Aprendizado Profundo (Deep Learning)

Recurrent Neural Networks

Dario Oliveira (dario.oliveira@fgv.br)

Recurrent Neural Network

- Introduction
- RNN Architecture
- RNN Training
- RNN Variants
- Extensions

Recurrent Neural Network (RNN)

RNNs are good at modeling **sequence data**: sequence to sequence translation.

- Speech recognition
- Language translation
- Image captioning
- Multitemporal image analysis
- Video analysis
- . . .

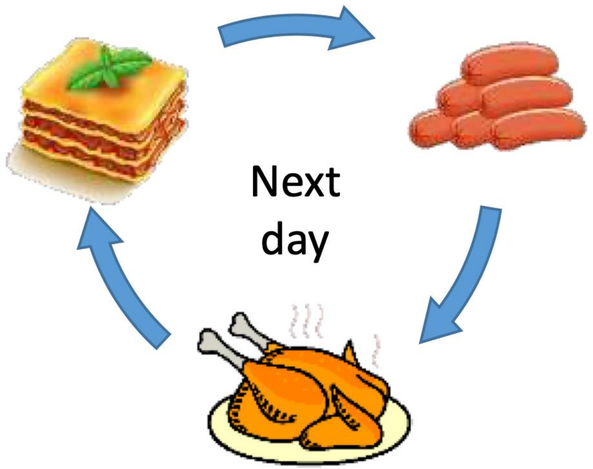
Sequence data

If you know the past…

…you may be able to predict the future.

Simple RNN

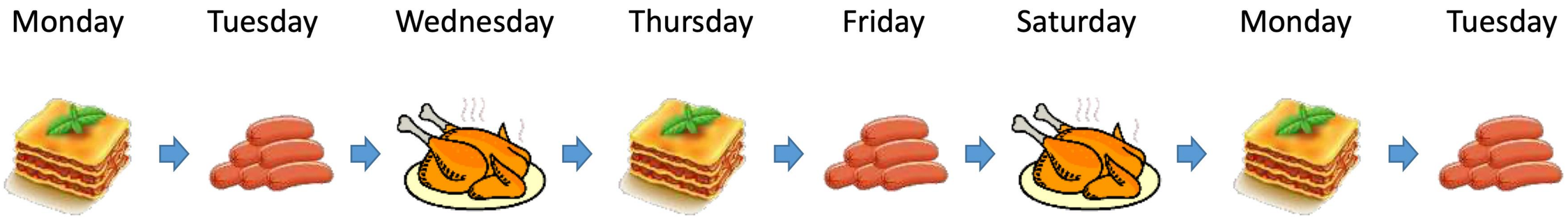
In the cafeteria…



Adapted from Luis Serrano : https://www.youtube.com/watch?v=UNmqTiOnRfg&t=1200s

Simple RNN

In the cafeteria…



Simple RNN

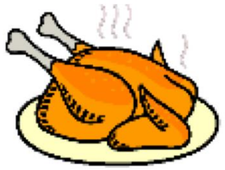
 $= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$  $= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$  $= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

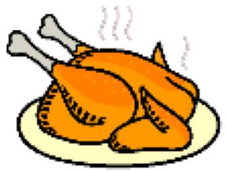
Simple RNN


$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} =$ 


$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} =$ 

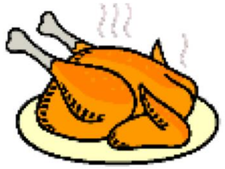

$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

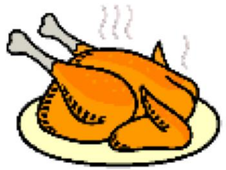
$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} =$ 

Simple RNN

$$= \begin{bmatrix}1\\0\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}1\\0\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0\\1\\0\end{bmatrix} =$$

$$= \begin{bmatrix}0\\1\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}0\\1\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0\\0\\1\end{bmatrix} =$$



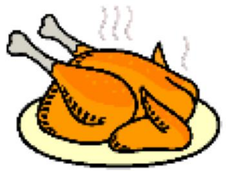


$$= \begin{bmatrix}0\\0\\1\end{bmatrix} \longrightarrow \begin{bmatrix}0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}0\\0\\1\end{bmatrix} \longrightarrow \begin{bmatrix}1\\0\\0\end{bmatrix} =$$

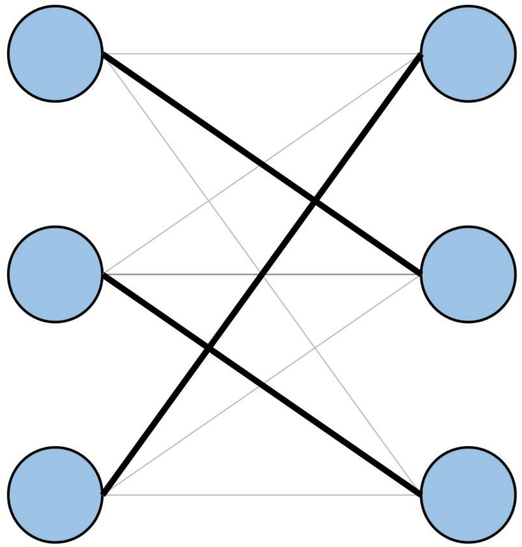
Simple RNN

 $= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

 $= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

 $= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

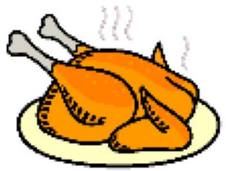
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



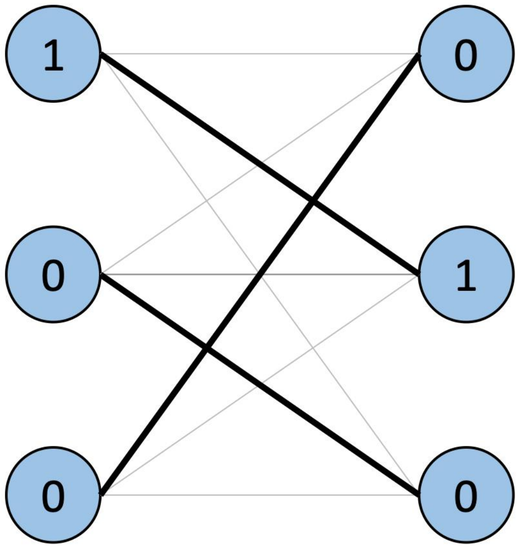
Simple RNN

 $= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

 $= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

 $= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

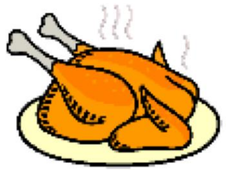
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



Simple RNN

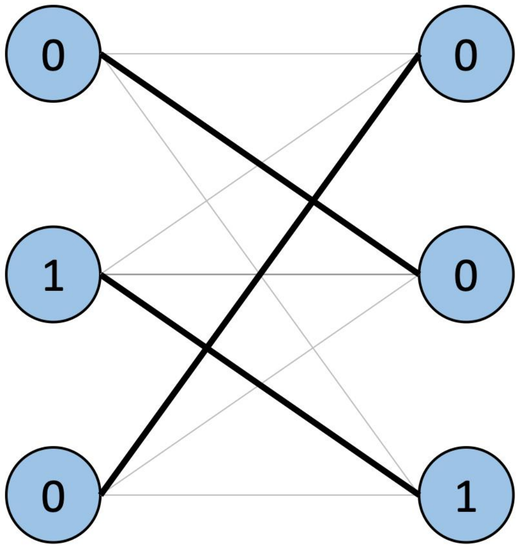
$$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

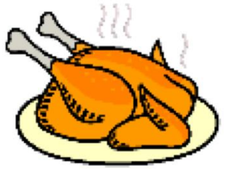
$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

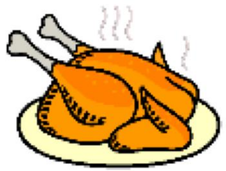




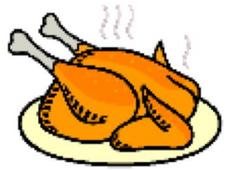
Simple RNN

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

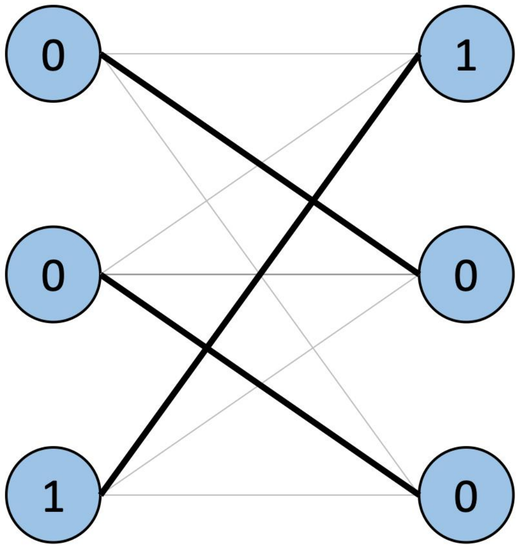
$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



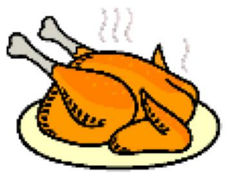
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



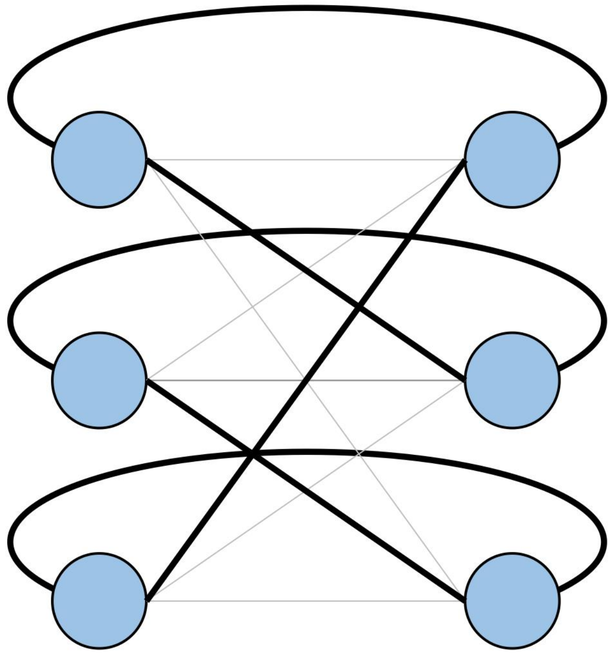
Simple RNN

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

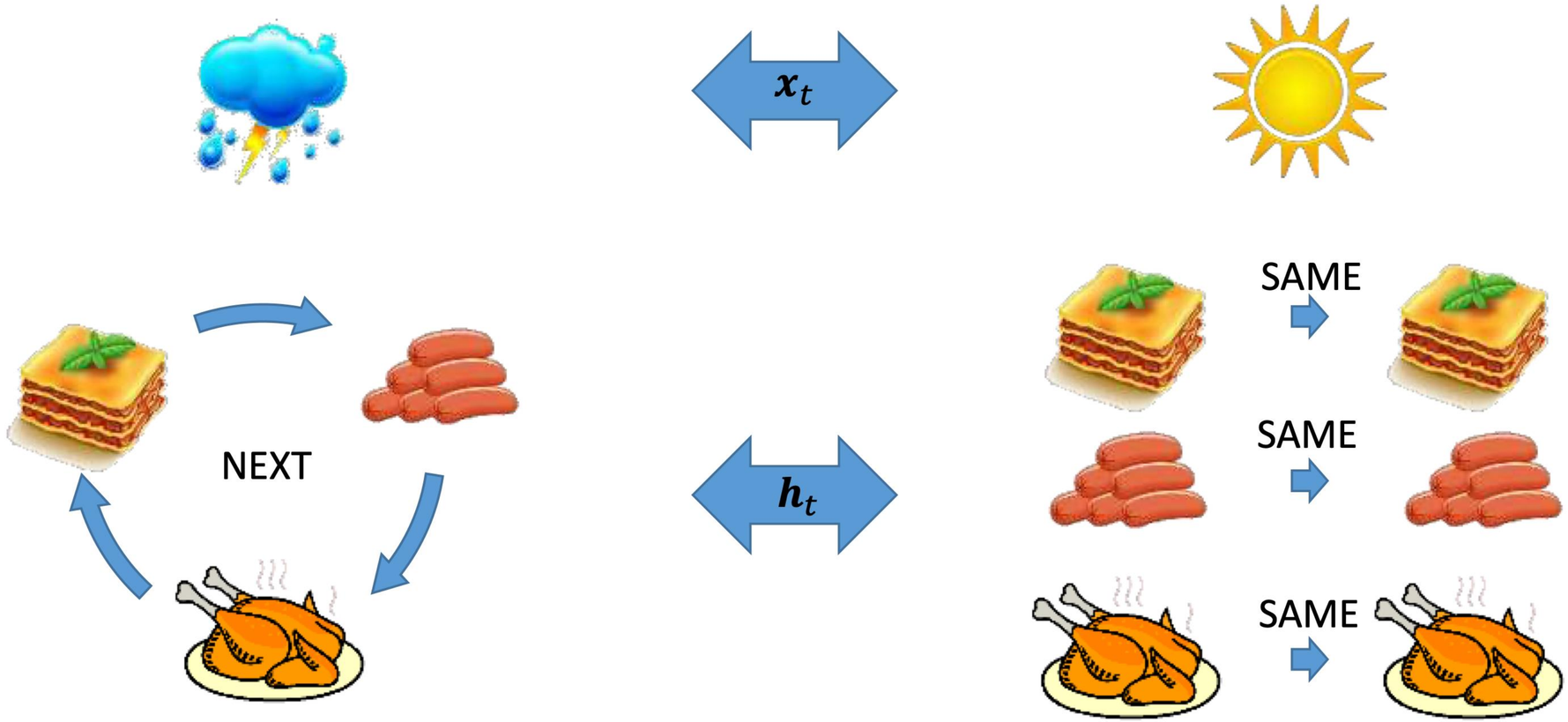


$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

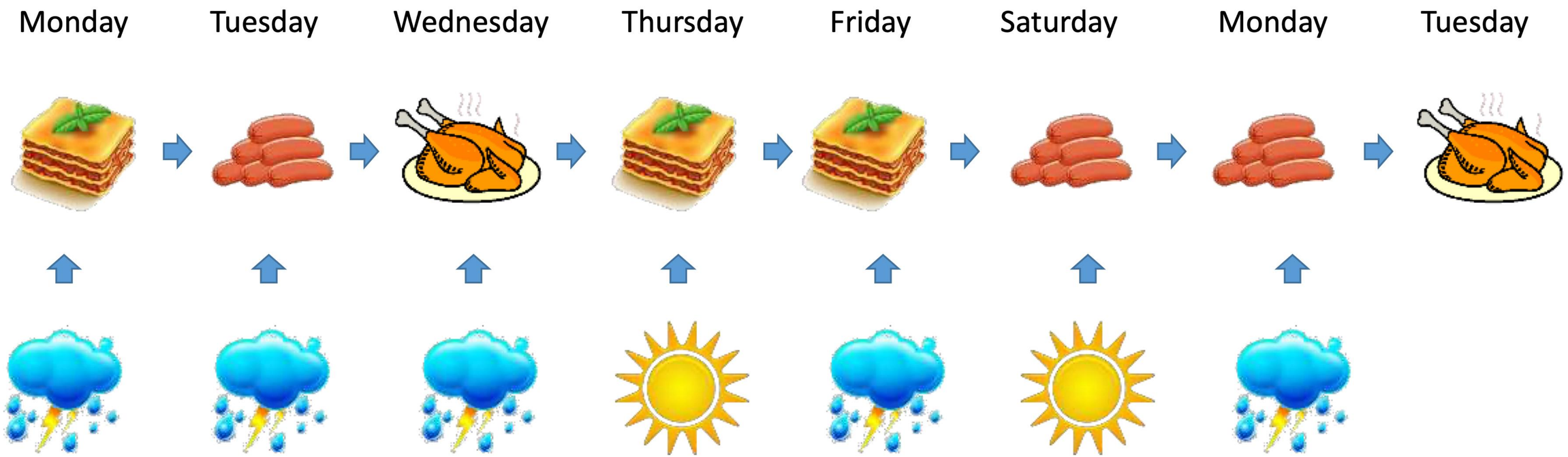


Simple RNN

In the cafeteria…



Simple RNN #2

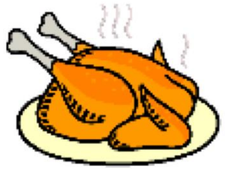

Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Monday
Tuesday

Simple RNN #2

$\boldsymbol{h_t}$

 $= \begin{bmatrix}1\\0\\0\end{bmatrix}$

 $= \begin{bmatrix}0\\1\\0\end{bmatrix}$

 $= \begin{bmatrix}0\\0\\1\end{bmatrix}$

$\boldsymbol{x_t}$

 $= \begin{bmatrix}1\\0\end{bmatrix}$

 $= \begin{bmatrix}0\\1\end{bmatrix}$

Simple RNN #2

$$\boldsymbol{W}_{hh}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{matrix} \text{SAME} \\ \\ \text{NEXT} \end{matrix}$$

food matrix

Which is the same food and which is the next food, depending on the current food (*hidden* state)?

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

SAME 

NEXT 

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

SAME 

NEXT 

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

SAME 

NEXT 

Simple RNN #2

$$W_{xh}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \hline 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

SAME

NEXT

weather matrix

Should I cook today's food or the next food, depending on the weather (input)?

