What can('t) we do we ConvNets?

Recurrent nets, Memory and attention

Karel Zimmermann
Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

Prerequisites: derivative of compound function

Introduce notation: $\dfrac{\partial f(a,b)}{\partial a} = \partial_0 f(a,b), \quad \dfrac{\partial f(a,b)}{\partial b} = \partial_1 f(a,b)$

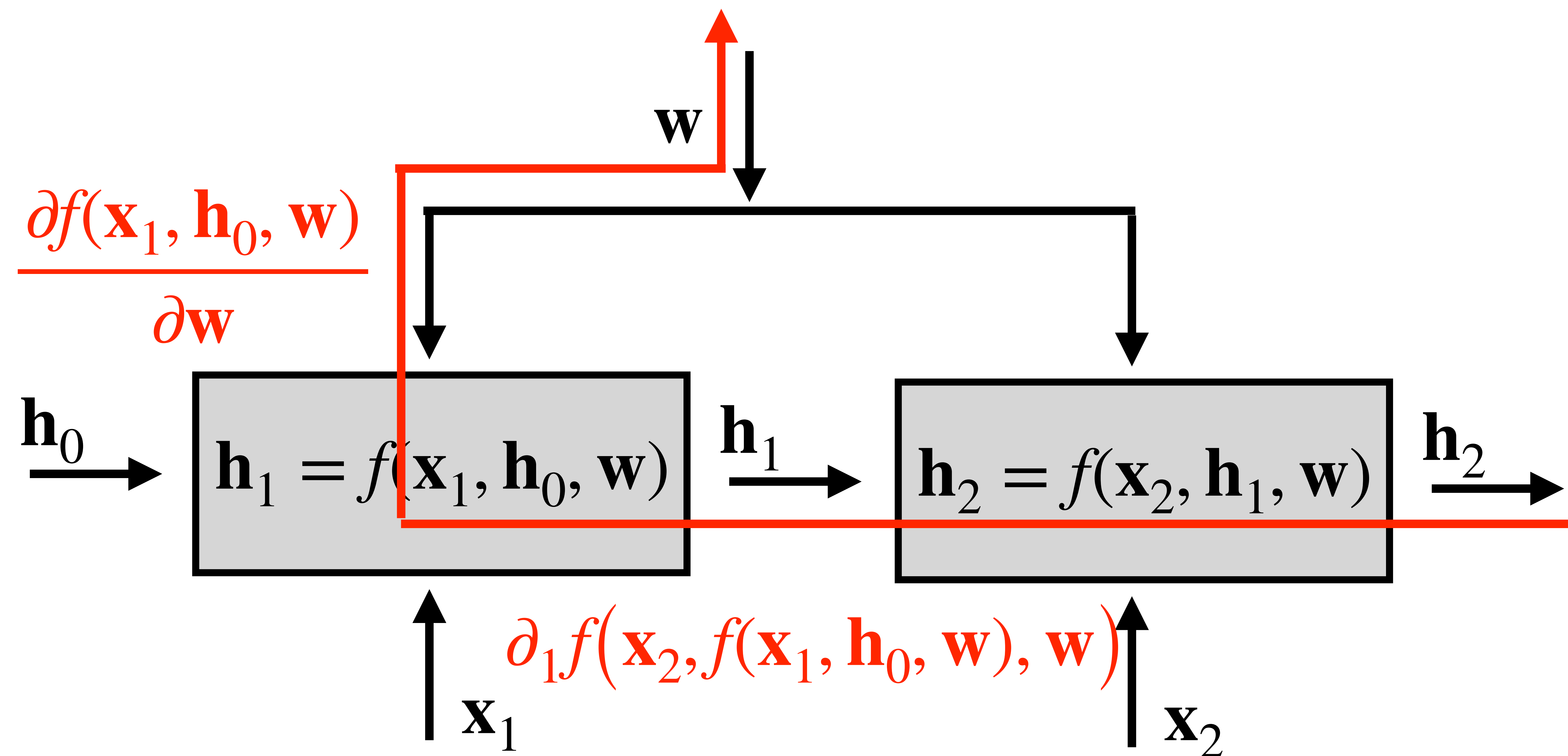
$$f(a,b) = a^2 - b^2 \;, \quad g(x) = \sin(x) \quad \Rightarrow \quad f\big(x, g(x)\big) = x^2 - \sin^2(x)$$

$$\frac{\partial f\big(x, g(x)\big)}{\partial x} = \partial_0 f\big(x, g(x)\big) + \partial_1 f\big(x, g(x)\big)\frac{\partial g(x)}{\partial x} = 2x - 2\sin(x)\cos(x)$$

Prerequisites: derivative of compound function

$$\mathbf{h}_1 = f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \quad \mathbf{h}_2 = f(\mathbf{x}_2, \mathbf{h}_1, \mathbf{w}) \quad \Rightarrow \quad \mathbf{h}_2 = f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big)$$

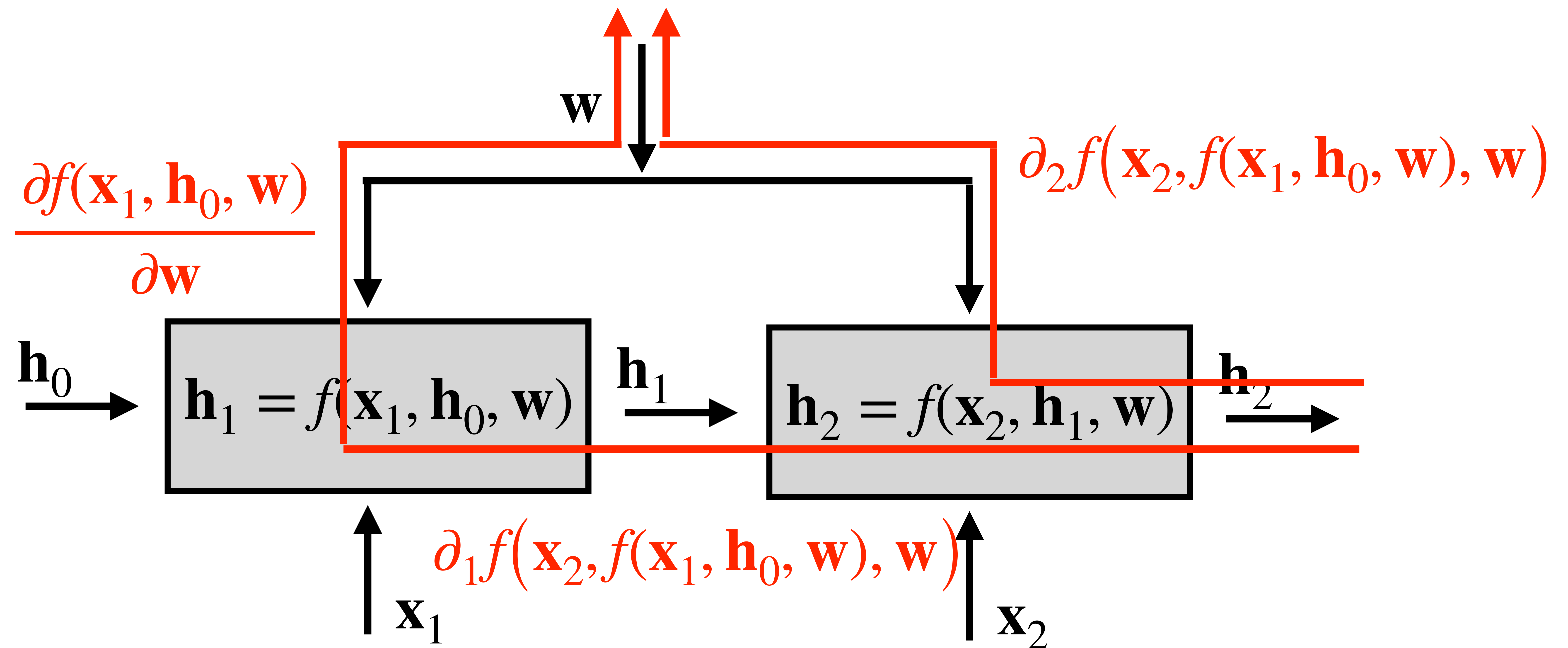
$$\frac{\partial f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big)}{\partial \mathbf{w}} = \partial_1 f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big) \frac{\partial f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w})}{\partial \mathbf{w}}$$



