

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

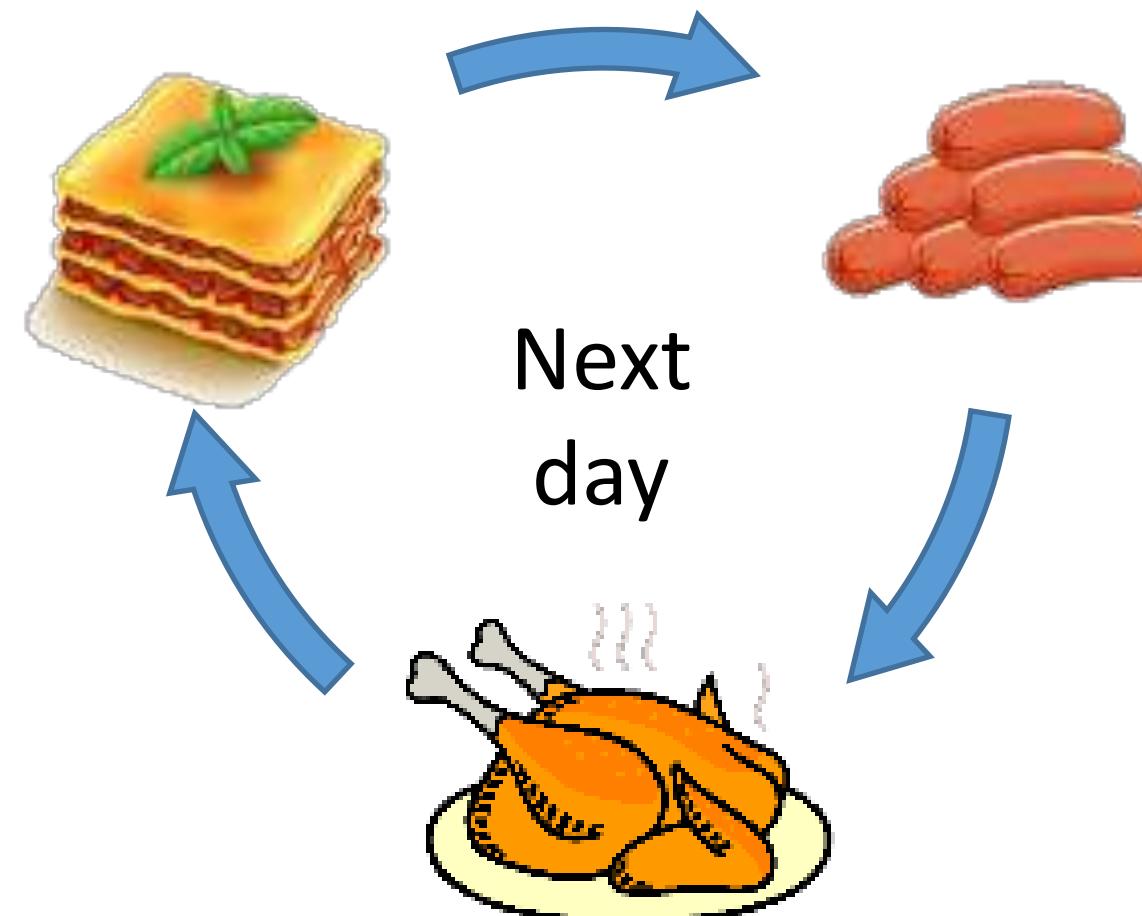
If you know the past...



...you may be able to predict the future.

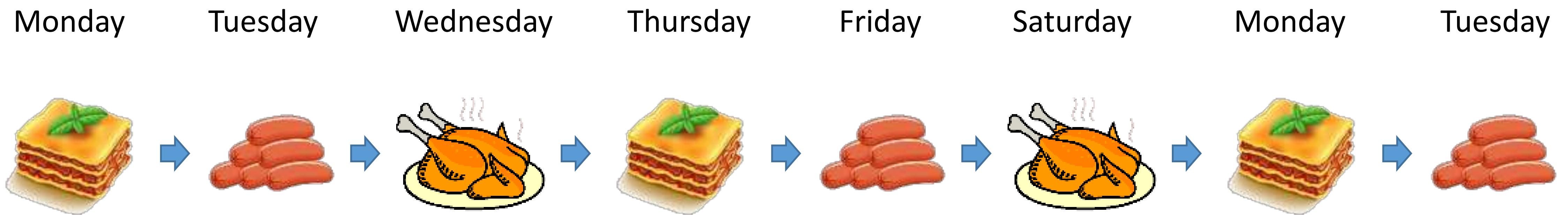
Simple RNN

In the cafeteria...



Simple RNN

In the cafeteria...



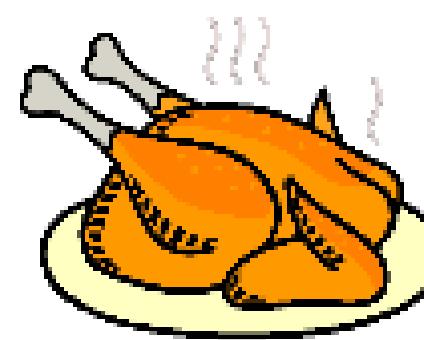
Simple RNN



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



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



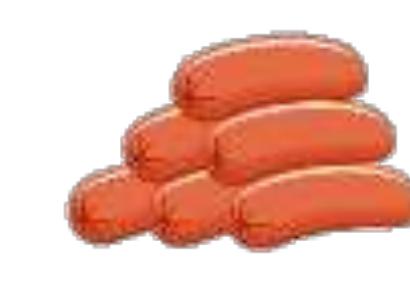
$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Simple RNN


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

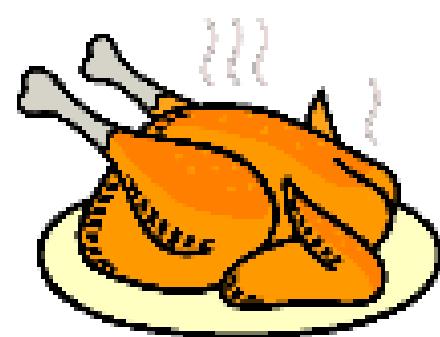


$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \text{Icon of sausages}$$


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



$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \text{Icon of roasted chicken}$$

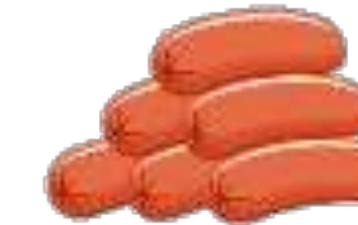

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

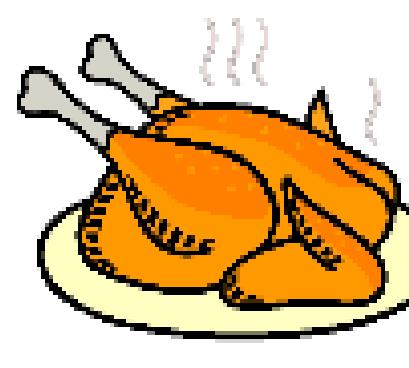


$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \text{Icon of sandwich}$$

Simple RNN


$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \text{Illustration of sausages}$$

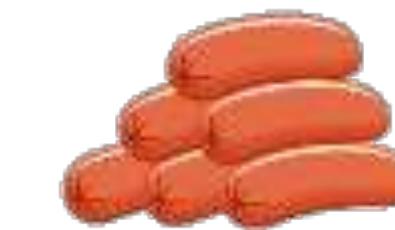

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \text{Illustration of chicken legs}$$


$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{\quad} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \text{Illustration of sandwich}$$

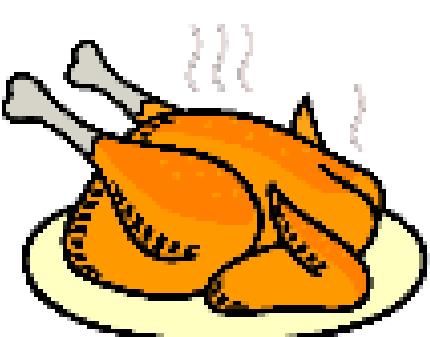
Simple RNN



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

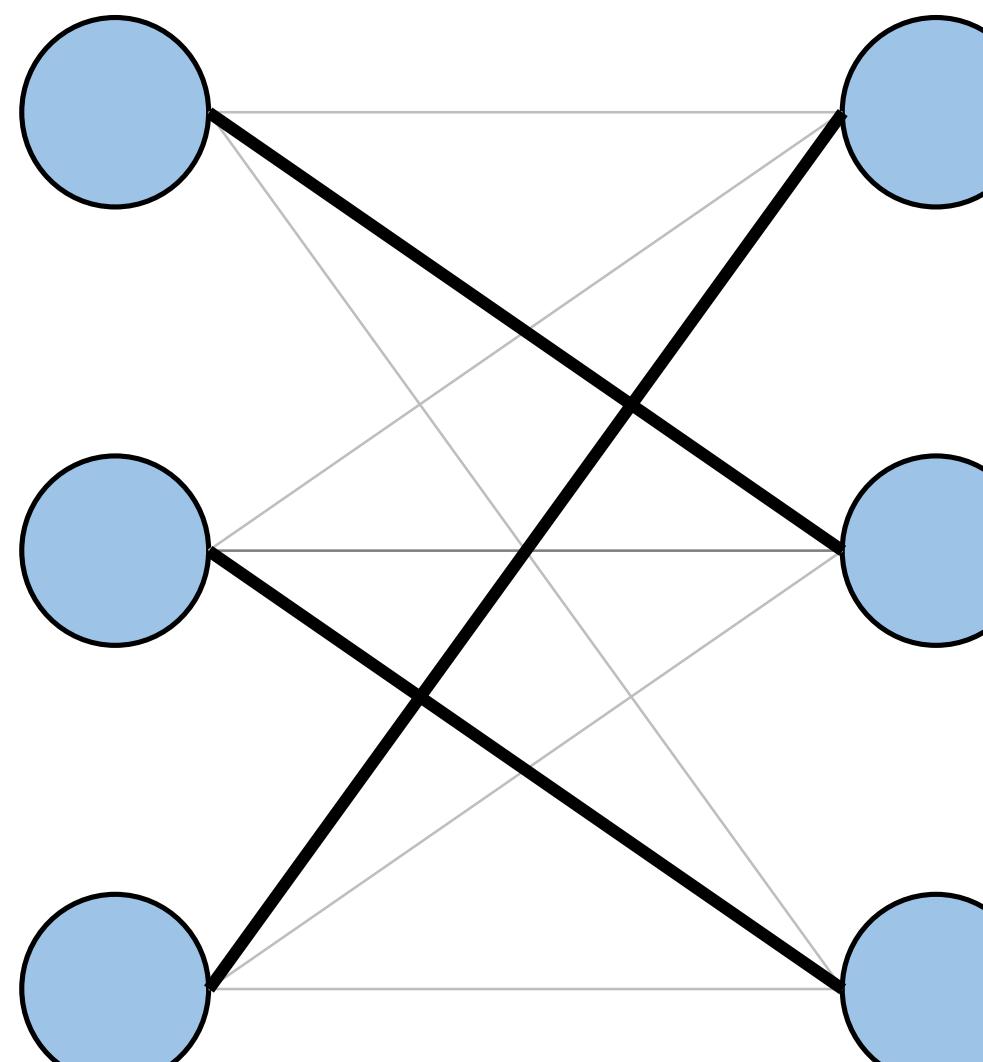


$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



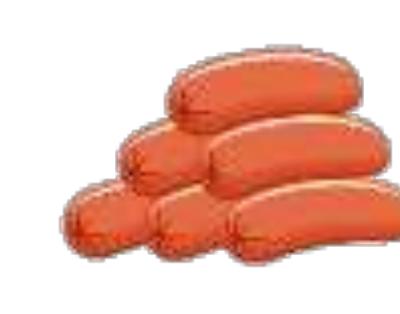
$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

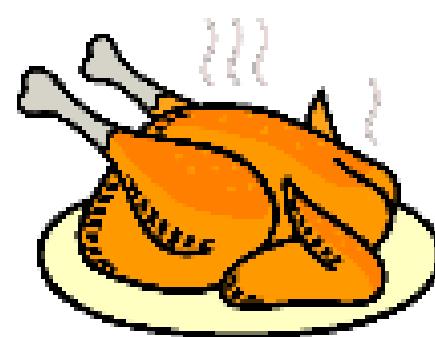
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



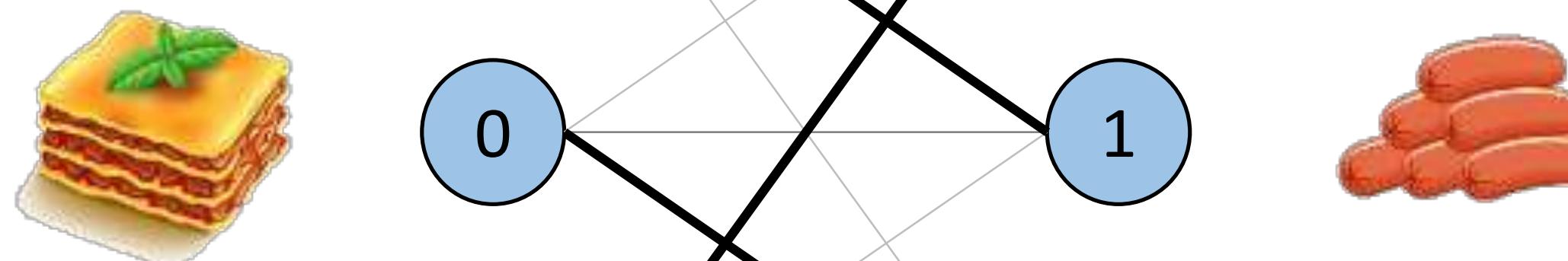
Simple RNN


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


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

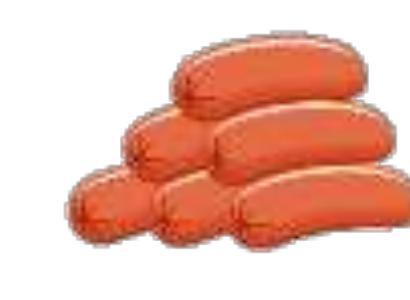

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

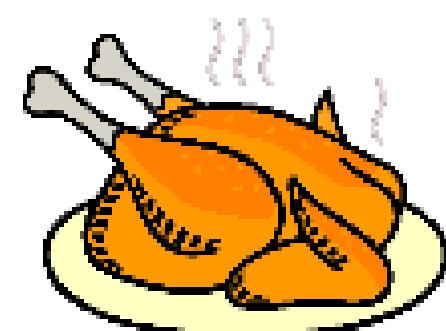
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



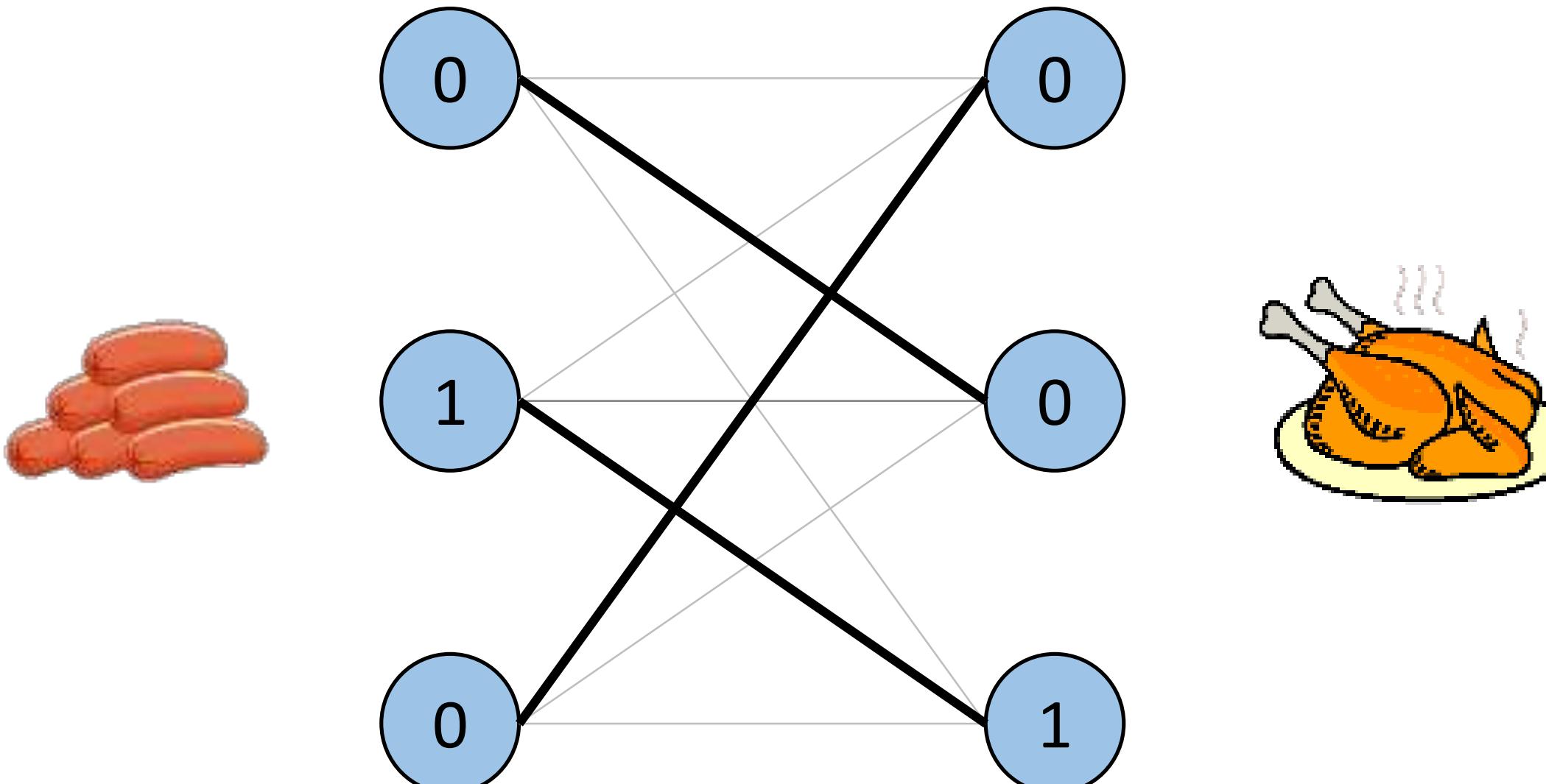
Simple RNN


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


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

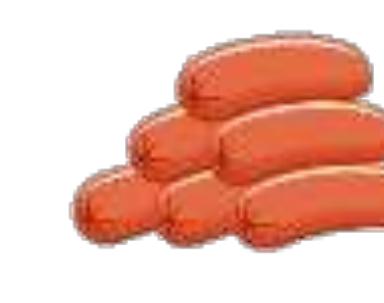

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

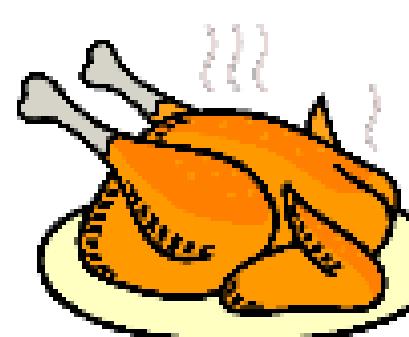
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



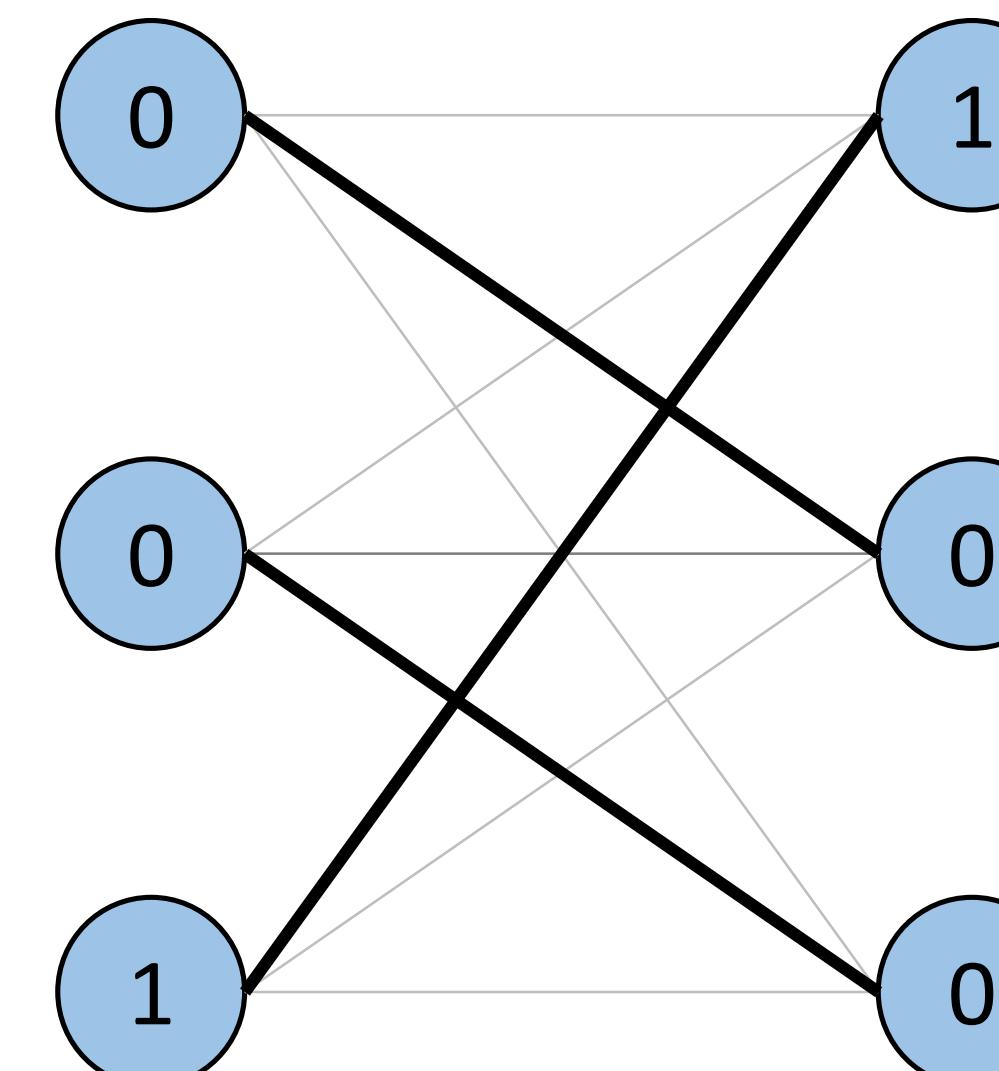
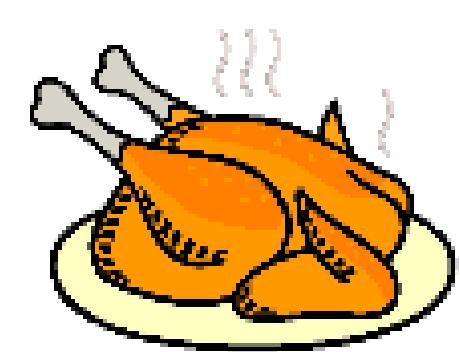
Simple RNN


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


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


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

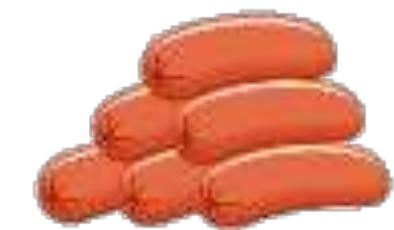
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$