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \hline 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \hline 0 \\ 0 \\ 0 \end{bmatrix}$$

SAME

NEXT

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \hline 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \hline 1 \\ 1 \\ 1 \end{bmatrix}$$

SAME

NEXT

Simple RNN #2

$$\underbrace{\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&1\\1&0&0\\0&1&0\end{bmatrix}}_{\boldsymbol{W}_{hh}} \times \underbrace{\begin{bmatrix}1\\0\\0\end{bmatrix}}_{\boldsymbol{h}_{t-1}} + \underbrace{\begin{bmatrix}1&0\\1&0\\1&0\\0&1\\0&1\\0&1\end{bmatrix}}_{\boldsymbol{W}_{xh}} \times \underbrace{\begin{bmatrix}1\\0\end{bmatrix}}_{\boldsymbol{x}_t} = \begin{bmatrix}1\\0\\0\\1\\0\\0\end{bmatrix} + \begin{bmatrix}1\\1\\1\\0\\0\\0\end{bmatrix} \begin{matrix}\text{SAME}\\ \\ \\ \\ \\ \text{NEXT}\end{matrix} = \begin{bmatrix}2\\1\\1\\0\\1\\0\end{bmatrix} \xrightarrow[\text{function } (NL)]{\text{non-linear}} \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{\text{merge}} \begin{bmatrix}1+0\\0+0\\0+0\end{bmatrix} = \begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\end{bmatrix}}_{\boldsymbol{W}_{ho}} \times \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\end{bmatrix}$$

merge

Simple RNN #2

$$\underbrace{\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&1\\1&0&0\\0&1&0\end{bmatrix}}_{\boldsymbol{W}_{hh}} \times \underbrace{\begin{bmatrix}1\\0\\0\end{bmatrix}}_{\boldsymbol{h}_{t-1}} + \underbrace{\begin{bmatrix}1&0\\1&0\\1&0\\0&1\\0&1\\0&1\end{bmatrix}}_{\boldsymbol{W}_{xh}} \times \underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{\boldsymbol{x}_t} = \begin{bmatrix}1\\0\\0\\0\\1\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\\1\\1\end{bmatrix} \begin{matrix}\\\\\text{SAME}\\\\\text{NEXT}\\\\\end{matrix} = \begin{bmatrix}1\\0\\0\\1\\2\\1\end{bmatrix} \xrightarrow[\text{function } (NL)]{\text{non-linear}} \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} \xrightarrow{\text{merge}} \begin{bmatrix}0+0\\0+1\\0+0\end{bmatrix} = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\end{bmatrix}}_{\boldsymbol{W}_{ho}} \times \begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

merge

Simple RNN #2

$$\underbrace{\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&1\\1&0&0\\0&1&0\end{bmatrix}}_{\boldsymbol{W}_{hh}} \times \underbrace{\begin{bmatrix}0\\1\\0\end{bmatrix}}_{\boldsymbol{h}_{t-1}} + \underbrace{\begin{bmatrix}1&0\\1&0\\1&0\\0&1\\0&1\\0&1\end{bmatrix}}_{\boldsymbol{W}_{xh}} \times \underbrace{\begin{bmatrix}1\\0\end{bmatrix}}_{\boldsymbol{x}_t} = \begin{bmatrix}0\\1\\0\\0\\0\\1\end{bmatrix} + \begin{bmatrix}1\\1\\1\\0\\0\\0\end{bmatrix} \begin{matrix}\text{SAME}\\ \\ \\ \\ \\ \text{NEXT}\end{matrix} = \begin{bmatrix}1\\2\\1\\0\\0\\1\end{bmatrix} \xrightarrow[\text{function } (NL)]{\text{non-linear}} \begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix} \xrightarrow{\text{merge}} \begin{bmatrix}0+0\\1+0\\0+0\end{bmatrix} = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\end{bmatrix}}_{\boldsymbol{W}_{ho}} \times \underbrace{\begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix}}_{\text{merge}} = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

Simple RNN #2

$$\underbrace{\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&1\\1&0&0\\0&1&0\end{bmatrix}}_{\boldsymbol{W}_{hh}} \times \underbrace{\begin{bmatrix}0\\1\\0\end{bmatrix}}_{\boldsymbol{h}_{t-1}} + \underbrace{\begin{bmatrix}1&0\\1&0\\1&0\\0&1\\0&1\\0&1\end{bmatrix}}_{\boldsymbol{W}_{xh}} \times \underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{\boldsymbol{x}_t} = \begin{bmatrix}0\\1\\0\\0\\0\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\\1\\1\\1\end{bmatrix} \begin{matrix}\text{SAME}\\ \\ \\ \\ \text{NEXT}\\ \\ \end{matrix} = \begin{bmatrix}0\\1\\0\\1\\1\\2\end{bmatrix} \xrightarrow[\text{function } (NL)]{\text{non-linear}} \begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix} \xrightarrow{\text{merge}} \begin{bmatrix}0+0\\0+0\\0+1\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\end{bmatrix}}_{\boldsymbol{W}_{ho}} \times \begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

merge

Simple RNN #2

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&1\\1&0&0\\0&1&0\end{bmatrix} \times \begin{bmatrix}0\\0\\1\end{bmatrix} + \begin{bmatrix}1&0\\1&0\\1&0\\0&1\\0&1\\0&1\end{bmatrix} \times \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\1\\1\\0\\0\end{bmatrix} + \begin{bmatrix}1\\1\\1\\0\\0\\0\end{bmatrix} \text{ SAME / NEXT} = \begin{bmatrix}1\\1\\2\\1\\0\\0\end{bmatrix} \xrightarrow{\text{non-linear function } (NL)} \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix} \xrightarrow{\text{merge}} \begin{bmatrix}0+0\\0+0\\1+0\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$\boldsymbol{W}_{hh}$ $\boldsymbol{h}_{t-1}$ $\boldsymbol{W}_{xh}$ $\boldsymbol{x}_t$

merge

$$\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\end{bmatrix} \times \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$\boldsymbol{W}_{ho}$

Simple RNN #2

$$\underbrace{\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\0&0&1\\1&0&0\\0&1&0\end{bmatrix}}_{\boldsymbol{W}_{hh}} \times \underbrace{\begin{bmatrix}0\\0\\1\end{bmatrix}}_{\boldsymbol{h}_{t-1}} + \underbrace{\begin{bmatrix}1&0\\1&0\\1&0\\0&1\\0&1\\0&1\end{bmatrix}}_{\boldsymbol{W}_{xh}} \times \underbrace{\begin{bmatrix}0\\1\end{bmatrix}}_{\boldsymbol{x}_t} = \begin{bmatrix}0\\0\\1\\1\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\1\\1\\1\\1\end{bmatrix} \begin{matrix}\\\\\text{SAME}\\\\\text{NEXT}\\\\\end{matrix} = \begin{bmatrix}0\\0\\1\\2\\1\\1\end{bmatrix} \xrightarrow[\text{function } (NL)]{\text{non-linear}} \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix} \xrightarrow{\text{merge}} \begin{bmatrix}0+1\\0+0\\0+0\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}1&0&0&1&0&0\\0&1&0&0&1&0\\0&0&1&0&0&1\end{bmatrix}}_{\boldsymbol{W}_{ho}} \times \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\end{bmatrix}$$

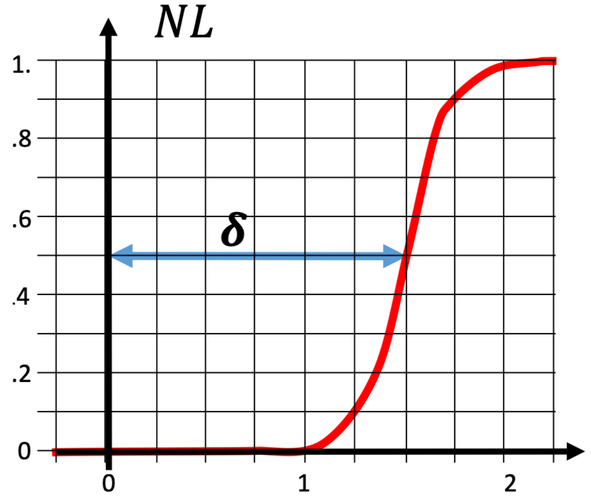
merge

Simple RNN #2

Actually, a Neural Network …

$$\boldsymbol{h}_t = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{W}_{ho}} \times NL \left\{ \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{\boldsymbol{W}_{hh}} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\boldsymbol{h}_{t-1}} + \underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}_{\boldsymbol{W}_{xh}} \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\boldsymbol{x}_t} \right\}$$

$$\boldsymbol{h}_t = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{W}_{ho}} \times NL \left\{ \begin{bmatrix} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{\boldsymbol{W}_{hh}} & \underbrace{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}_{\boldsymbol{W}_{xh}} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\} \begin{matrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{matrix}$$



Simple RNN #2

Actually, a Neural Network …

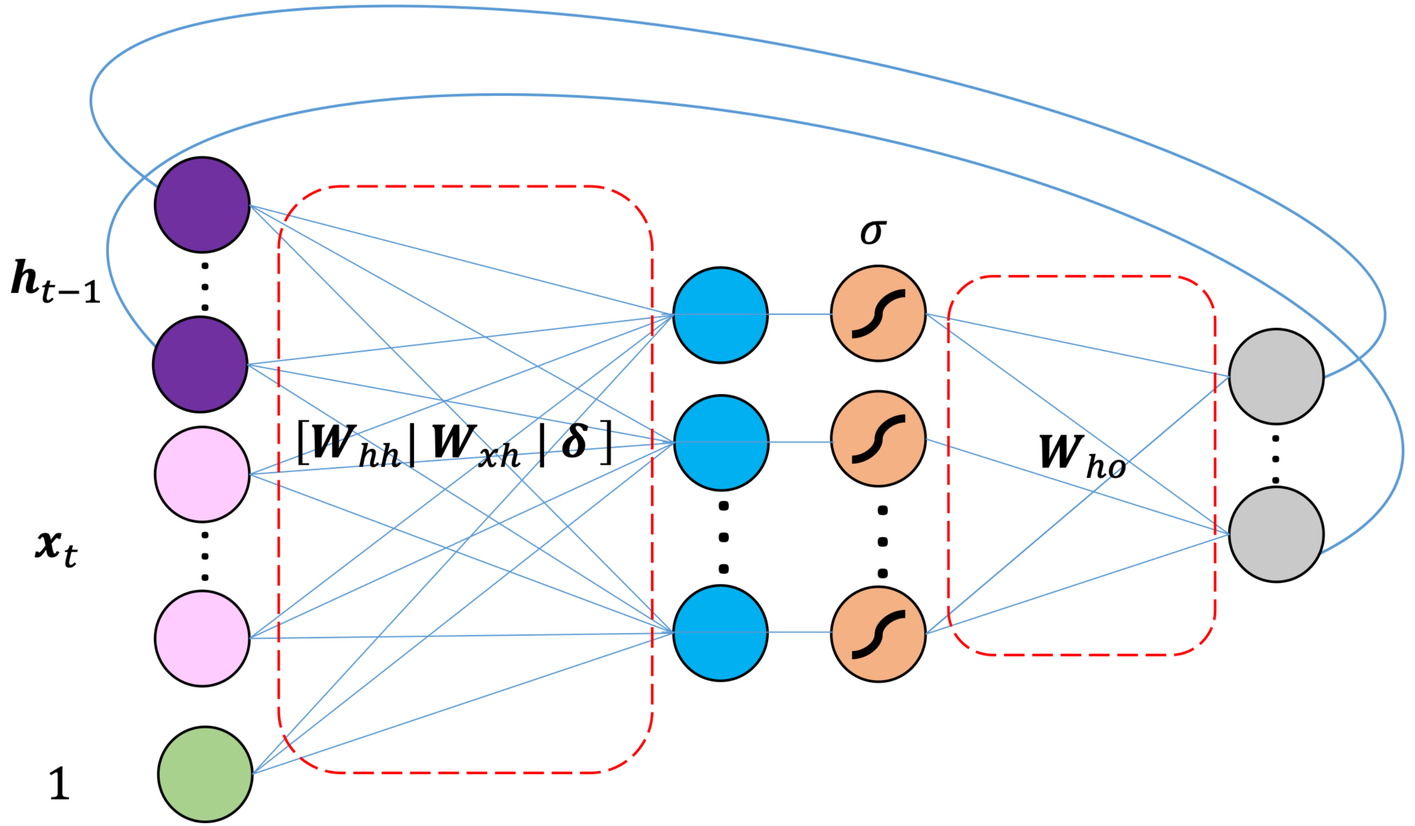
$$\boldsymbol{h}_t = \overbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}^{\boldsymbol{W}_{ho}} \times NL \left\{ \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}^{\boldsymbol{W}_{hh}} \overbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}^{\boldsymbol{h}_{t-1}} + \overbrace{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}^{\boldsymbol{W}_{xh}} \overbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}^{\boldsymbol{x}_t} + \overbrace{\begin{bmatrix} \delta \\ \delta \\ \delta \\ \delta \\ \delta \\ \delta \end{bmatrix}}^{\boldsymbol{\delta}} \right\}$$

$$\boldsymbol{h}_t = \overbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}^{\boldsymbol{W}_{ho}} \times NL \left\{ \begin{bmatrix} \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}^{\boldsymbol{W}_{hh}} & \overbrace{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}}^{\boldsymbol{W}_{xh}} & \overbrace{\begin{bmatrix} \delta \\ \delta \\ \delta \\ \delta \\ \delta \\ \delta \end{bmatrix}}^{\boldsymbol{\delta}} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \right\} \begin{matrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \\ 1 \end{matrix}$$

$$\boldsymbol{h}_t = \boldsymbol{W}_{ho} \ \sigma \left[\boldsymbol{W}_{hh} \middle| \boldsymbol{W}_{xh} \middle| \boldsymbol{\delta} \right] \begin{bmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \\ 1 \end{bmatrix}$$

Simple RNN #2

Actually, a **Recurrent Neural Network**…

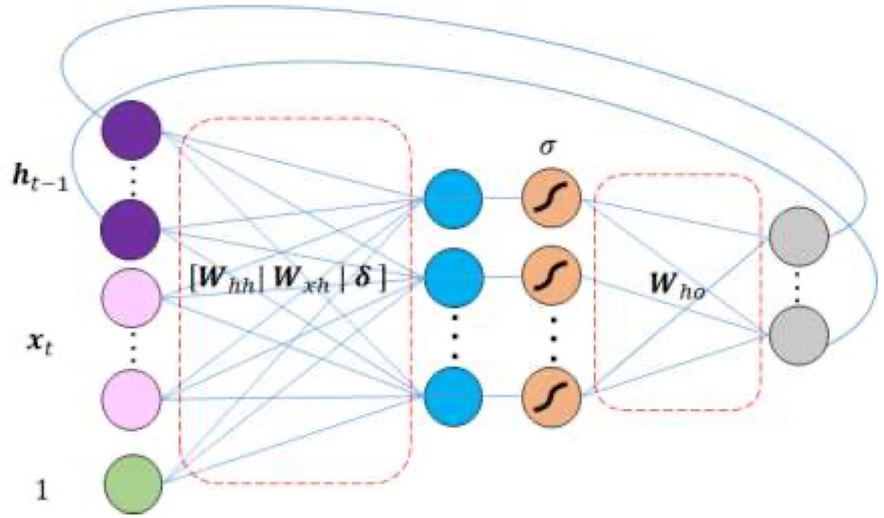


$$h_t = W_{ho}\ \sigma\ [W_{hh} | W_{xh} | \delta] \begin{bmatrix} h_{t-1} \\ x_t \\ 1 \end{bmatrix}$$

Simple RNN

The RNN output can be only the last one, or the whole sequence of outputs.

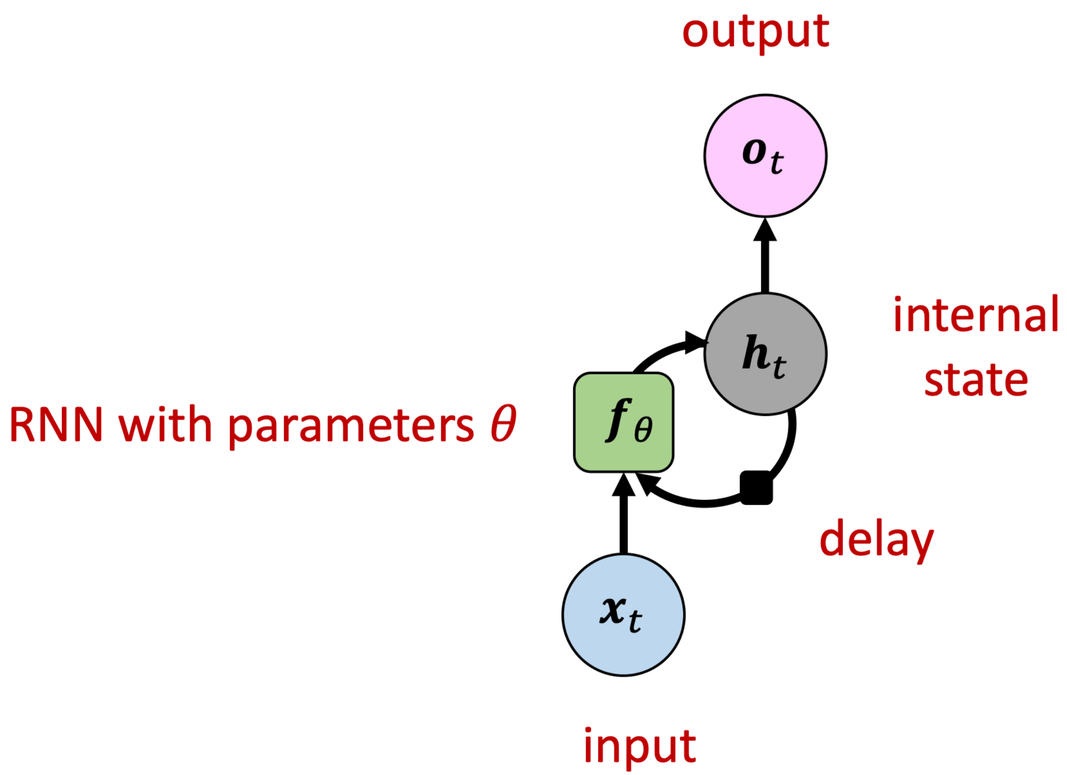
```
# x_t shape is (32, 10, 3) -> (batch size, time steps, feature dims.)
inputs = x_t

# returning the last output
simple_rnn = tf.keras.layers.SimpleRNN(2) # 2 is the output vector dims.
output = simple_rnn(inputs)  # output has shape (32, 2)

# returning the whole sequence of outputs
simple_rnn = tf.keras.layers.SimpleRNN(2, return_sequences=True)
outputs = simple_rnn(inputs) # output has shape (32, 10, 2)
```



Recurrent Neural Network

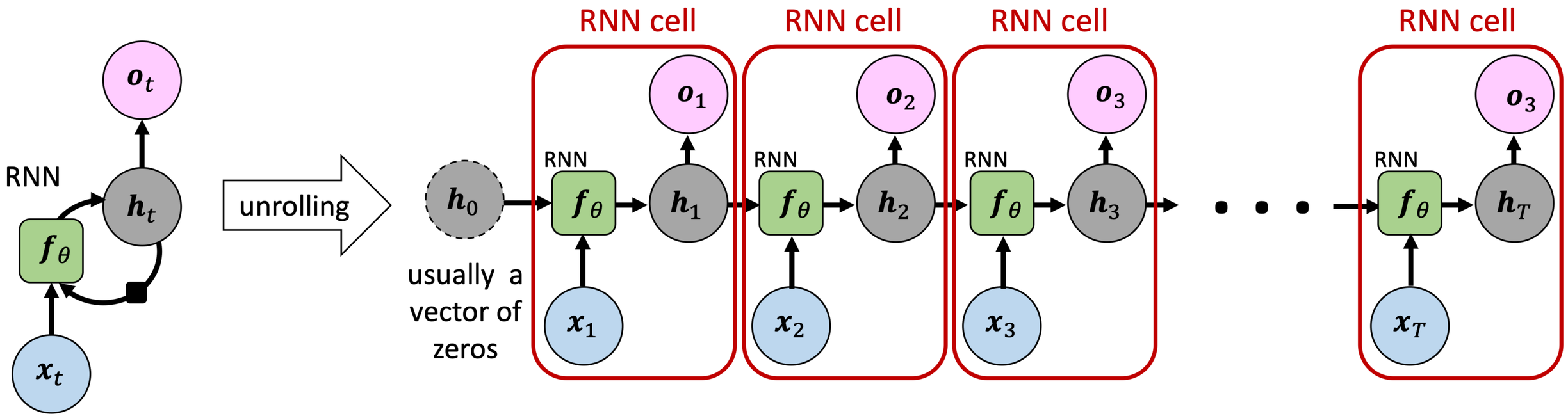
- Introduction
- RNN Architecture
- RNN Training
- RNN Variants
- Extensions