Prerequisites: derivative of compound function

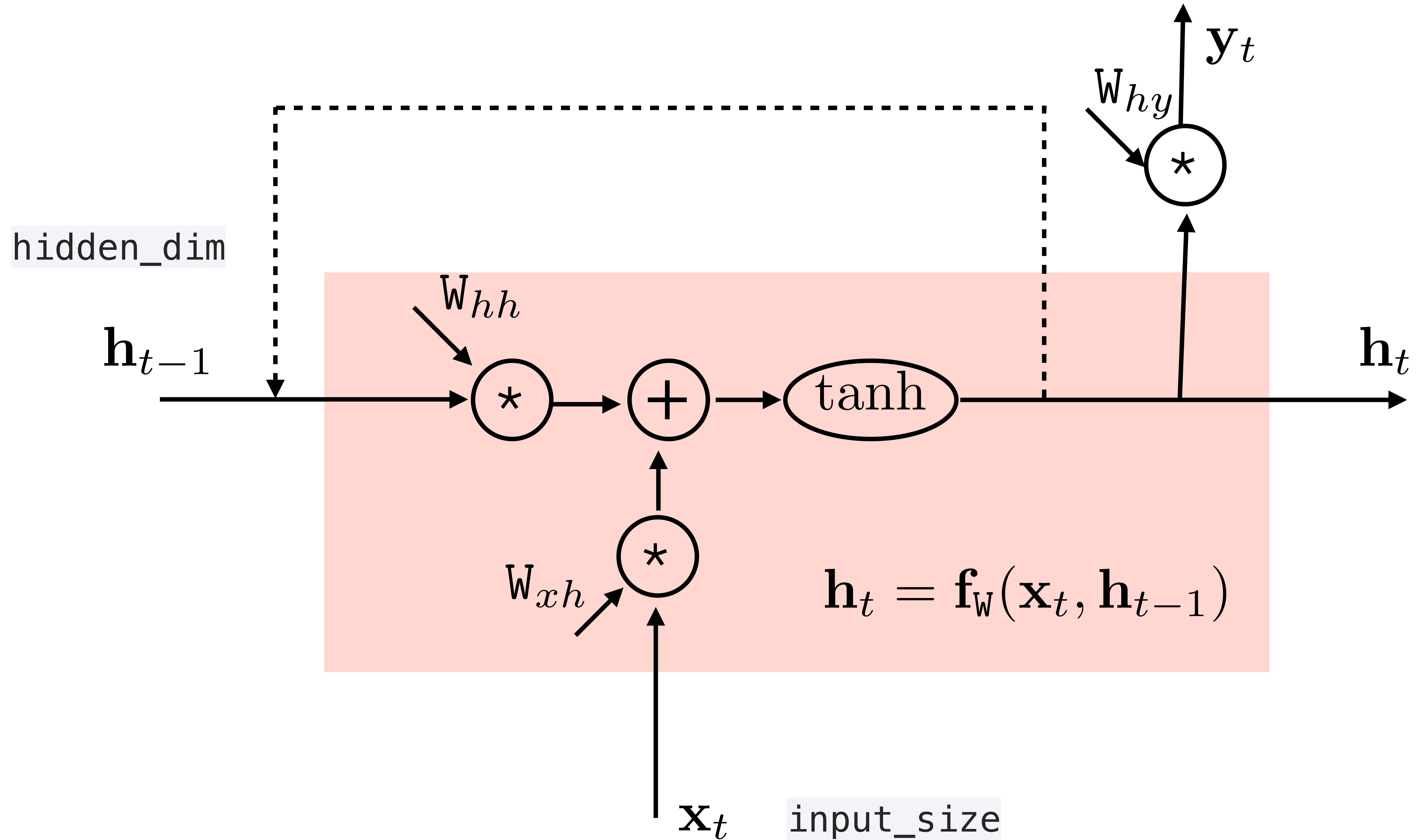
$$\mathbf{h}_1 = f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \quad \mathbf{h}_2 = f(\mathbf{x}_2, \mathbf{h}_1, \mathbf{w}) \quad \Rightarrow \quad \mathbf{h}_2 = f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big)$$

$$\frac{\partial f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big)}{\partial \mathbf{w}} = \partial_1 f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big) \frac{\partial f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w})}{\partial \mathbf{w}} + \partial_2 f\big(\mathbf{x}_2, f(\mathbf{x}_1, \mathbf{h}_0, \mathbf{w}), \mathbf{w}\big)$$



Simple recurrent block

```
torch.nn.RNN(input_size, hidden_dim, n_layers)
```



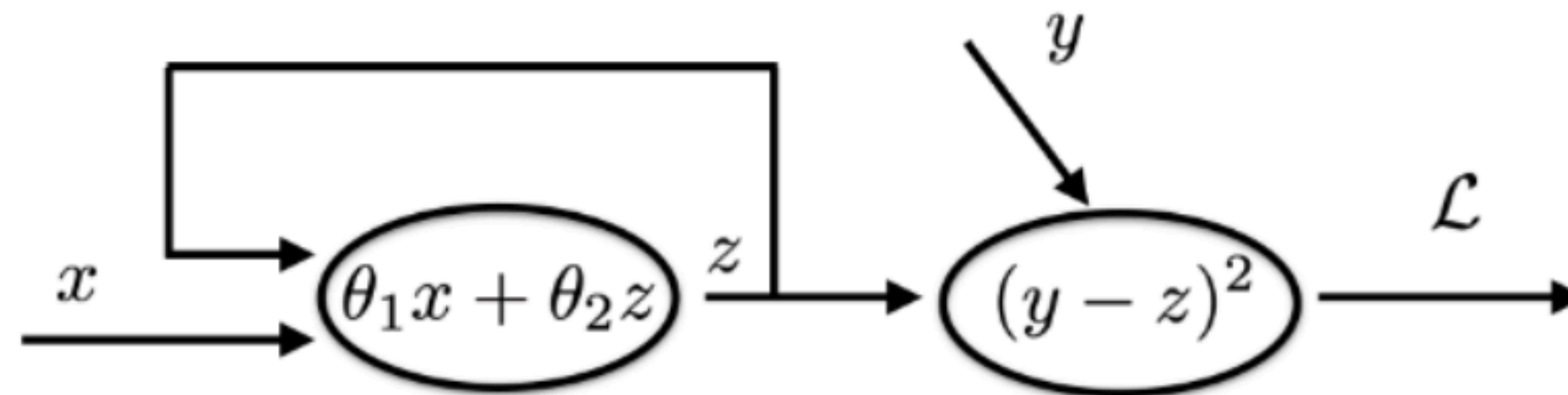
PyTorch: https://pytorch.org/docs/stable/nn.html

RNN example with backprop

Consider linear recurrent neural network with L2 loss depicted on the image below. The network is initialized with parameters $\theta_1 = 1, \theta_2 = 0, z_0 = 0$. You are given the following training sequence:

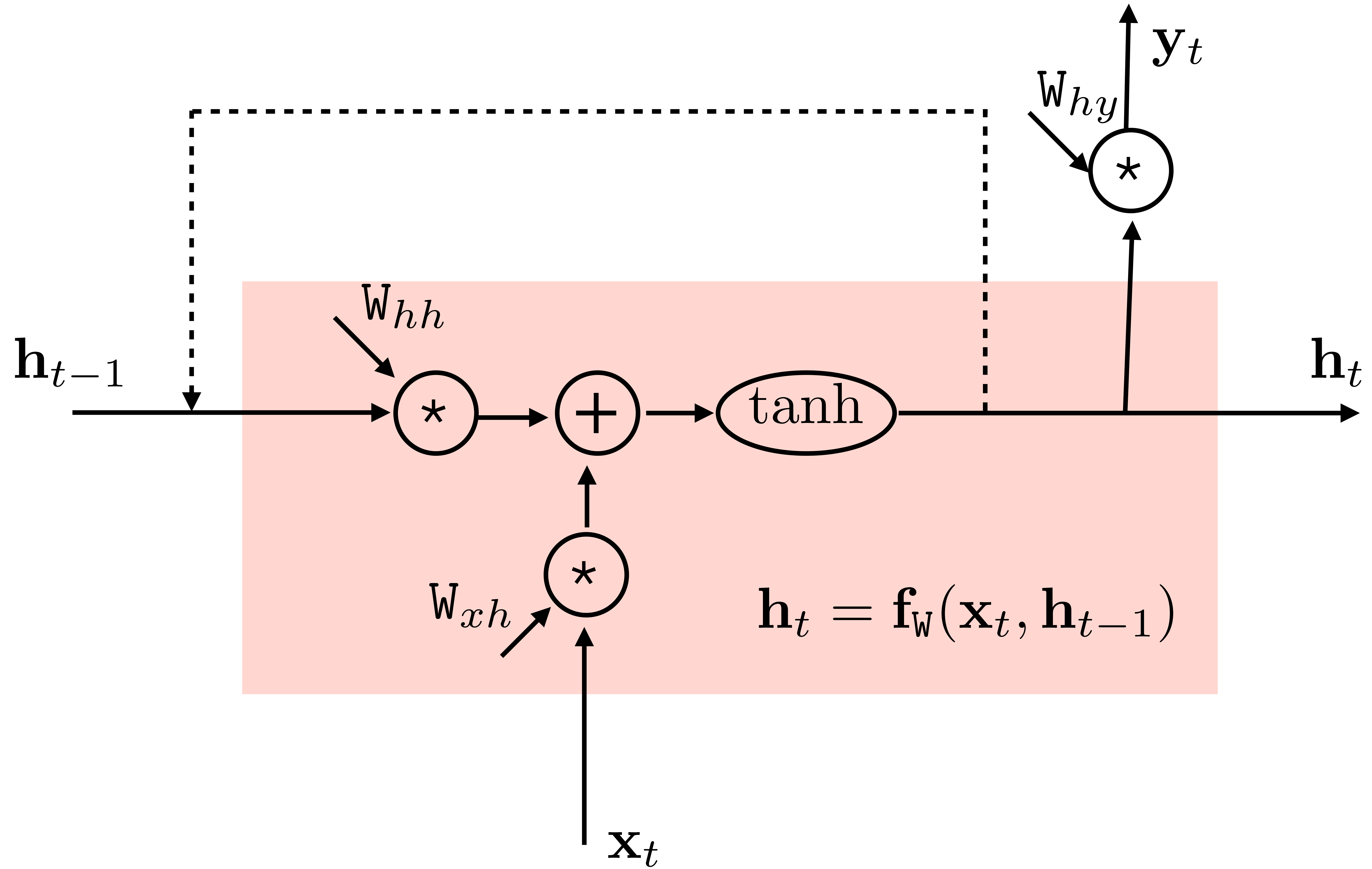
time=1	time=2
$x_1 = 0$	$x_2 = 1$
$y_1 = 1$	$y_2 = 3$

Estimate gradient of the overall loss (computed over all available outputs y_i for both available times $i = 1, 2$) with respect to θ_1.