Simple RNN



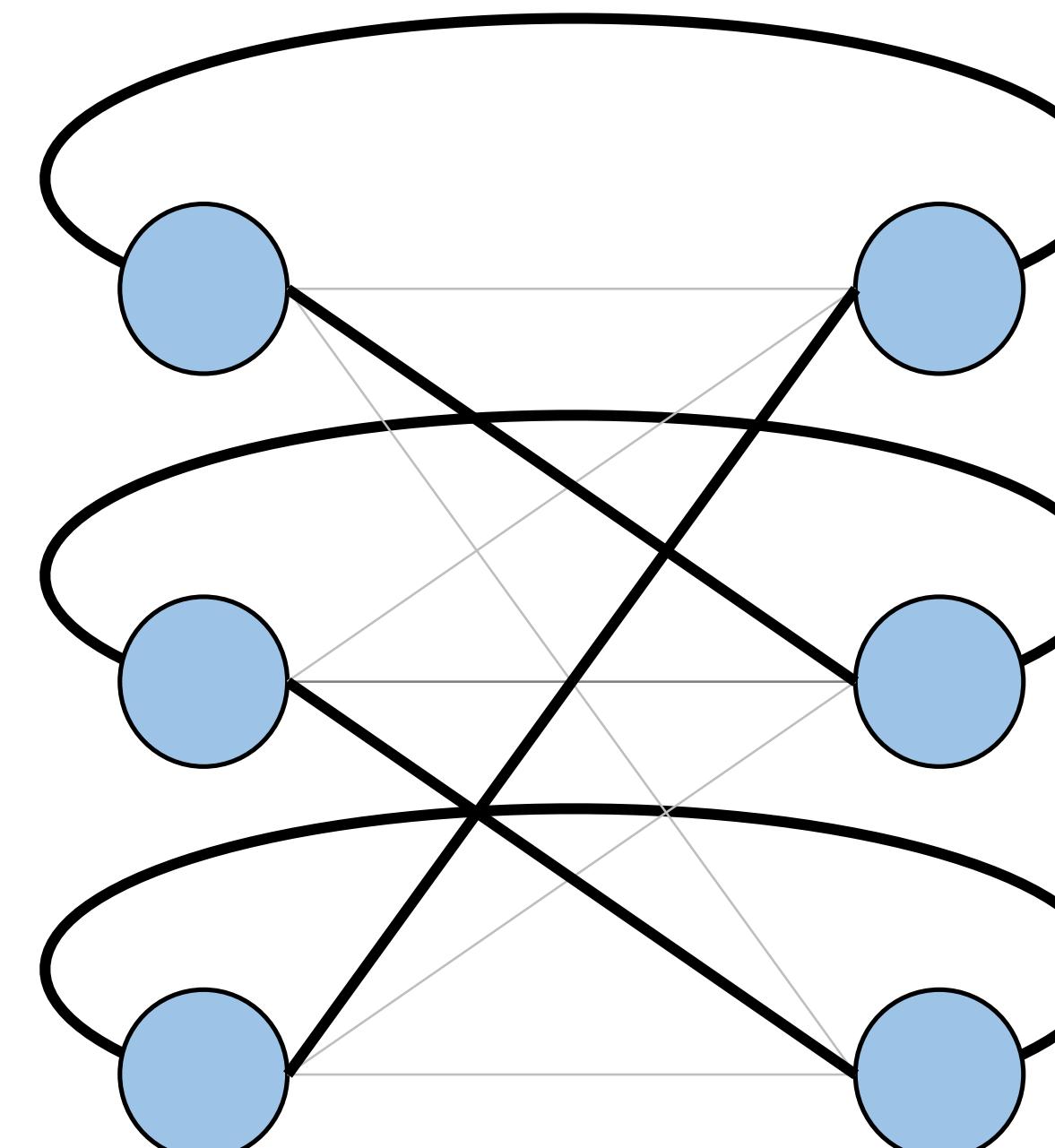
$$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



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

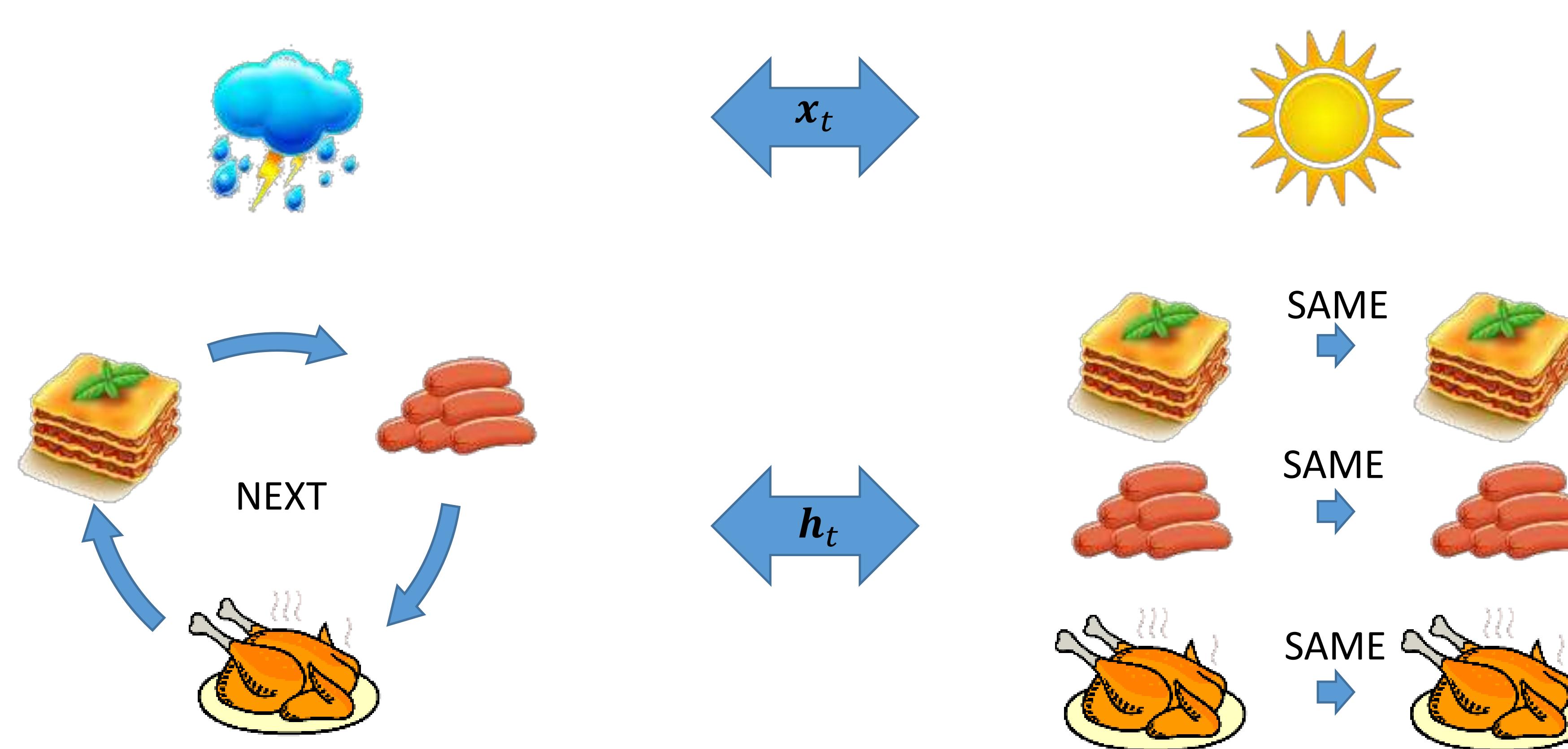


$$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

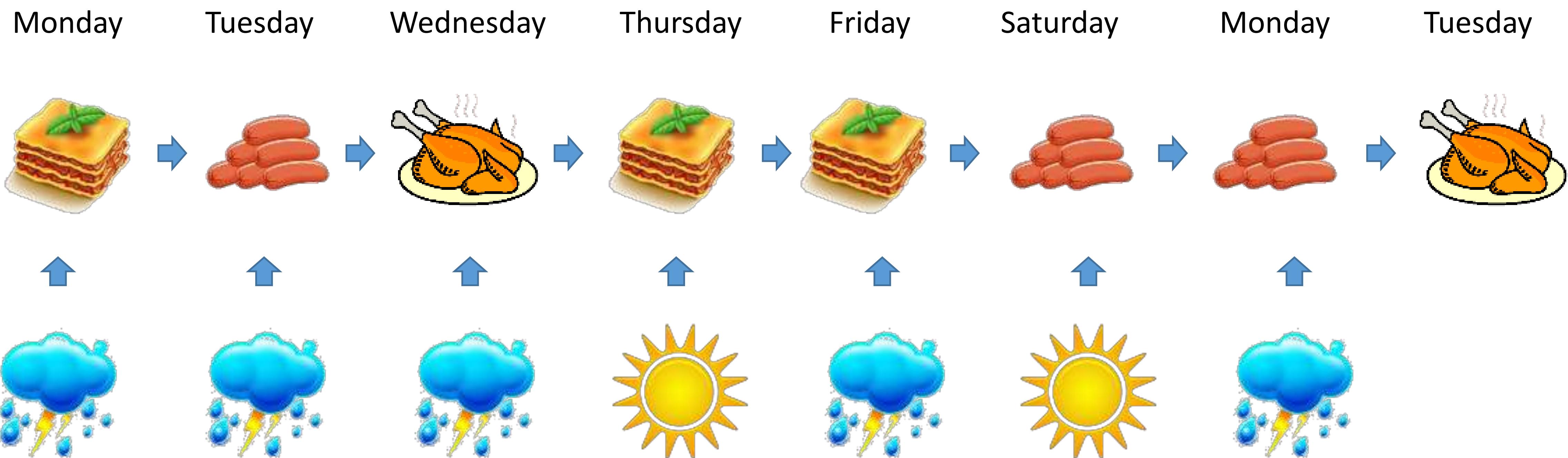


Simple RNN

In the cafeteria...



Simple RNN #2

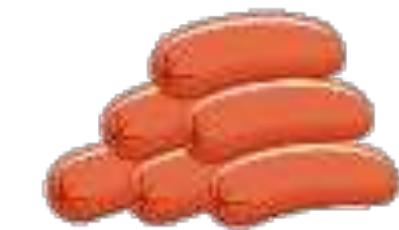


Simple RNN #2

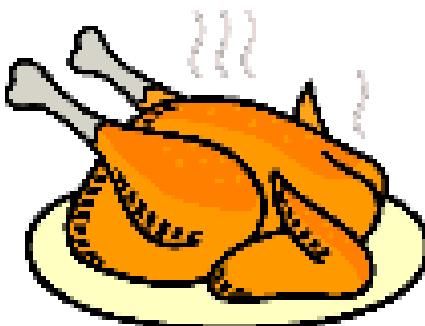
h_t



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



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



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

x_t



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



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

Simple RNN #2

$$W_{hh}$$

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

SAME
NEXT

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}

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

SAME
NEXT

weather matrix

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

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

SAME 
NEXT 

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

SAME 
NEXT 

Simple RNN #2

$$\begin{aligned}
 & \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right] \times \begin{matrix} \text{cake} \\ \hbar_{t-1} \end{matrix} + \left[\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{array} \right] \times \begin{matrix} \text{sun} \\ x_t \end{matrix} = \\
 & \quad \begin{matrix} \text{cake} \\ \text{ham} \end{matrix} \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right] + \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \text{SAME} \\
 & \quad \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right] \text{NEXT} \\
 & = \left[\begin{array}{c} 2 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{array} \right] \xrightarrow{\substack{\text{non-linear} \\ \text{function } (NL)}} \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \xrightarrow{\text{merge}} \left[\begin{array}{c} 1+0 \\ 0+0 \\ 0+0 \end{array} \right] = \left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array} \right]
 \end{aligned}$$

W_{hh}

W_{xh}

$$\begin{aligned}
 & \text{merge} \\
 & \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \times \begin{matrix} \text{cake} \\ W_{ho} \end{matrix} = \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right]
 \end{aligned}$$

Simple RNN #2

$$\begin{aligned}
 & \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right] \times \begin{matrix} \text{cake} \\ \text{x}_t \end{matrix} + \left[\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{array} \right] \times \begin{matrix} \text{cloud} \\ \text{x}_t \end{matrix} = \\
 & \quad \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right] \text{SAME} + \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{array} \right] \text{NEXT} = \\
 & \quad \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \end{array} \right] \xrightarrow{\text{non-linear function (NL)}} \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{array} \right] \xrightarrow{\text{merge}} \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right] = \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \end{array} \right]
 \end{aligned}$$

$$\begin{aligned}
 & \text{merge} \\
 & \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \times \begin{matrix} \text{carrots} \\ \text{W}_{ho} \end{matrix} = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{array} \right]
 \end{aligned}$$

Simple RNN #2

$$\begin{aligned}
 & \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right] \times \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \left[\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{array} \right] \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \\
 & \quad \text{clouds} \quad \text{sun} \quad \text{roast chicken} \quad \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{array} \right] + \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \xrightarrow{\text{SAME}} \left[\begin{array}{c} 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} \right] \\
 & \quad \text{clouds} \quad \text{sun} \quad \text{roast chicken} \quad \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] + \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{array} \right] \xrightarrow{\text{NEXT}} \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \\
 & \quad \text{non-linear} \quad \text{function (NL)} \quad \text{merge} \quad \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] \xrightarrow{\text{merge}} \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] = \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right] = \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right]
 \end{aligned}$$

W_{hh} W_{xh}

$$\left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right]$$

merge W_{ho}

Simple RNN #2

The diagram illustrates the forward pass of an LSTM cell. It starts with the previous hidden state h_{t-1} (represented as a 3x3 matrix) and the current input x_t (represented as a 3x1 vector). These are multiplied by weight matrices W_{hh} and W_{xh} respectively, and then summed. The result is then combined with the previous cell state c_{t-1} (represented as a 3x1 vector) via a forget gate f_t (represented as a 3x1 vector). The resulting hidden state h_t and cell state c_t are then passed through a non-linear function (NL) to produce the final output.

merge

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

w_{ho}



Simple RNN #2

The diagram illustrates the forward pass of an LSTM cell. It starts with the computation of the hidden state h_t and cell state c_t from the previous hidden state h_{t-1} and current input x_t . The computation involves two multiplications: one between W_{hh} and h_{t-1} , and another between W_{xh} and x_t . These results are then added together. The diagram also shows the addition of hidden states and the application of a non-linear function NL . Finally, the cell state c_t is merged with the hidden state h_t to produce the final output.

merge

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

w_{ho}



Simple RNN #2

$$\begin{aligned}
 & \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array} \right] \times \begin{matrix} \text{roast chicken} \\ \text{on a plate} \end{matrix} + \left[\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{array} \right] \times \begin{matrix} \text{cloud} \\ \text{with rain} \end{matrix} = \\
 & \quad \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{array} \right] + \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{array} \right] \xrightarrow{\text{SAME}} \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{array} \right] \\
 & \quad \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{array} \right] \xrightarrow{\text{NEXT}} \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ 2 \\ 1 \\ 1 \end{array} \right] \xrightarrow{\substack{\text{non-linear} \\ \text{function } (NL)}} \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array} \right] \xrightarrow{\text{merge}} \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{array} \right] = \left[\begin{array}{c} \text{sandwich} \end{array} \right]
 \end{aligned}$$

W_{hh} W_{xh}

$$\begin{aligned}
 & \text{merge} \\
 & \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \times \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array} \right] = \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right]
 \end{aligned}$$

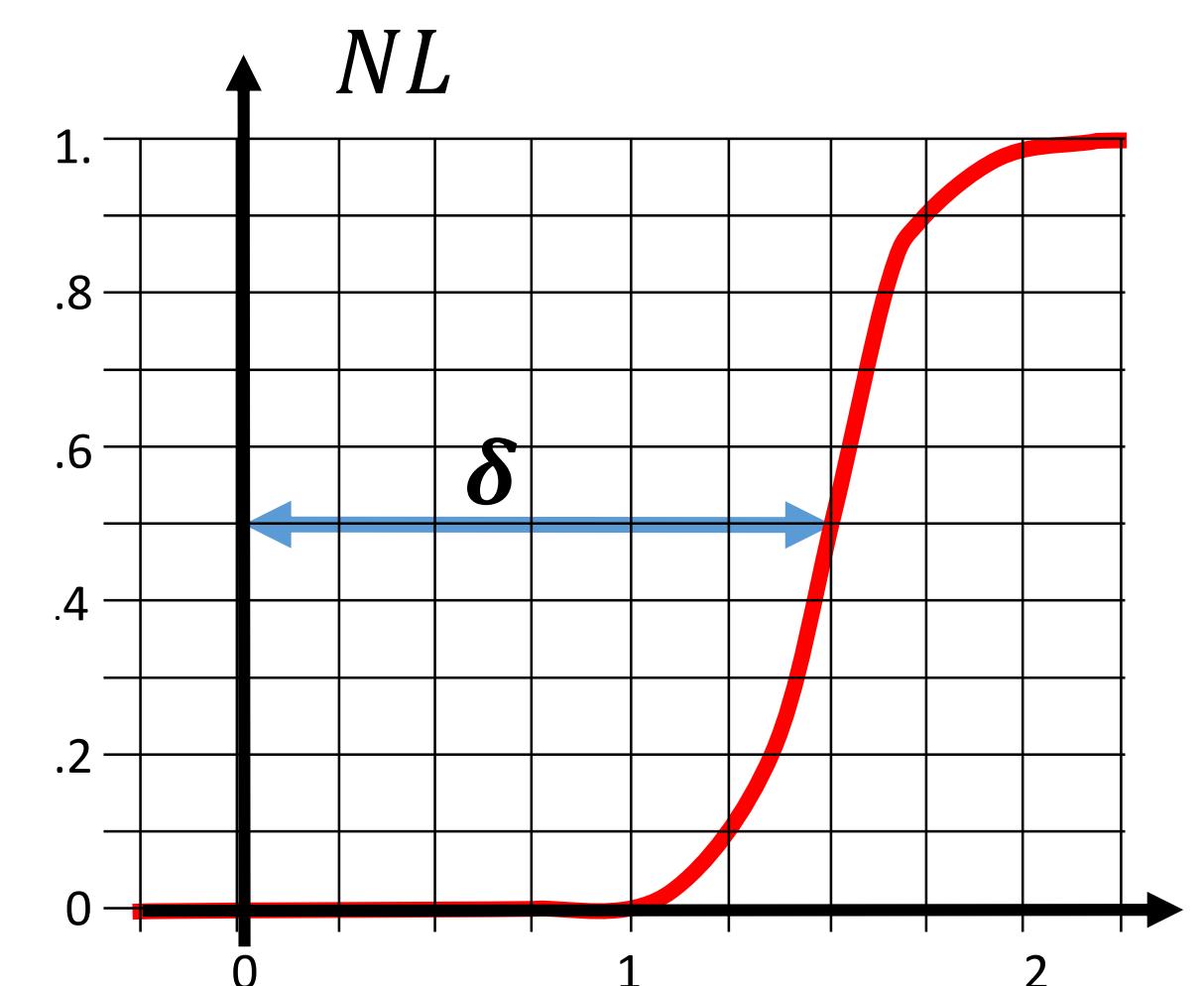
W_{ho}

Simple RNN #2

Actually, a Neural Network ...

$$\mathbf{h}_t = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \times NL \left\{ \begin{array}{c} \mathbf{W}_{ho} \\ \boxed{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}} \mathbf{h}_{t-1} \\ + \boxed{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}} \mathbf{x}_t \end{array} \right\}$$

$$\mathbf{h}_t = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \times NL \left\{ \begin{array}{c} \mathbf{W}_{hh} \\ \boxed{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}} \mathbf{h}_{t-1} \\ + \boxed{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}} \mathbf{x}_t \end{array} \right\}$$



Simple RNN #2

Actually, a Neural Network ...

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

Diagram illustrating the computation of \mathbf{h}_t . The input vector \mathbf{x}_t is shown as a column vector with elements 1, 0, 1, 0, 1, 0. The hidden state \mathbf{h}_{t-1} is shown as a column vector with elements 0, 0, 1, 0, 1, 0. The bias vector $\boldsymbol{\delta}$ is shown as a column vector with elements δ, δ, δ, δ, δ, δ. The weight matrices \mathbf{W}_{hh} and \mathbf{W}_{xh} are shown as 6x6 matrices. The final output \mathbf{h}_t is the result of the weighted sum of \mathbf{h}_{t-1} , \mathbf{x}_t , and $\boldsymbol{\delta}$.

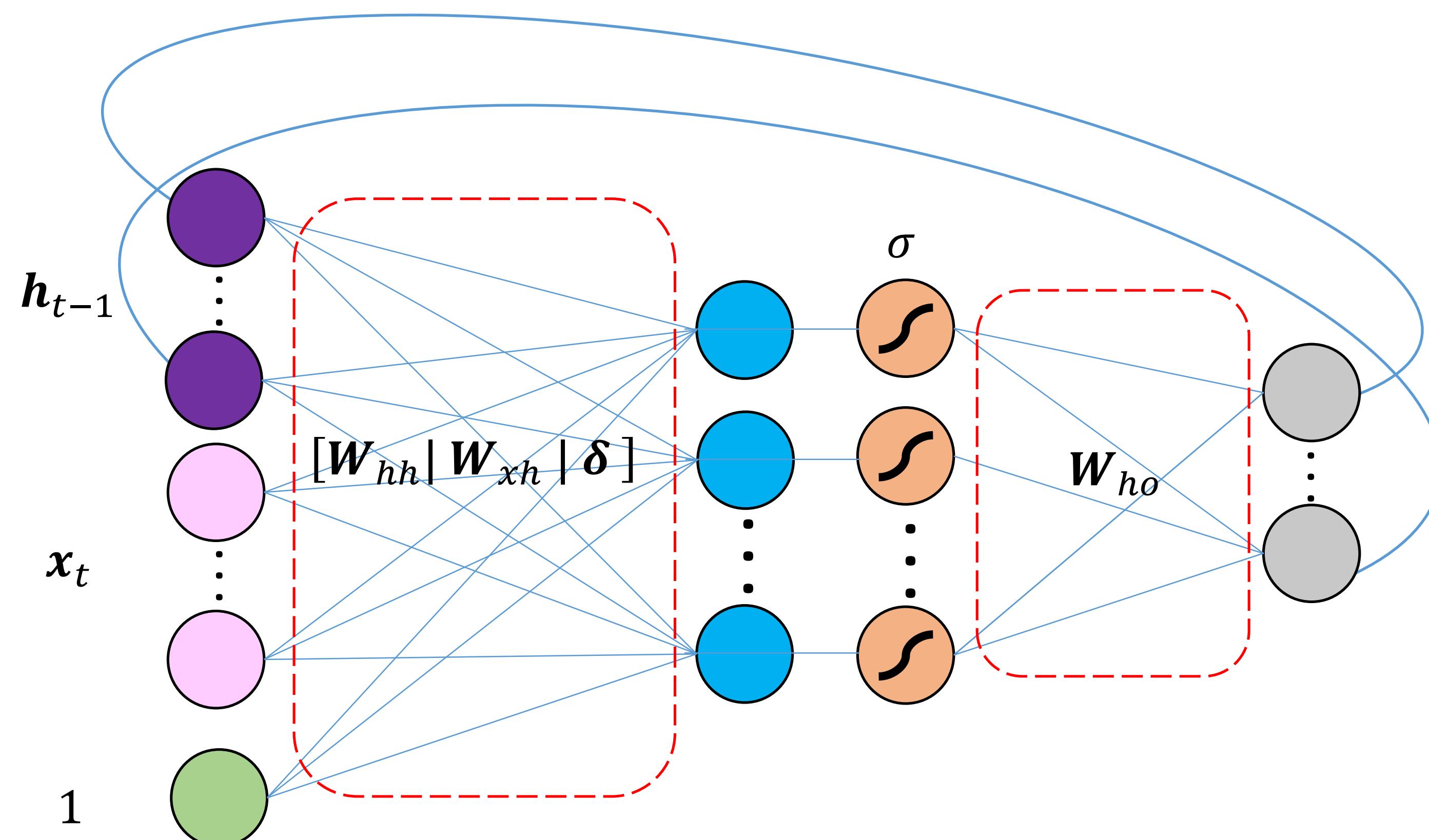
$$\mathbf{h}_t = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \times NL \left\{ \begin{bmatrix} \mathbf{W}_{hh} \\ \mathbf{W}_{xh} \\ \boldsymbol{\delta} \end{bmatrix} \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \\ 1 \end{bmatrix} \right\}$$

Diagram illustrating the computation of \mathbf{h}_t . The input vector \mathbf{x}_t is shown as a column vector with elements 1, 0, 1, 0, 1, 0. The hidden state \mathbf{h}_{t-1} is shown as a column vector with elements 0, 0, 1, 0, 1, 0. The bias vector $\boldsymbol{\delta}$ is shown as a column vector with elements δ, δ, δ, δ, δ, δ. The weight matrices \mathbf{W}_{hh} and \mathbf{W}_{xh} are shown as 6x6 matrices. The final output \mathbf{h}_t is the result of the weighted sum of \mathbf{h}_{t-1} , \mathbf{x}_t , and $\boldsymbol{\delta}$.