RNN architecture



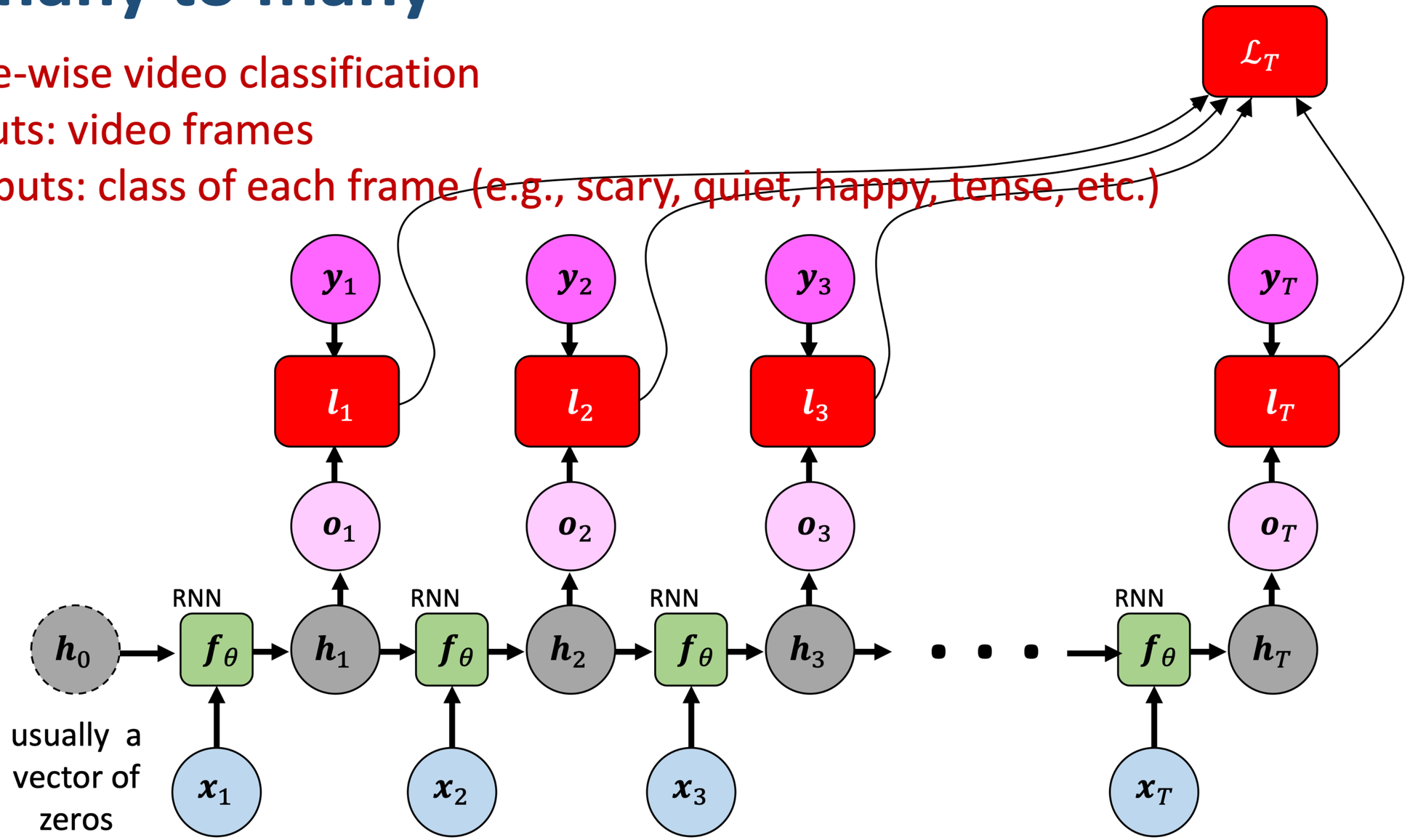
RNN unrolling

θ fixed for all time steps (cells)



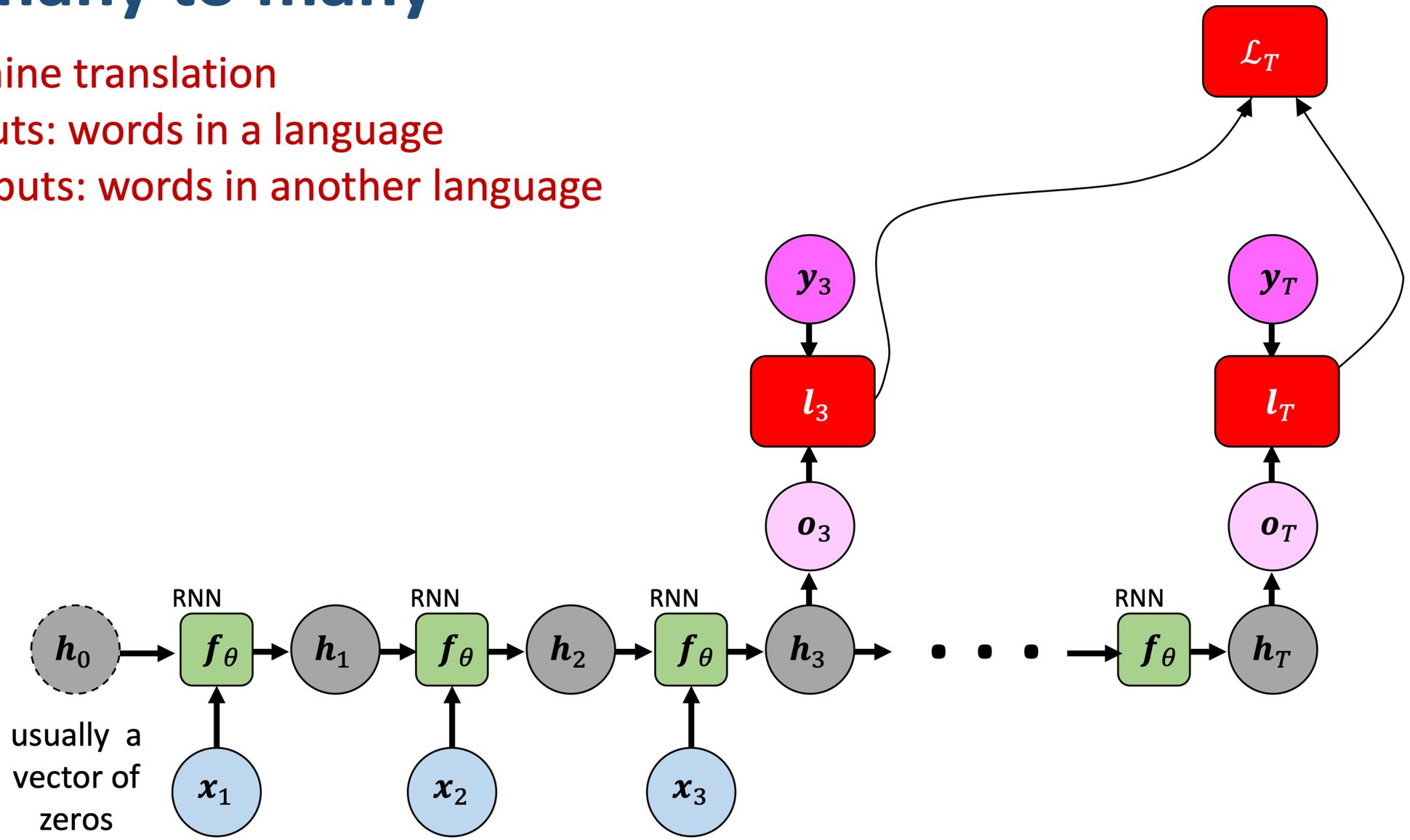
RNN many to many

e.g., frame-wise video classification
Many inputs: video frames
Many outputs: class of each frame (e.g., scary, quiet, happy, tense, etc.)



RNN many to many

e.g., machine translation
Many inputs: words in a language
Many outputs: words in another language

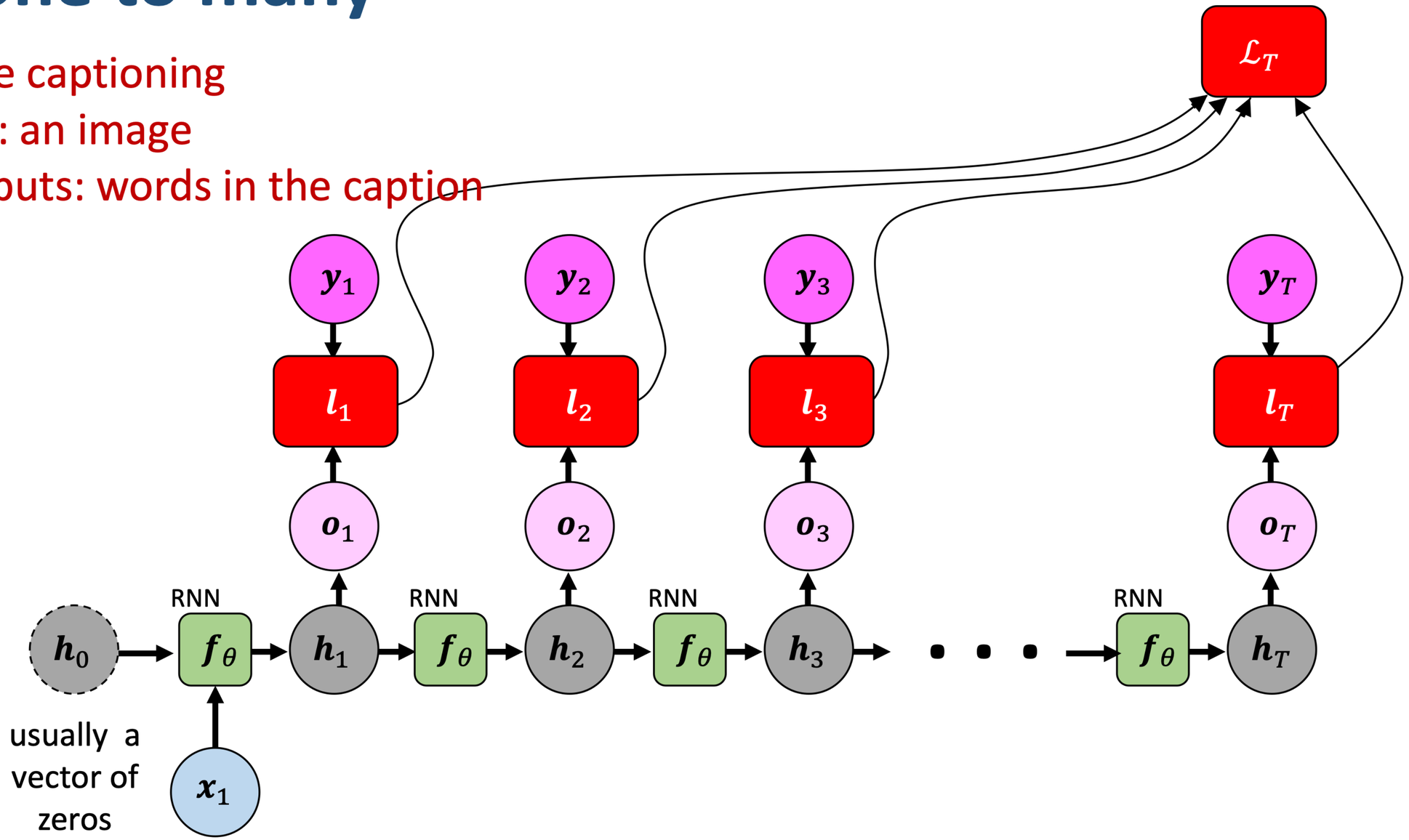


RNN one to many

e.g., image captioning

One input: an image

Many outputs: words in the caption

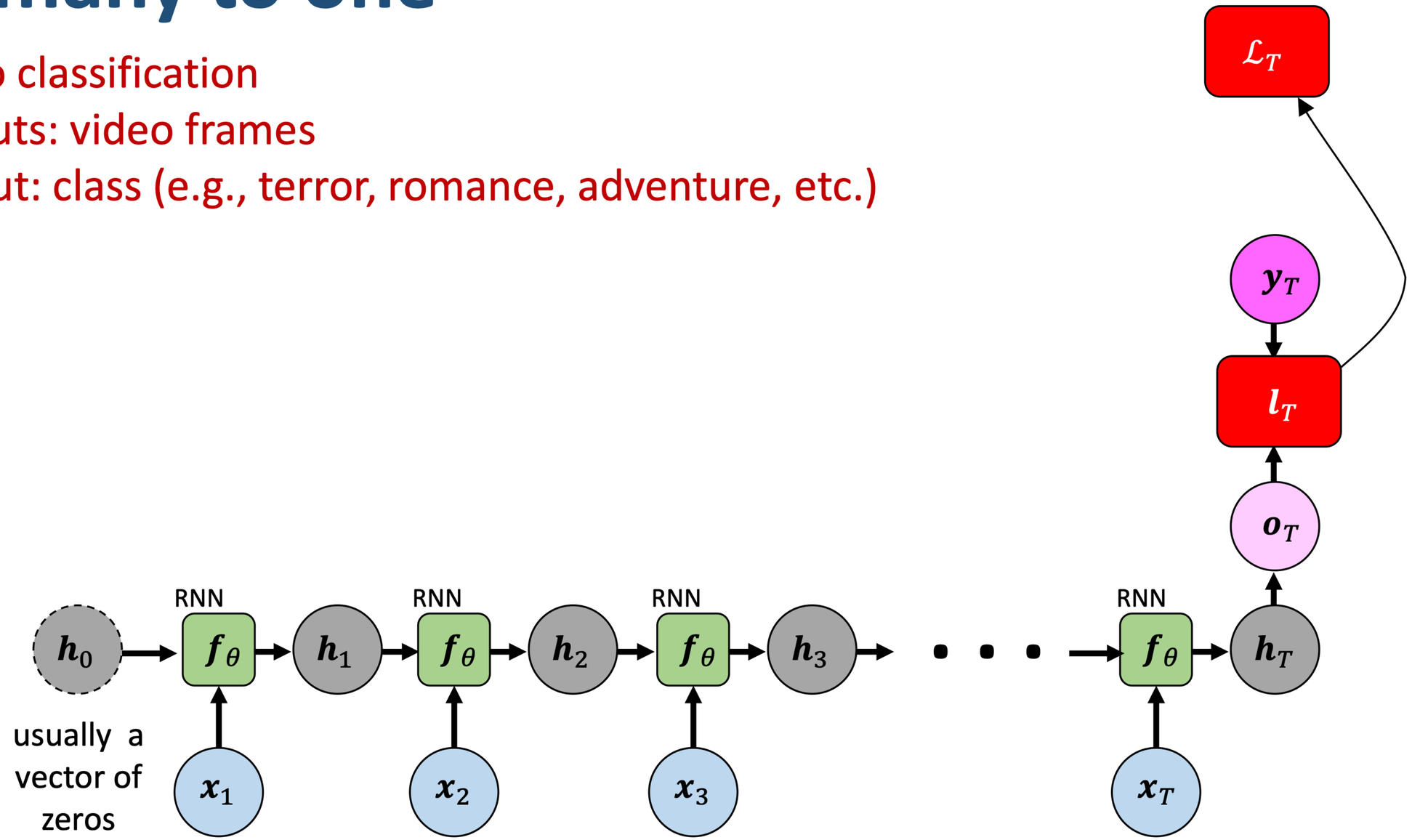


RNN many to one

e.g., video classification

Many inputs: video frames

One output: class (e.g., terror, romance, adventure, etc.)



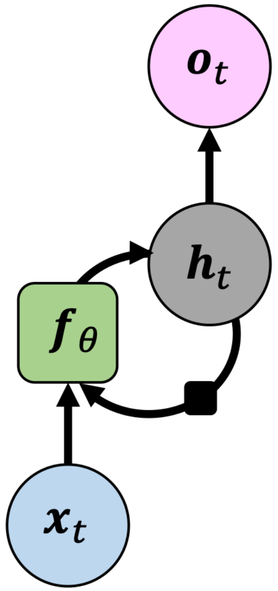
Recurrent Neural Network

We can process a sequence of vectors $\boldsymbol{x}_t$ by applying a recurrence formula at every time step:

new state — prior state

$$\boldsymbol{h}_t = f_\theta(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

some function with parameters θ — input vector at the current step



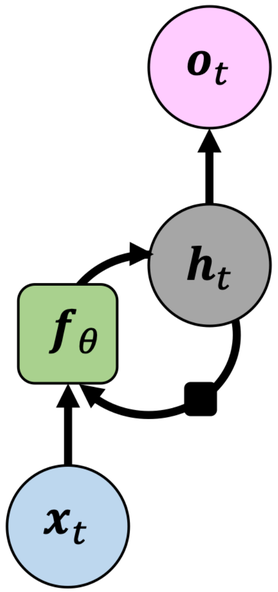
The same function $\boldsymbol{f_\theta}$ and the same set of parameters $\boldsymbol{\theta}$ are used at every time step.