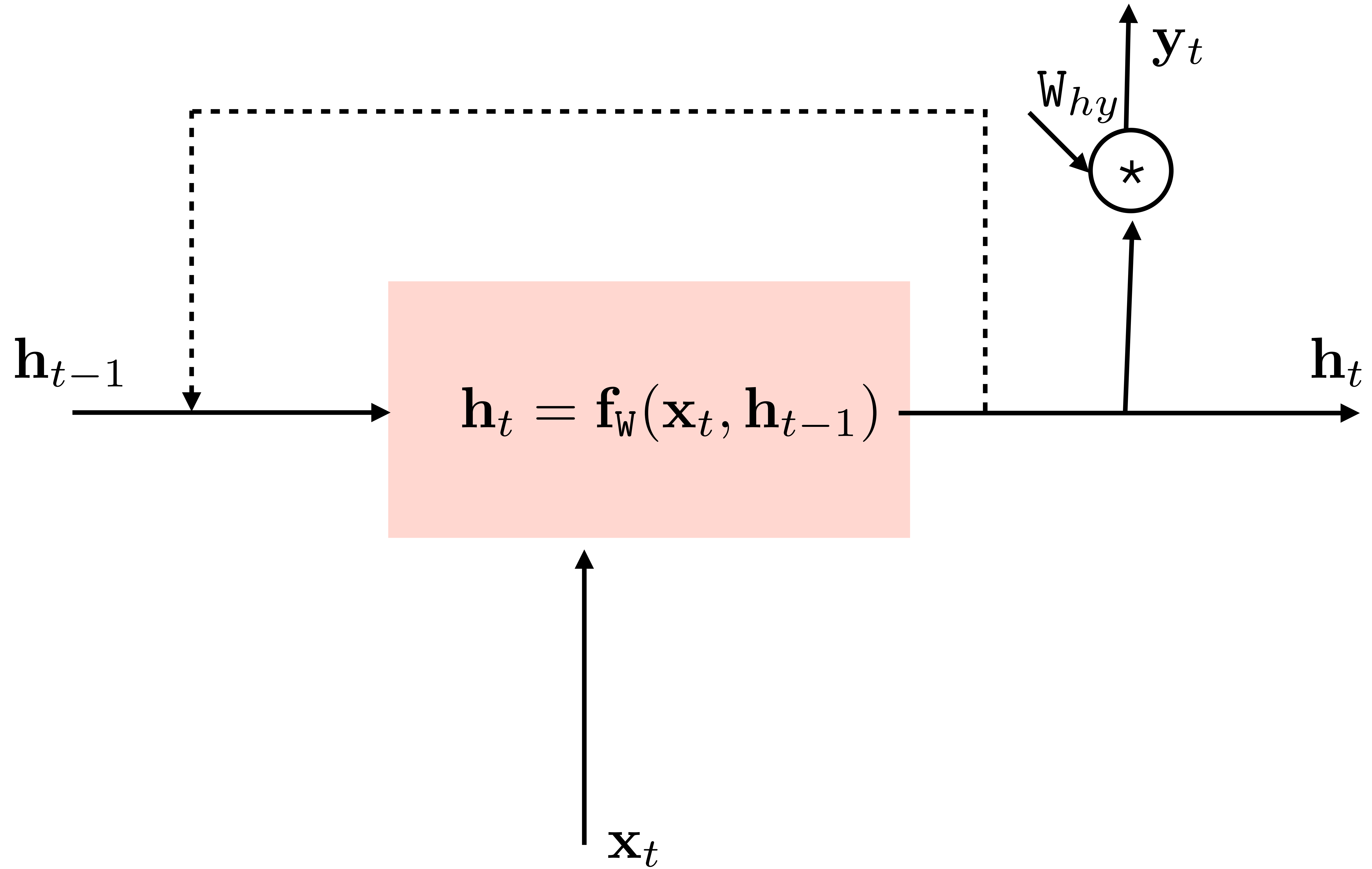


Hint: Unroll the network in time, to obtain a usual feedforward network with two loss nodes. Do the backpropagation as usual.

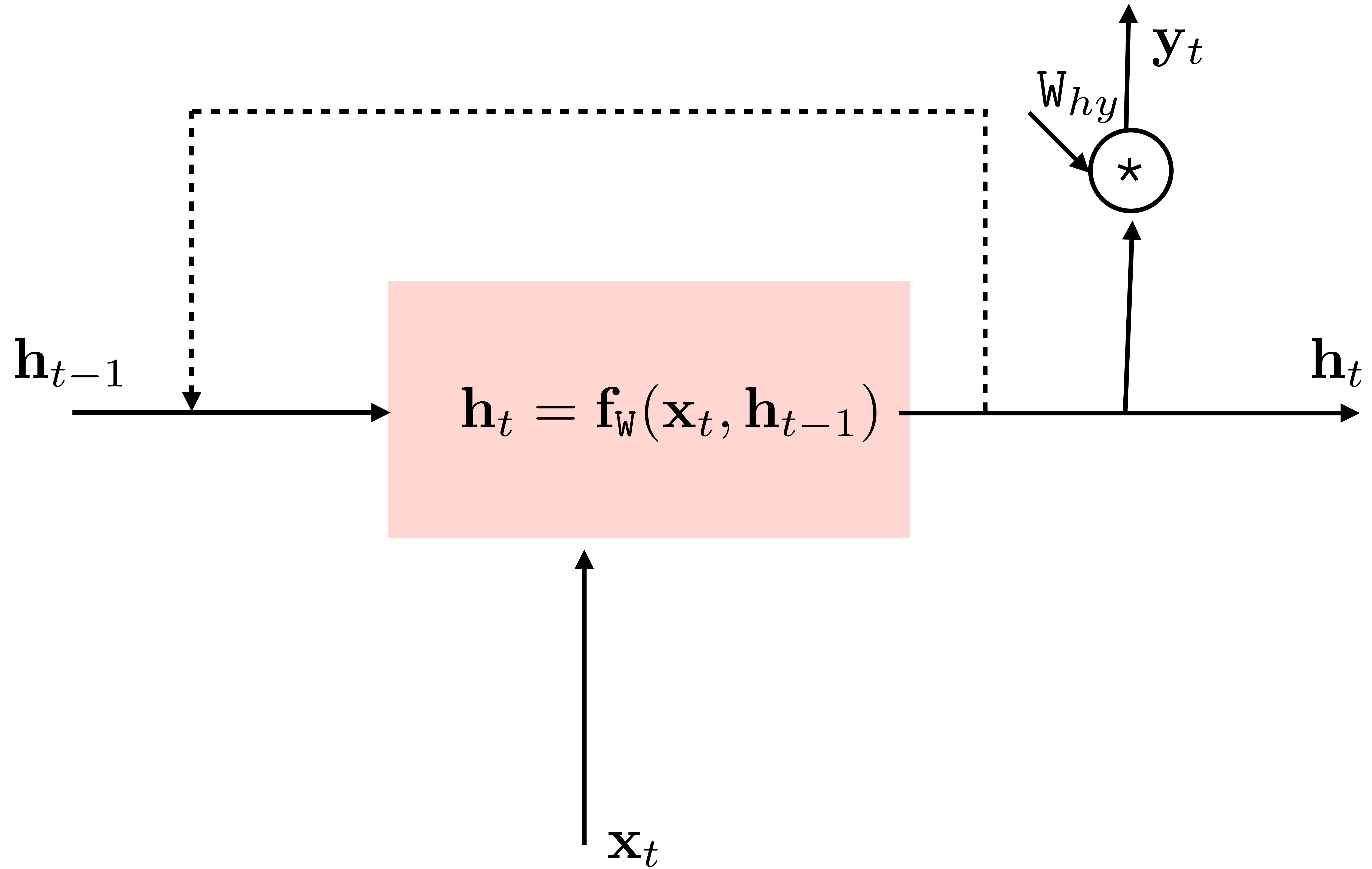
Simple recurrent block - feed-forward pass



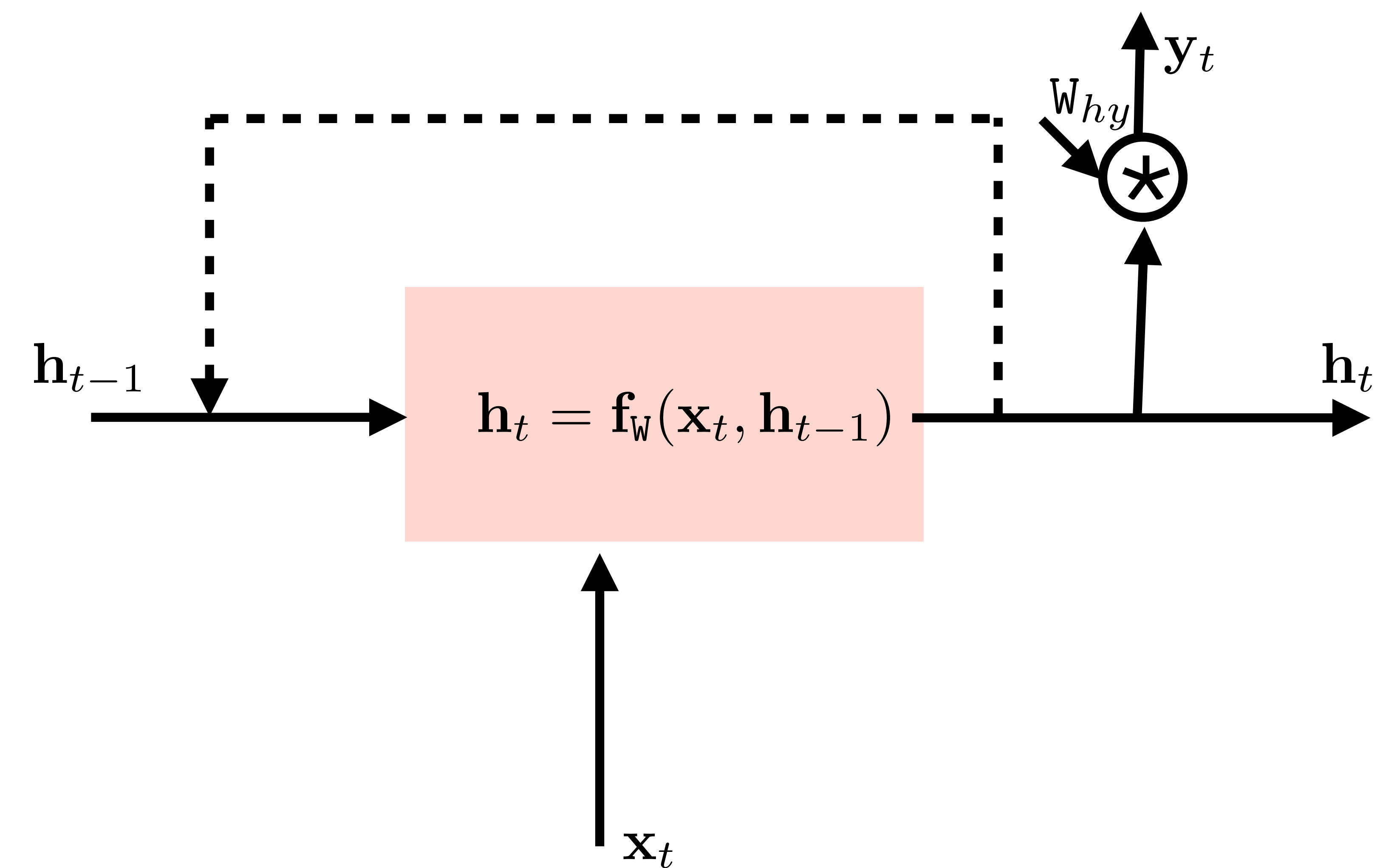
Simple recurrent block - feed-forward pass