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

Simple RNN #2

Actually, a Recurrent Neural Network...



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

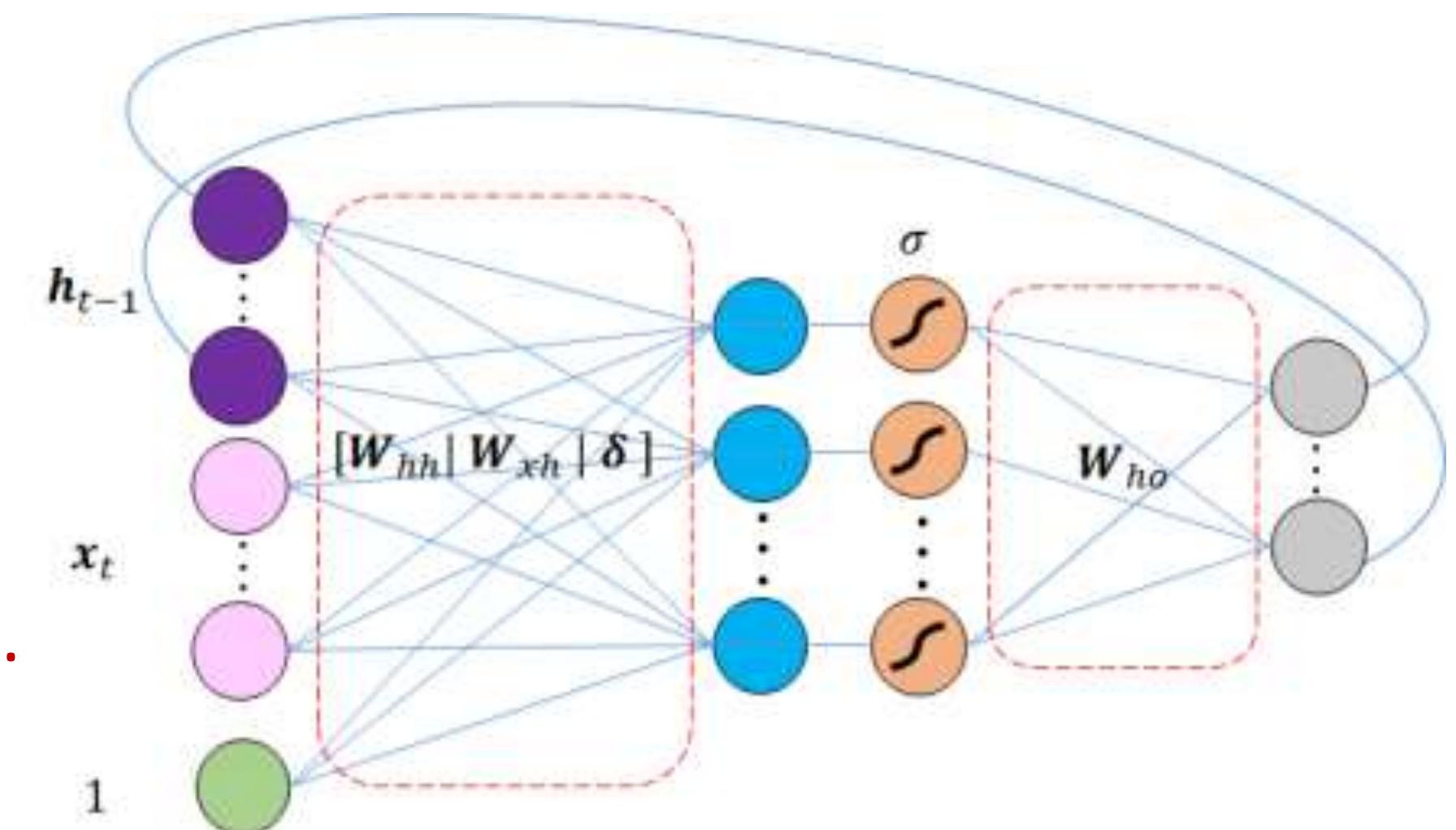
Simple RNN

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)
```

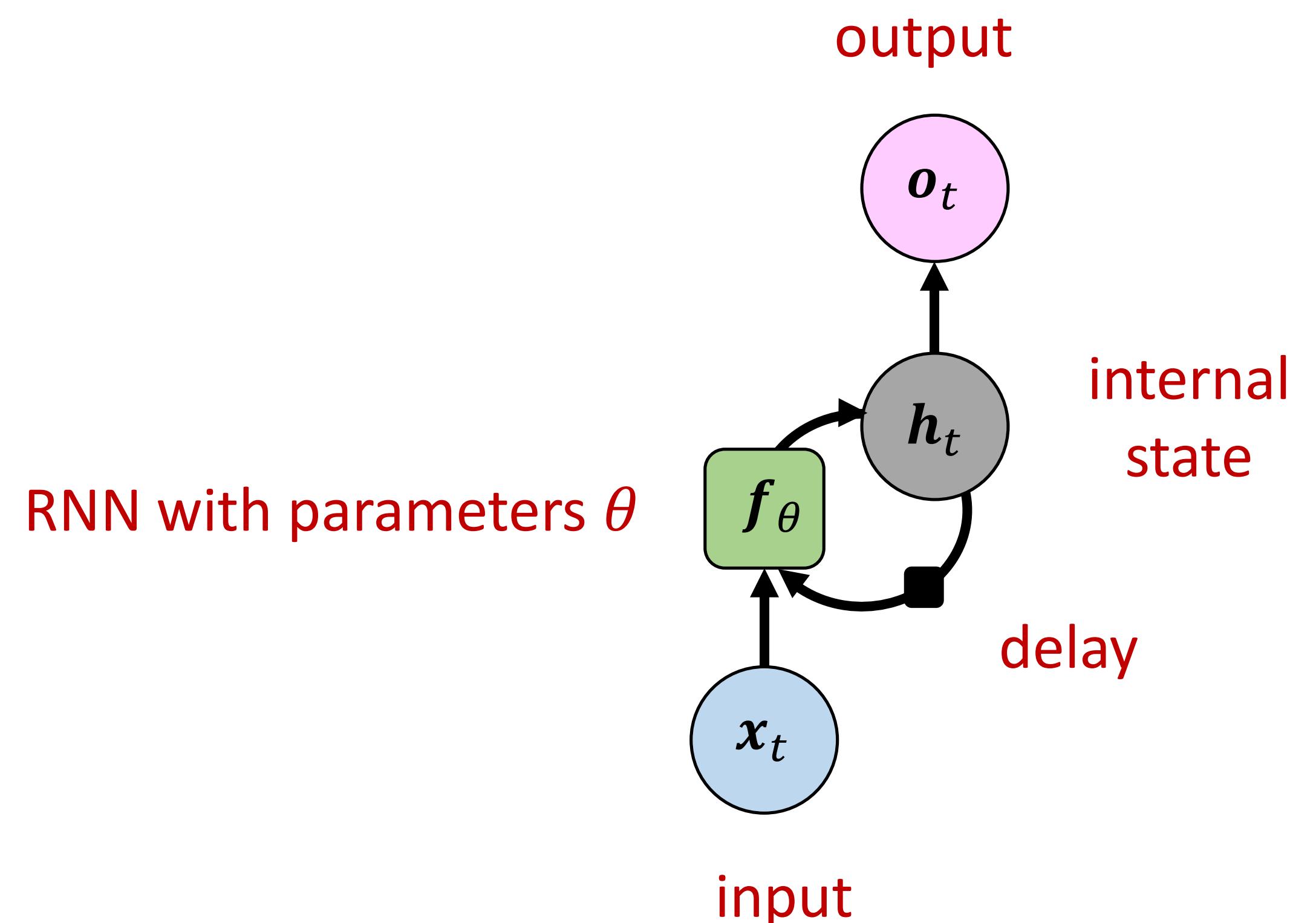
```
# returning the whole sequence of outputs  
simple_rnn = tf.keras.layers.SimpleRNN(2, return_sequences=True)  
outputs = simple_rnn(inputs) # output has shape (32, 10, 2)
```



Recurrent Neural Network

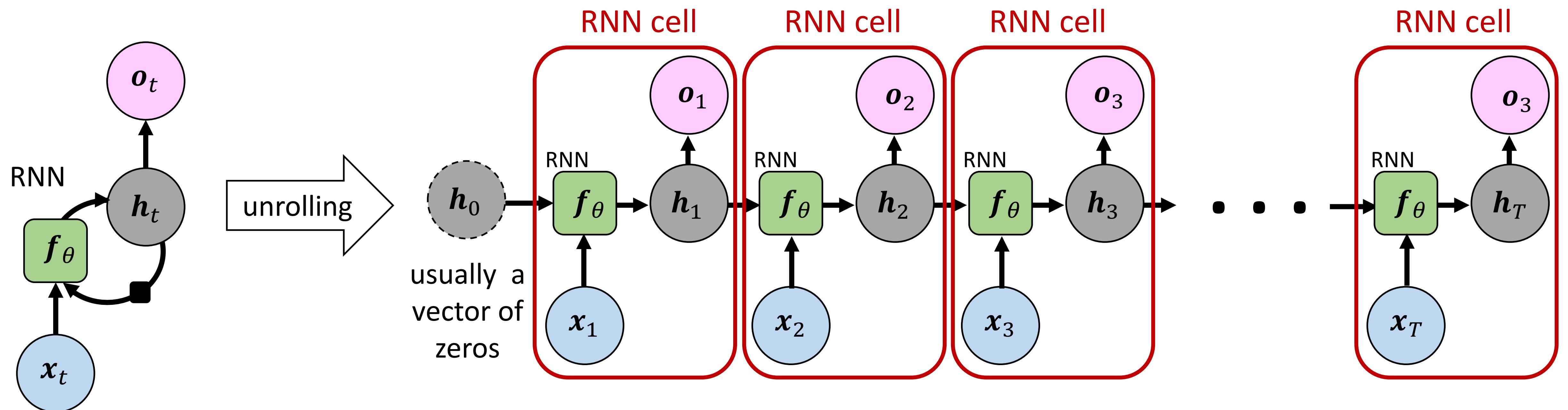
- Introduction
- RNN Architecture
- RNN Training
- RNN Variants
- Extensions

RNN architecture



RNN unrolling

θ fixed for all time steps (cells)

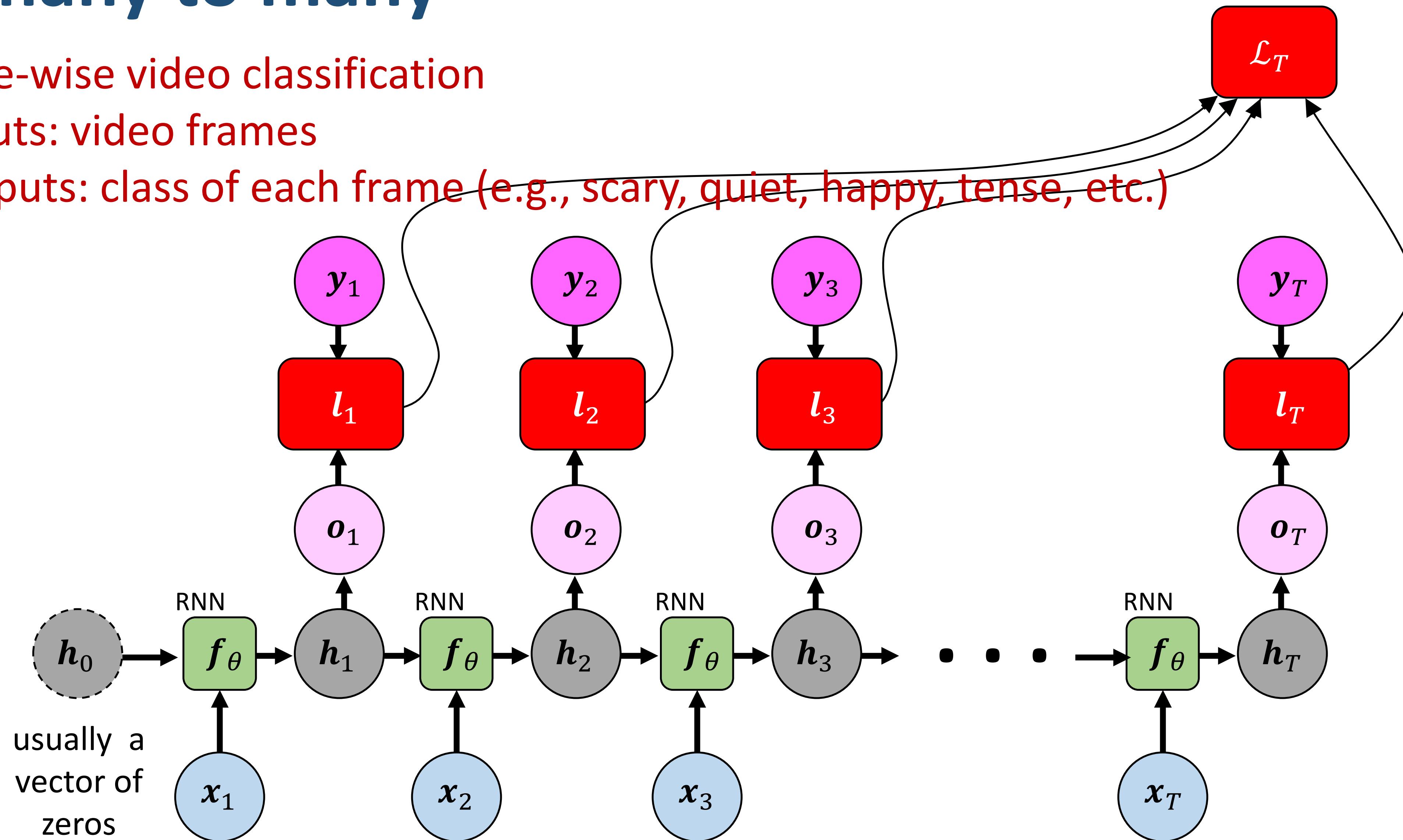


RNN many to many

e.g., frame-wise video classification

Many inputs: video frames

Many outputs: class of each frame (e.g., scary, quiet, happy, tense, etc.)

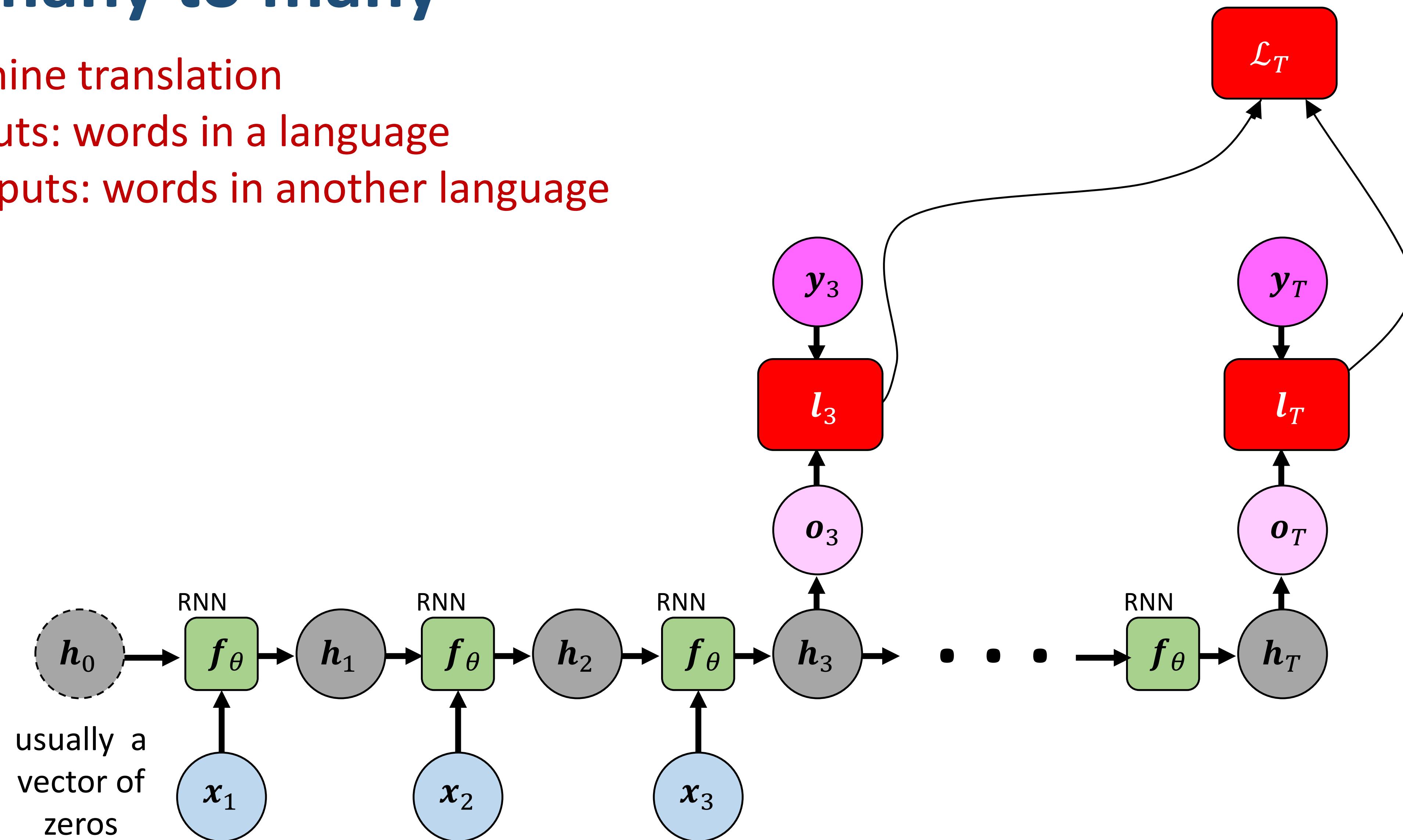


RNN many to many

e.g., machine translation

Many inputs: words in a language

Many outputs: words in another language

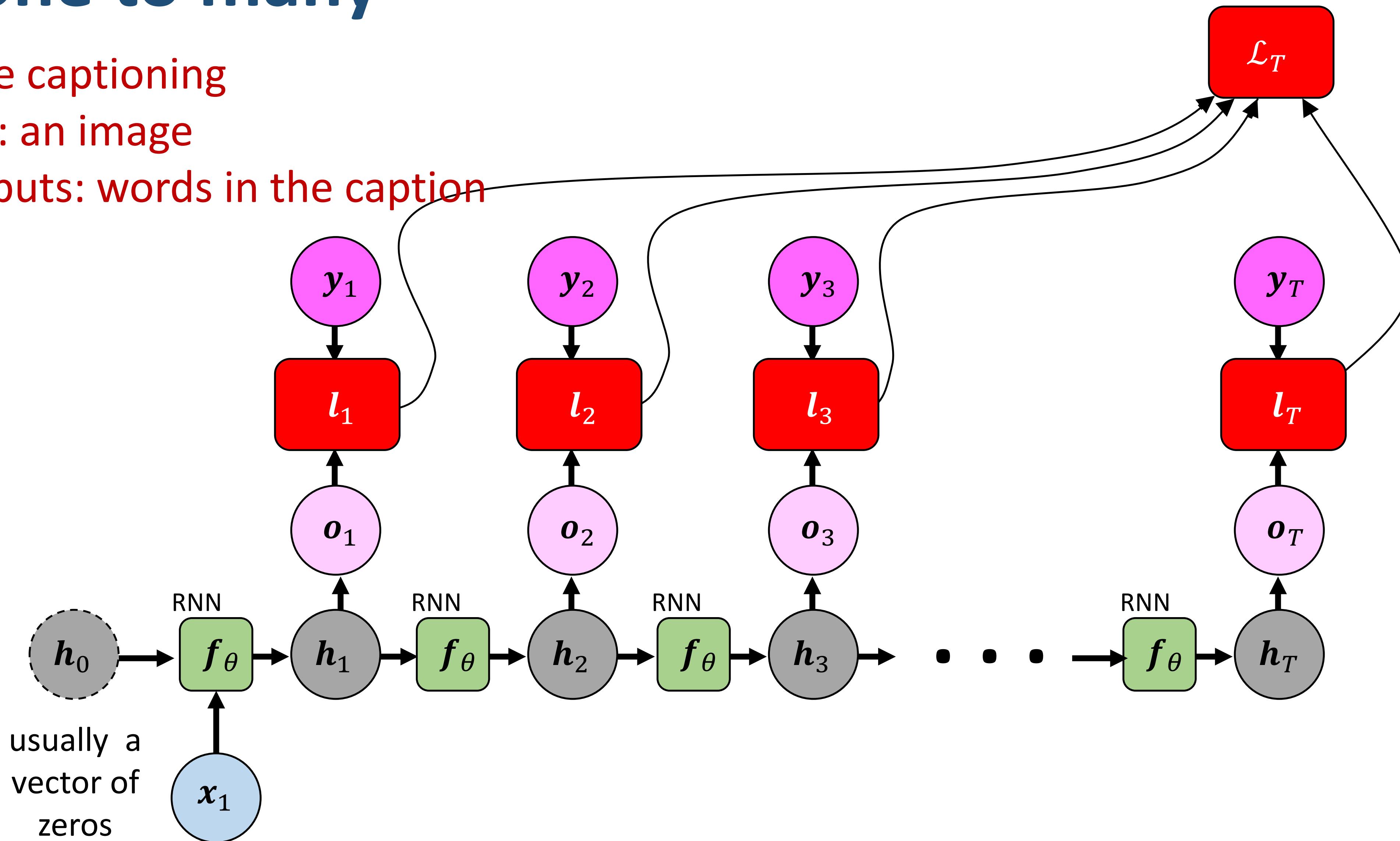


RNN one to many

e.g., image captioning

One input: an image

Many outputs: words in the caption

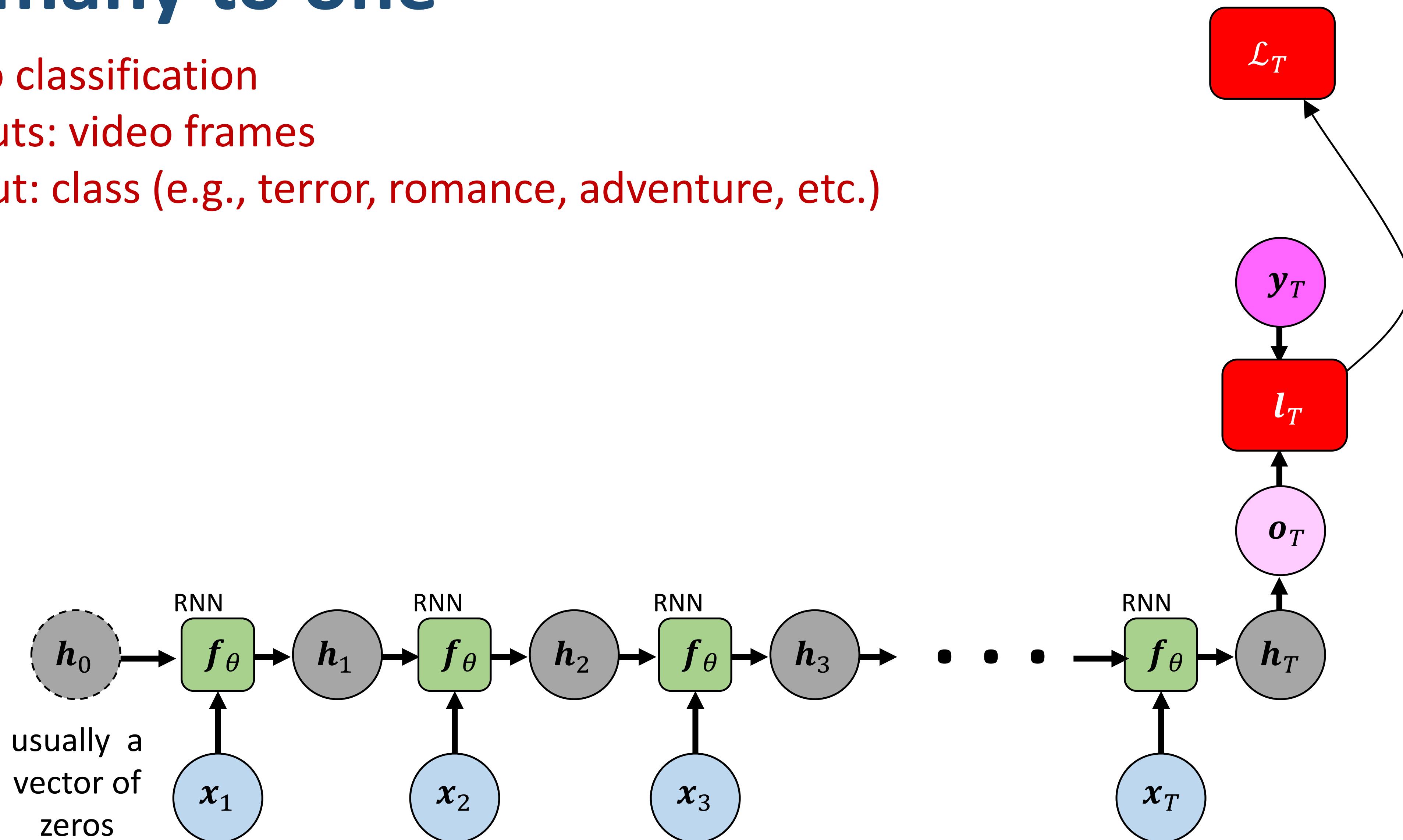


RNN many to one

e.g., video classification

Many inputs: video frames

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

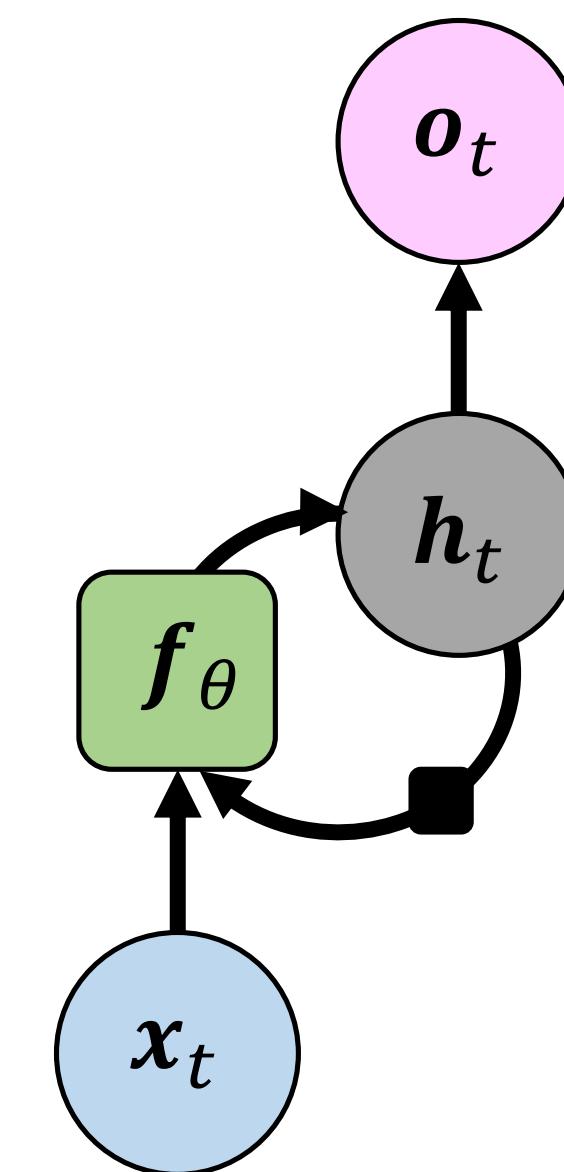


Recurrent Neural Network

We can process a sequence of vectors x_t by applying a recurrence formula at every time step:

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

new state prior state
some function with input vector at the
parameters θ current step



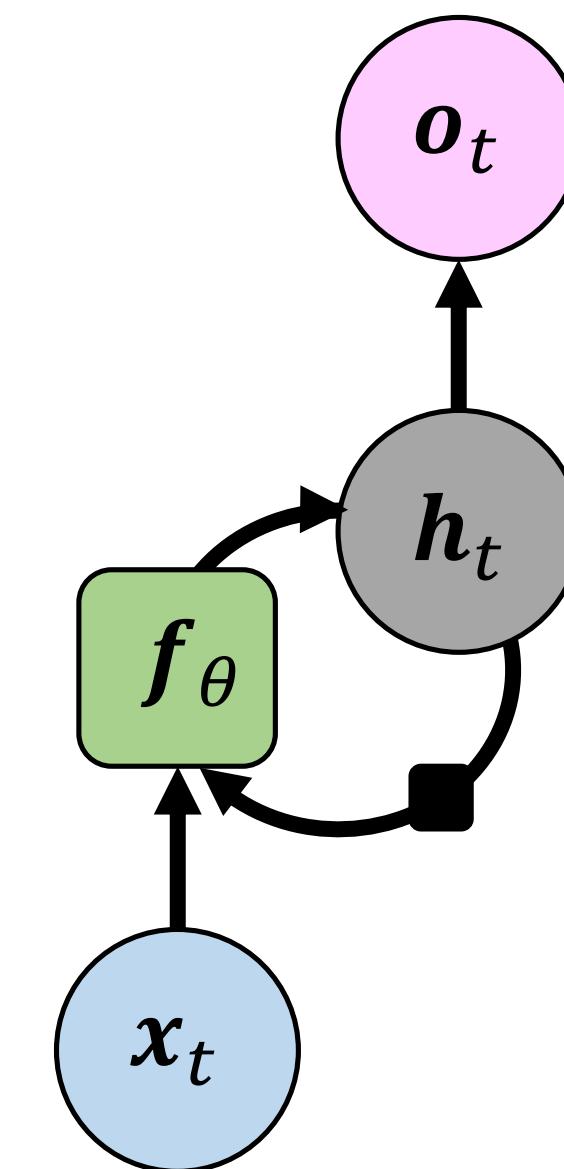