Recurrence: recursivity

For $T = 3$: $\boldsymbol{h}_3 = f_\theta(\boldsymbol{h}_2, \boldsymbol{x}_3)$

$= f_\theta(f_\theta(\boldsymbol{h}_1, \boldsymbol{x}_2), \boldsymbol{x}_3)$

$= f_\theta(f_\theta(f_\theta(\boldsymbol{h}_0, \boldsymbol{x}_1), \boldsymbol{x}_2), \boldsymbol{x}_3)$



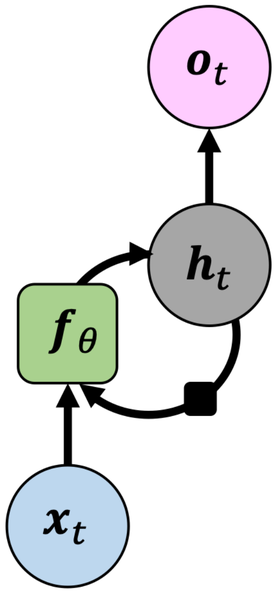
Let's assume the following simple RNN:

$$\boldsymbol{h}_t = f_\theta(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t)$$

$$\boldsymbol{o}_t = \boldsymbol{W}_{ho}\boldsymbol{h}_t$$

Recurrence

Assume the following simple RNN for $T = 3$:



$$\boldsymbol{h}_t = f_\theta(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t)$$

$$\boldsymbol{h}_3 = tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_2 + \boldsymbol{W}_{xh}\boldsymbol{x}_3)$$

$$= tanh\left(\boldsymbol{W}_{hh}\left(tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_1 + \boldsymbol{W}_{xh}\boldsymbol{x}_2)\right) + \boldsymbol{W}_{xh}\boldsymbol{x}_3\right)$$

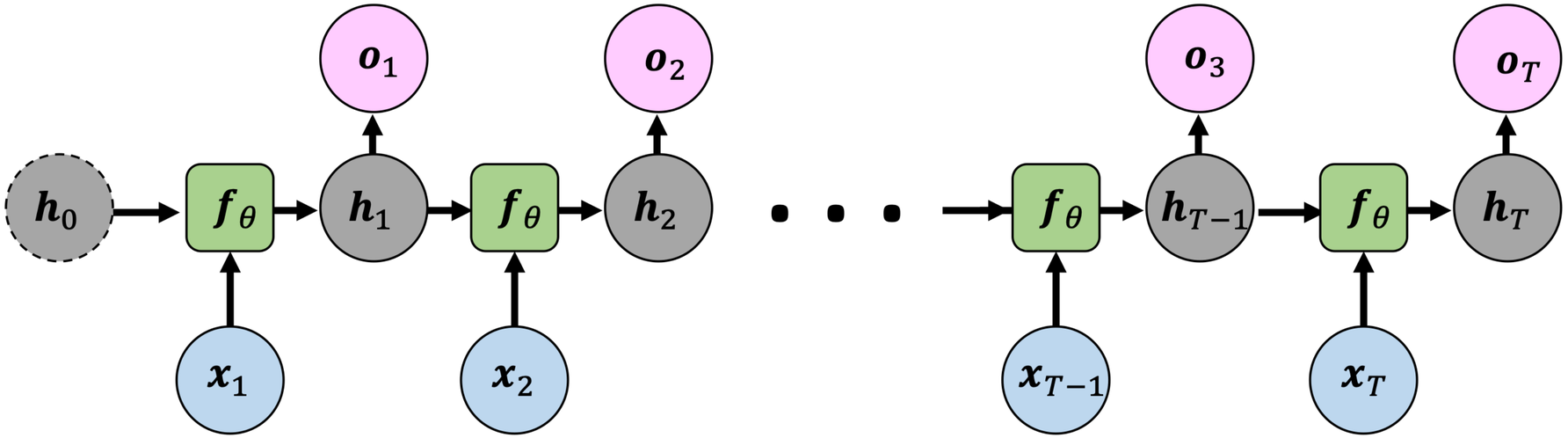
$$= tanh\left(\boldsymbol{W}_{hh}\left(tanh\left(\boldsymbol{W}_{hh}\left(tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_0 + \boldsymbol{W}_{xh}\boldsymbol{x}_1)\right) + \boldsymbol{W}_{xh}\boldsymbol{x}_1\right)\right) + \boldsymbol{W}_{xh}\boldsymbol{x}_3\right)$$

Recurrent Neural Network

- Introduction
- RNN Architecture
- RNN Training
- RNN Variants
- Extensions

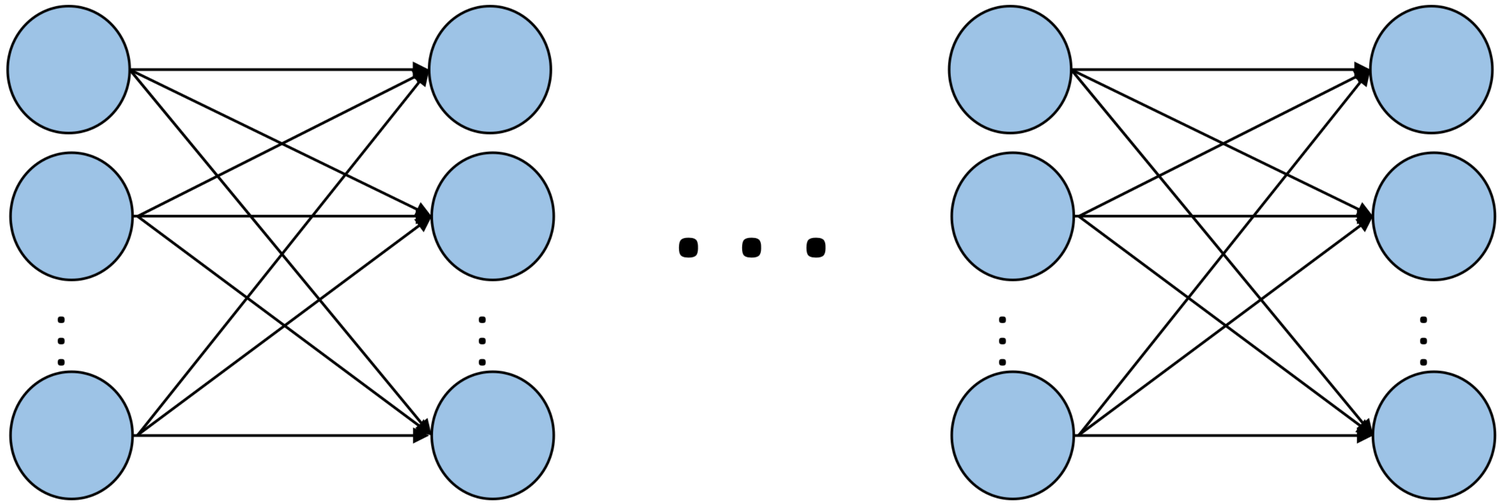
Backpropagation Through Time (BPTT)

As we saw, each RNN cell is a NN layer.



Backpropagation Through Time (BPTT)

As we saw, each RNN cell is a NN layer:



- So, the unrolled RNN is a deep NN.
- Then, the backpropagation algorithm can be applied for training RNNs.
- The only difference is that in RNN the parameters are the same in all time steps.

Backpropagation Through Time (BPTT)

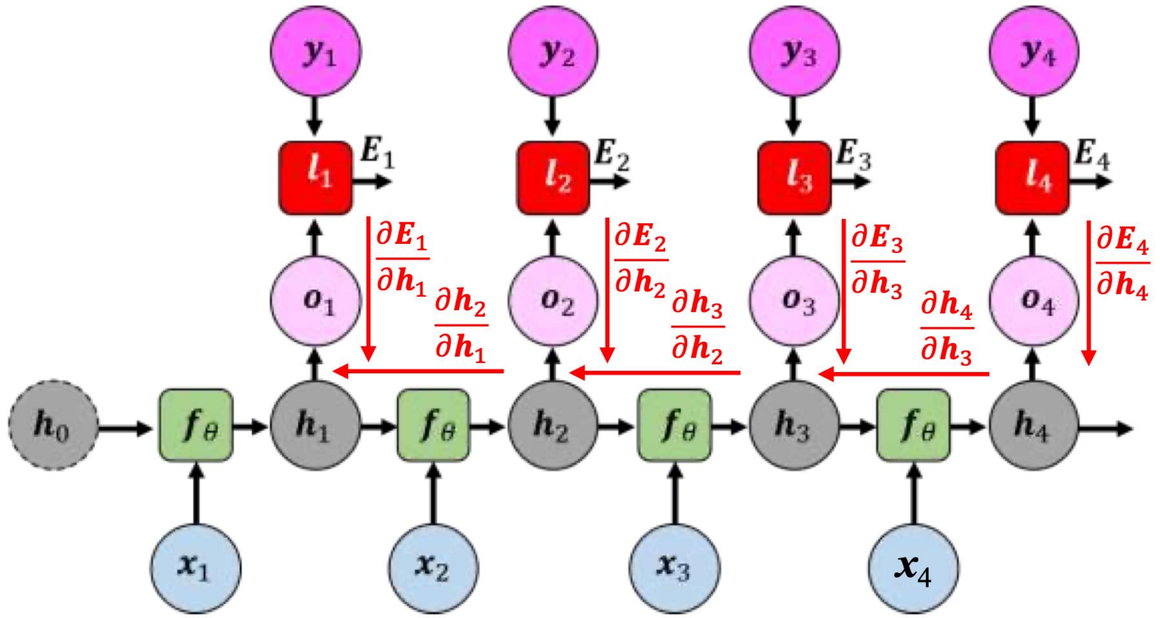
We can summarize the algorithm as follows:

- Present a sequence of time steps of input and output (reference) to the network.
- Unroll the network then calculate and accumulate errors across each time step.
- Roll-up the network and update weights.
- Repeat for the next training batch.

Described at: https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

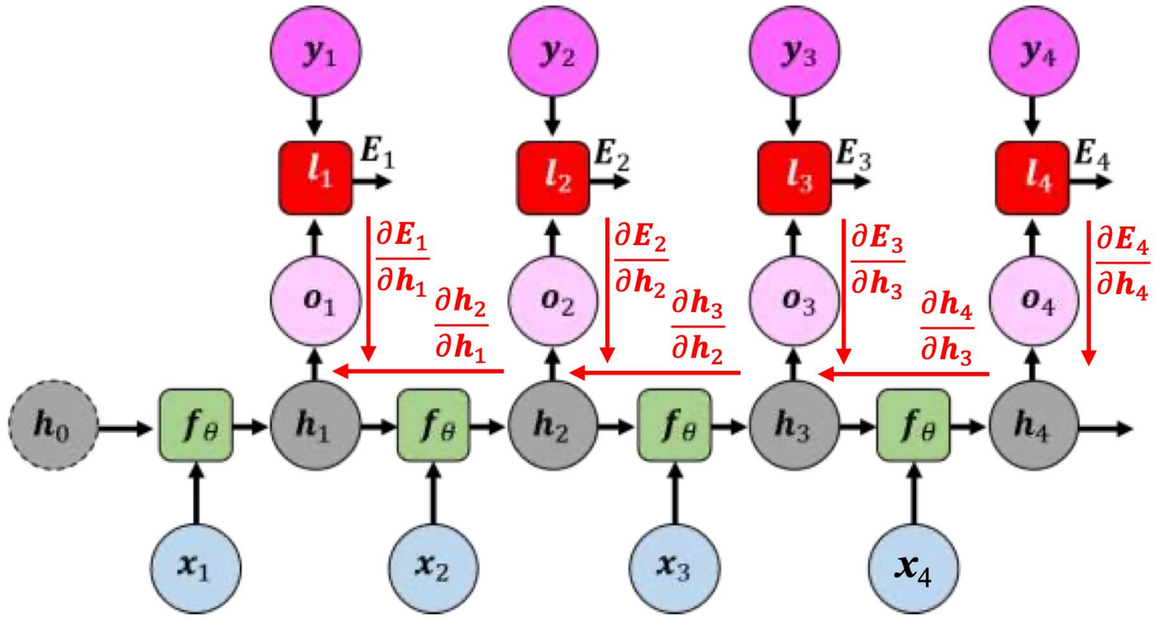
Backpropagation Through Time (BPTT)

We can summarize the algorithm as follows:



Problem with BPTT

During training the gradient of error must propagate all the way back to the first time step.

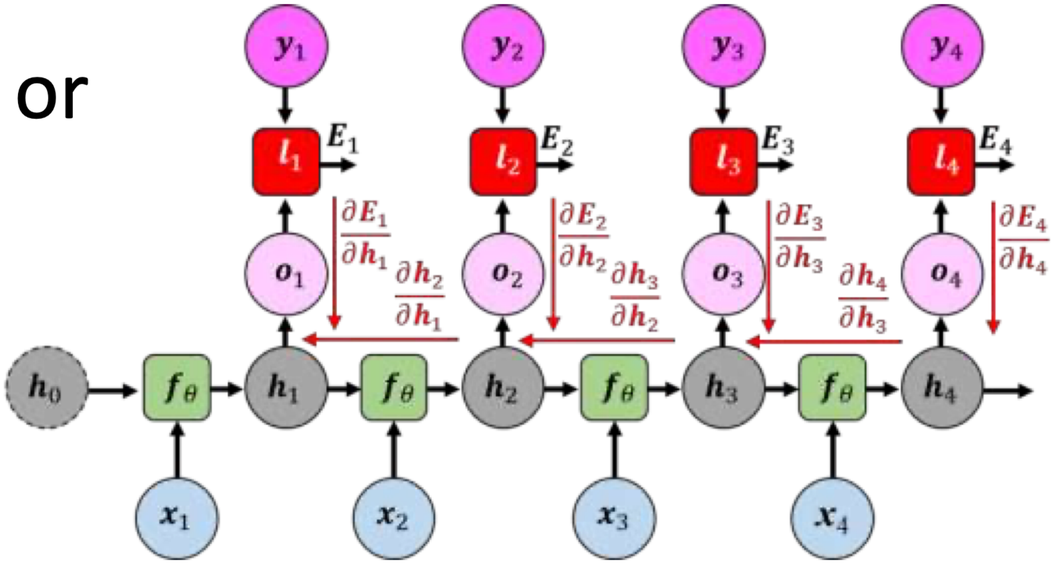


Thus, for long sequences, each batch computation, as well as the memory demand, might become unacceptably high.

Problem with BPTT

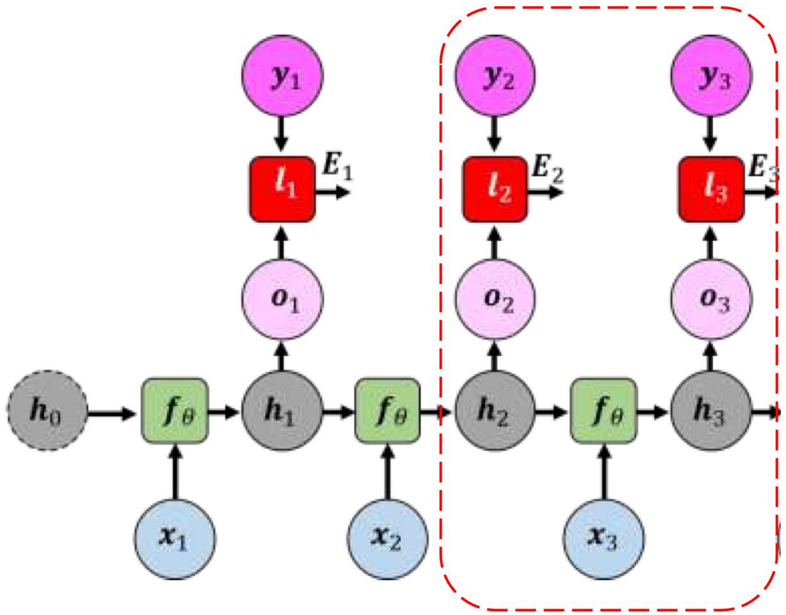
During training the gradient of error must propagate all the way back to the first time step.

- If input sequences comprises thousands of time steps, then this will be the number of derivatives required for a single weight update.
- This can cause weights to vanish or explode (go to zero or overflow) and make learning slow.



Truncated BPTT (TBPTT)

- Run the network for k_1 time steps.
- Update considering only the last k_2 time steps, with $k_2 < k_1$.
- Example: k_1 =3 and k_2 =2



Recurrent Neural Network

- Introduction
- RNN Architecture
- RNN Training
- RNN Variants
- Extensions

Limitations of RNN

Distance from meaningful information:

The clouds are in the ___

I grew up in France… I speak fluent _____

Limitations of RNN

Selecting meaningfully information:

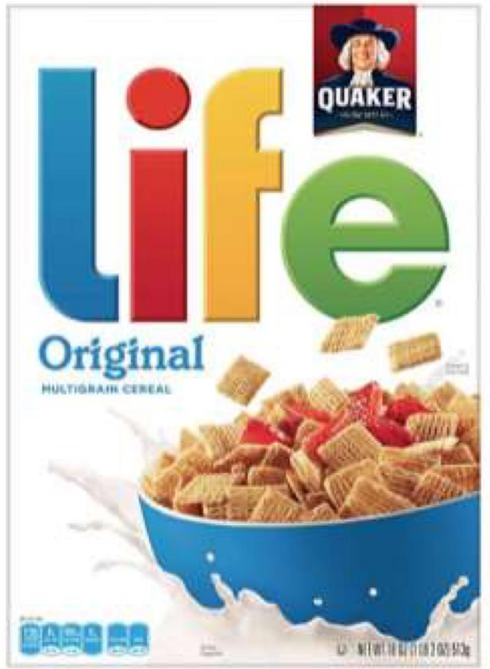
Customers Review 2,491

Thanos

September 2018

Verified Purchase

Amazing! This box of cereal gave me a **perfectly balanced breakfast**, as all things should be. I only ate half of it but **will definitely be buying again!**



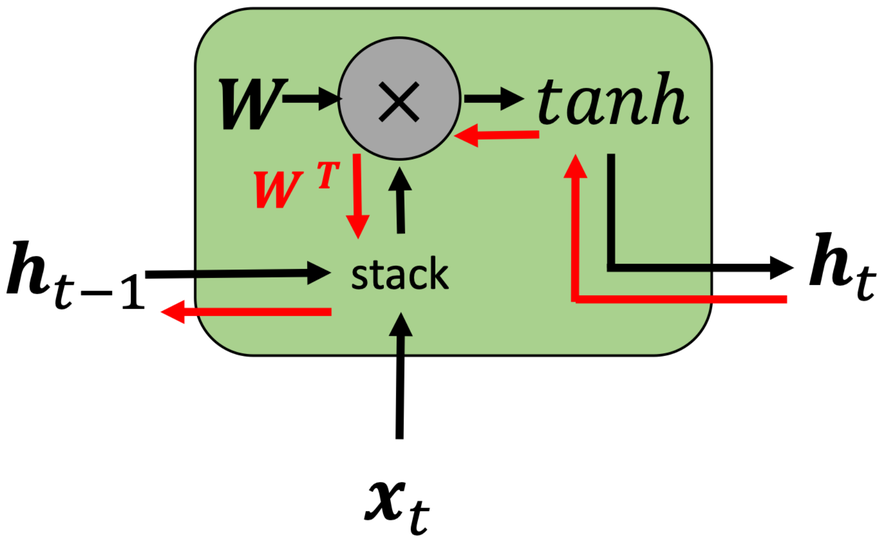
A Box of Cereal

$3.99

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

The short term memory problem

Consider the basic RNN so far:



Backpropagation from $\boldsymbol{h_t}$ to $\boldsymbol{h_{t-1}}$ multiplies by $\boldsymbol{W^T}$

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

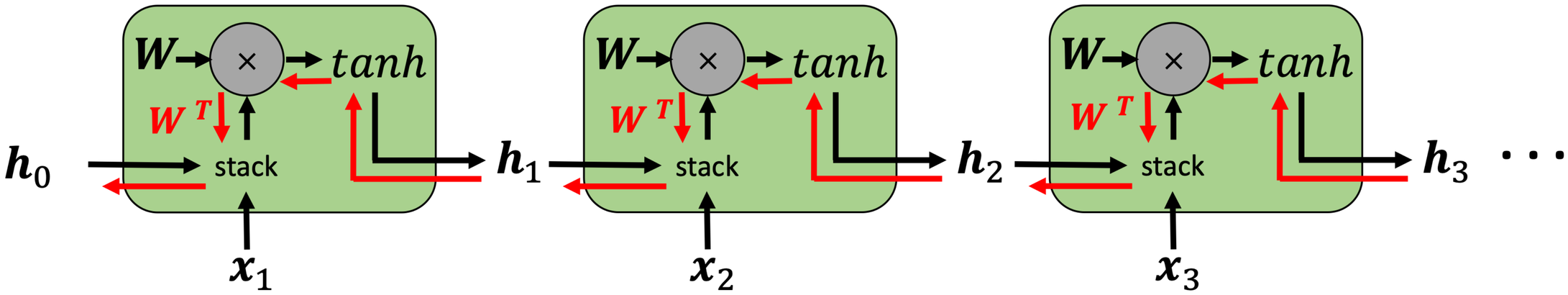
$$= tanh\left((\boldsymbol{W_{hh}} \quad \boldsymbol{W_{xh}})\begin{pmatrix}\boldsymbol{h_{t-1}}\\ \boldsymbol{x_t}\end{pmatrix}\right)$$

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

See further discussion at: https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem

The short term memory problem

Gradient flow…



Computing gradient of $\boldsymbol{h_0}$ involves many factors of $\boldsymbol{W^T}$ (and repeated $tanh$)

- Largest singular value > 1: **exploding gradients**
  - → **Gradient clipping**: scale the gradient if its norm exceeds a threshold
- Largest singular value < 1: **vanishing gradients**
  - → **Change RNN architecture**

See further discussion at: https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem

Long Short Term Memory (LSTM)

- A special kind of RNN.
- Introduced in 1997 by Hochreiter and Schmidhuber.
- Designed tackle the long-term dependency and vanishing gradient problems.
- Cells have two outputs h_t(hidden/output state) and c_t (cell state).

