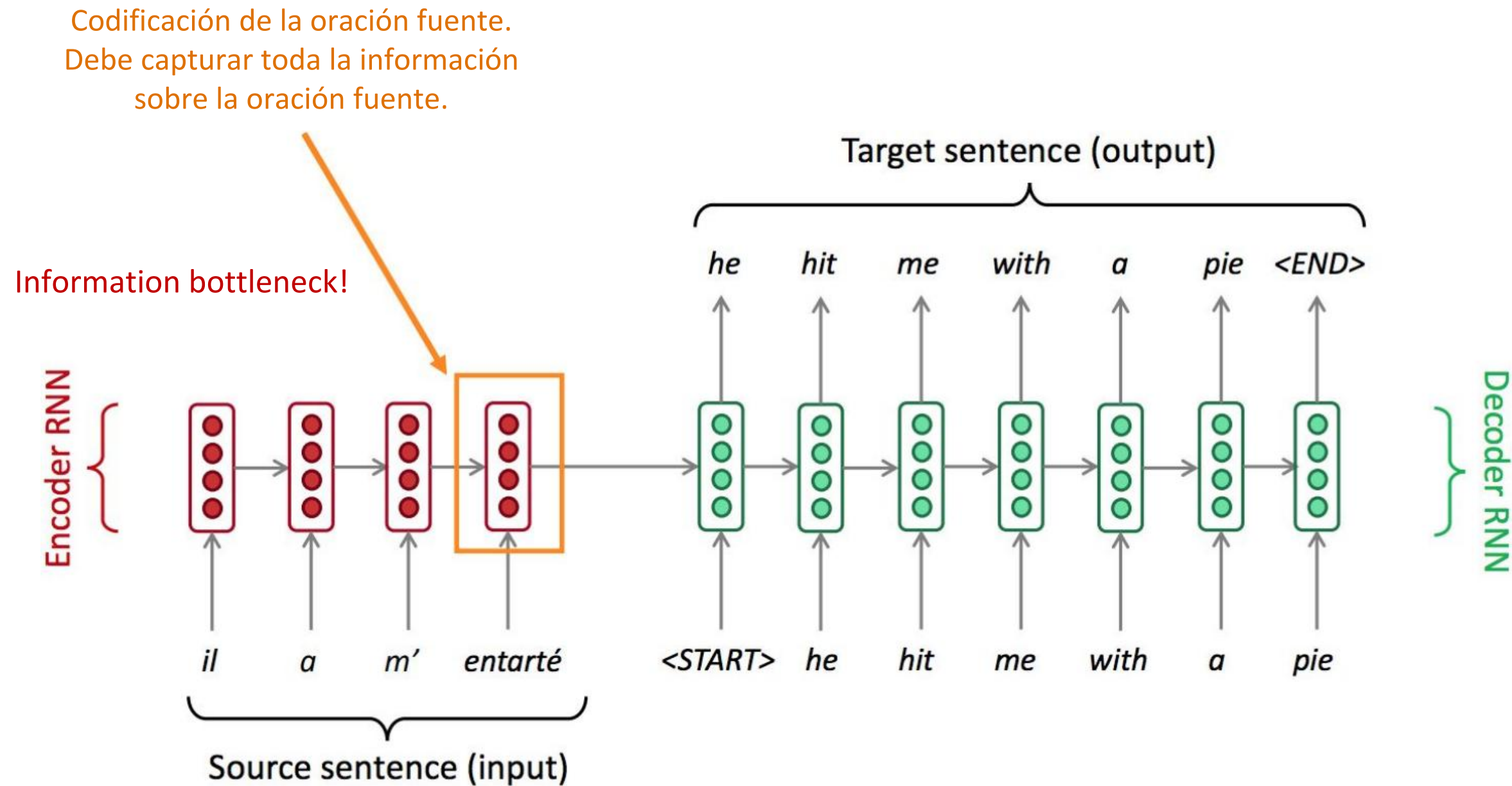
Sequence-to-sequence models



Sequence-to-sequence models



Sequence-to-sequence models



GRACIAS

Victor Flores Benites