The same function f_θ and the same set of parameters θ are used at every time step.

Recurrence: recursivity

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

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

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



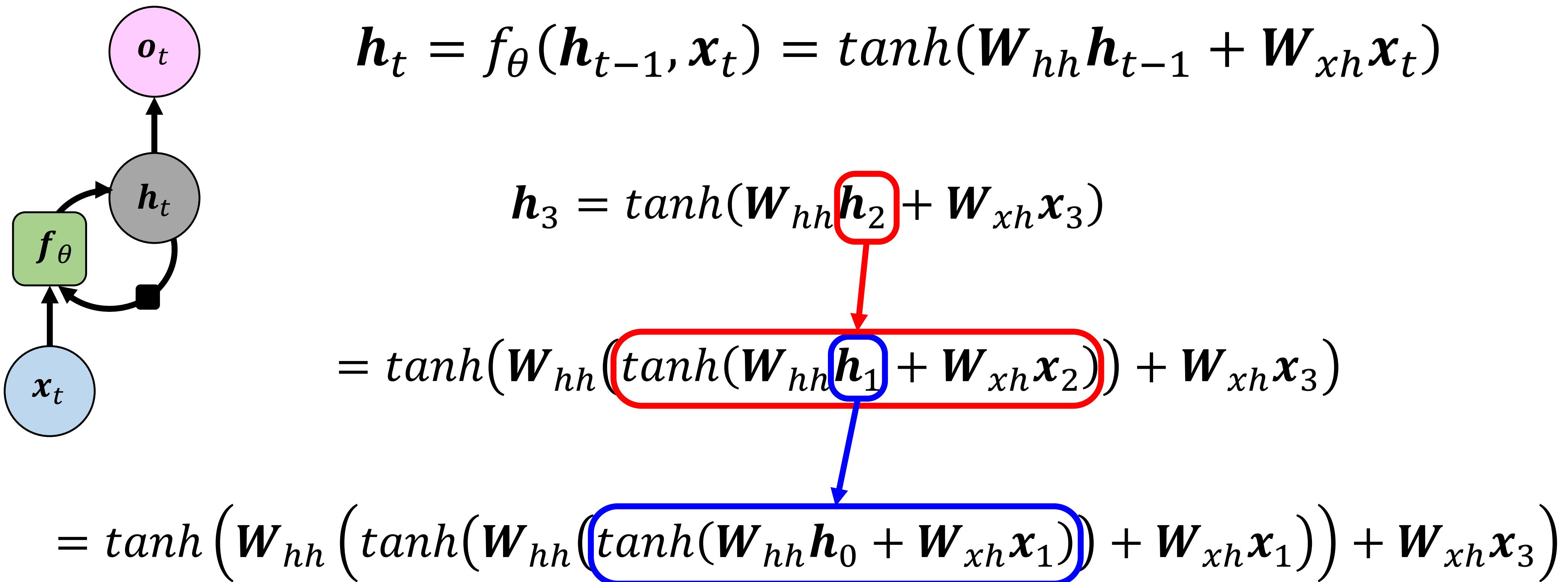
Let's assume the following simple RNN:

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

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

Recurrence

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

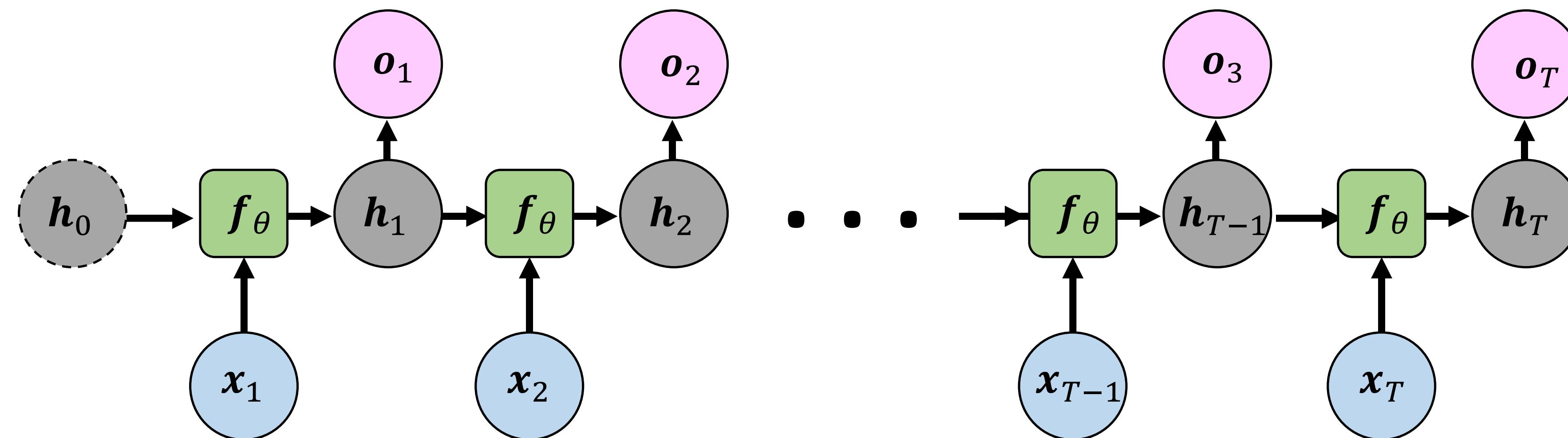


Recurrent Neural Network

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

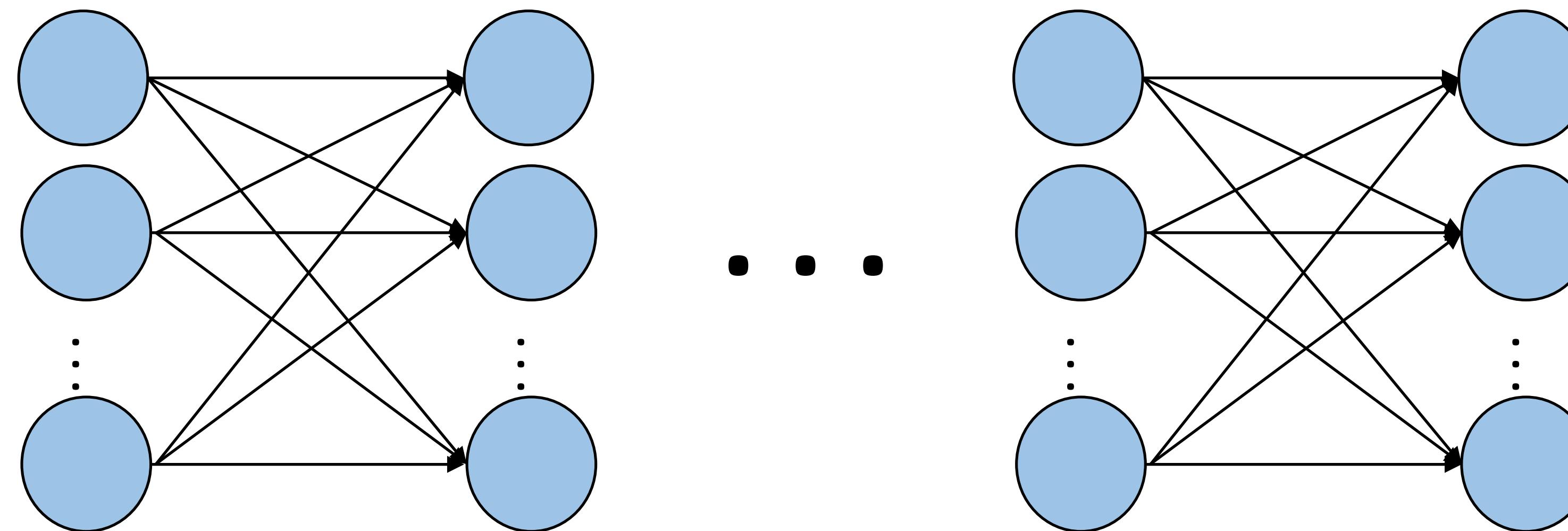
Backpropagation Through Time (BPTT)

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.

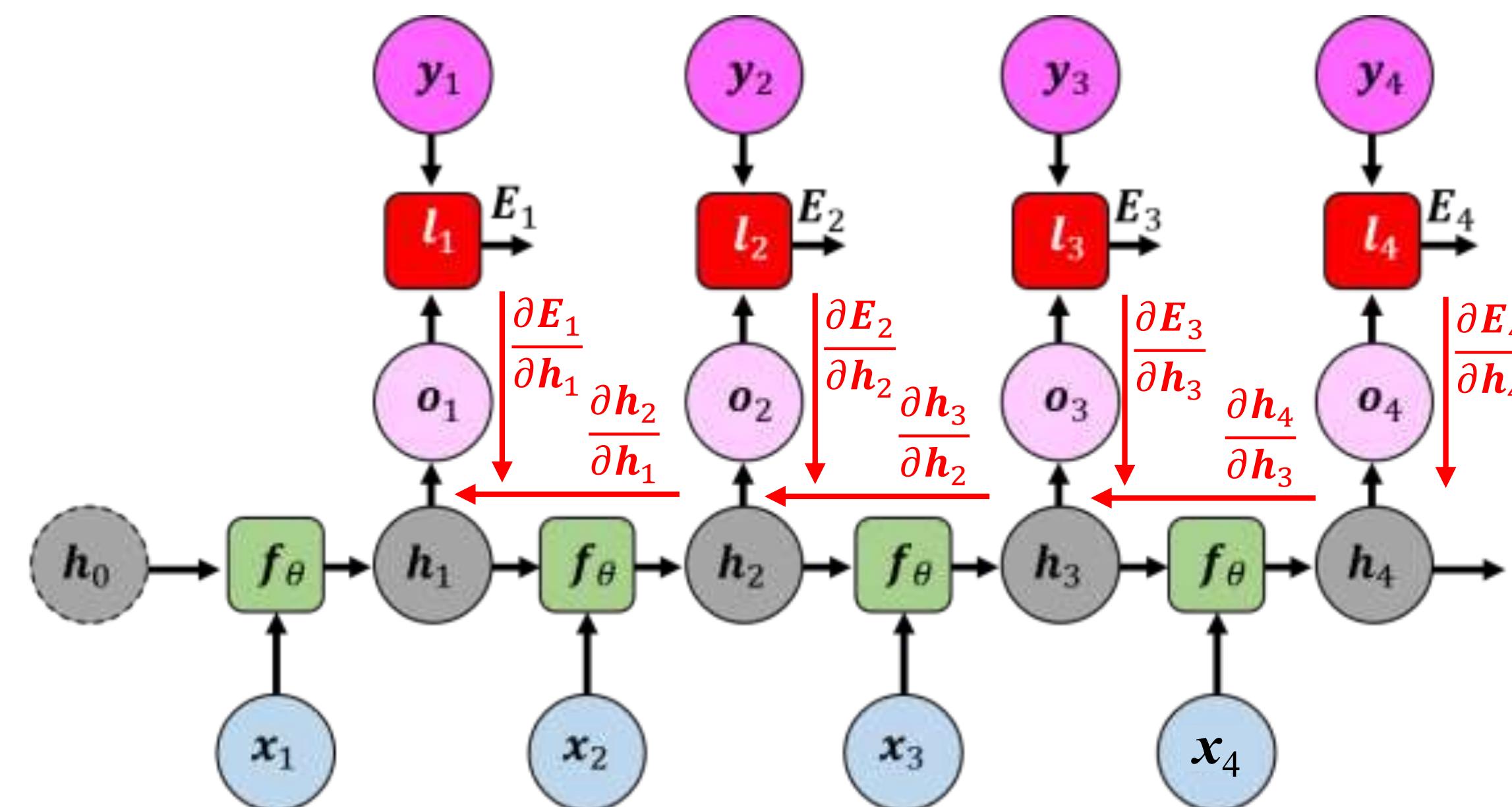
Backpropagation Through Time (BPTT)

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.

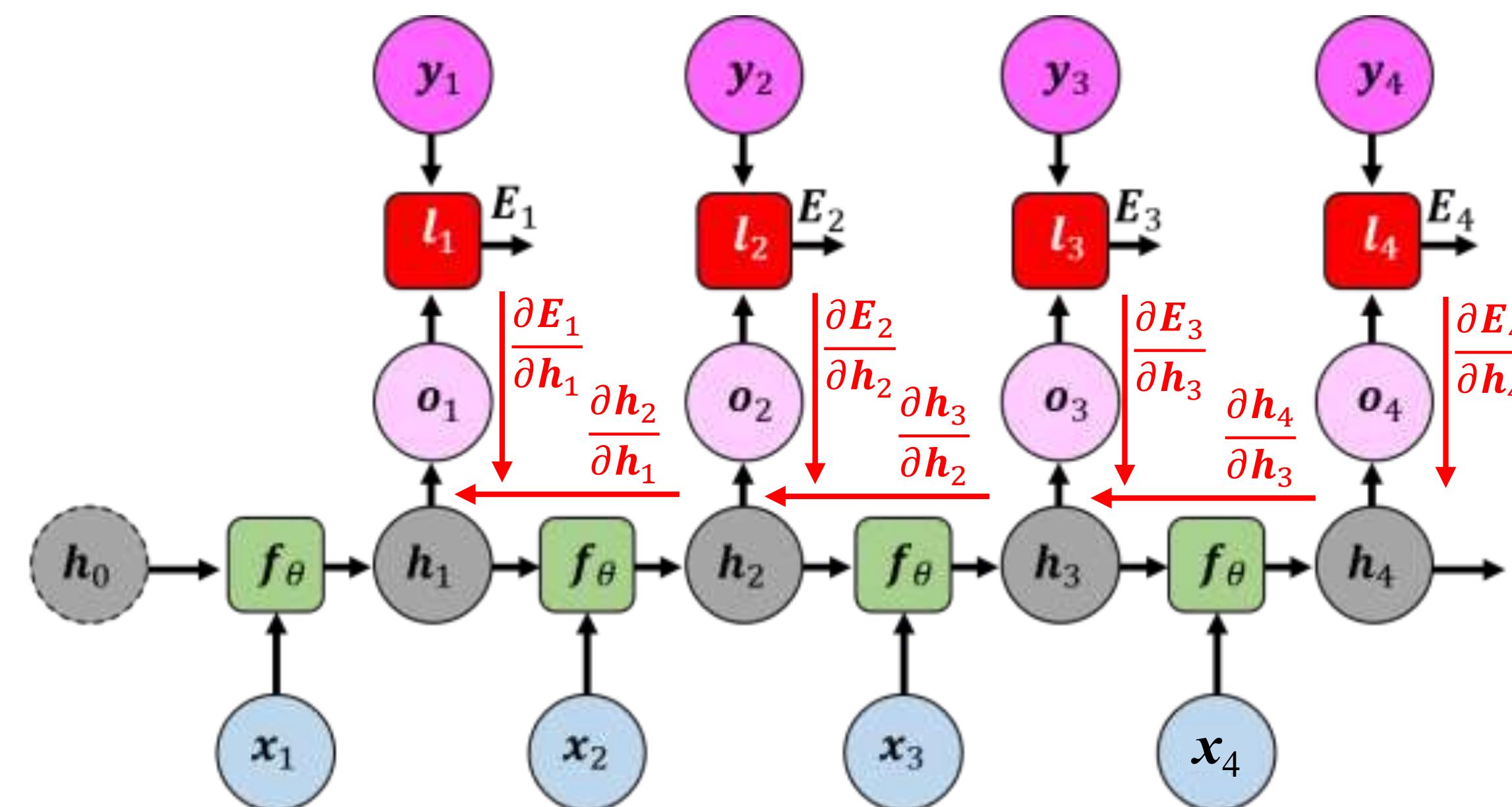
Backpropagation Through Time (BPTT)

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.

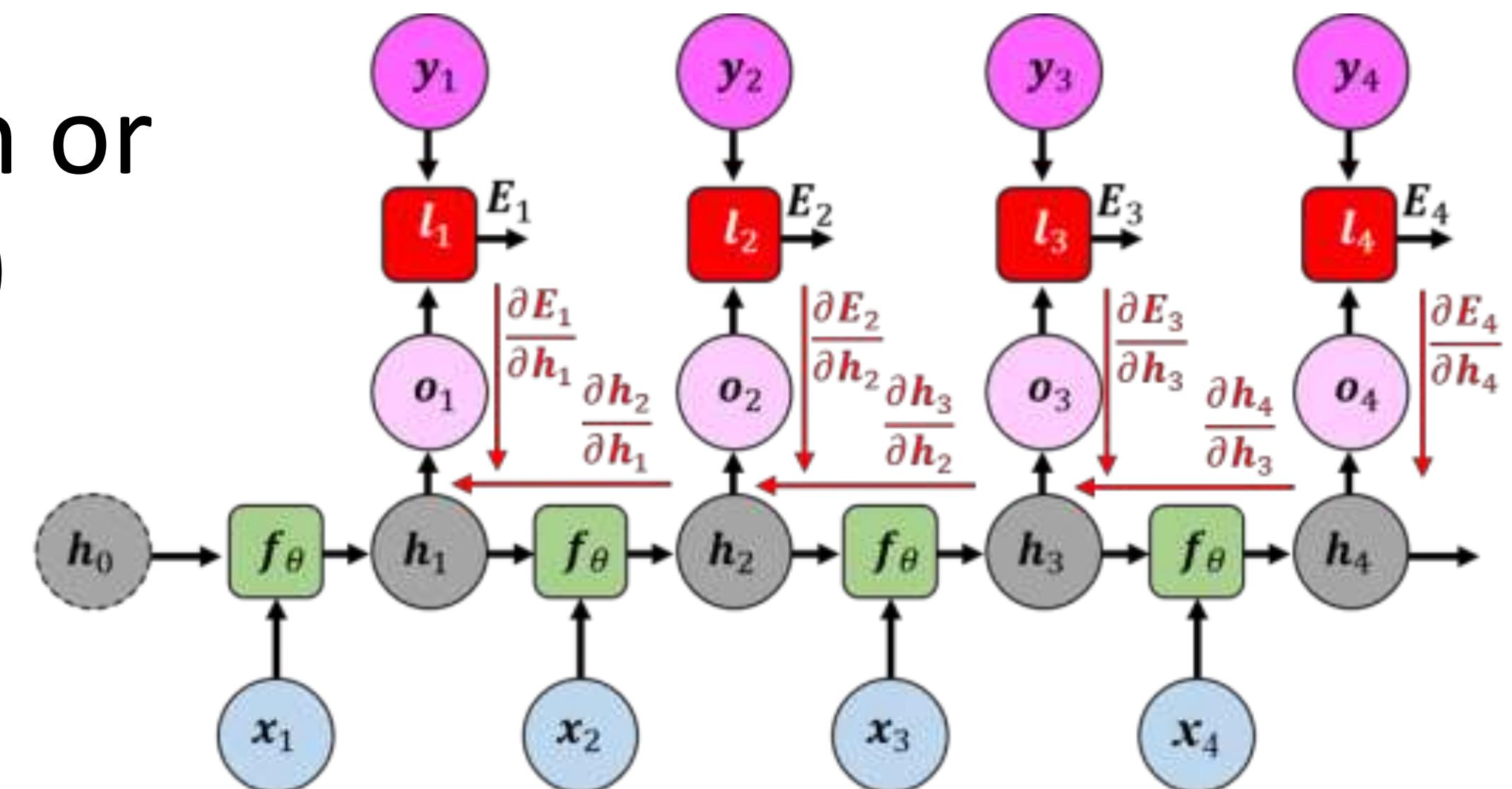


Thus, for long sequences, each batch computation, as well as the memory demand, might become unacceptably high.

Problem with BPTT

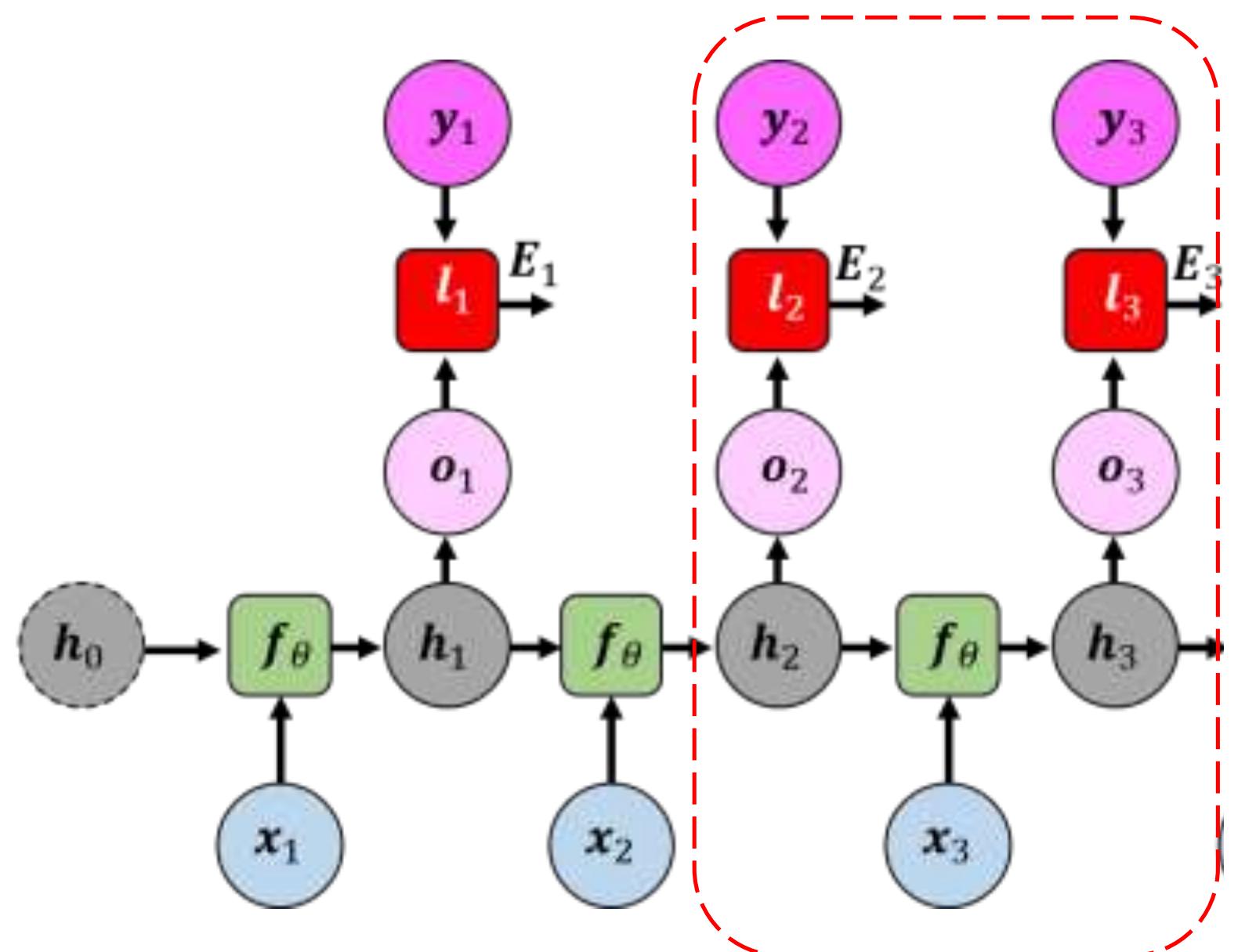
During training the gradient of error must propagate all the way back to the first time step.

- 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

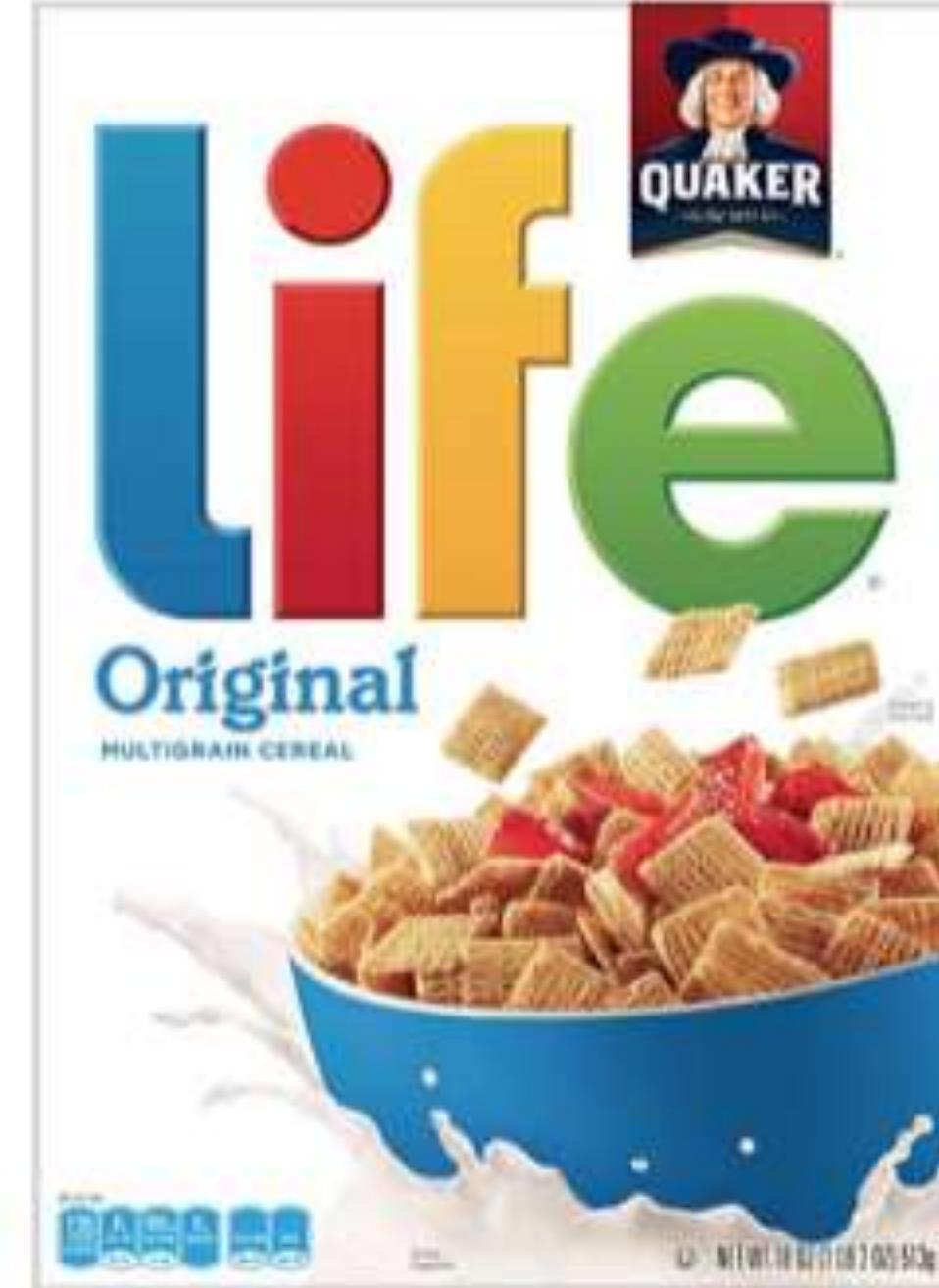


Thanos

September 2018

Verified Purchase

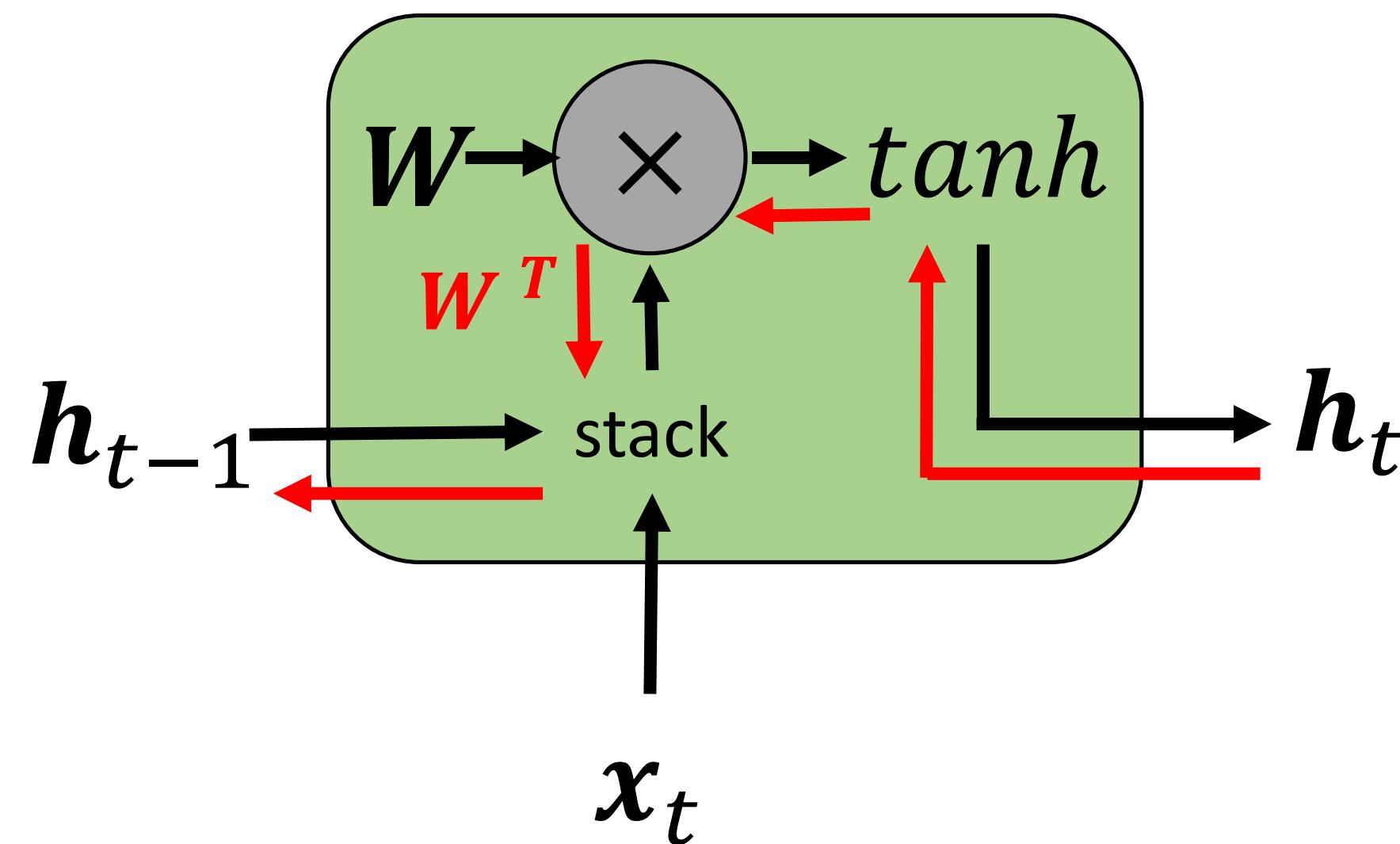
Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!



A Box of Cereal
\$3.99

The short term memory problem

Consider the basic RNN so far:

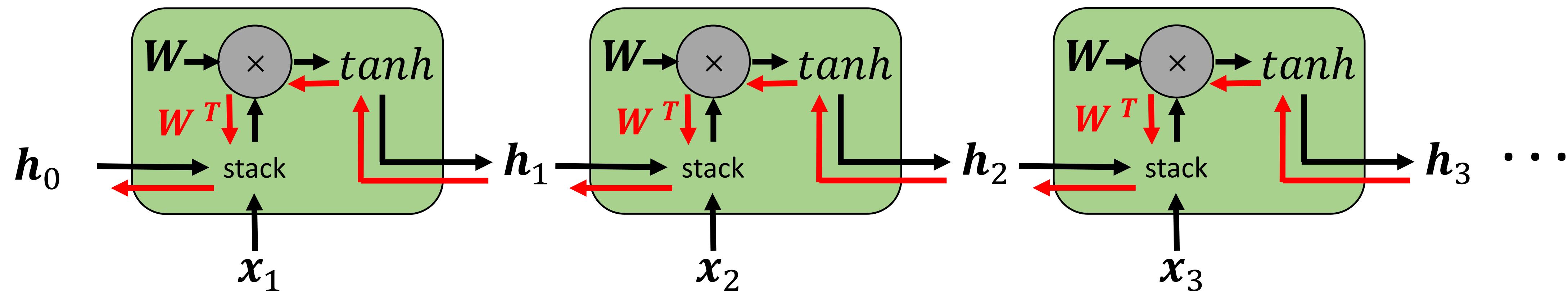