Hochreiter & Schmidhuber 1997 available at : http://www.bioinf.jku.at/publications/older/2604.pdf

Long Short Term Memory (LSTM)

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{W_f} \\ \boldsymbol{W_i} \\ \boldsymbol{W_s} \\ \boldsymbol{W_c} \end{pmatrix}$$

Vanilla RNN

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

LSTM

$$\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{i} \\ \boldsymbol{s} \\ \tilde{\boldsymbol{c}}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} \boldsymbol{W} \begin{pmatrix} \boldsymbol{h_{t-1}} \\ \boldsymbol{x_t} \end{pmatrix}$$

$$\boldsymbol{c_t} = \boldsymbol{f} \times \boldsymbol{c_{t-1}} + \boldsymbol{i} \times \tilde{\boldsymbol{c}}_t$$

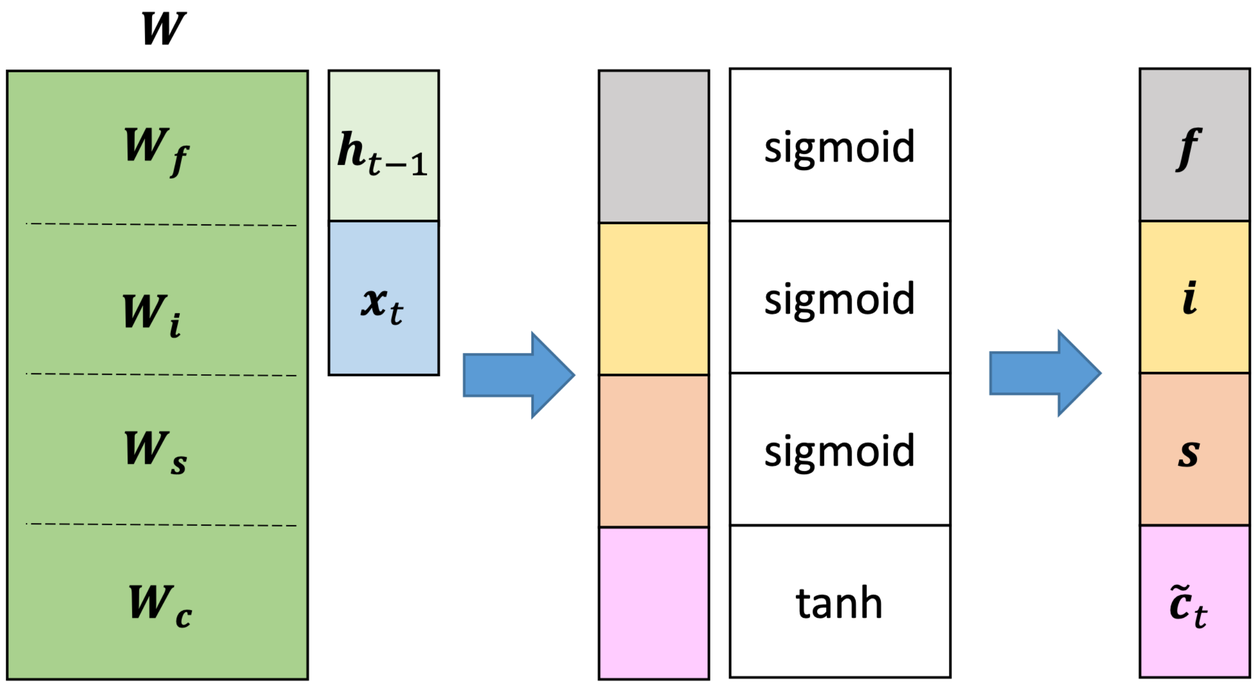
$$\boldsymbol{h_t} = \boldsymbol{s} \times tanh(\boldsymbol{c_t})$$

Long Short Term Memory (LSTM)

Forget gate (f): how much to erase from the past cell state.

Ignore gate (i): how much to ignore from the (candidate) new cell state.

Select gate (s): how much of the new cell state to reveal as output.

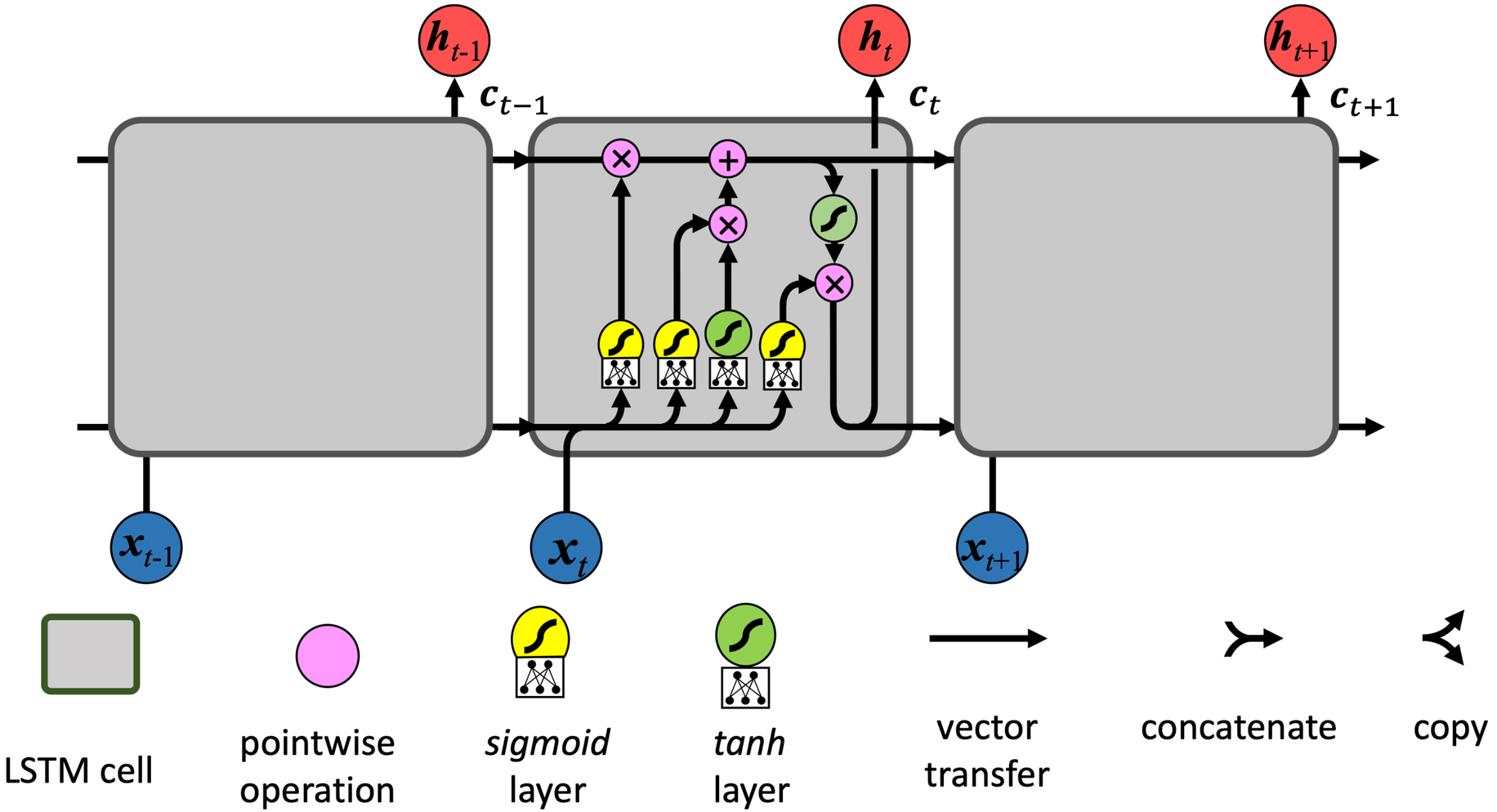


$$\begin{pmatrix} f \\ i \\ s \\ \tilde{c}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \times c_{t-1} + i \times \tilde{c}_t$$

$$h_t = s \times tanh(c_t)$$

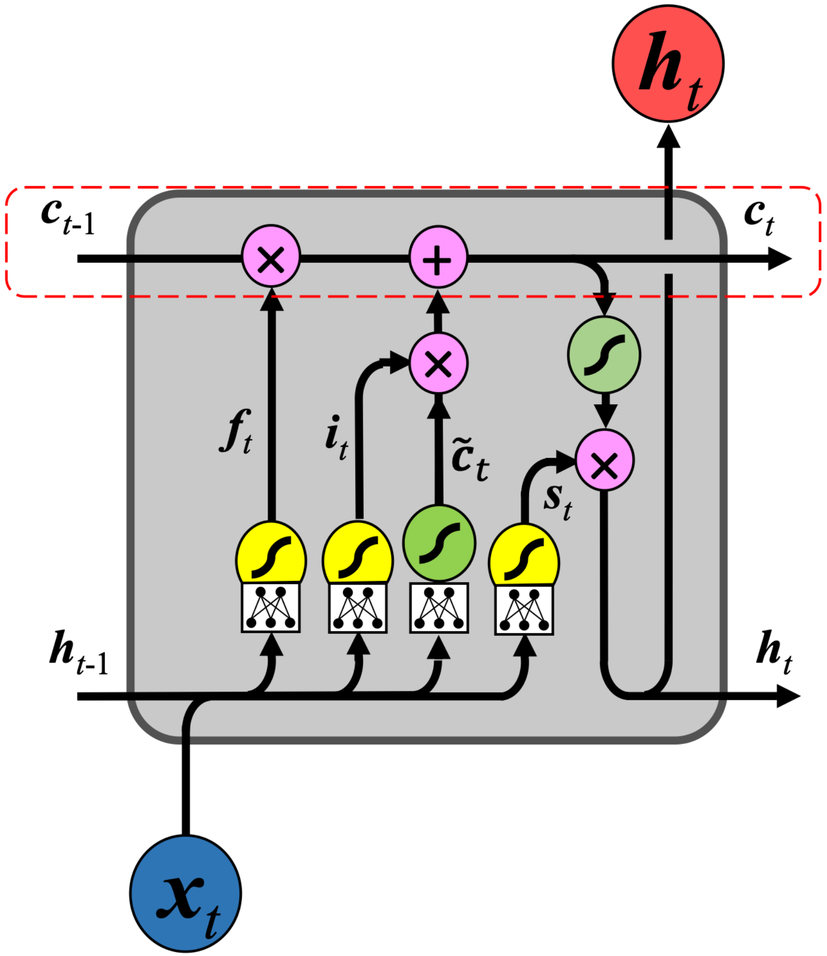
LSTM interacting cells



Core idea behind LSTMs

The cell state ($\boldsymbol{c}_t$):

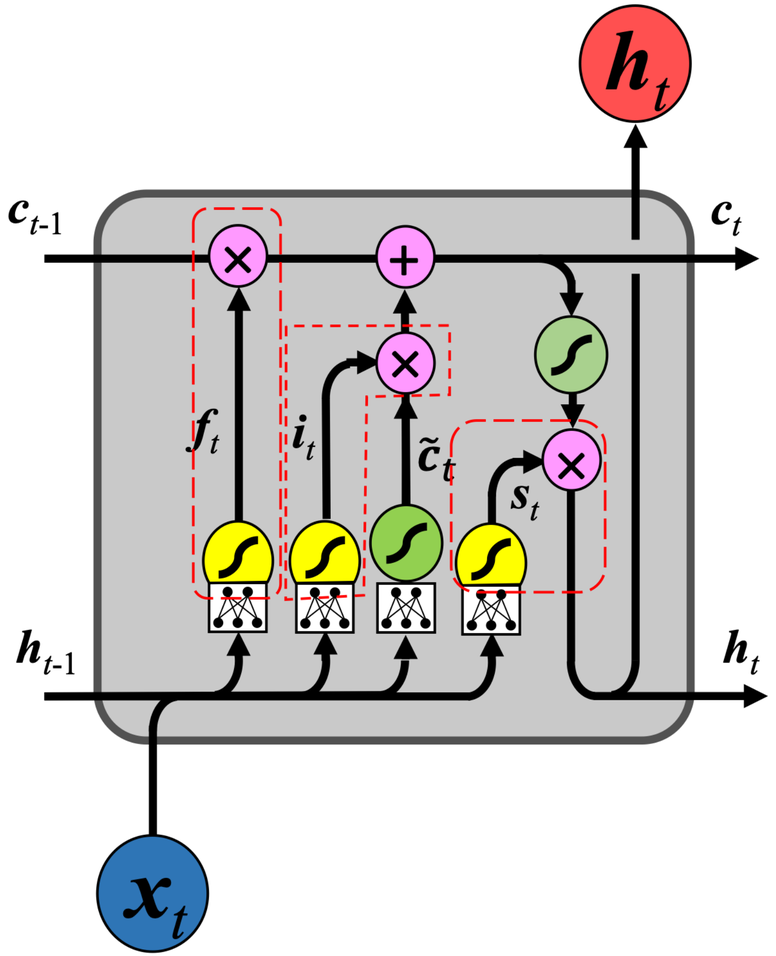
- Retains the relevant information from the past network operation.
- Kind of a conveyor belt.
- Information can flow along it unchanged.
- But possible to remove information from it (by multiplying by zero).
- Also possible to add information to it.
- No vanishing gradient along the cell state!



The core idea behind LSTMs

Gates:

- LSTMs have three gates, to protect and control the cell state.
- Each gate: a sigmoid activated neural net layer and a point-wise multiplication operation.
- Tell how much of each component should be let through (sigmoids):
  - zero $\rightarrow$ blocks everything
  - one $\rightarrow$ lets everything pass



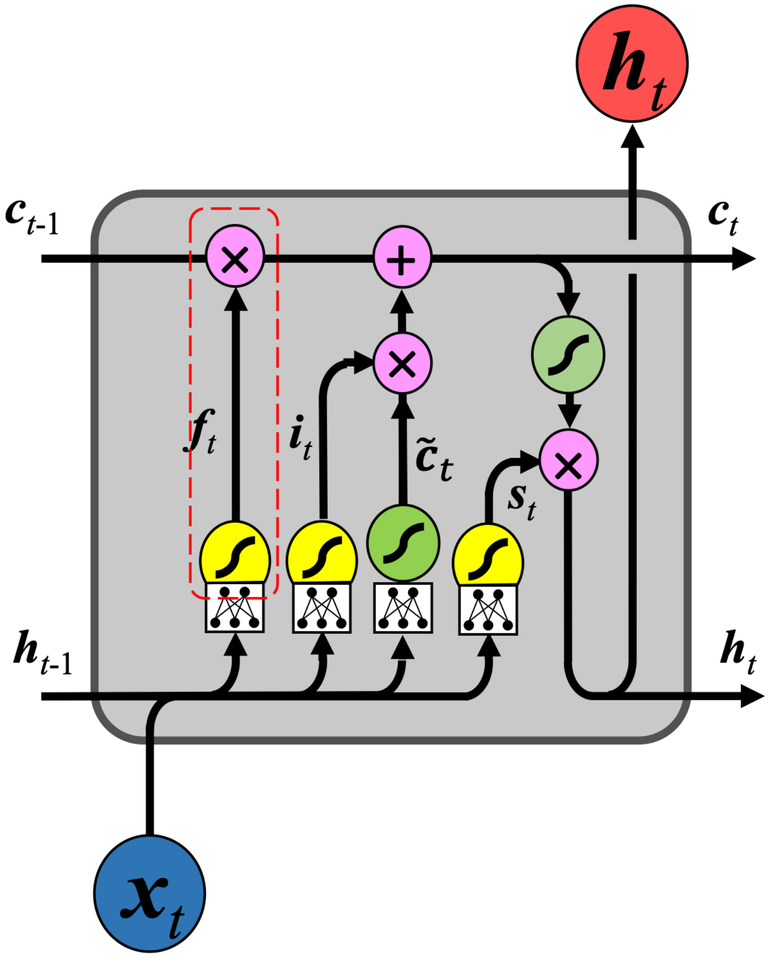
Walk through a LSTM

Forget gate:

- It decides when and which information in the cell state $\boldsymbol{c}_{t\text{-}1}$ must be kept ($\boldsymbol{f}_t = 1$) or dropped ($\boldsymbol{f}_t = 0$) based on $\boldsymbol{h}_{t\text{-}1}$ and $\boldsymbol{x}_t$:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_f\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right]\right)$$