Simple recurrent block - feed-forward pass



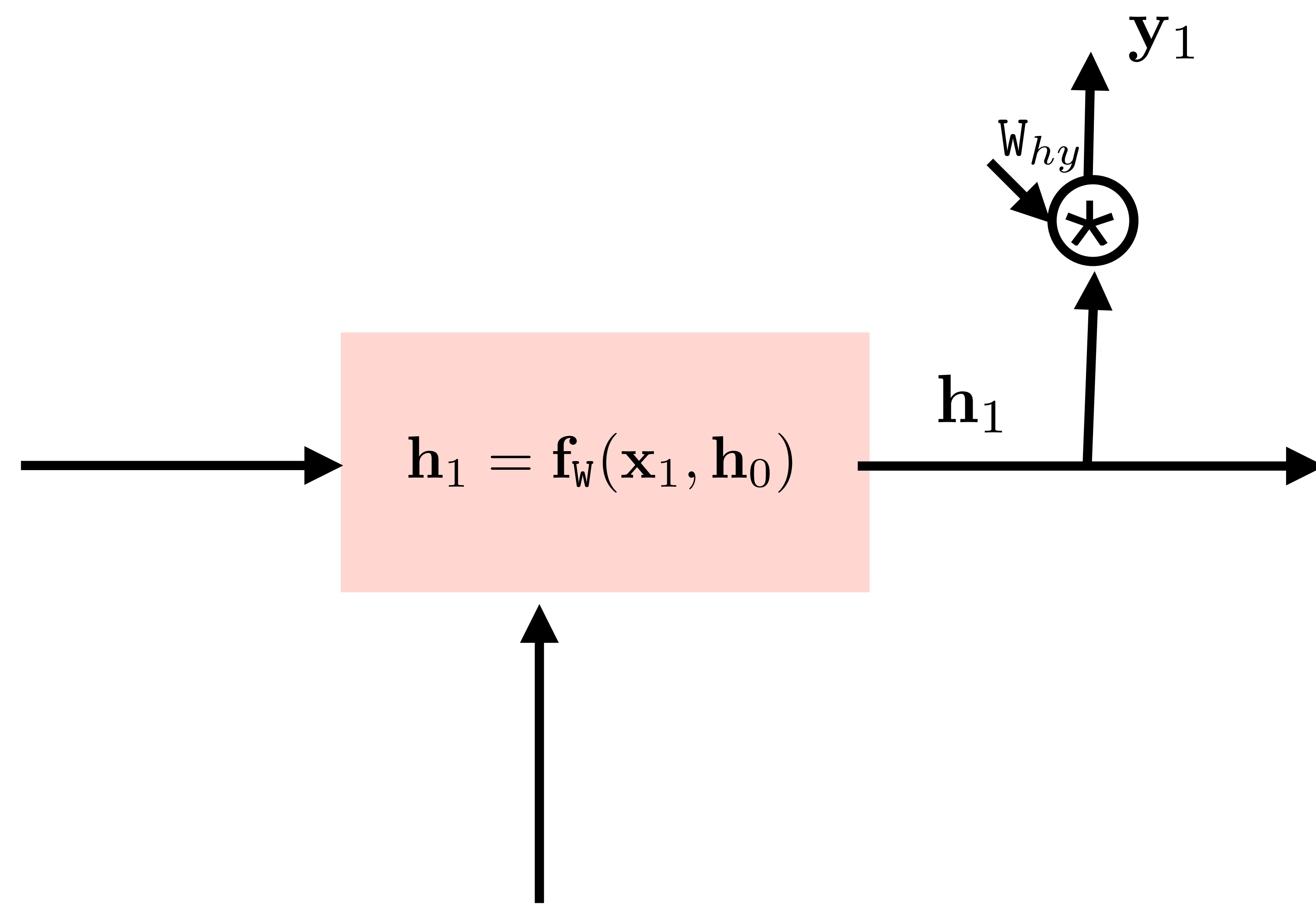
Simple recurrent block - feed-forward pass



Given a finite input sequence: $\mathbf{h}_0 \ \mathbf{x}_1 \ \mathbf{x}_2 \ \mathbf{x}_3$
we remove the recurrent connection by:

- successive substitution of inputs and
- unrolling the net

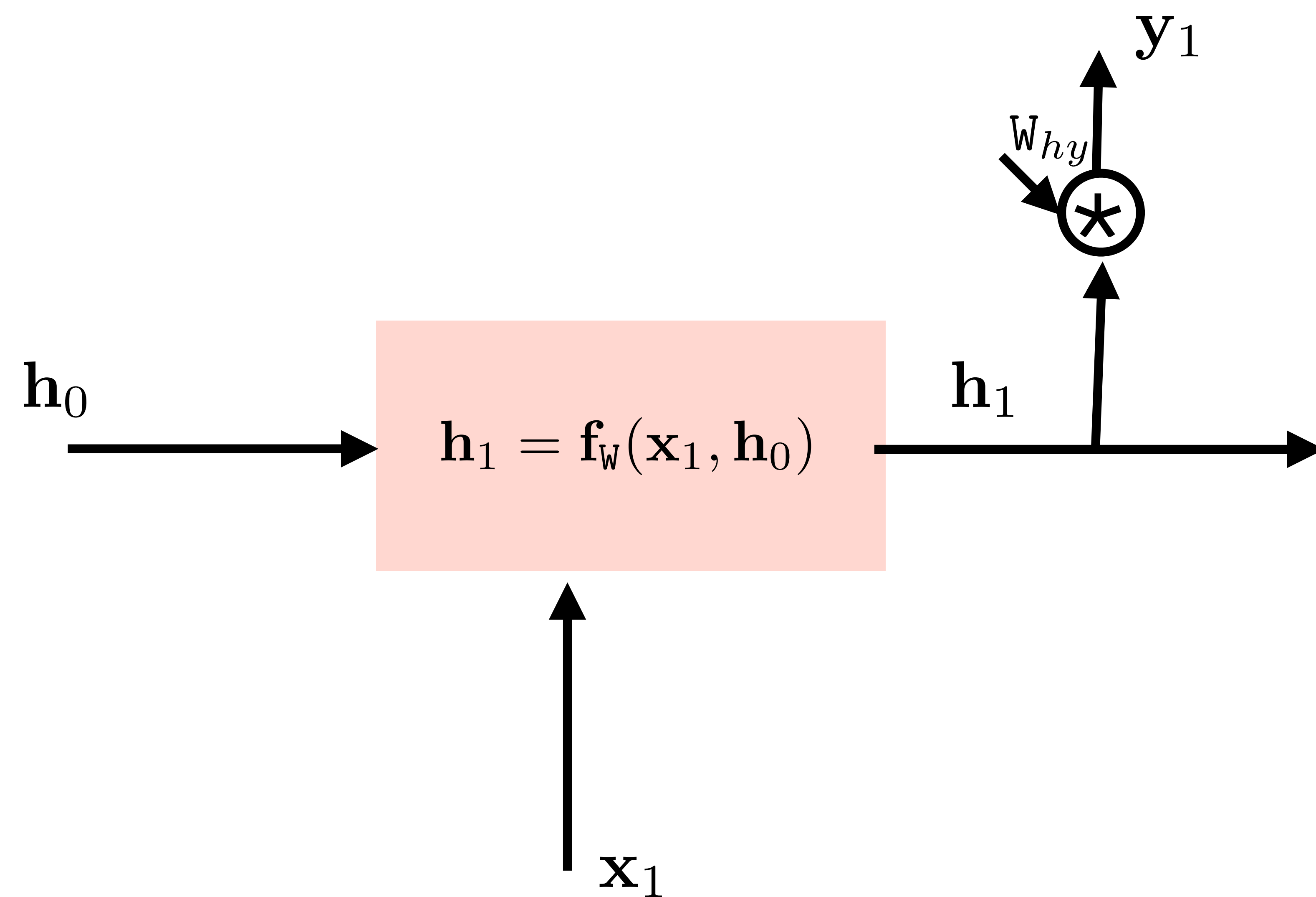
Simple recurrent block - feed-forward pass



Given a finite input sequence: $\mathbf{h}_0 \ \mathbf{x}_1 \ \mathbf{x}_2 \ \mathbf{x}_3$
we remove the recurrent connection by:

- successive substitution of inputs and
- unrolling the net

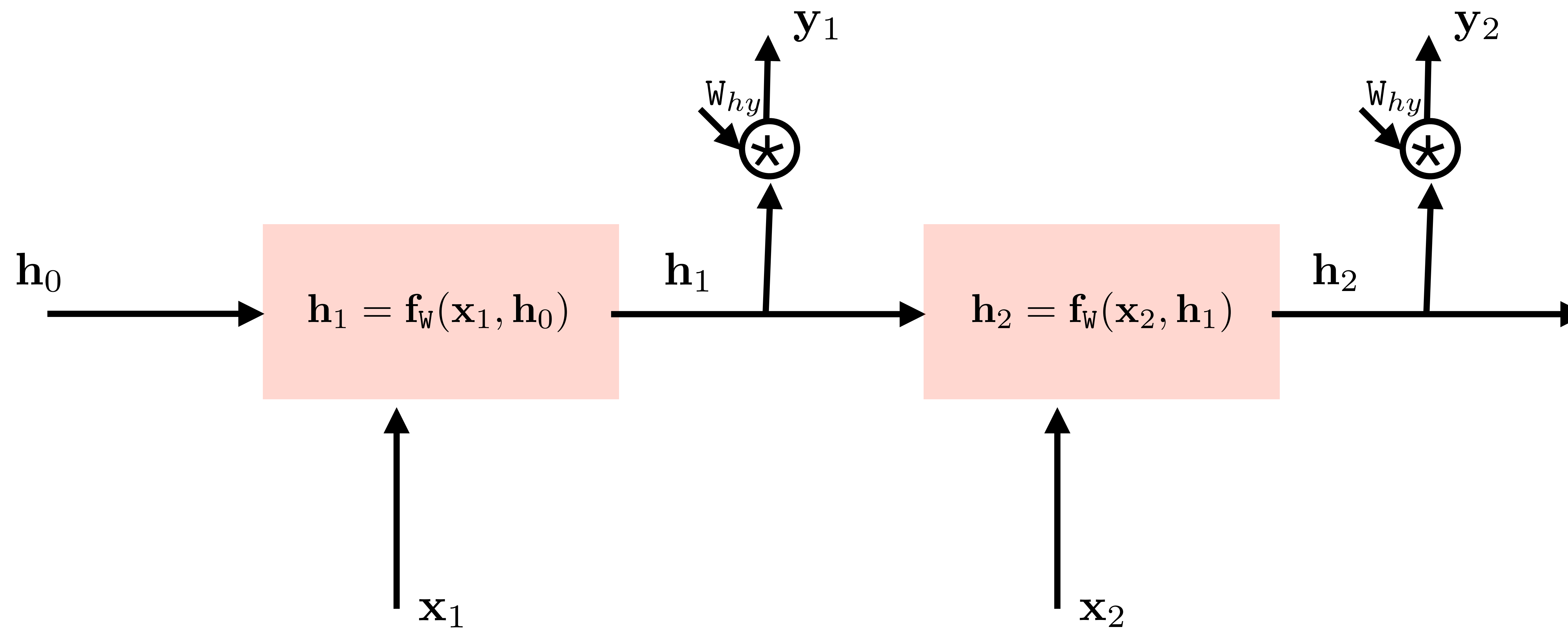
Simple recurrent block - feed-forward pass



Given a finite input sequence: $\mathbf{x}_2$ $\mathbf{x}_3$
we remove the recurrent connection by:

- successive substitution of inputs and
- unrolling the net

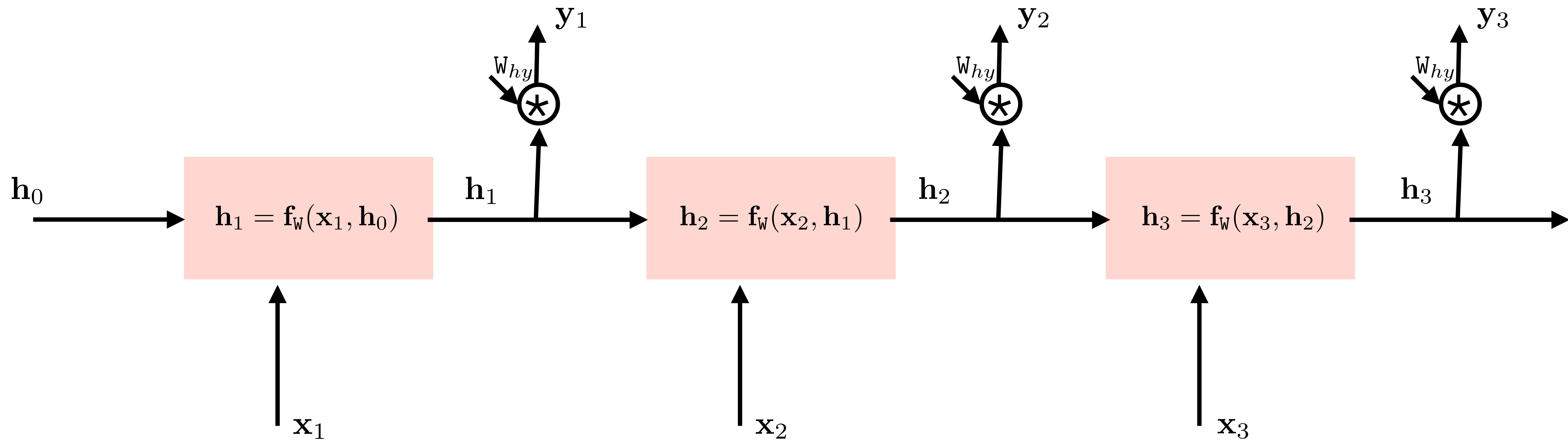
Simple recurrent block - feed-forward pass



Given a finite input sequence: $\mathbf{x}_3$
we remove the recurrent connection by:

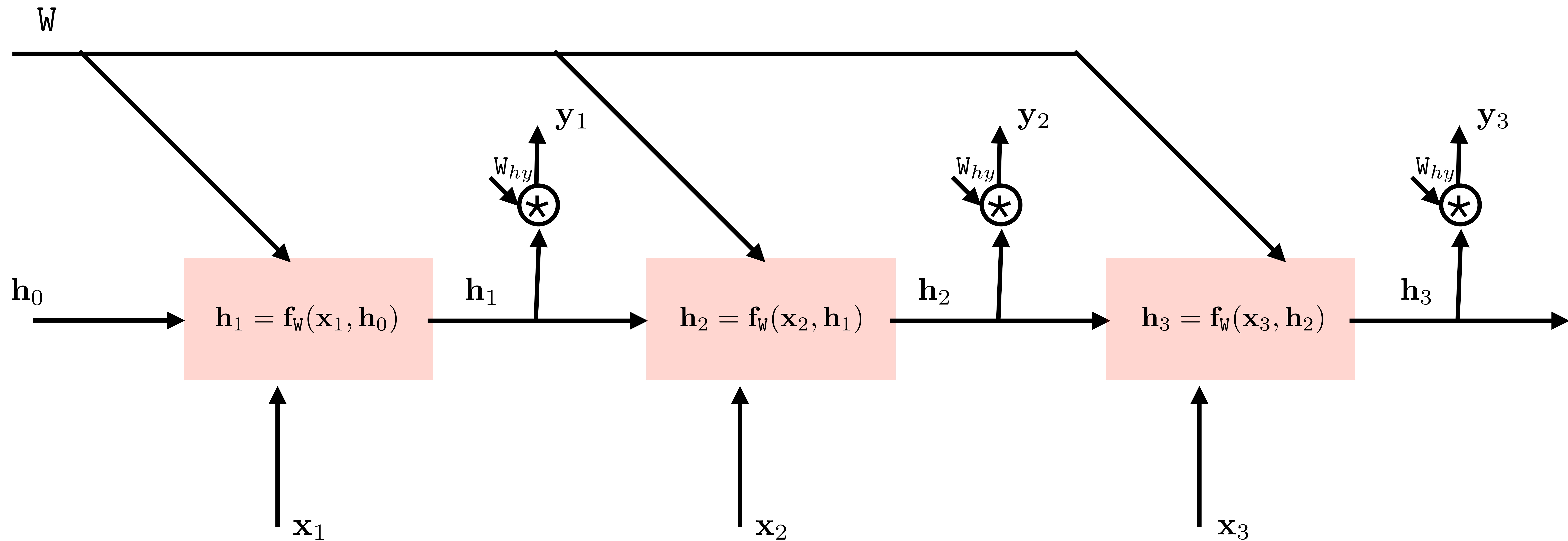
- successive substitution of inputs and
- unrolling the net

Simple recurrent block - feed-forward pass



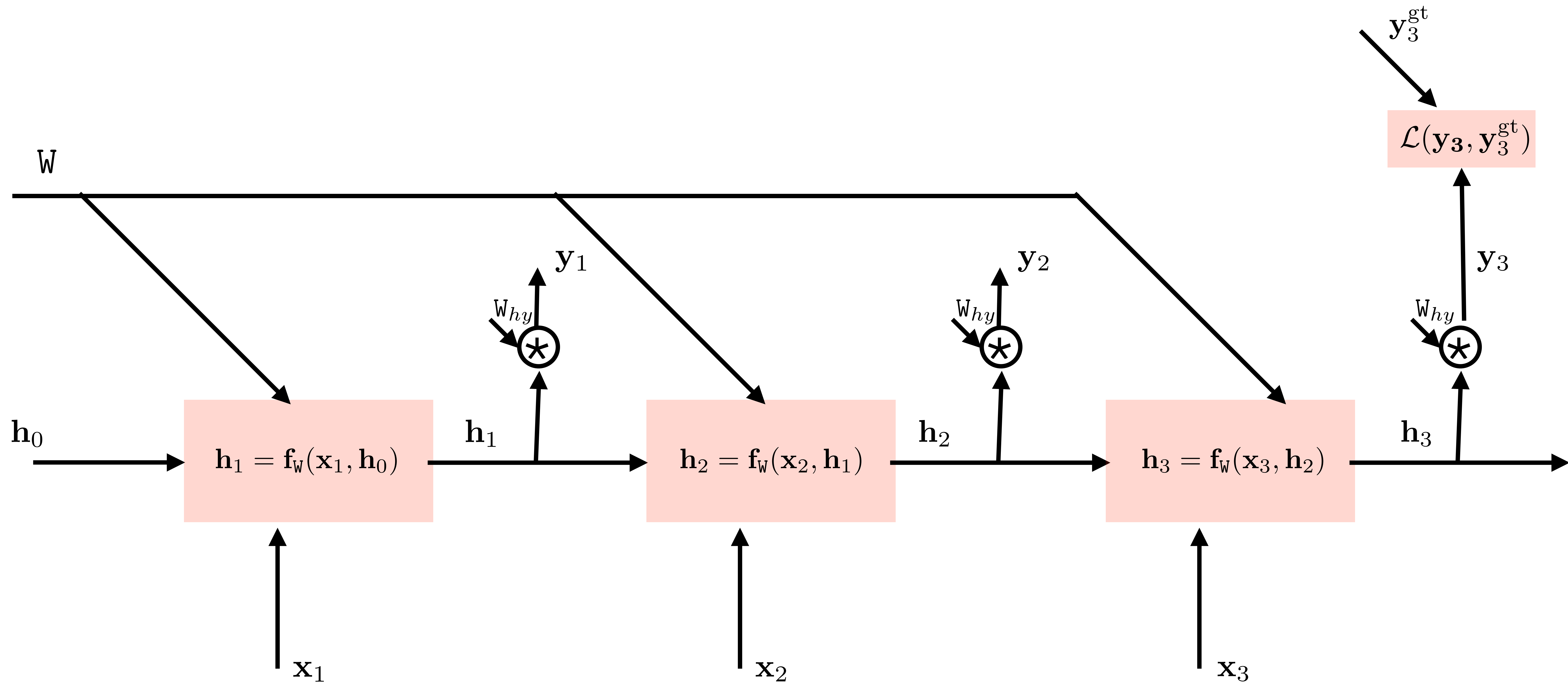
- Unrolled computational graph:
  - it is normal feedforward network