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh\left(\begin{pmatrix} W_{hh} & W_{xh} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \end{aligned}$$

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

The short term memory problem

Gradient flow...



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

Largest singular value > 1 :
exploding gradients

Largest singular value < 1 :
vanishing gradients

- **Gradient clipping**: scale the gradient if its norm exceeds a threshold
- **Change RNN architecture**

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).

Long Short Term Memory (LSTM)

Vanilla RNN

$$\mathbf{h}_t = \tanh\left(W\begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} f \\ i \\ s \\ \tilde{\mathbf{c}}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix}$$

$$\begin{aligned} \mathbf{c}_t &= f \times \mathbf{c}_{t-1} + i \times \tilde{\mathbf{c}}_t \\ \mathbf{h}_t &= s \times \tanh(\mathbf{c}_t) \end{aligned}$$

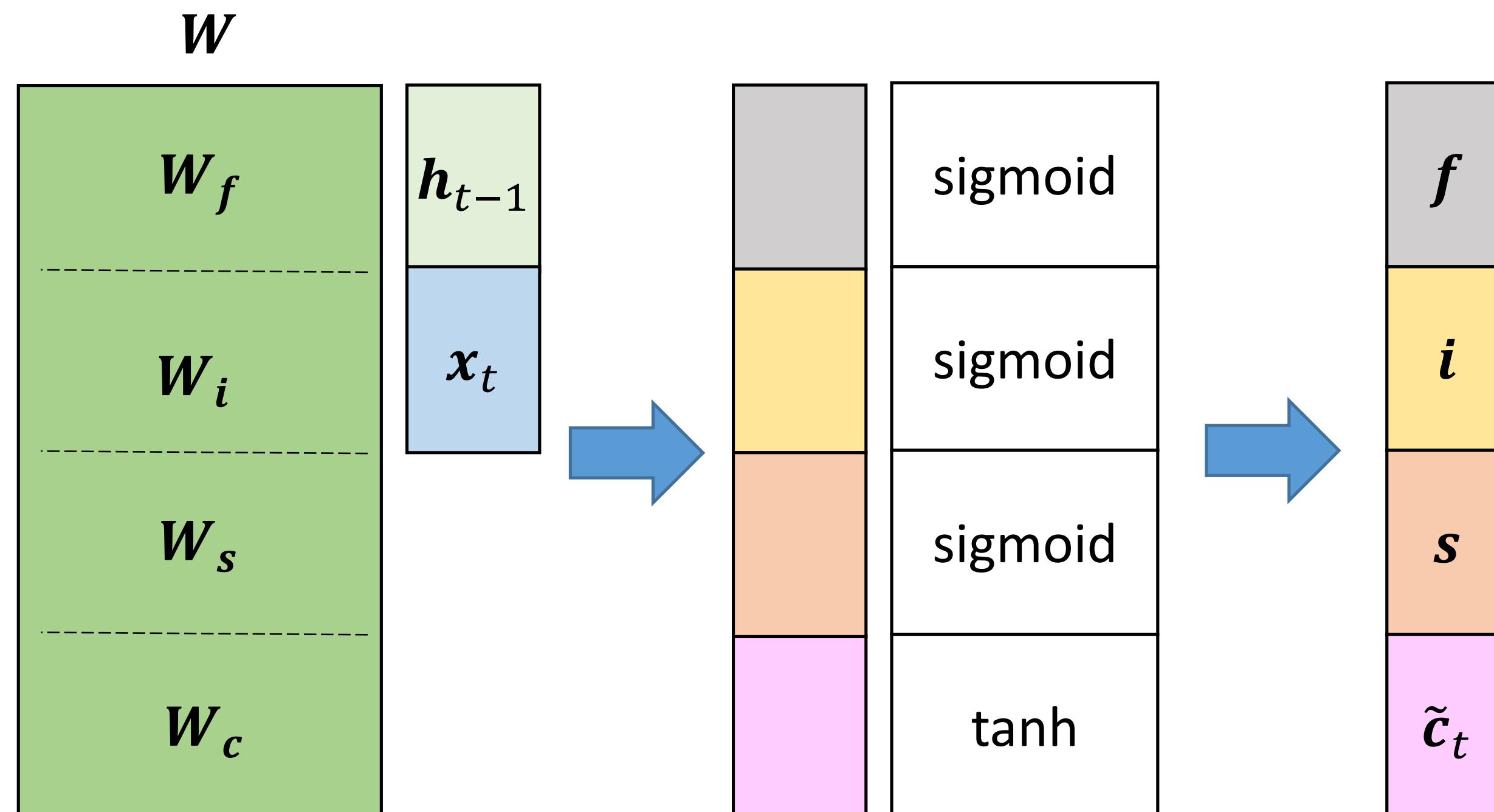
$$W = \begin{pmatrix} \mathbf{W}_f \\ \mathbf{W}_i \\ \mathbf{W}_s \\ \mathbf{W}_c \end{pmatrix}$$

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.

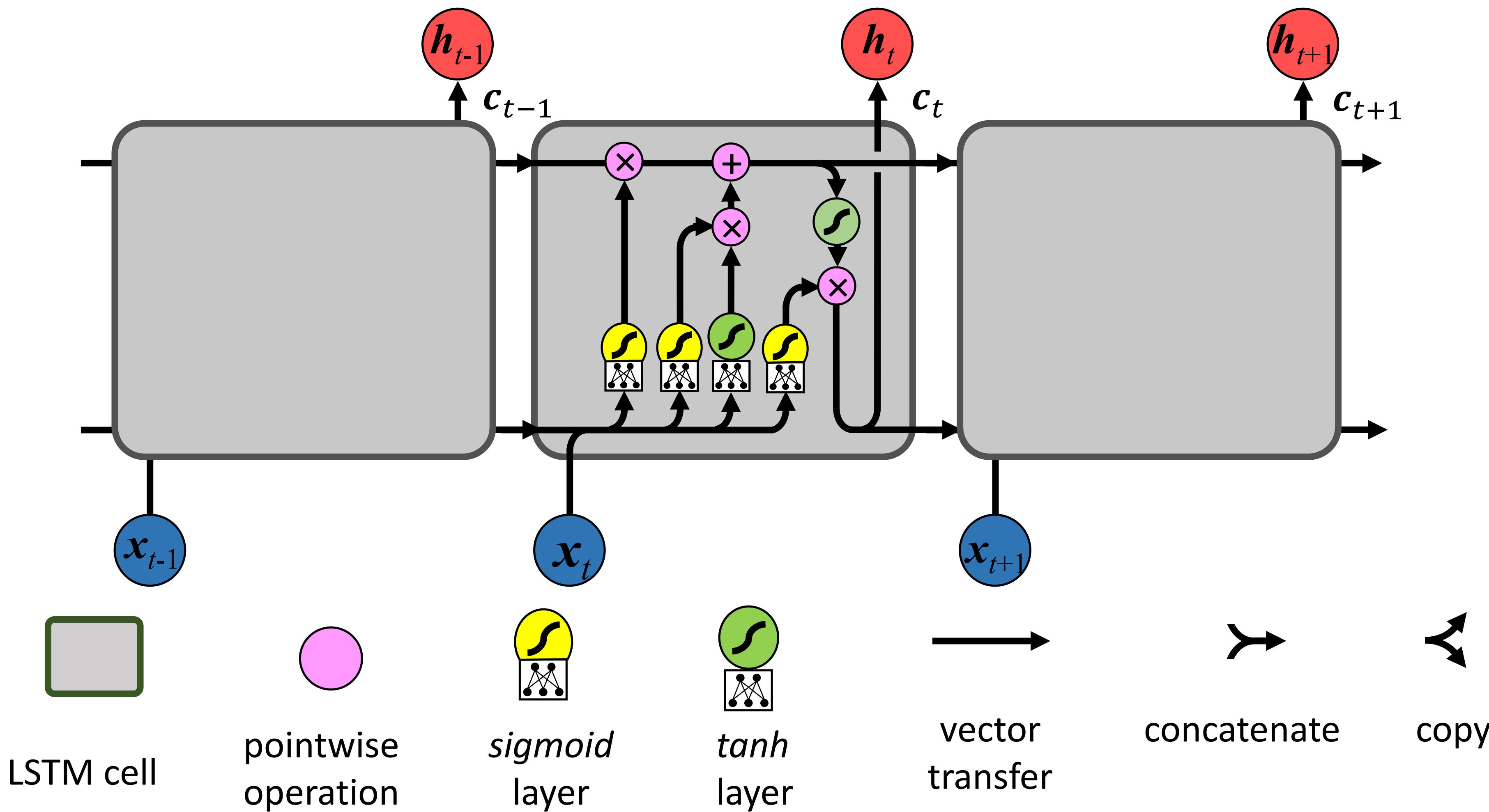


$$\begin{pmatrix} f \\ i \\ s \\ \tilde{c}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

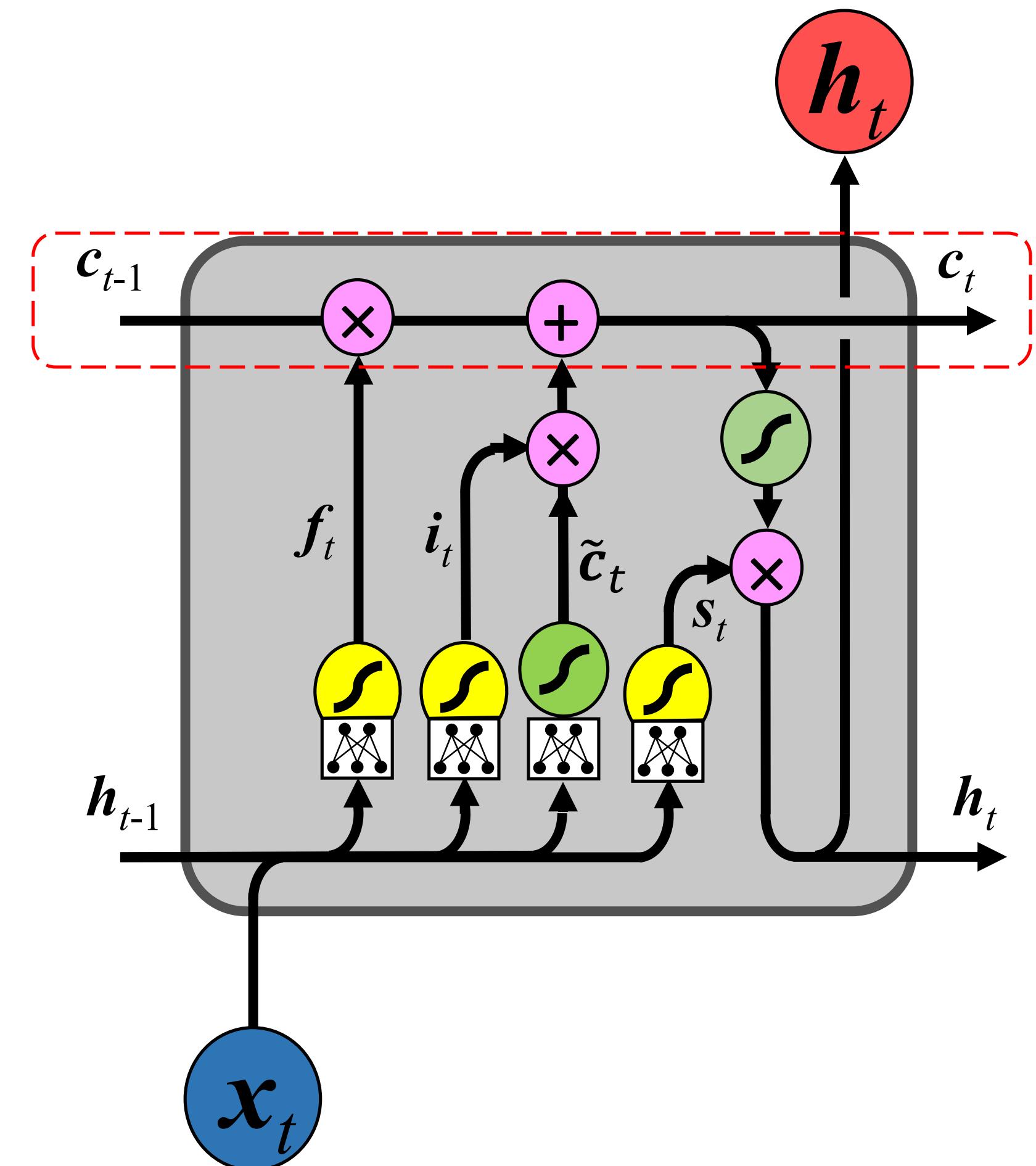
LSTM interacting cells



Core idea behind LSTMs

The cell state (c_t):

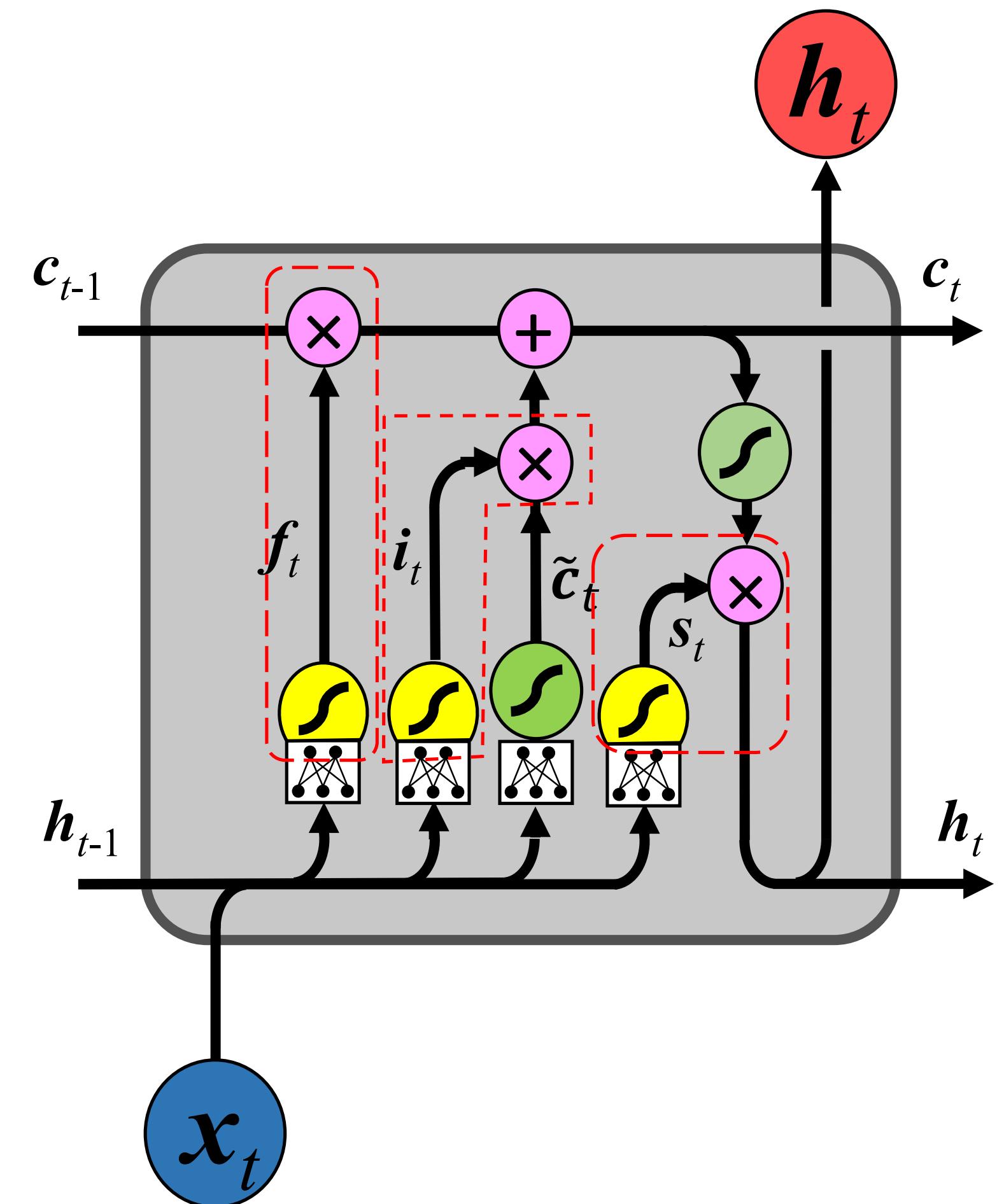
- Retains the **relevant information from the past** network operation.
- Kind of a conveyor belt.
- Information can flow along it unchanged.
- But possible to remove information from it (by multiplying by zero).
- Also possible to add information to it.
- **No vanishing gradient** along the cell state!