- This is done for each element of $\boldsymbol{c}_{t\text{-}1}$

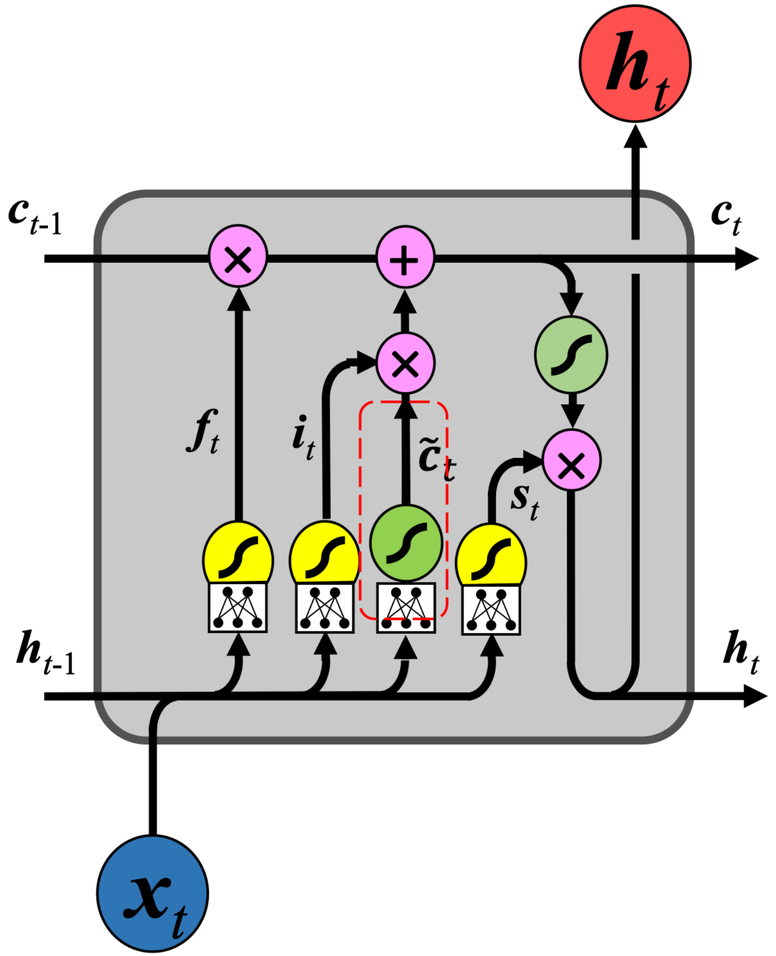


Walk through a LSTM

Candidate values:

- Based on $\boldsymbol{h}_{t\text{-}1}$ and $\boldsymbol{x}_t$, the LSTM cell creates a new vector of candidate values $\tilde{\boldsymbol{c}}_t$:

$$\tilde{\boldsymbol{c}}_t = tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$



Walk through a LSTM

Ignore gate:

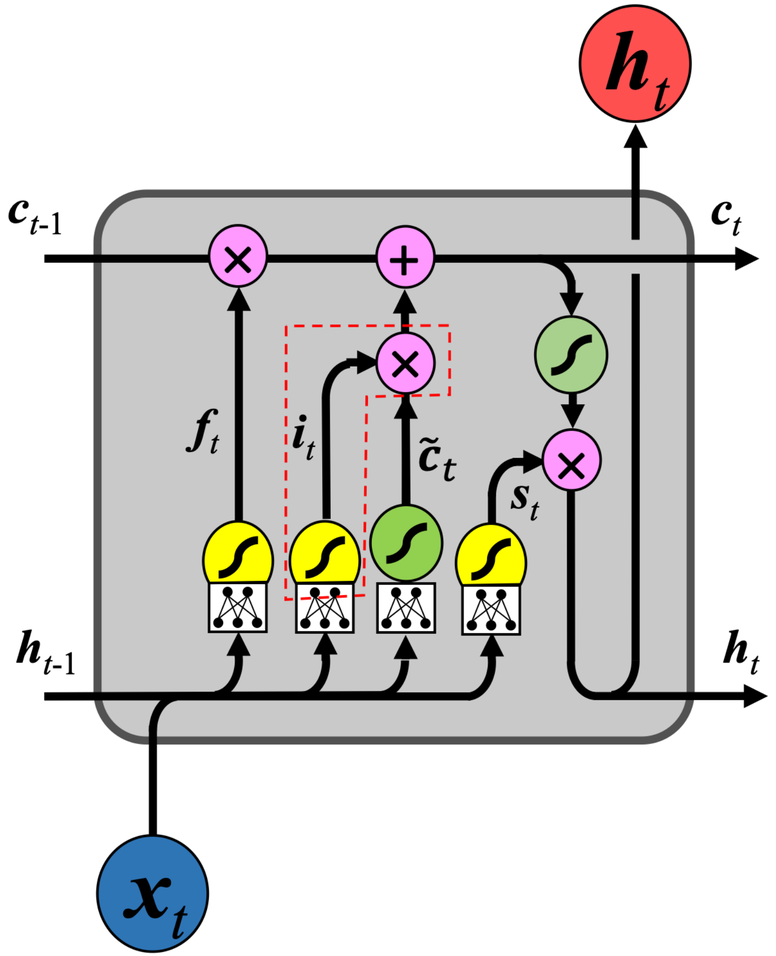
- Again, based on $\boldsymbol{h}_{t\text{-}1}$ and $\boldsymbol{x}_t$, the ignore gate tells which candidate values in $\tilde{\boldsymbol{c}}_t$ shall pass through:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

Cell update:

- These values are then added to what passed through the cell gate so far:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \times \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \times \tilde{\boldsymbol{c}}_t$$



Walk through a LSTM

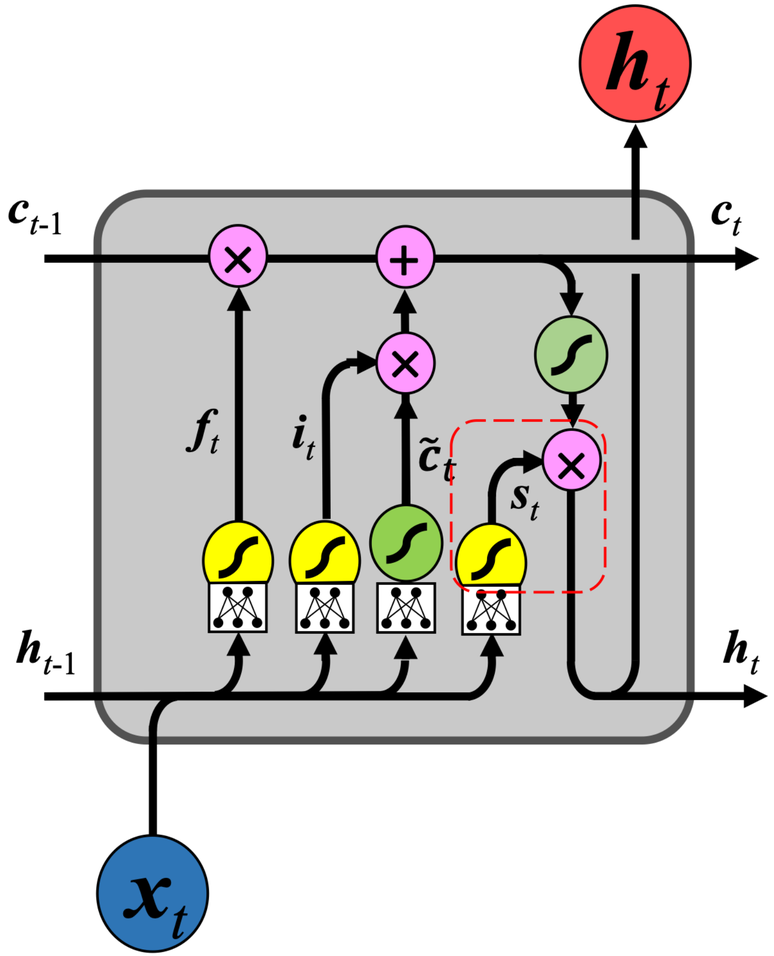
Output update:

- First, a *tanh* squashes the cell values between -1 and 1, which go through the select filter.

Select filter:

- Then, a sigmoid layer decides which parts of the cell state must be preserved:

$$\boldsymbol{s}_t = \sigma(\boldsymbol{W}_s[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

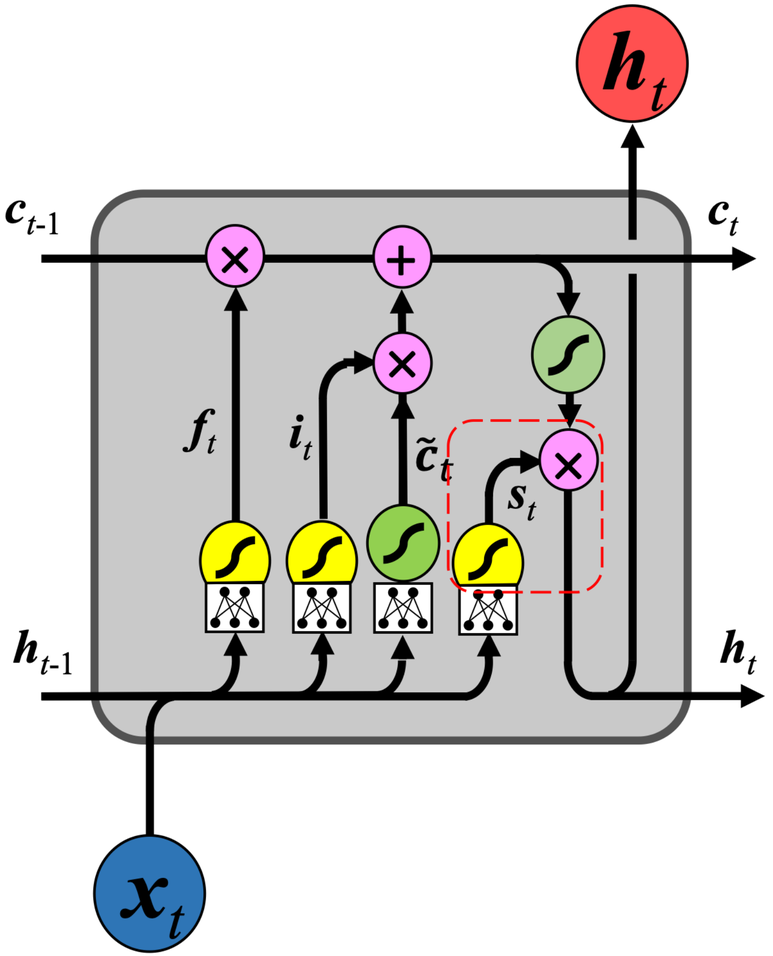


Walk through a LSTM

Output:

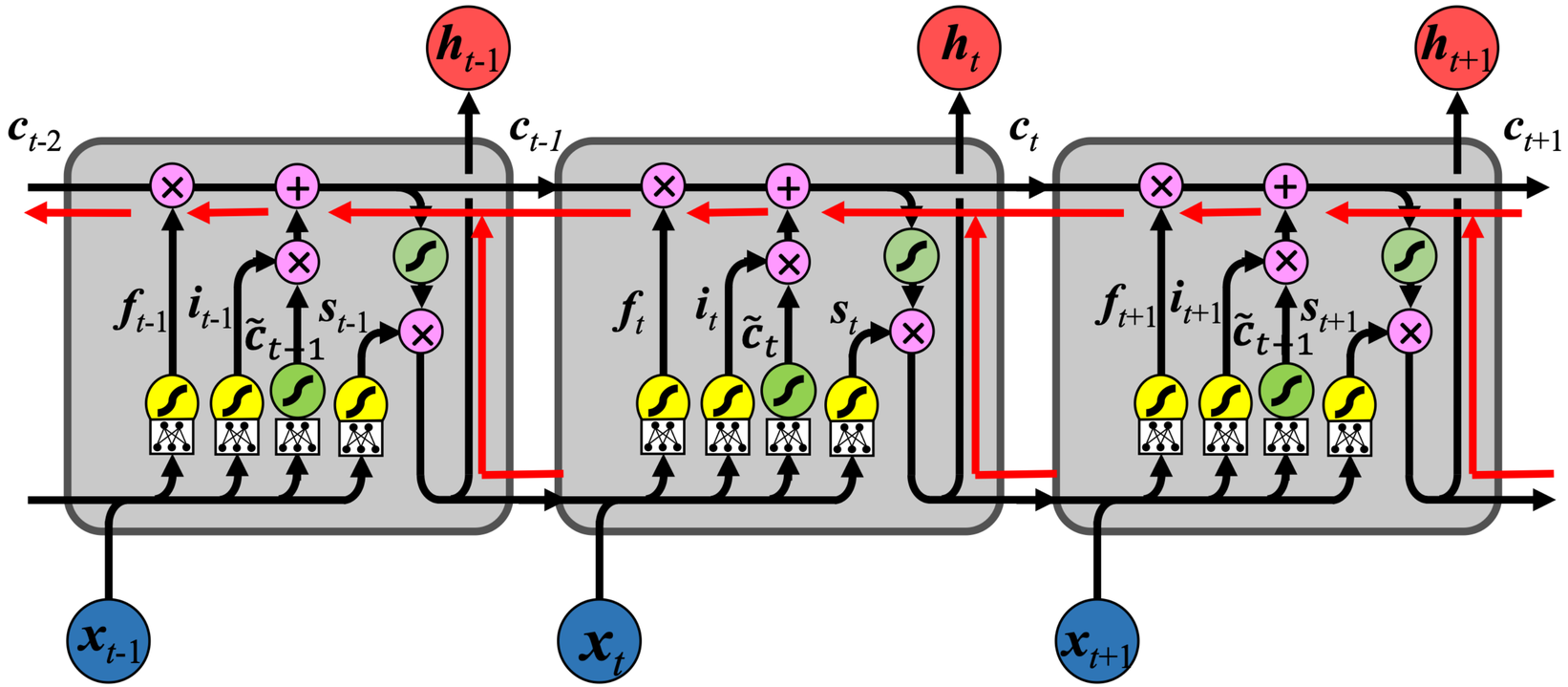
- The output will be a filtered version of the cell state.

$$\boldsymbol{h_t = s_t \times tanh(c_t)}$$



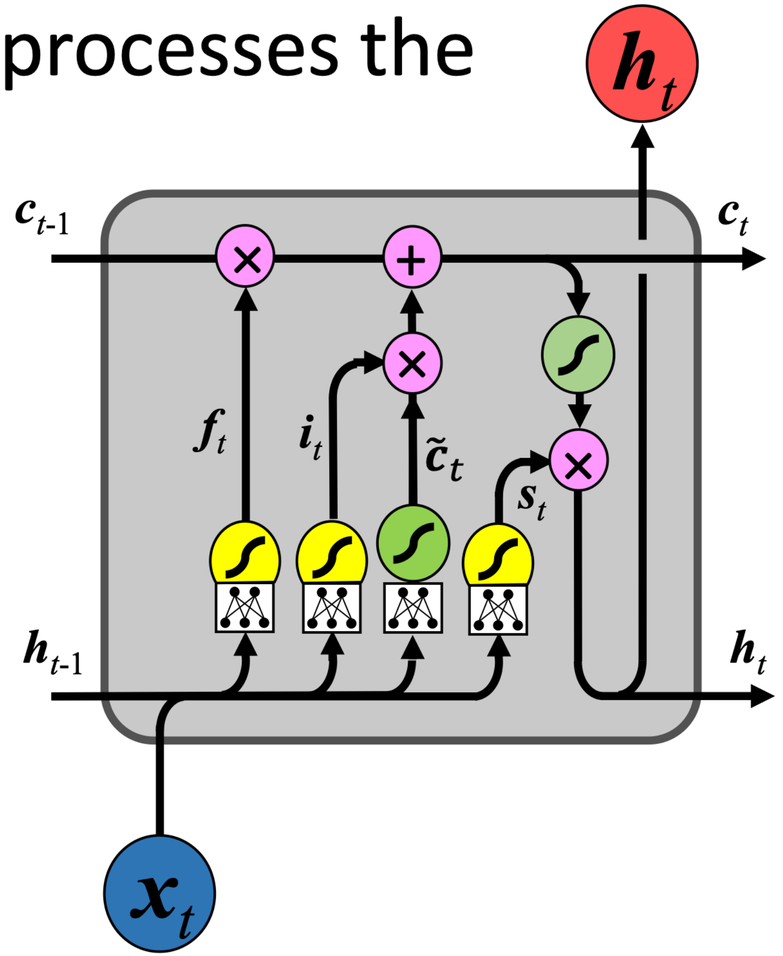
Gradient Flow in LSTM

- Backpropagation from c_{t-1} to c_t: only elementwise multiplication by f_t, no matrix multiplication by W^T.
- Multiplied by a different f_t, at each time step.



Long Short Term Memory (LSTM)

Simple model that processes sequences of integers, embeds each integer into a 64-dimensional vector, then processes the sequence of vectors using a LSTM layer.




```
model = keras.Sequential()
# Add an Embedding layer expecting input vocab of size 1000, and
# output embedding dimension of size 64.
model.add(layers.Embedding(input_dim=1000, output_dim=64))

# Add a LSTM layer with 128 internal units.
model.add(layers.LSTM(128))

# Add a Dense layer with 10 units.
model.add(layers.Dense(10))
```

https://keras.io/guides/working_with_rnns/

Long Short Term Memory (LSTM)

Example: samples produced with LSTM trained with Leo Tolstoy's War and Peace.

- After 100 iterations:

  tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng

- After 700 iterations:

  Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort how, and Gogition is so overelical and ofter.