Simple recurrent block - feed-forward pass



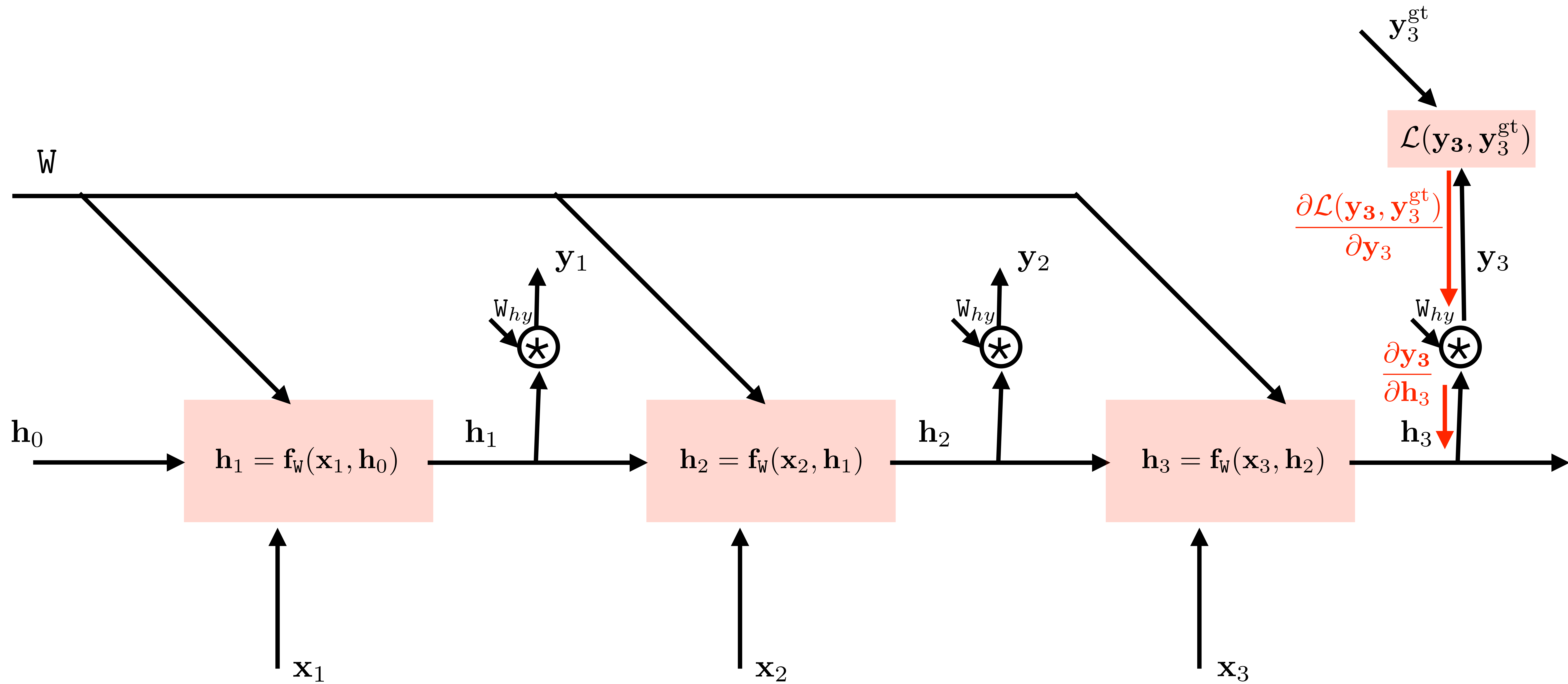
- Unrolled computational graph:
  - it is normal feedforward network
  - it consists of several same blocks with the same weights!

Simple recurrent block - backward pass



- Loss function:
  - cross-entropy loss on the last output only (for simplicity)

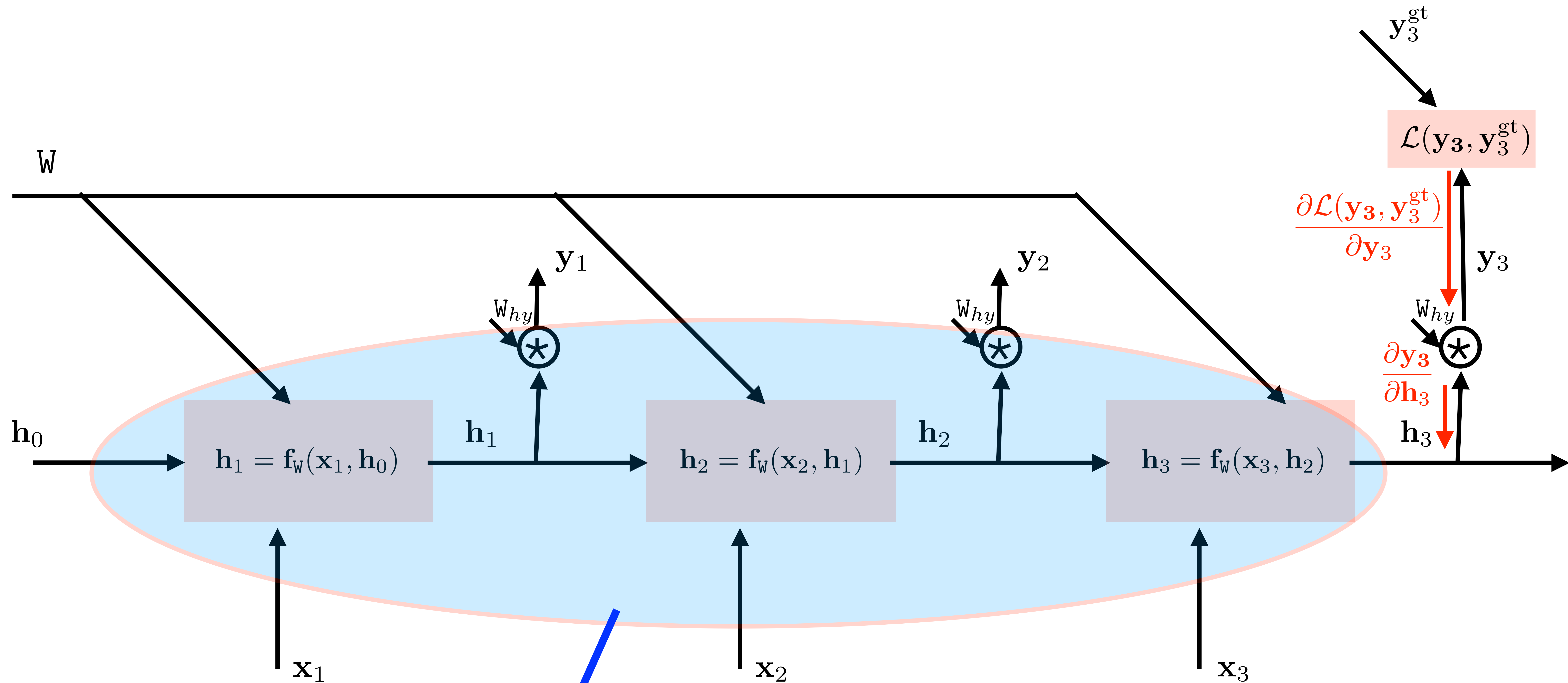
Simple recurrent block - backward pass



- Backprop:

$$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\mathrm{gt}})}{\partial \mathtt{W}} = \quad ?$$