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 → blocks everything
 - one → lets everything pass



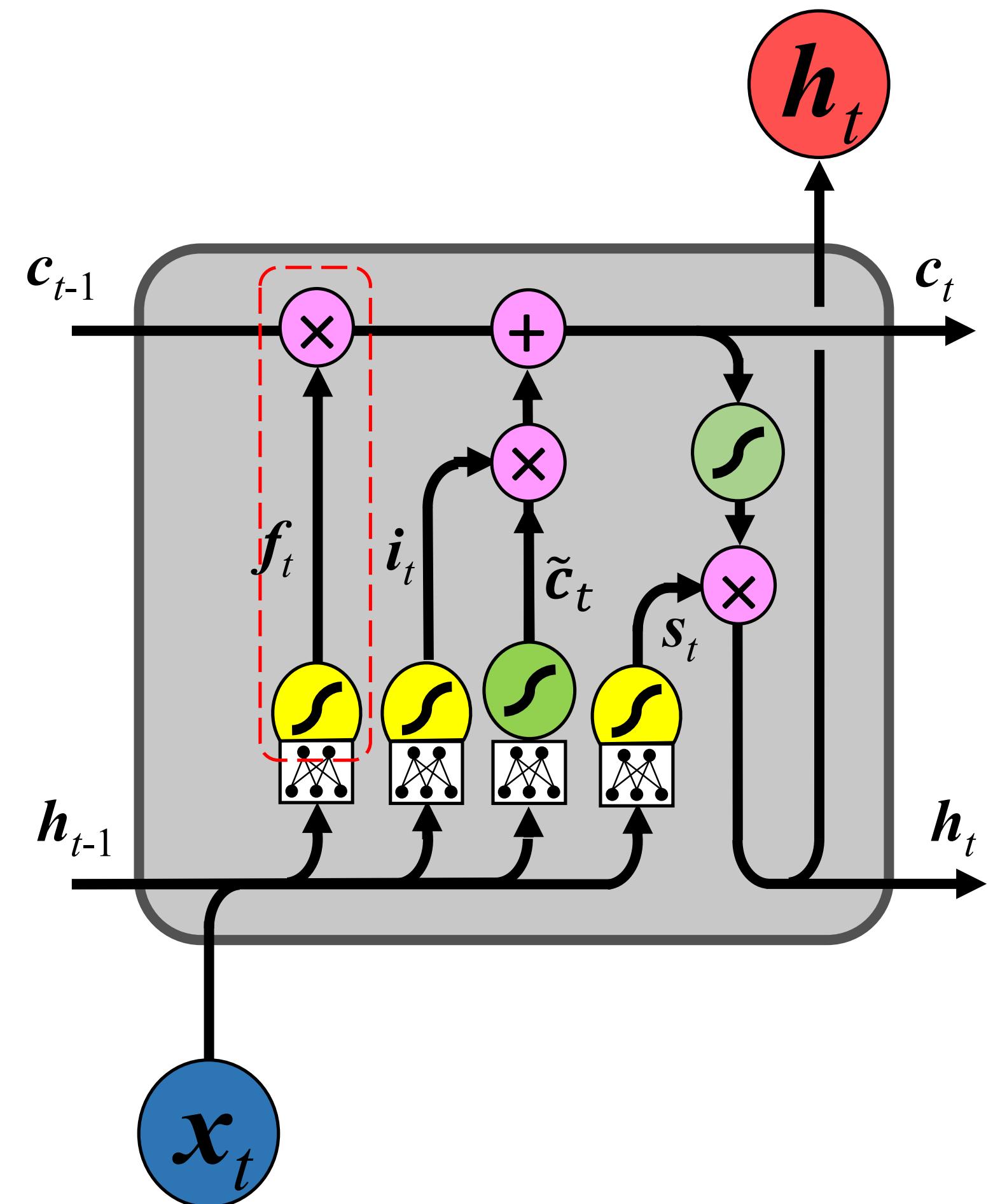
Walk through a LSTM

Forget gate:

- It decides when and which information in the cell state c_{t-1} must be kept ($f_t = 1$) or dropped ($f_t = 0$) based on h_{t-1} and x_t :

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- This is done for each element of c_{t-1}

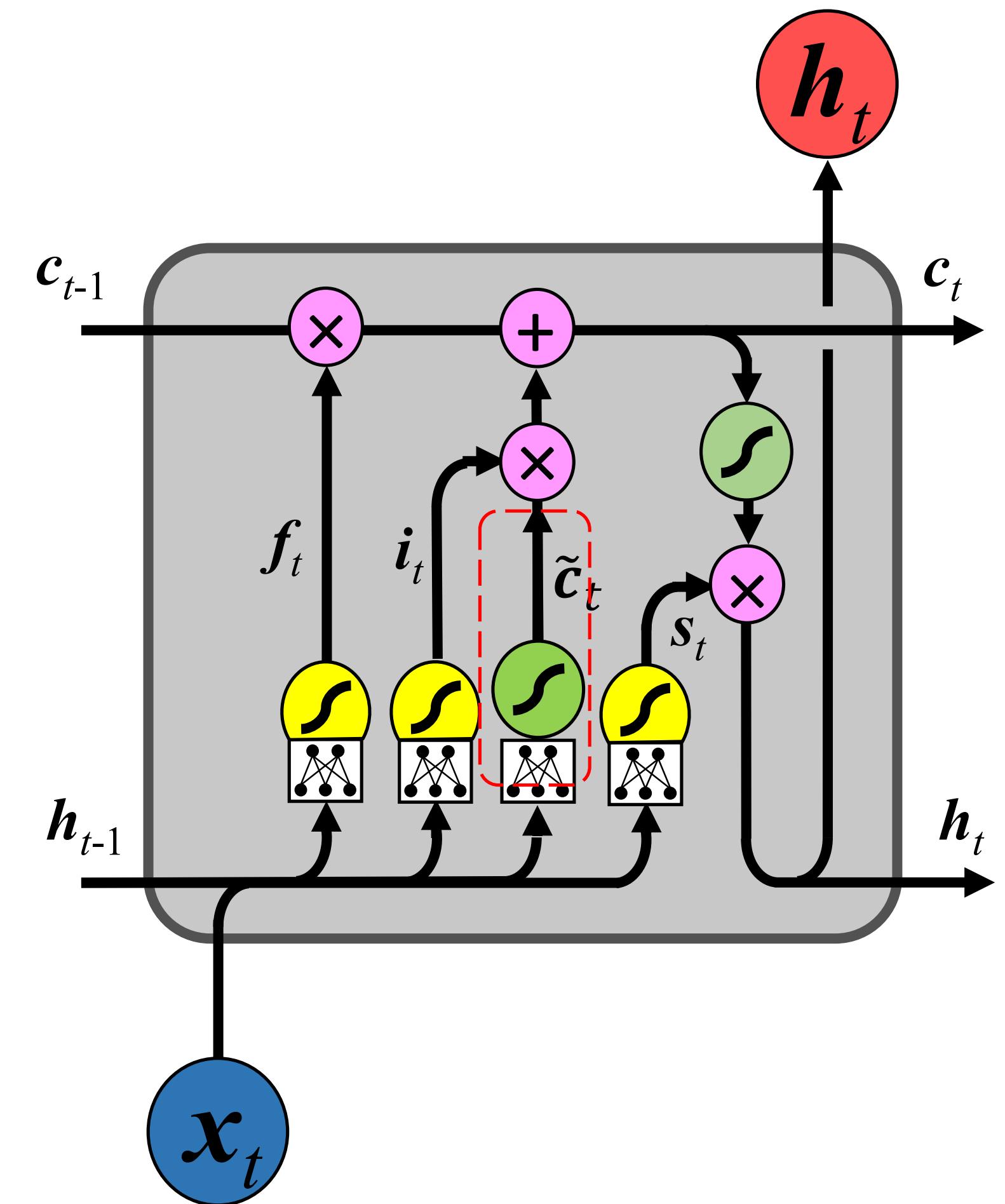


Walk through a LSTM

Candidate values:

- Based on h_{t-1} and x_t , the LSTM cell creates a new vector of candidate values \tilde{c}_t :

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t])$$



Walk through a LSTM

Ignore gate:

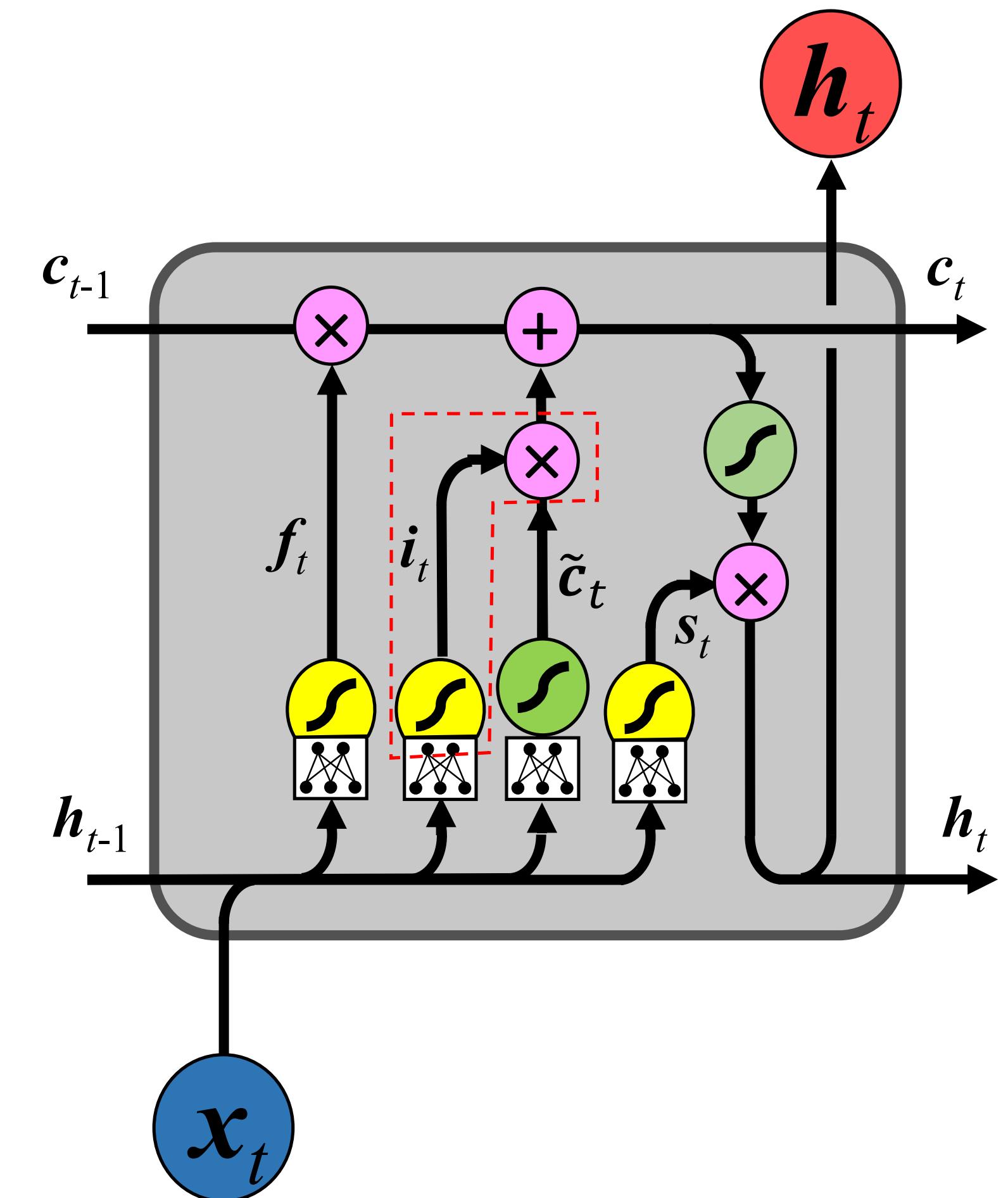
- Again, based on h_{t-1} and x_t , the ignore gate tells which candidate values in \tilde{c}_t shall pass through:

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

Cell update:

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

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$$



Walk through a LSTM

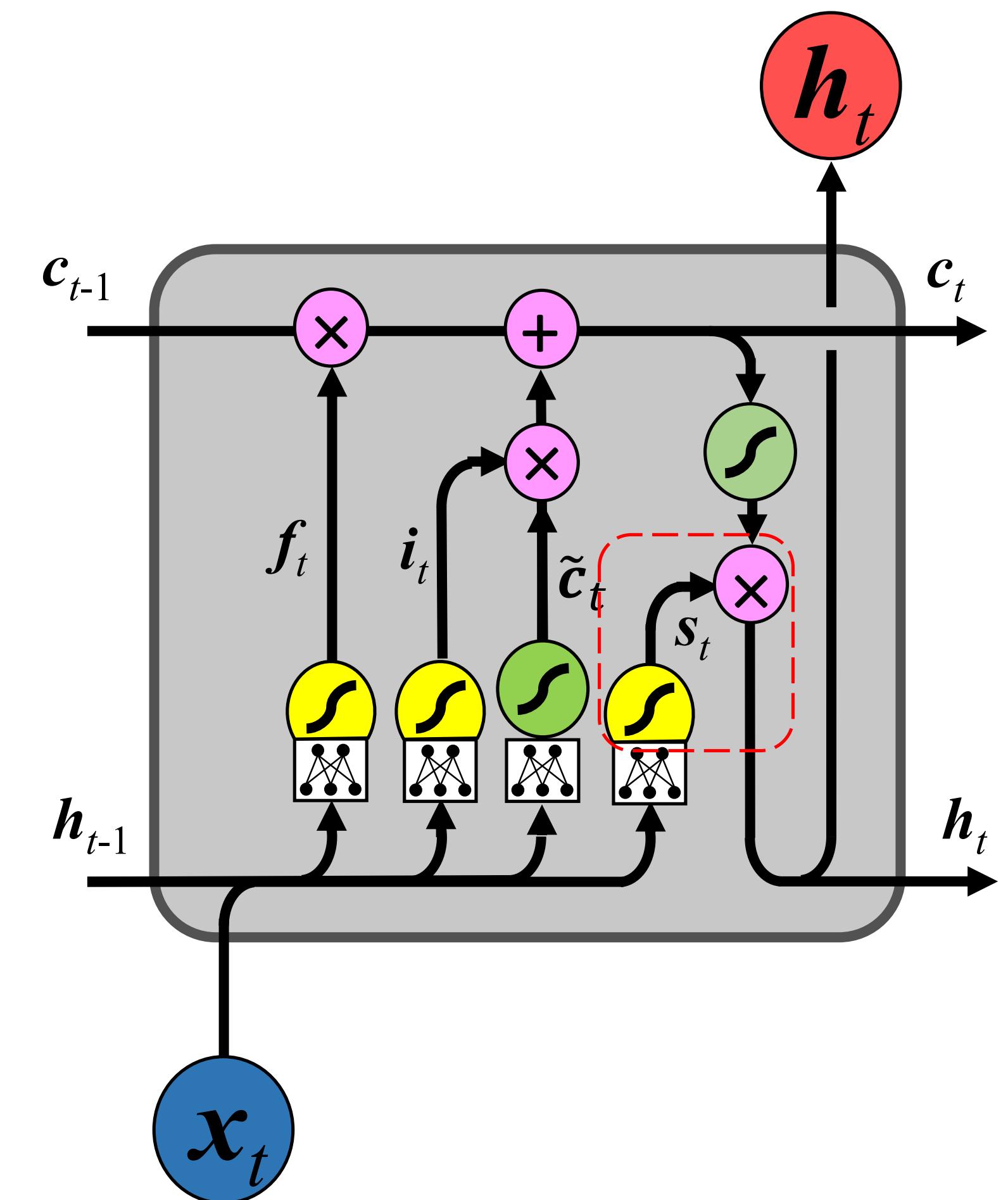
Output update:

- 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:

$$s_t = \sigma(W_s[h_{t-1}, x_t])$$

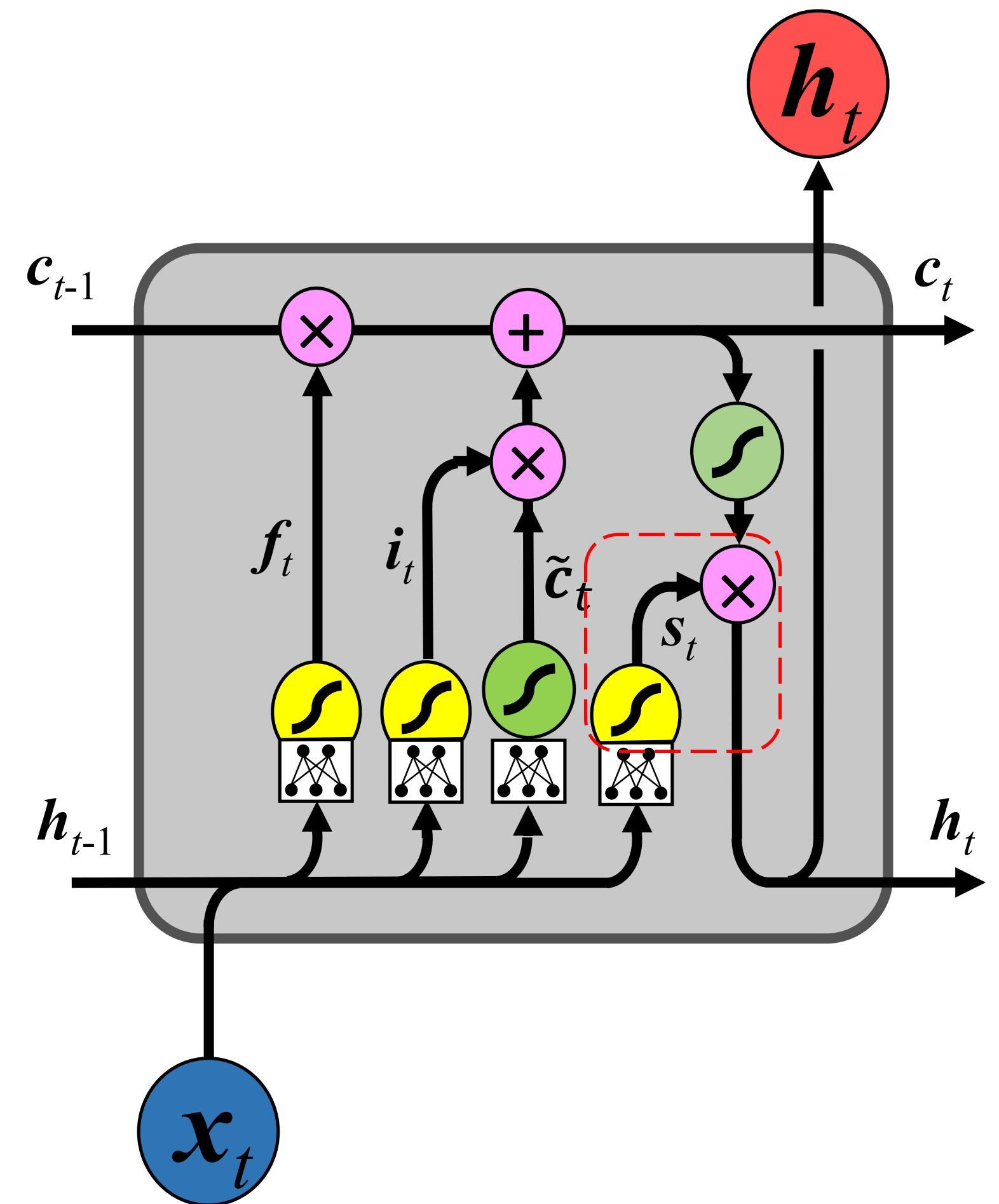


Walk through a LSTM

Output:

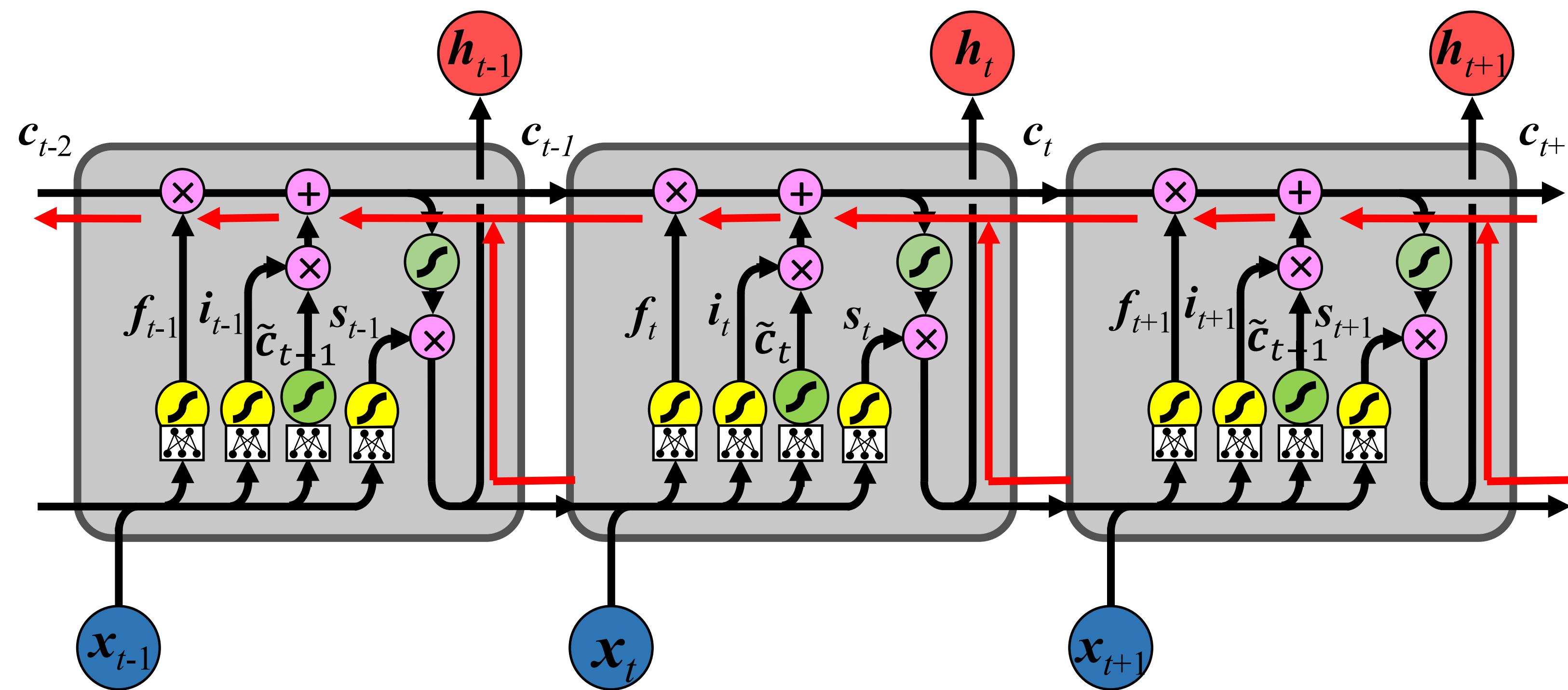
- The output will be a **filtered** version of the cell state.

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



Gradient Flow in LSTM

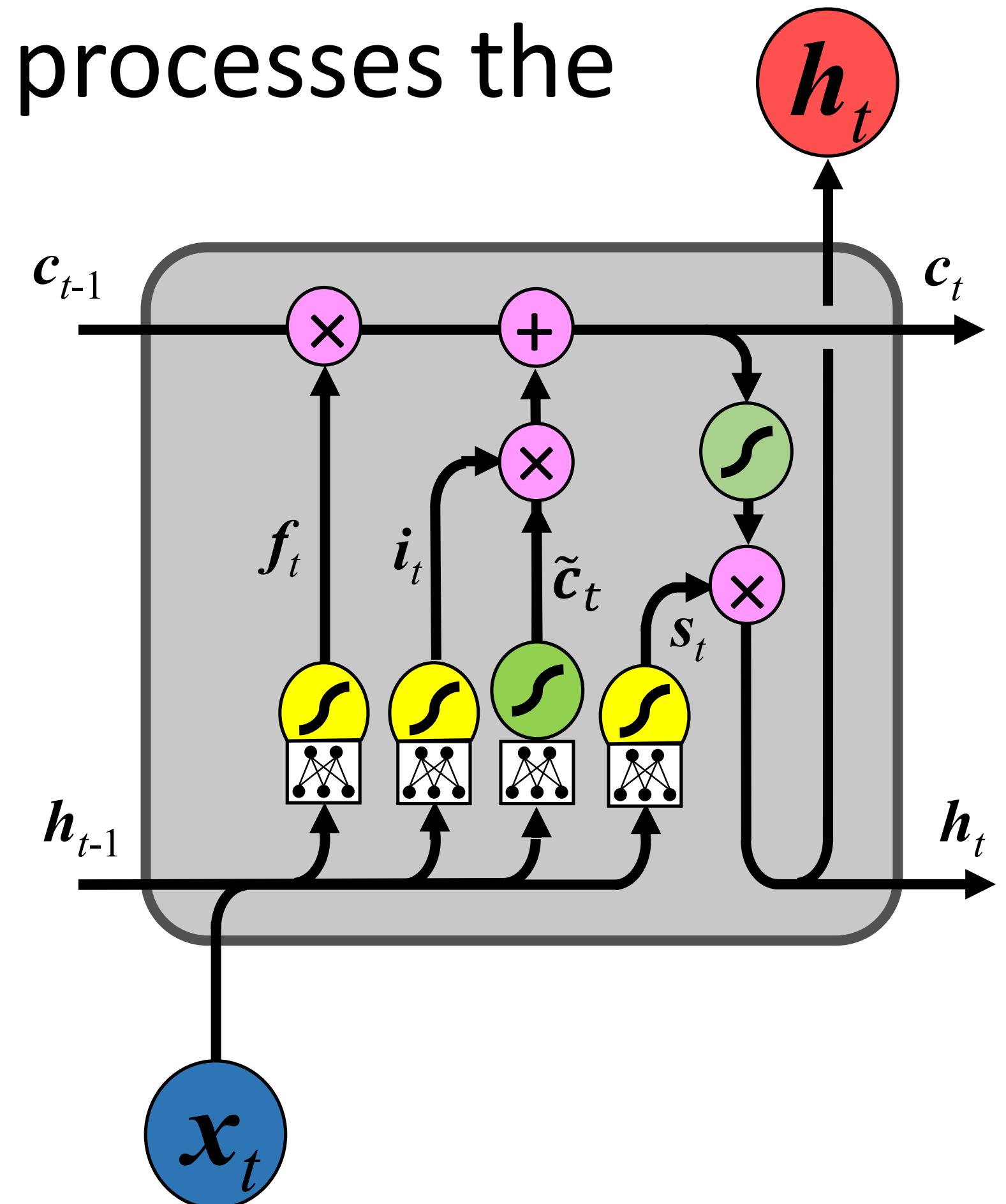
- Backpropagation from c_{t-1} to c_t : only elementwise multiplication by f_t , no matrix multiplication by W^T .
- Multiplied by a different f_t , at each time step.



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))
```