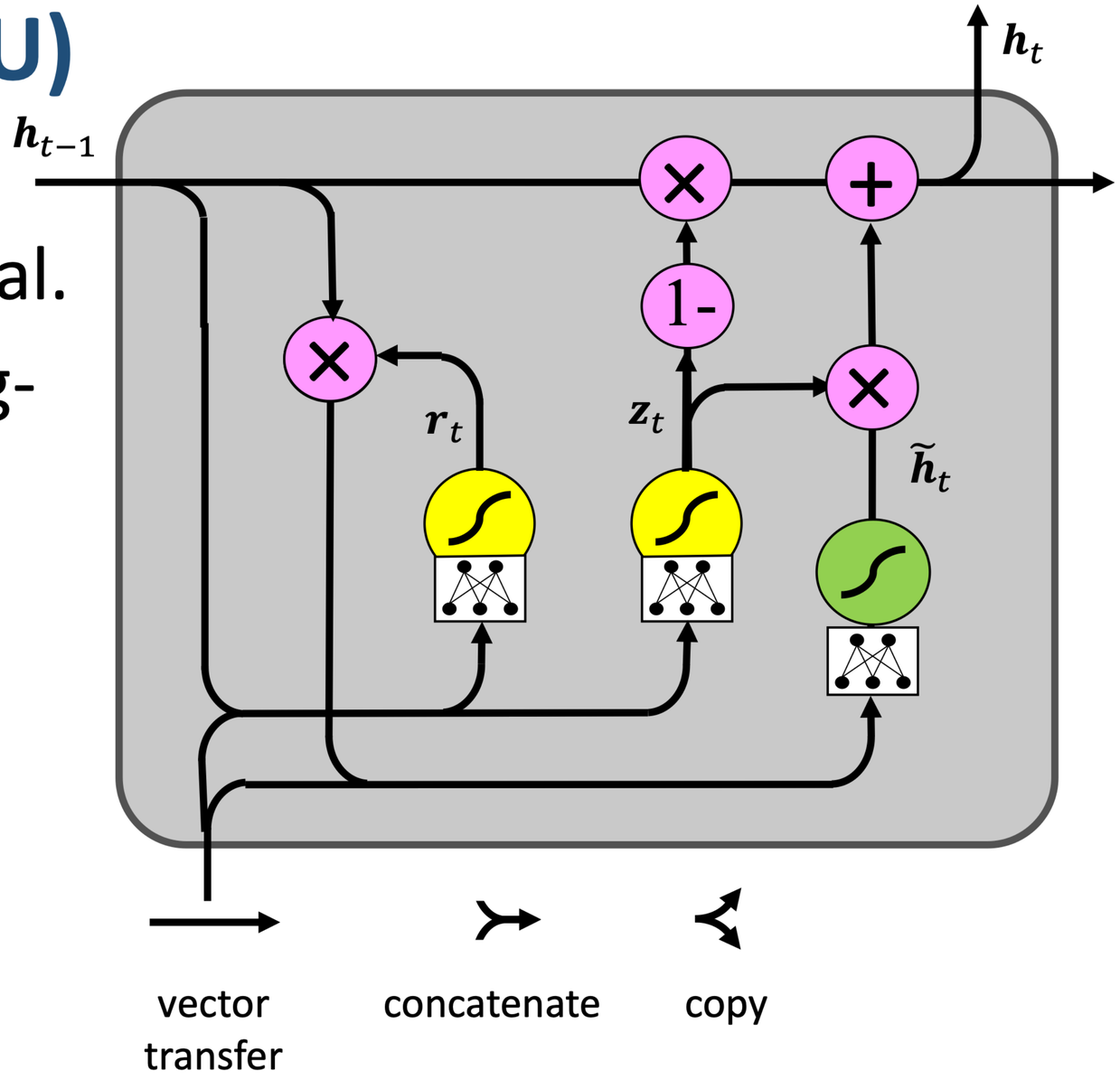
- After 2000 iterations:

  "Why do what that day," replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him. Pierre aking his soul came to the packs and drove up his father-in-law women

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Gated Recurrent Unit (GRU)

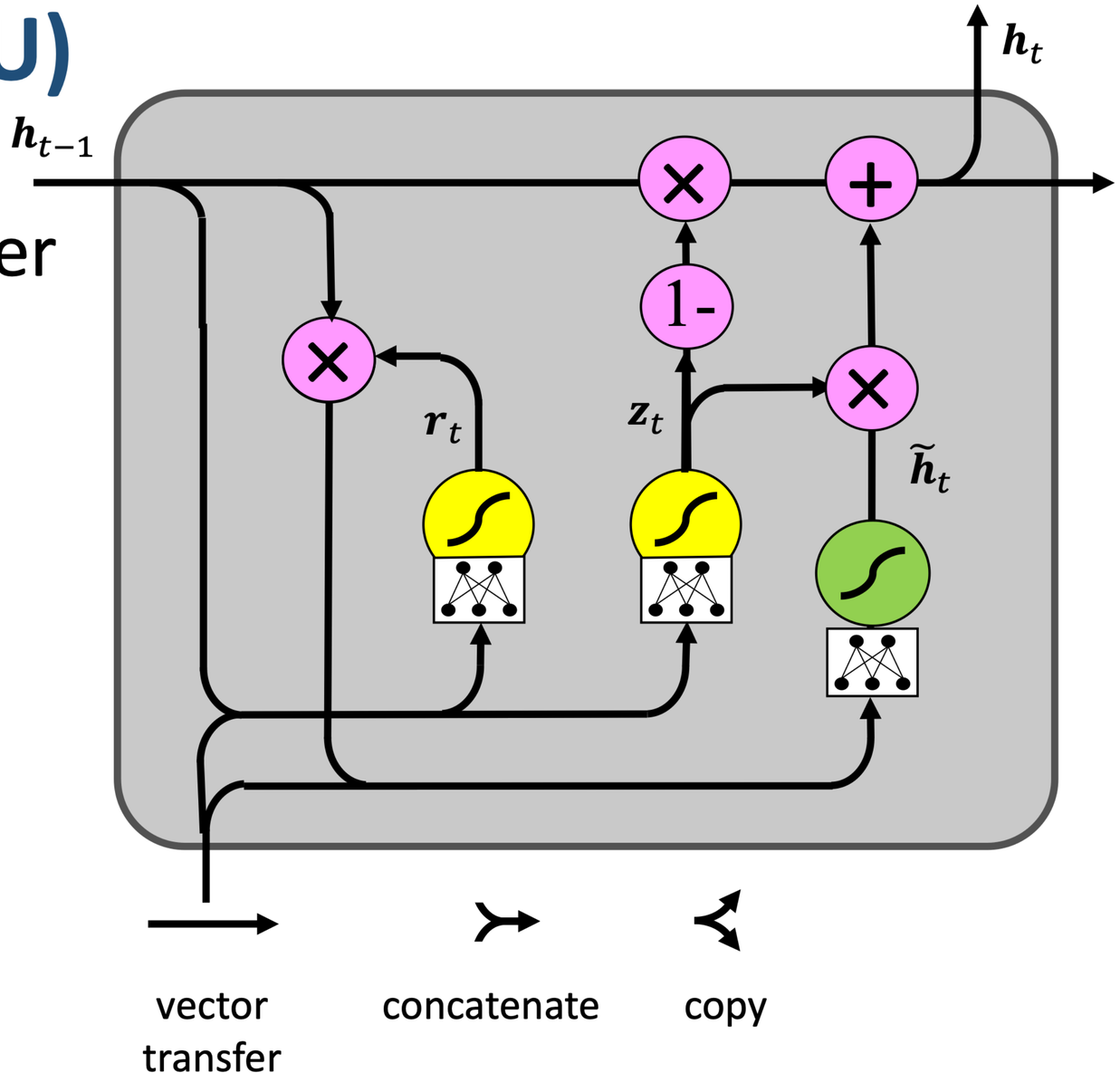
- A simpler variation of LSTM.
- Introduced in 2014 by Cho, et al.
- Also designed to avoid the long-term dependency problem.
- A little faster than LSTM.



Cho et al., 2014 available at : https://arxiv.org/pdf/1406.1078v3.pdf

Gated Recurrent Unit (GRU)

- Got rid of the cell state and used the hidden state to transfer information.
- Two gates: **reset gate** ($\boldsymbol{r}$) and **update gate** ($\boldsymbol{z}$).

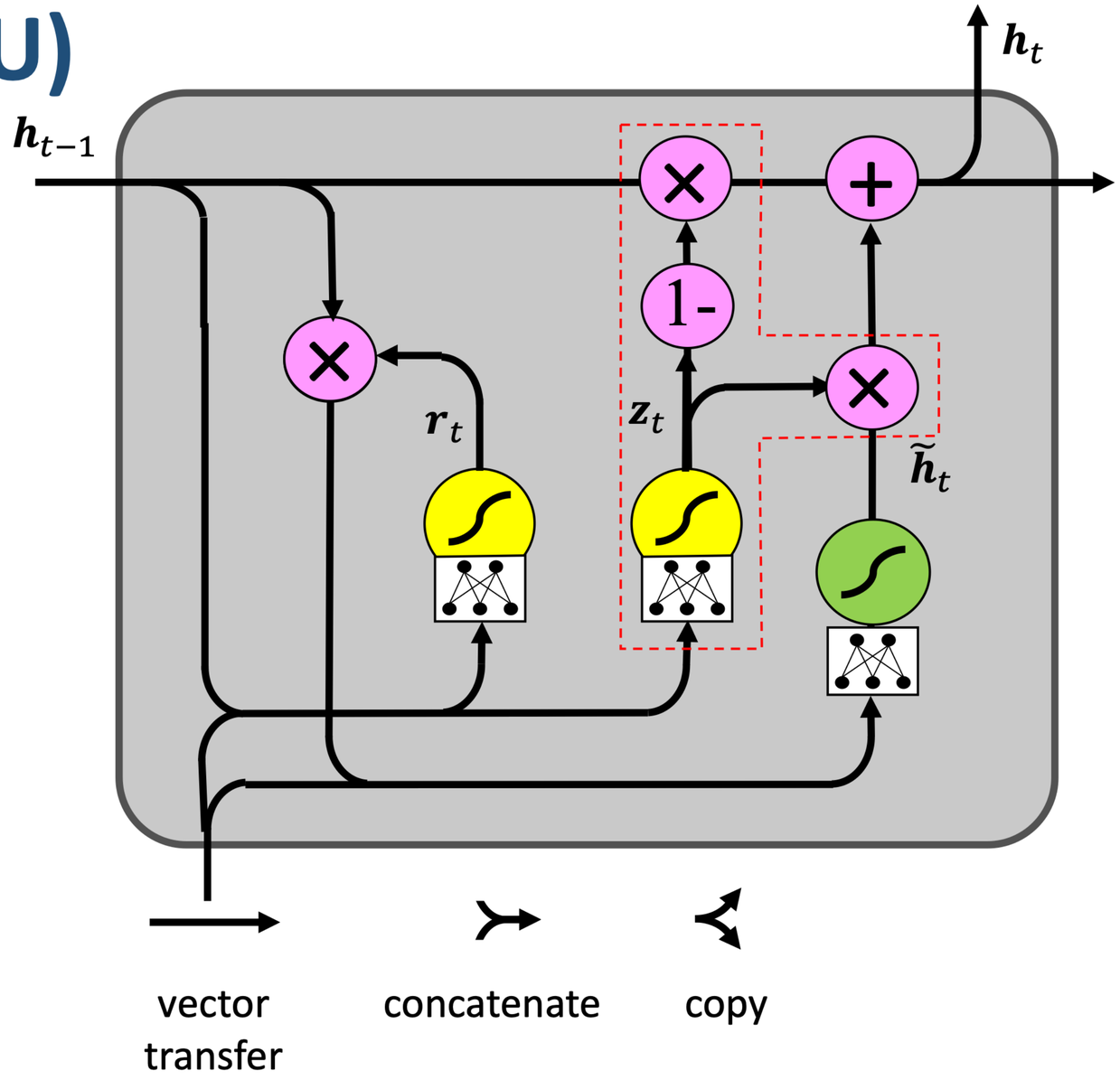


https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Gated Recurrent Unit (GRU)

Update Gate:

- Acts similar to the forget and input gate of an LSTM.
- Decides what information to throw away and what new information to add.

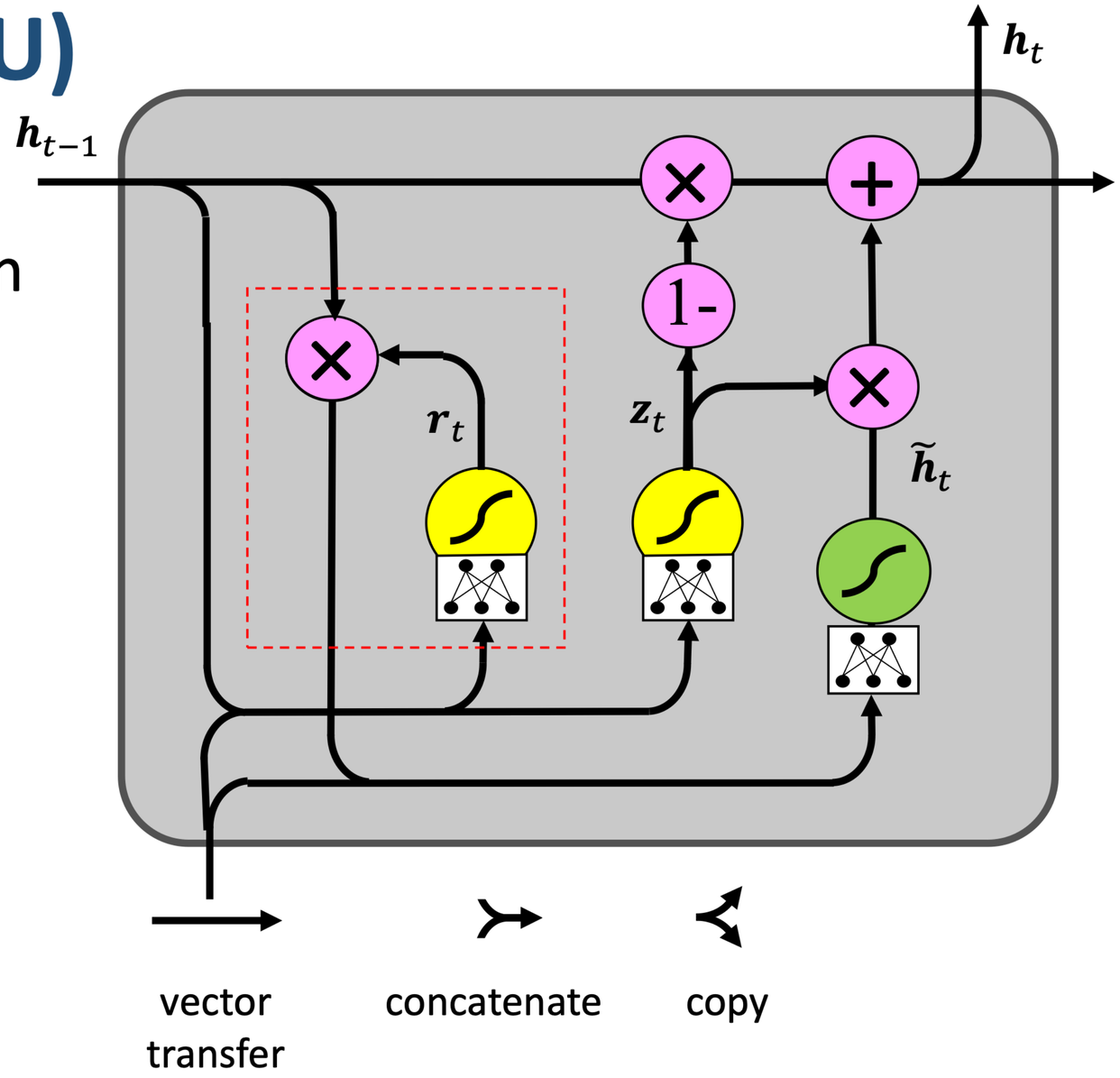


https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Gated Recurrent Unit (GRU)

Reset Gate:

- Decides how much past information to forget.



https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

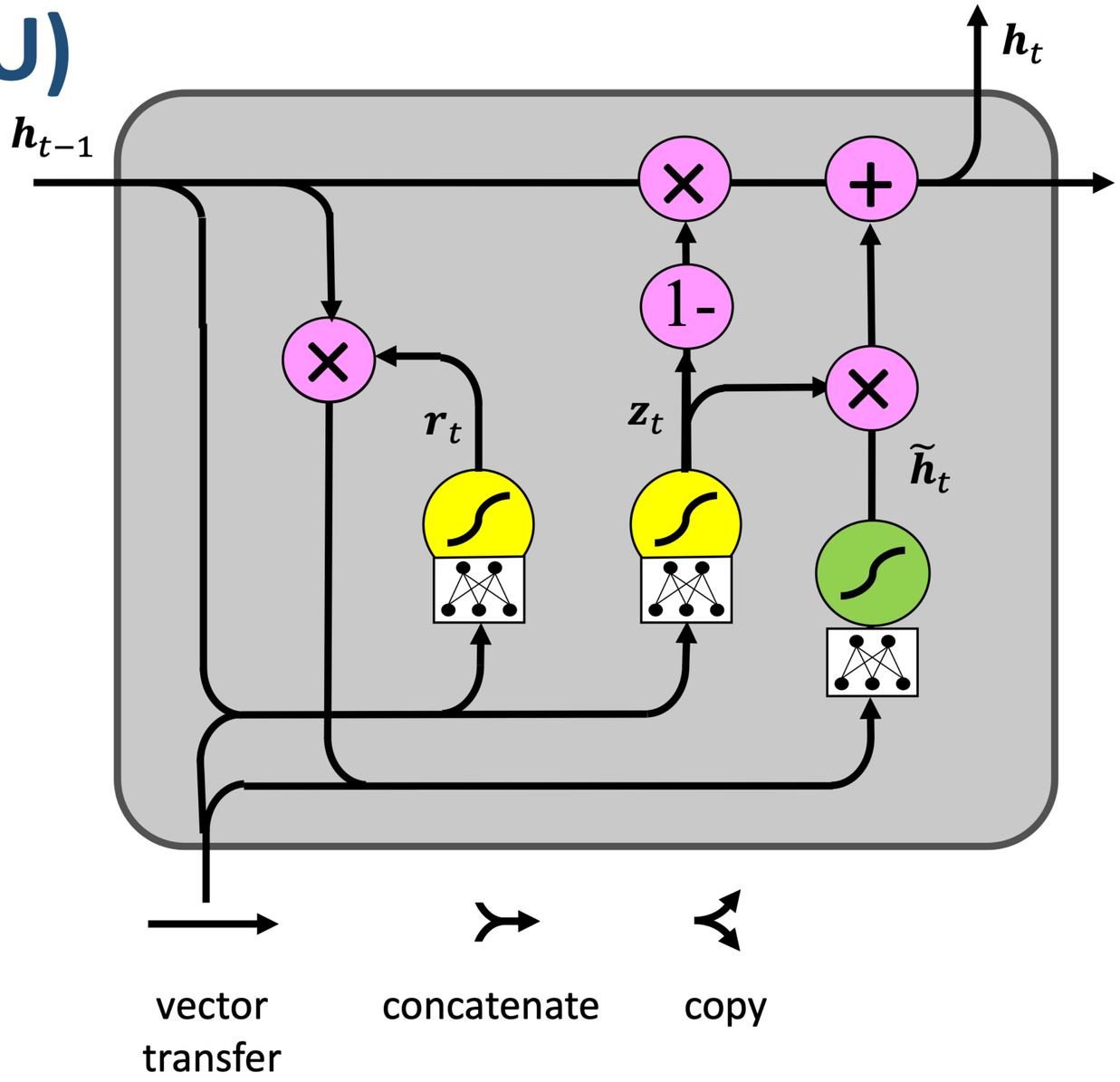
Gated Recurrent Unit (GRU)

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z \boldsymbol{z}_t + \boldsymbol{W}_z \boldsymbol{h}_{t-1})$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{W}_r \boldsymbol{h}_{t-1})$$

$$\tilde{\boldsymbol{h}}_t = tanh(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{r}_t * \boldsymbol{W}_h \boldsymbol{h}_{t-1})$$

$$\boldsymbol{h}_t = \boldsymbol{z}_t * \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t)\tilde{\boldsymbol{h}}_t$$



Cho et al., 2014 available at : https://arxiv.org/pdf/1406.1078v3.pdf

Other RNN Variants

"In this paper, we present the first large-scale analysis of eight LSTM variants on three representative tasks: speech recognition, handwriting recognition, and polyphonic music modeling …

Our results show that none of the variants can improve upon the standard LSTM architecture significantly."