Simple recurrent block - backward pass

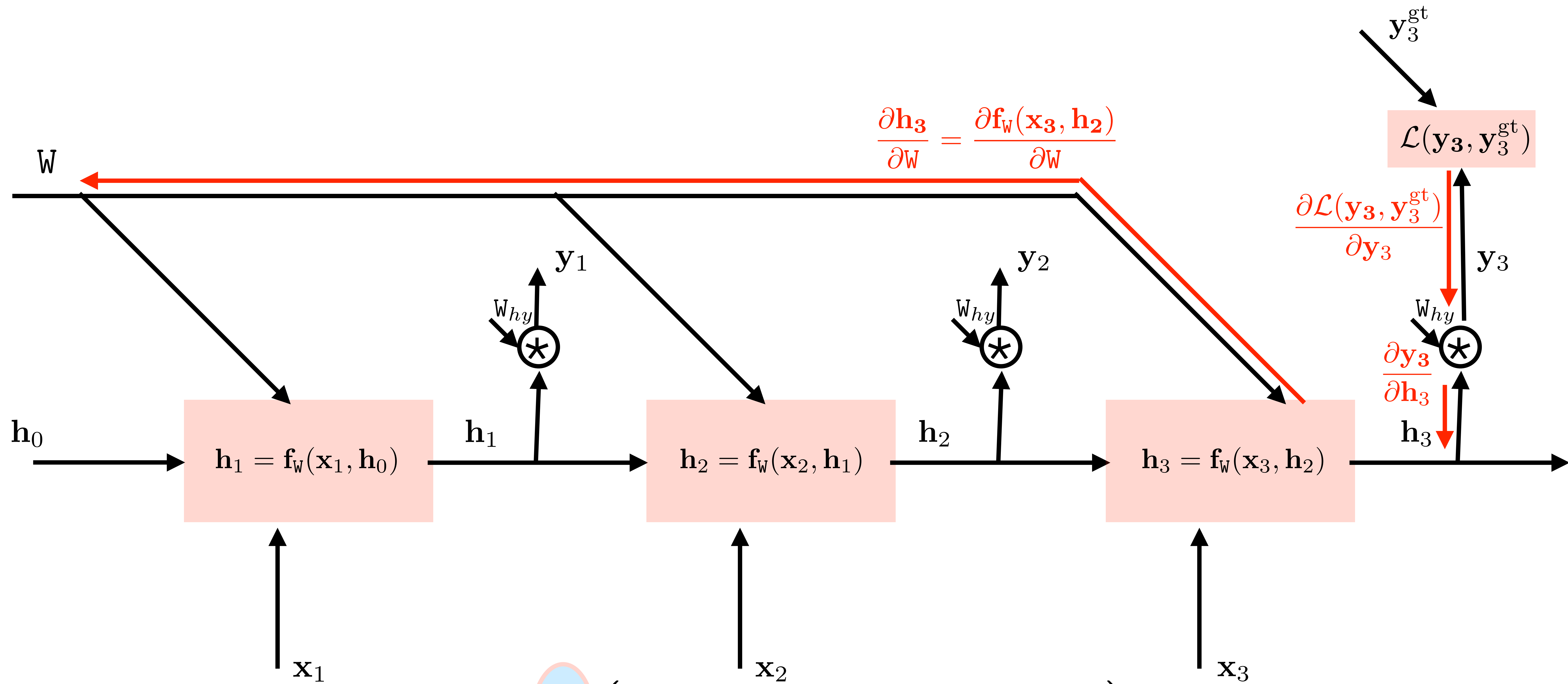


- Backprop:
  - differentiation of multi-dimensional composite function:

$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{gt})}{\partial \mathtt{W}} =$?

$$\mathbf{h}_3 = \mathbf{f}_{\mathtt{W}}\Big(\mathbf{x}_3, \mathbf{f}_{\mathtt{W}}\big(\mathbf{x}_2, \mathbf{f}_{\mathtt{W}}(\mathbf{x}_1, \mathbf{h}_0)\big)\Big)$$

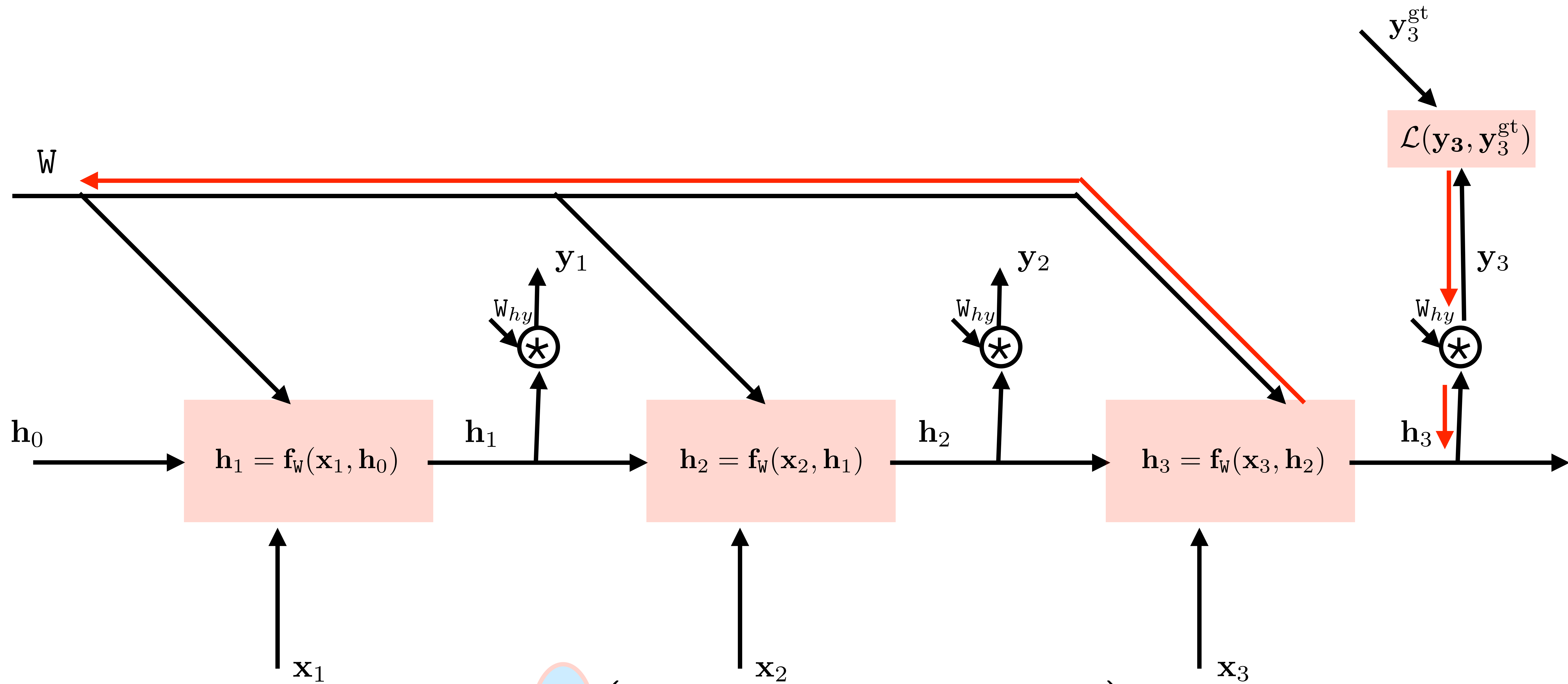
Simple recurrent block - backward pass



$$\mathbf{h}_3 = \mathbf{f}_{\mathtt{W}}\Big(\mathbf{x}_3, \mathbf{f}_{\mathtt{W}}\big(\mathbf{x}_2, \mathbf{f}_{\mathtt{W}}(\mathbf{x}_1, \mathbf{h}_0)\big)\Big)$$

$$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\mathrm{gt}})}{\partial \mathtt{W}} = \quad ?$$

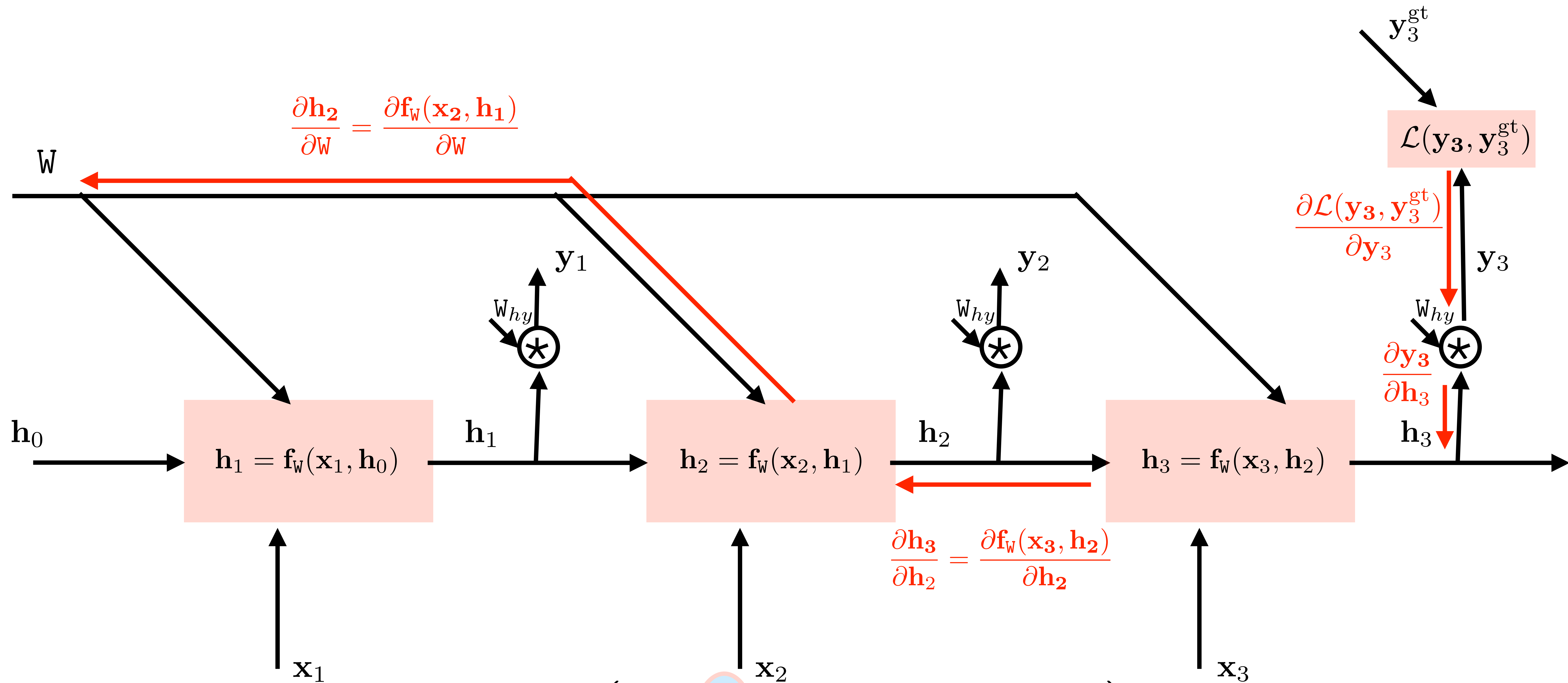
Simple recurrent block - backward pass



$$\mathbf{h}_3 = \mathbf{f}_W\Big(\mathbf{x}_3, \mathbf{f}_W\big(\mathbf{x}_2, \mathbf{f}_W(\mathbf{x}_1, \mathbf{h}_0)\big)\Big)$$

$$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\mathrm{gt}})}{\partial W} = \frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\mathrm{gt}})}{\partial \mathbf{y}_3} \frac{\partial \mathbf{y_3}}{\partial \mathbf{h}_3} \frac{\partial \mathbf{f}_W(\mathbf{x_3}, \mathbf{h_2})}{\partial W} +$$