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 Ing

- After 700 iterations:

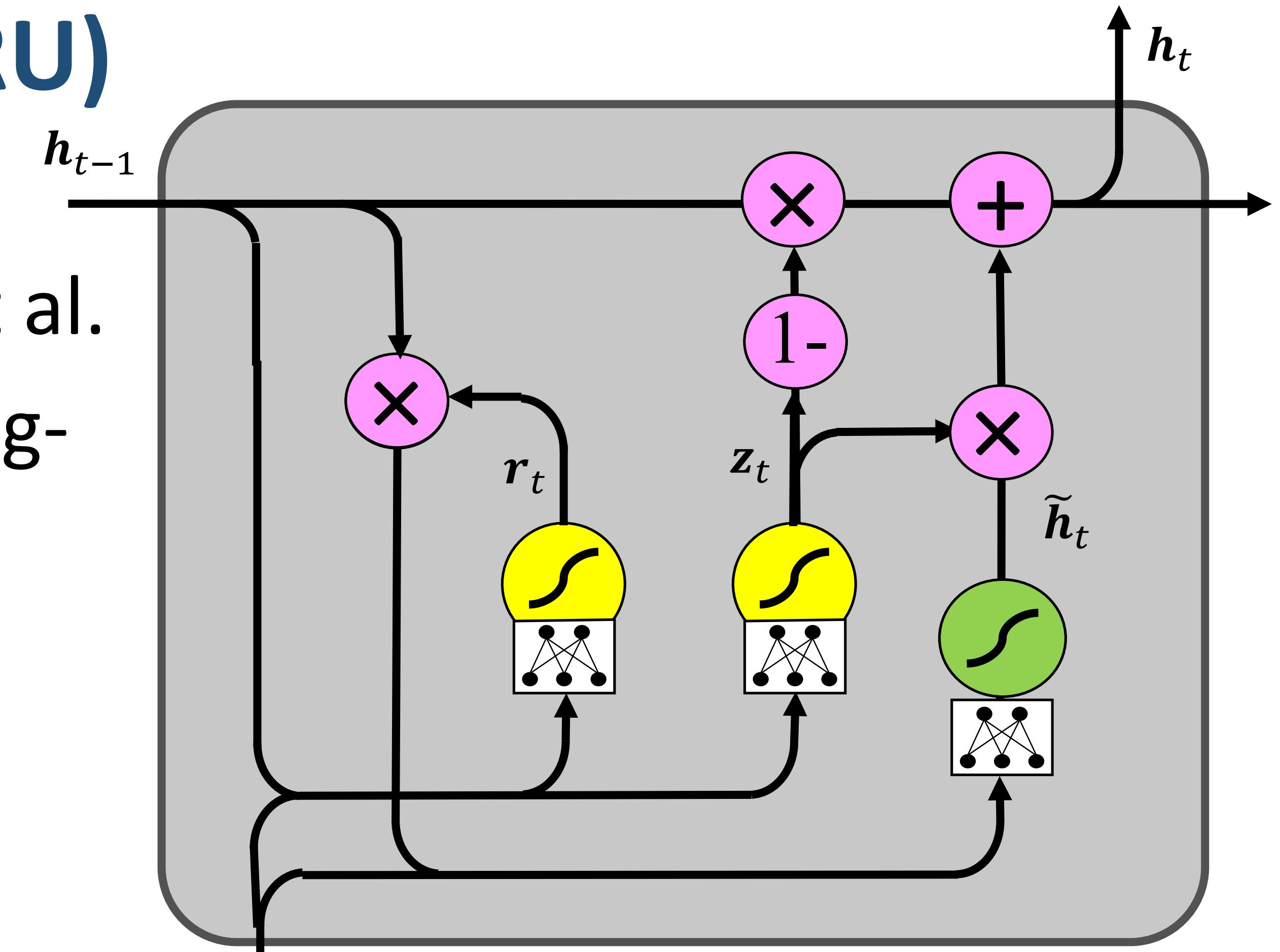
Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort how, and Gogition is so overelical and ofter.

- 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

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.



LSTM layer

pointwise operation

sigmoid layer

tanh layer

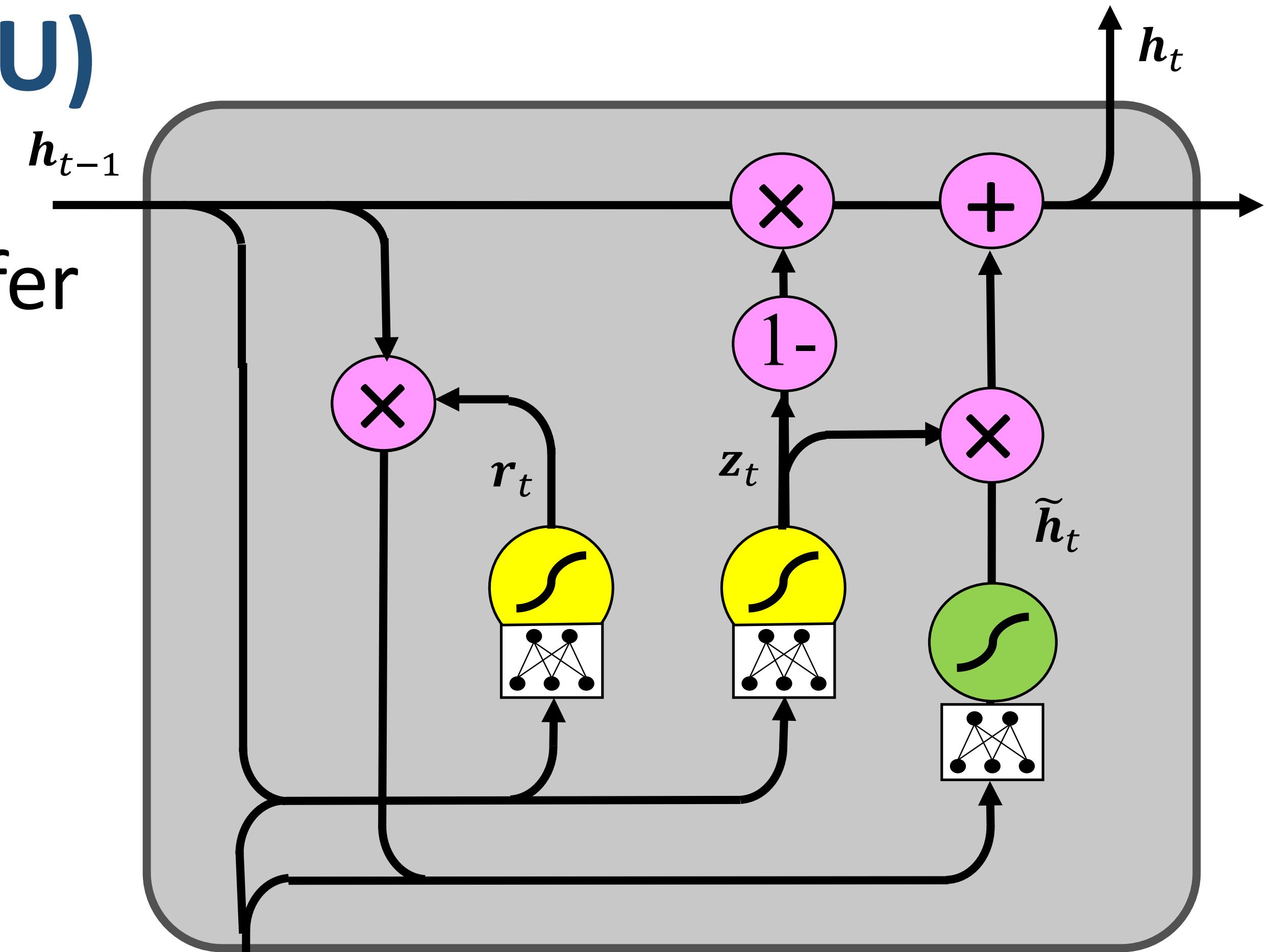
vector transfer

concatenate

copy

Gated Recurrent Unit (GRU)

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

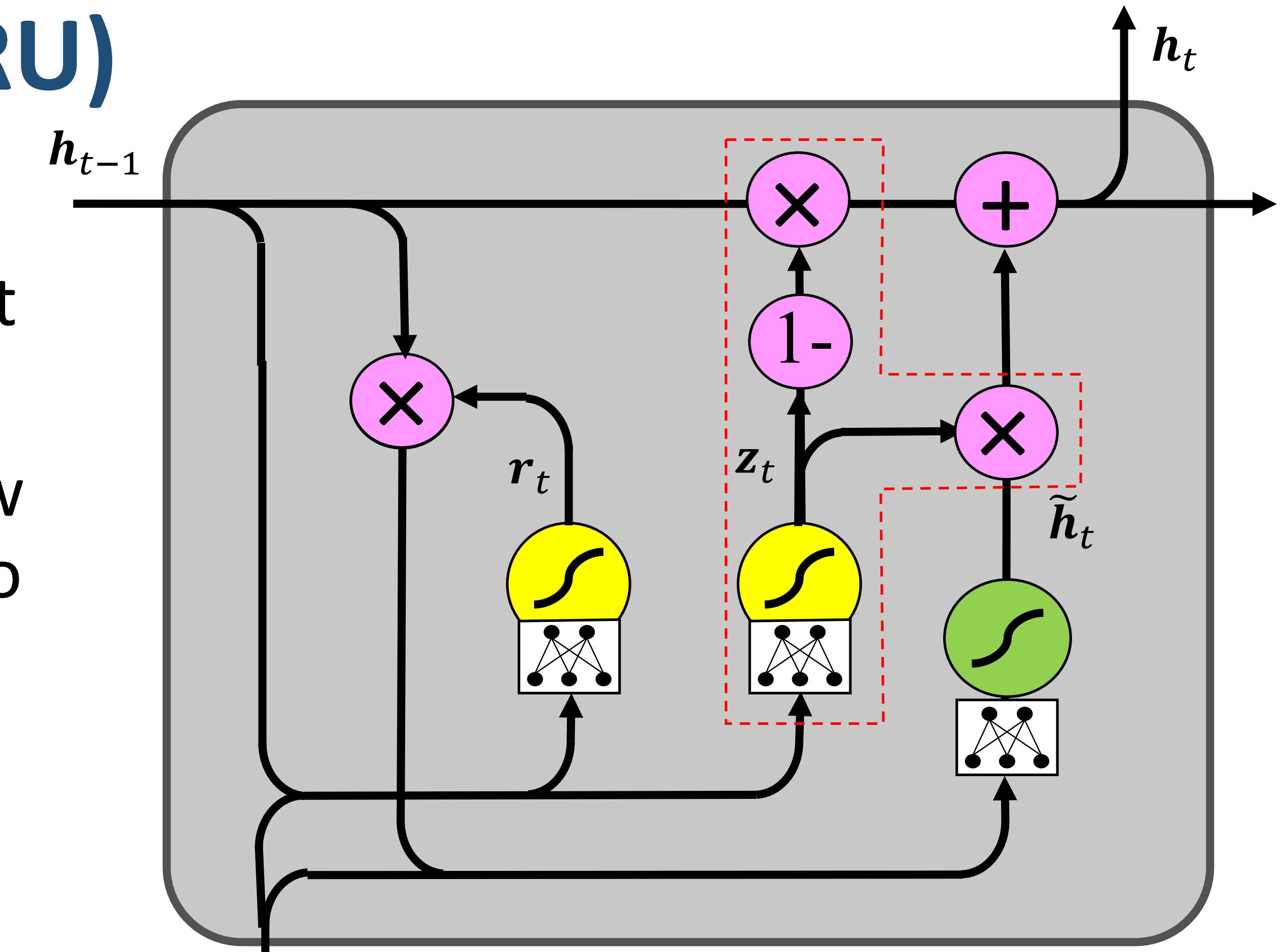


LSTM layer	pointwise operation	<i>sigmoid</i> layer	<i>tanh</i> layer	vector transfer	concatenate	copy

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.