Greff et al., 2015, LSTM: *A Search Space Odyssey*.

Greff el al., 2015 available at https://ieeexplore.ieee.org/document/7508408

Other RNN Variants

"We have evaluated a variety of recurrent neural network architectures in order to find an architecture that reliably outperforms the LSTM.

Though there were architectures that outperformed the LSTM on some problems, we were unable to find an architecture that consistently beat the LSTM and the GRU in all experimental conditions."

Jozefowicz et al., 2015, *An Empirical Exploration of Recurrent Network Architectures*.

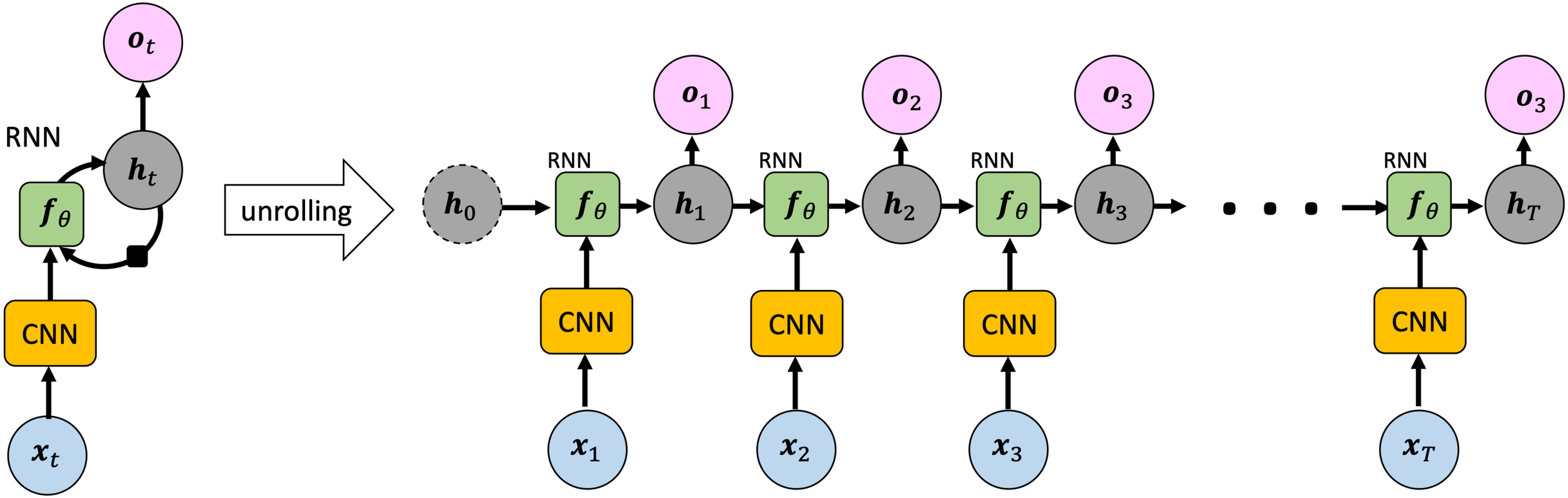
Jozefowicz el al., 2015 available at https://ieeexplore.ieee.org/document/7508408

Recurrent Neural Network

- Introduction
- RNN Architecture
- RNN Training
- RNN Variants
- Extensions

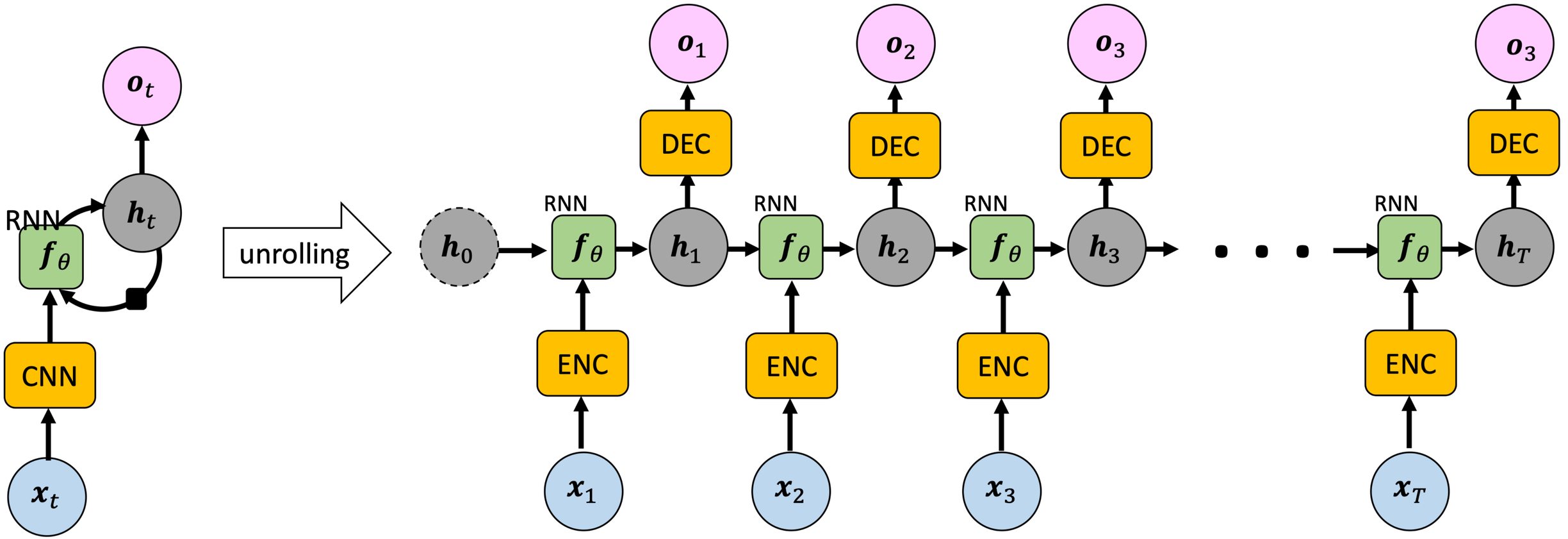
Combining CNN and RNN

A CNN learns data representation to the RNN input.

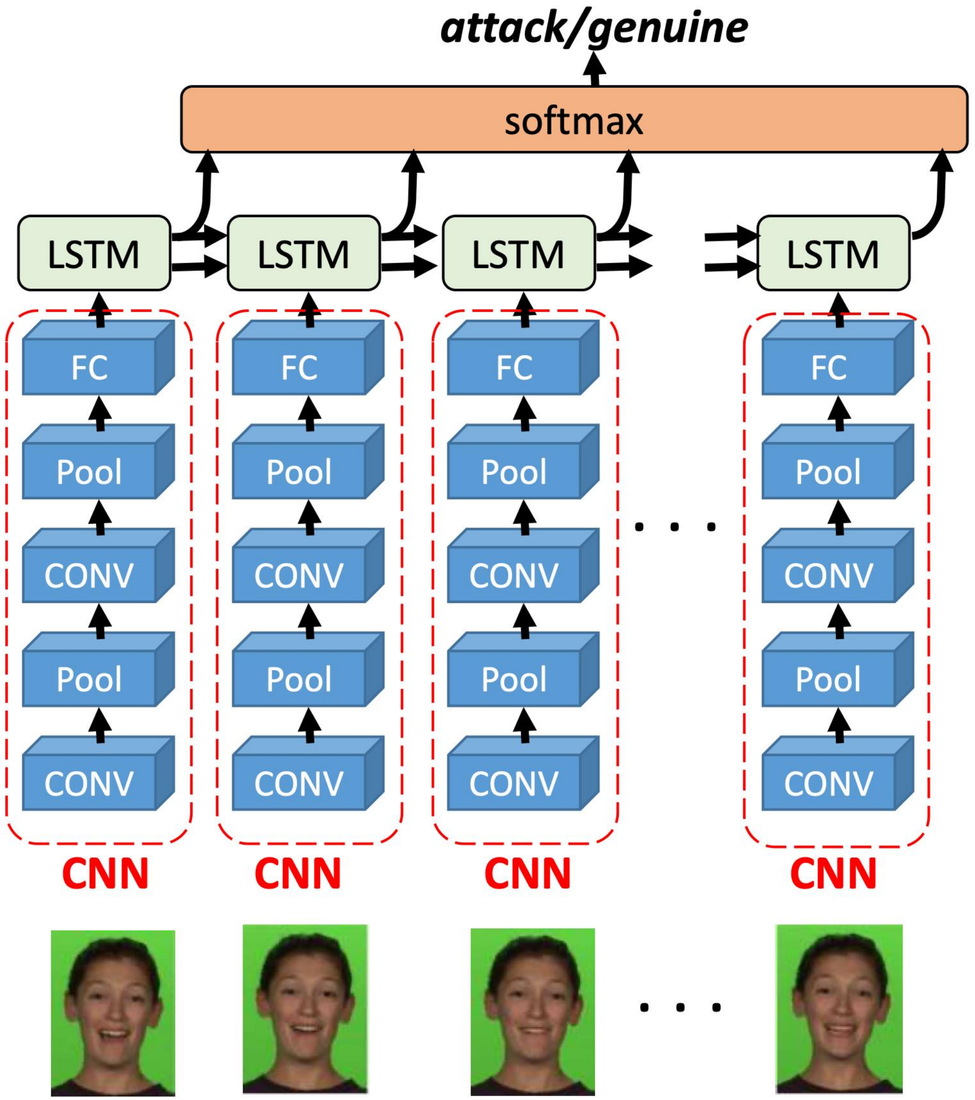


Combining CNN and RNN

- A first CNN (encoder) learns data representation to the RNN input.
- A second CNN (decoder) provides the RNN output.



Combining CNN and RNN



Face anti-spoofing from video:

- A CNN learns a representation and provides the input to a LSTM.
- LSTM learns temporal structure.
- The softmax decides if it is an ***attack*** or ***genuine***.

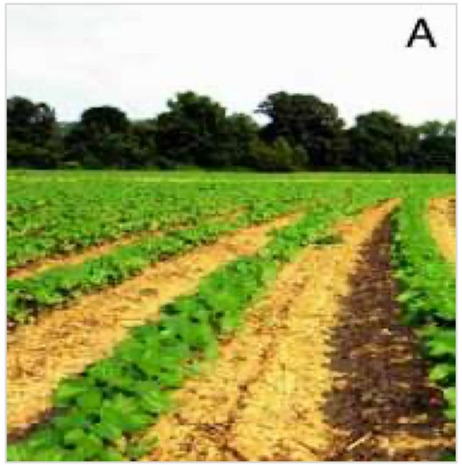
Xu et al., 2015: https://ieeexplore.ieee.org/document/7486482

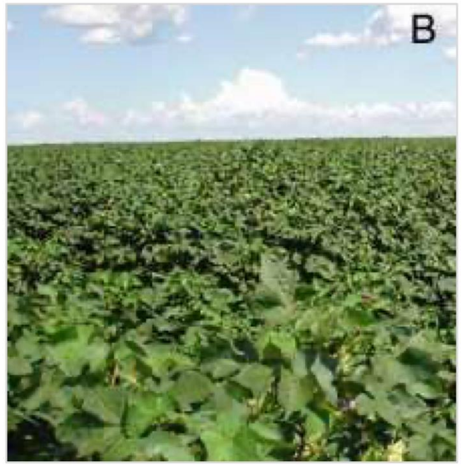
Bidirectional RNN (BRNN)

A positive or negative remark?

“You can always count on the Americans to do the right thing,

What kind of crop?



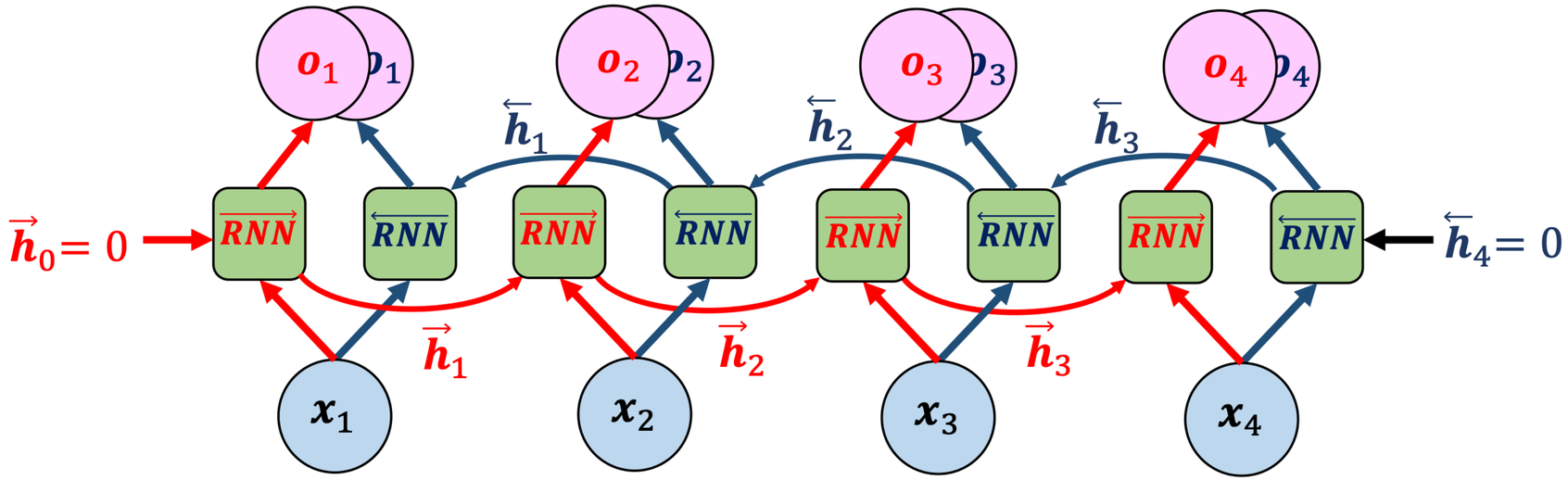


Bidirectional RNN (BRNN)

- Standard RNNs take only the “past” into account.
- In some applications the full sequence is available:
  - Non real-time language translation
  - Off line video analysis
  - Remote sensing images from different epochs
  - …
- Why not take advantage of later time steps to infer what happens in earlier ones?

Bidirectional RNN (BRNN)

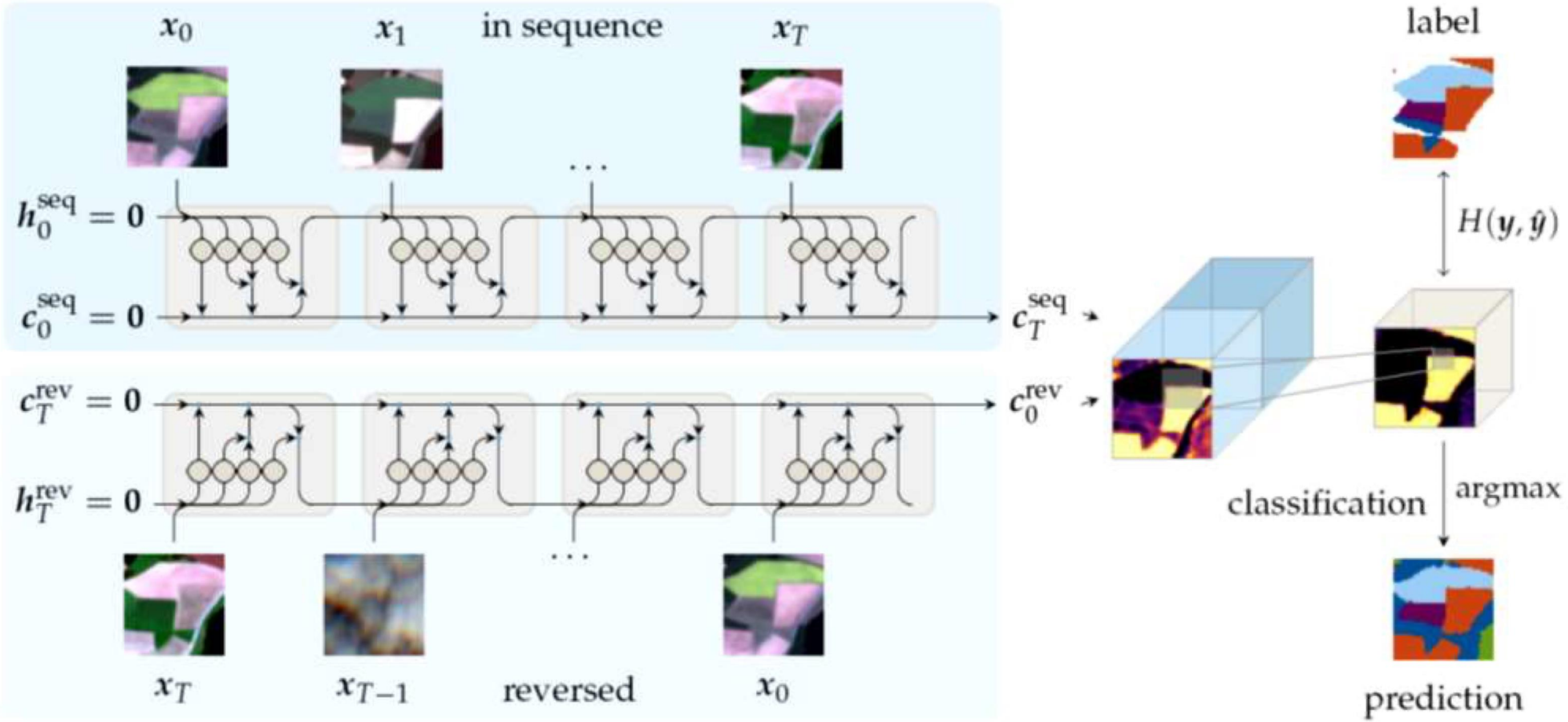
- $\overrightarrow{RNN}$ and $\overleftarrow{RNN}$ share the same parameters.



- The outcome at each time step is the combination/concatenation of the results produced by the forward and the backward passes.
- The RNN units can be LSTM, GRU or any other RNN variant.

Bidirectional RNN (BRNN)

Example: crop recognition



Rußwurm & Körner., 2018: https://www.mdpi.com/2220-9964/7/4/129

Online Lectures/Tutorials

- A Beginner's Guide to LSTMs and Recurrent Neural Networks
- Understanding LSTM Networks
- The Unreasonable Effectiveness of Recurrent Neural Networks – Andrej Karpathy
- Illustrated Guide to Recurrent Neural Networks
- Illustrated Guide to LSTM’s and GRU’s: A step by step explanation
- Recurrent Neural Networks, Stanford University, 2017
- A lecture presenting the Convolutional LSTM for image sequence processing, with application to medical and self-driving car images (Keras source code is shown at minute 15)

Next Lecture

Thursday
Lab LSTM

See you next class!