LSTM layer

pointwise operation

sigmoid layer

tanh layer

vector transfer

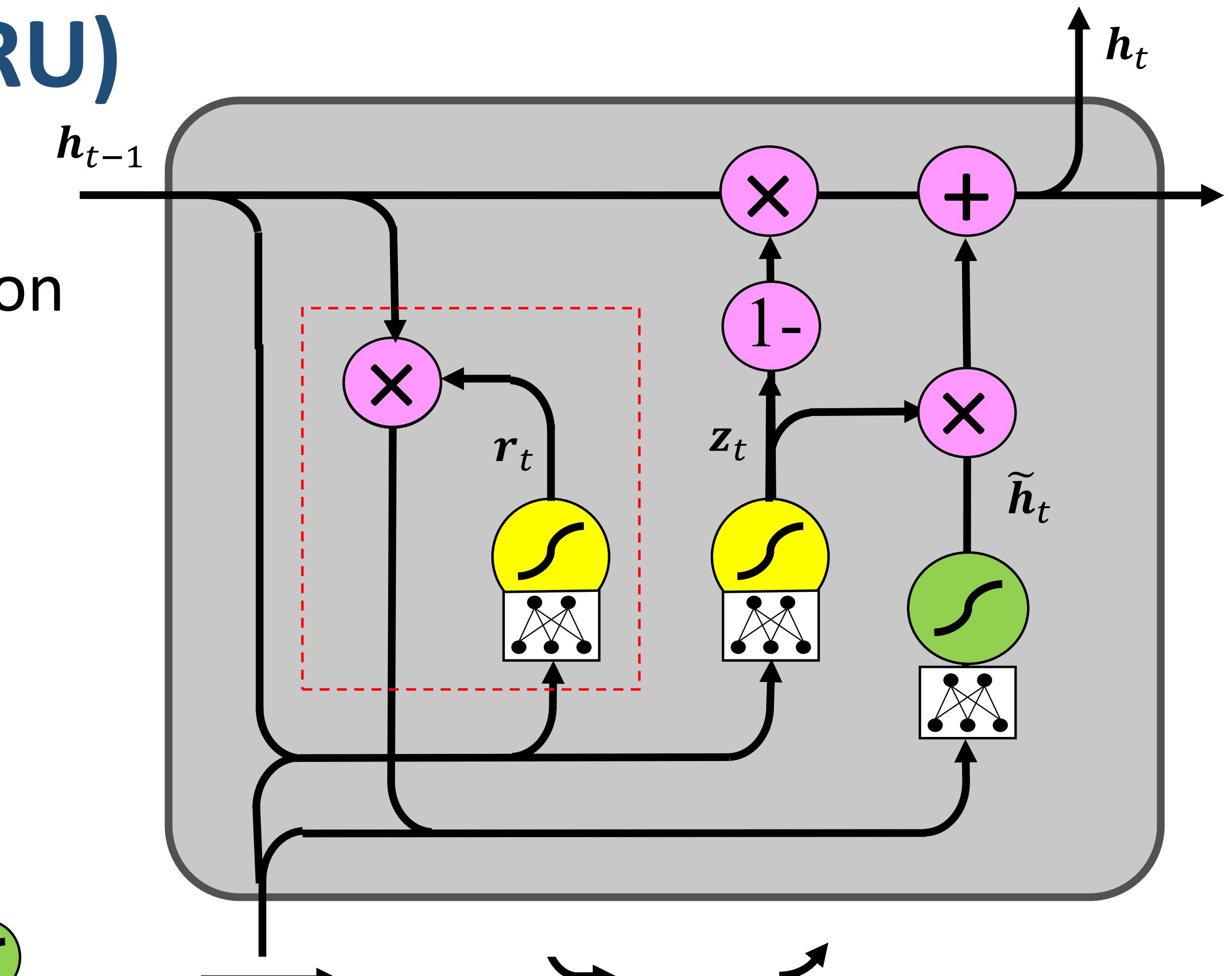
concatenate

copy

Gated Recurrent Unit (GRU)

Reset Gate:

- Decides how much past information to forget.



LSTM layer

pointwise operation

sigmoid layer

tanh layer

vector transfer

concatenate

copy

Gated Recurrent Unit (GRU)

$$z_t = \sigma(W_z x_t + W_z h_{t-1})$$

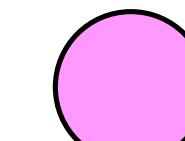
$$r_t = \sigma(W_r x_t + W_r h_{t-1})$$

$$\tilde{h}_t = \tanh(W_h x_t + r_t * W_h h_{t-1})$$

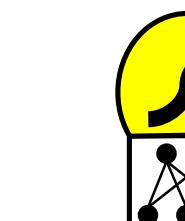
$$h_t = z_t * \tilde{h}_t + (1 - z_t) h_{t-1}$$



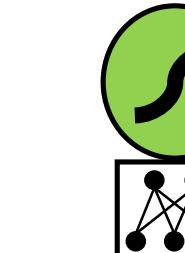
LSTM layer



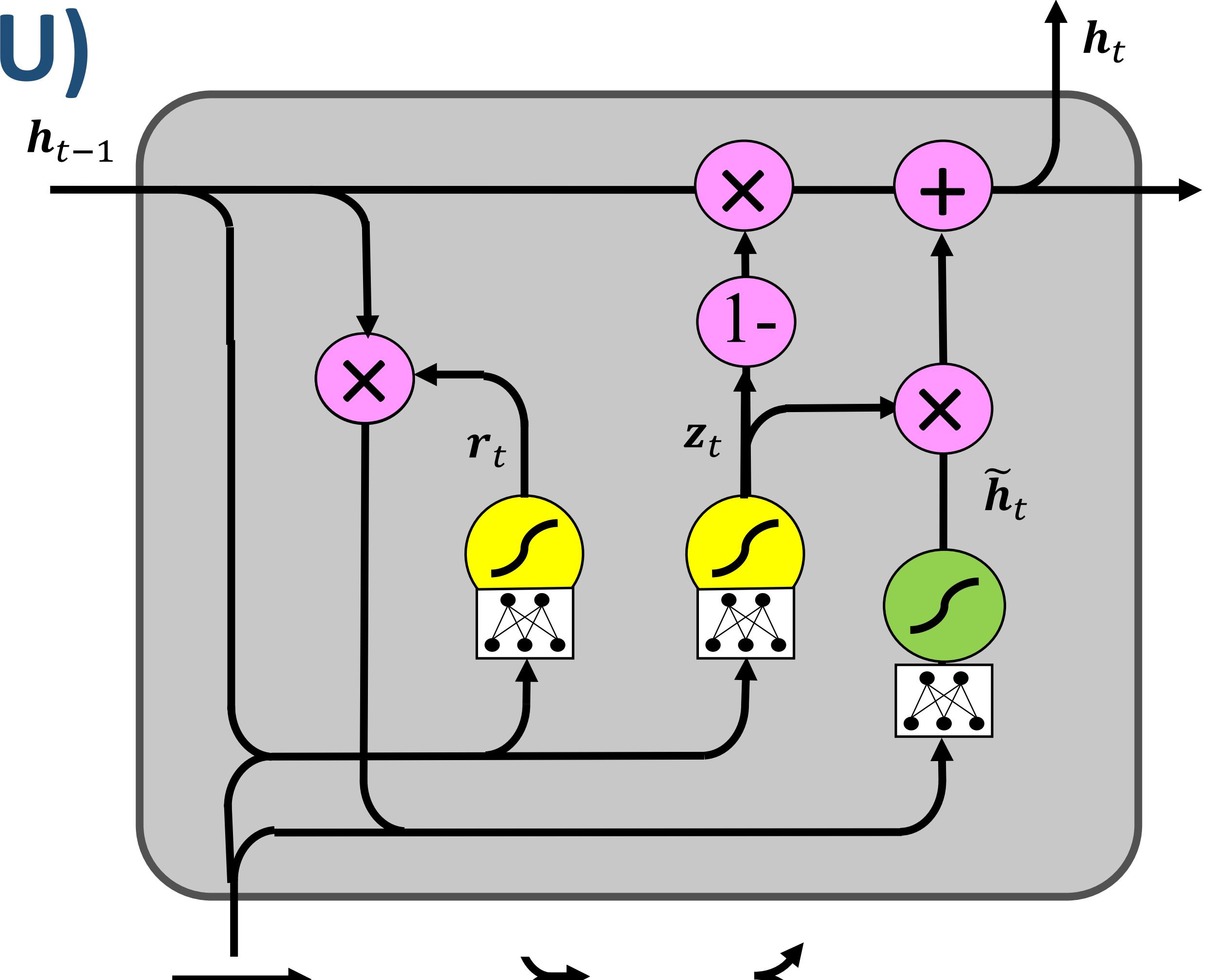
pointwise
operation



sigmoid
layer



tanh
layer



vector
transfer

concatenate

copy

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.

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.”

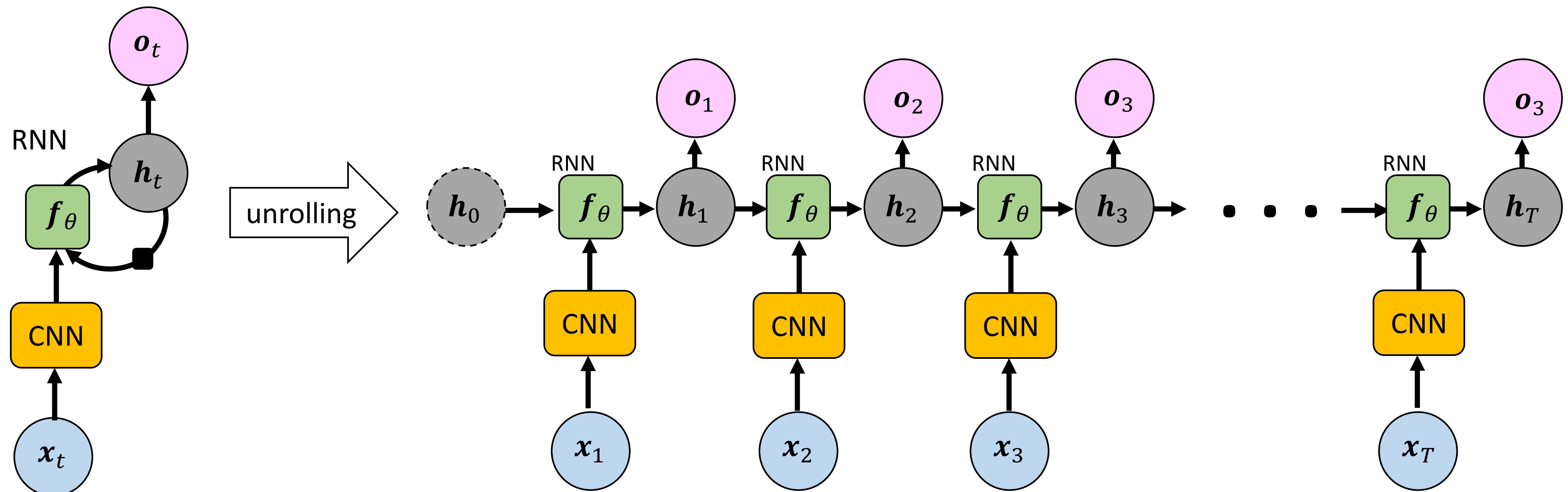
Jozefowicz et al., 2015, *An Empirical Exploration of Recurrent Network Architectures.*

Recurrent Neural Network

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

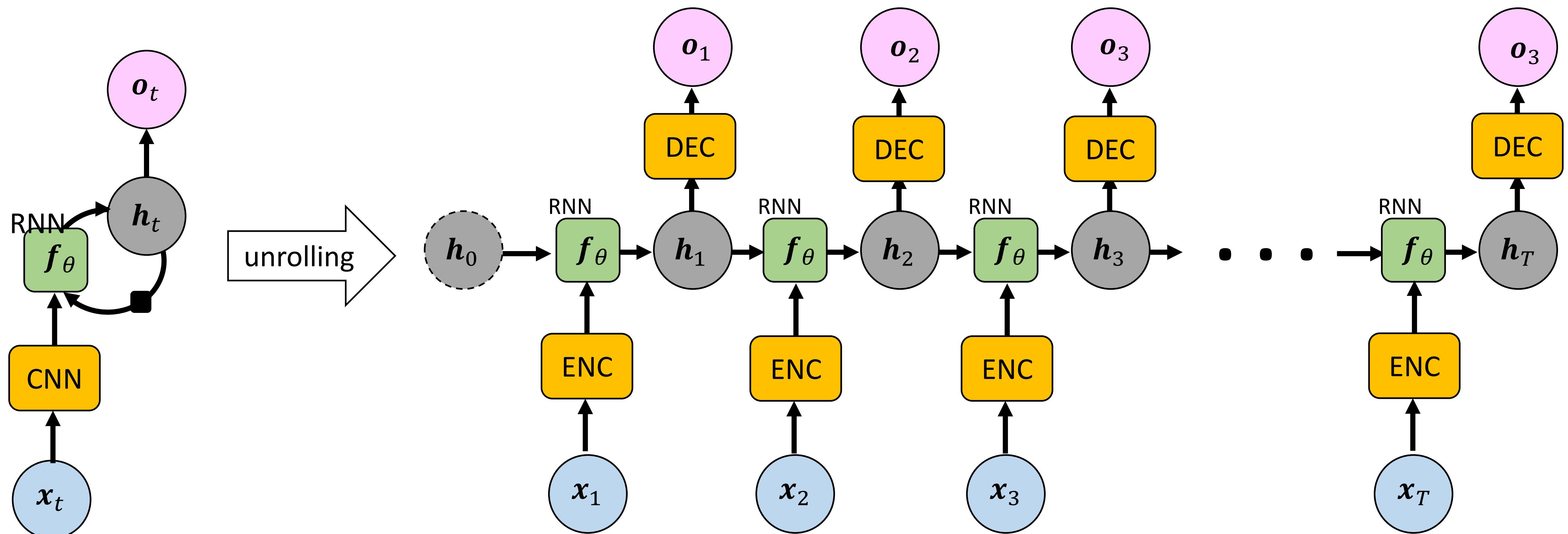
Combining CNN and RNN

A CNN learns data representation to the RNN input.

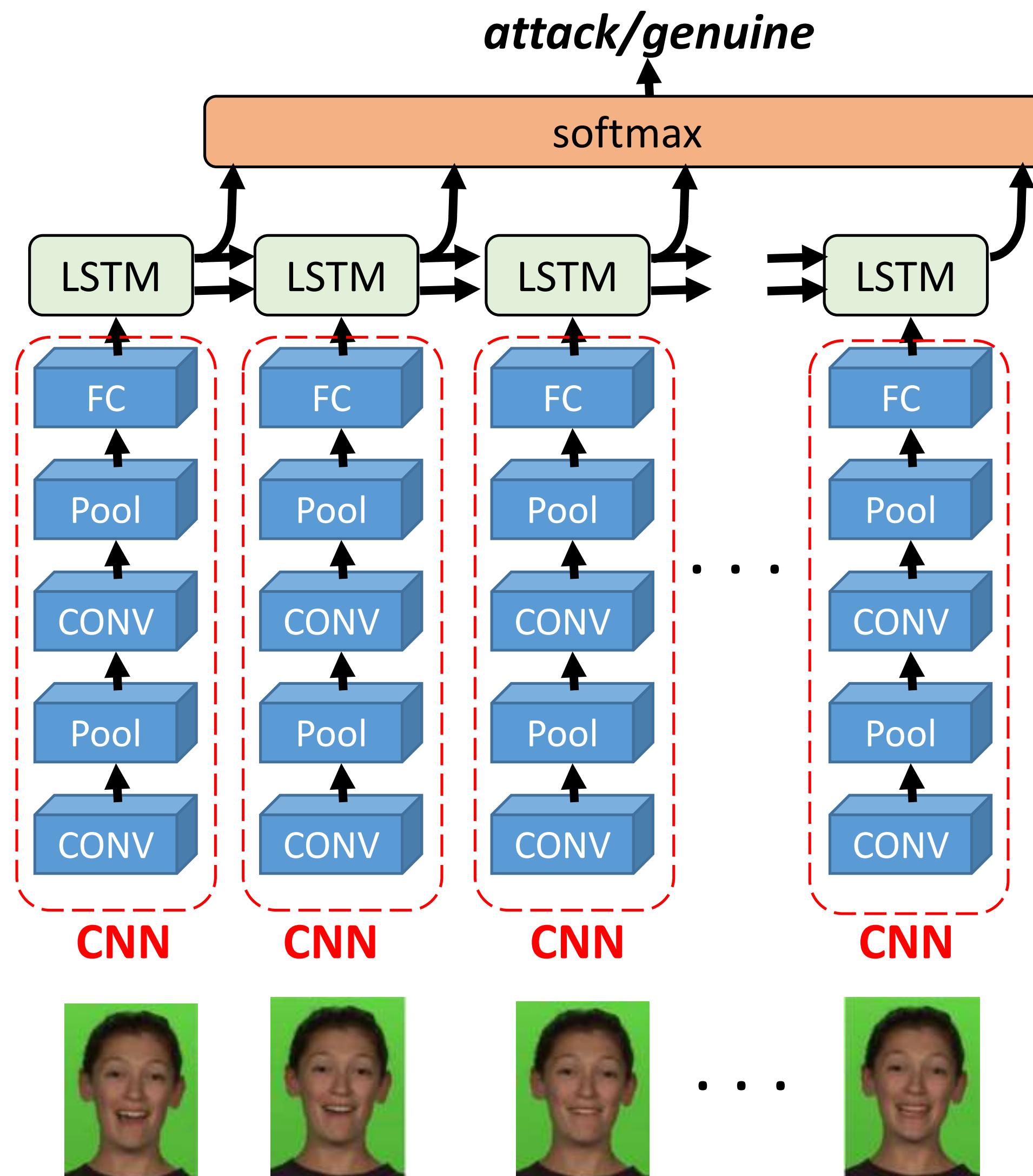


Combining CNN and RNN

- 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*.

Bidirectional RNN (BRNN)

A positive or negative remark?

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

What kind of crop?

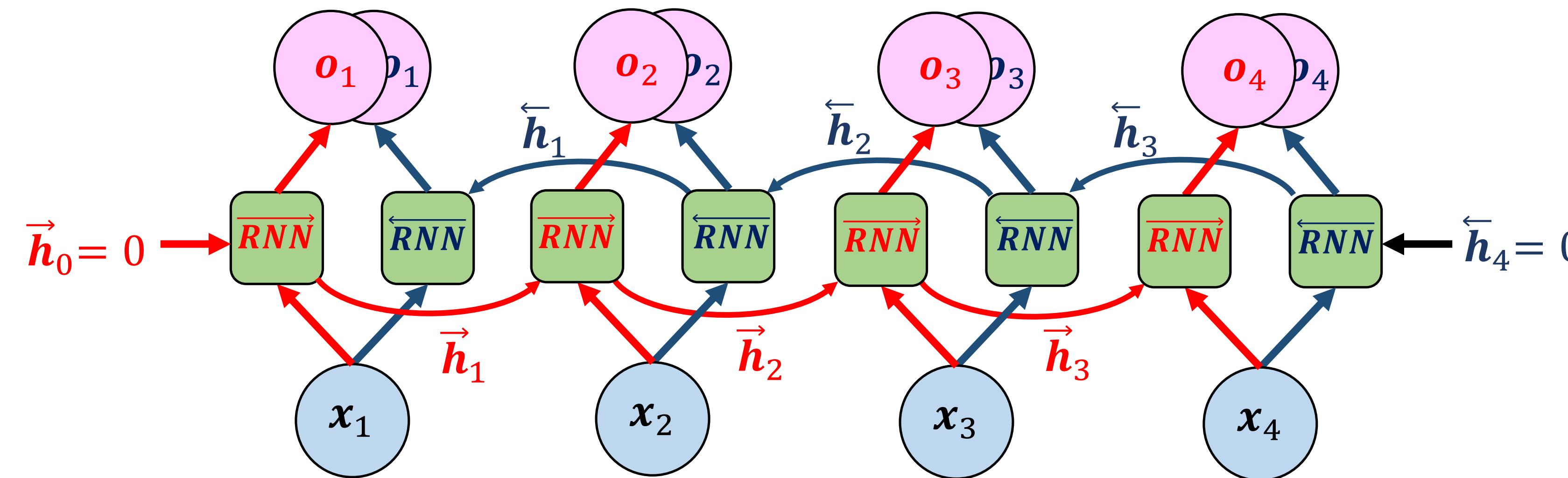


Bidirectional RNN (BRNN)

- 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)

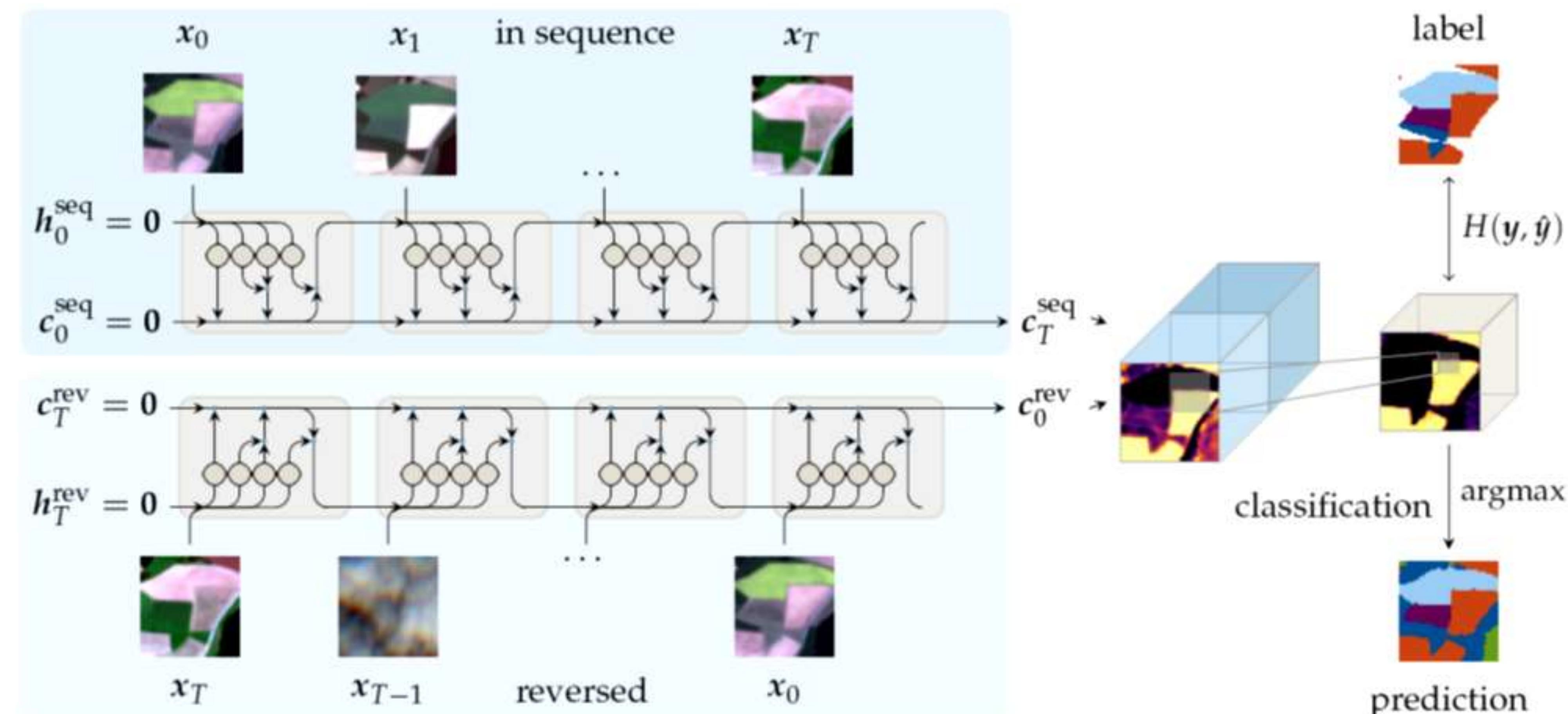
- \overrightarrow{RNN} and \overleftarrow{RNN} share the same parameters.



- 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



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 Mask R-CNN**

See you next class!