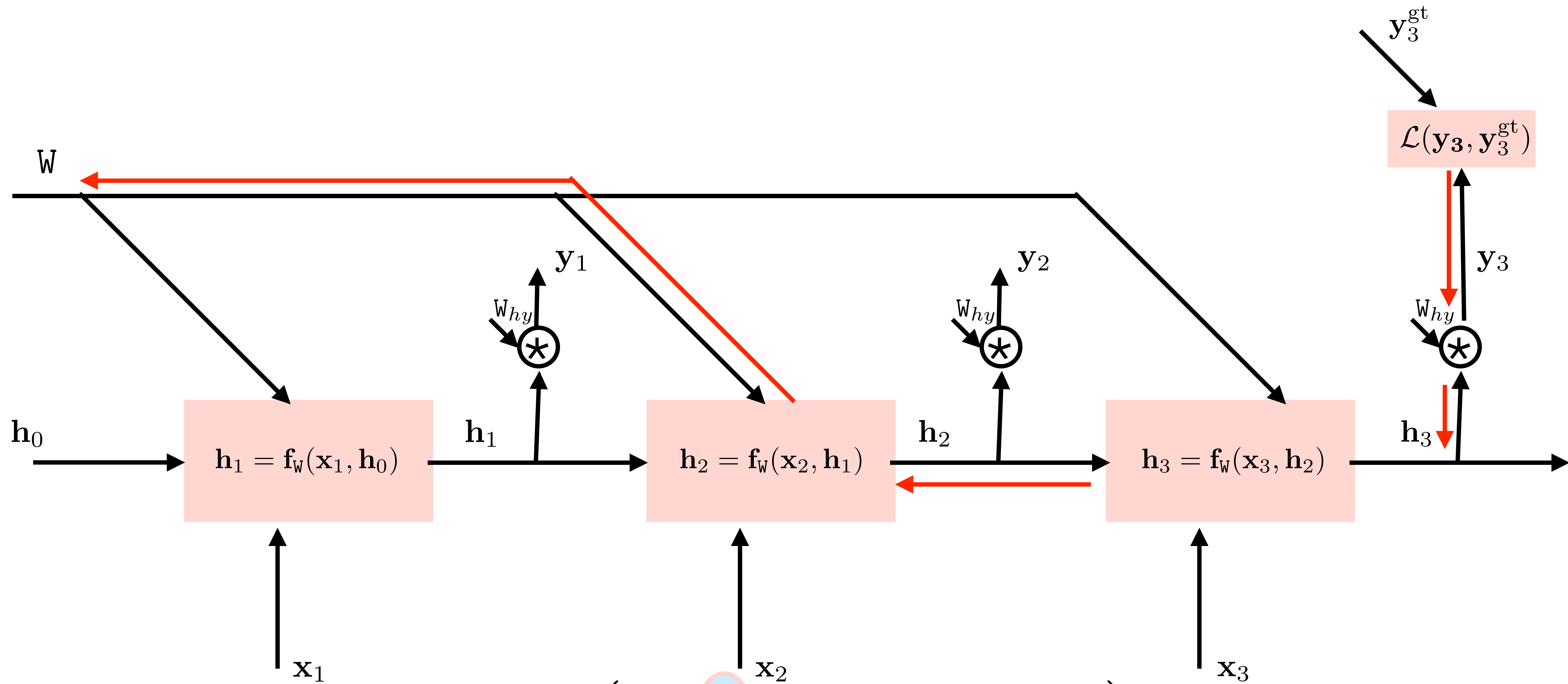
Simple recurrent block - backward pass



$$\mathbf{h}_3 = \mathbf{f}_{\mathtt{W}}\Big(\mathbf{x}_3, \mathbf{f}_{\mathtt{W}}(\mathbf{x}_2, \mathbf{f}_{\mathtt{W}}(\mathbf{x}_1, \mathbf{h}_0))\Big)$$

$$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\mathrm{gt}})}{\partial \mathtt{W}} = \frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\mathrm{gt}})}{\partial \mathbf{y}_3} \frac{\partial \mathbf{y_3}}{\partial \mathbf{h}_3} \frac{\partial \mathbf{f}_{\mathtt{W}}(\mathbf{x_3}, \mathbf{h_2})}{\partial \mathtt{W}} +$$

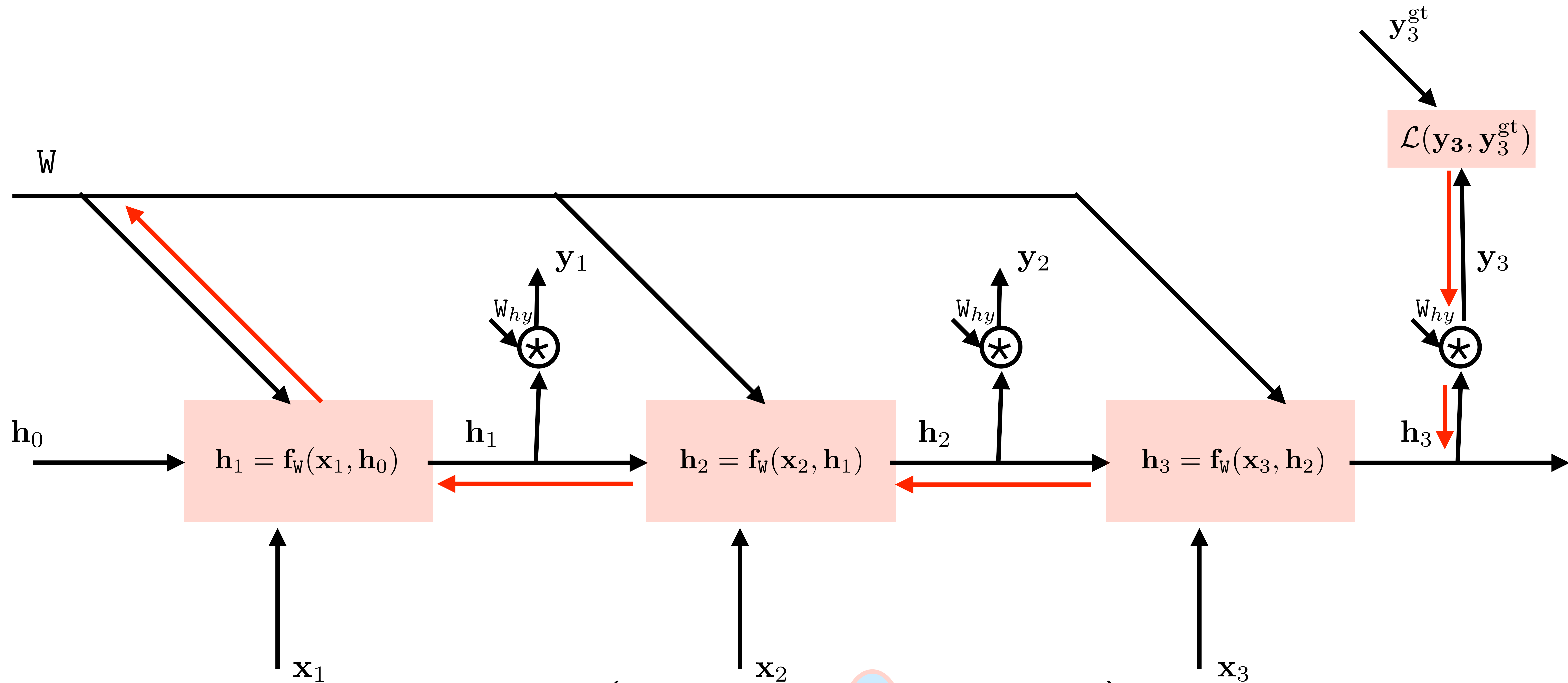
Simple recurrent block - backward pass



$$\mathbf{h}_3 = \mathbf{f}_{\mathtt{W}}\Big(\mathbf{x}_3, \mathbf{f}_{\mathtt{W}}\big(\mathbf{x}_2, \mathbf{f}_{\mathtt{W}}(\mathbf{x}_1, \mathbf{h}_0)\big)\Big)$$

$$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\text{gt}})}{\partial \mathtt{W}} = \frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\text{gt}})}{\partial \mathbf{y}_3} \frac{\partial \mathbf{y_3}}{\partial \mathbf{h}_3} \frac{\partial \mathbf{f}_{\mathtt{W}}(\mathbf{x_3}, \mathbf{h_2})}{\partial \mathtt{W}} + \frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\text{gt}})}{\partial \mathbf{y}_3} \frac{\partial \mathbf{y_3}}{\partial \mathbf{h}_3} \frac{\partial \mathbf{f}_{\mathtt{W}}(\mathbf{x_3}, \mathbf{h_2})}{\partial \mathbf{h_2}} \frac{\partial \mathbf{f}_{\mathtt{W}}(\mathbf{x_2}, \mathbf{h_1})}{\partial \mathtt{W}} +$$

Simple recurrent block - backward pass



$$\mathbf{h}_3 = \mathbf{f}_W\Big(\mathbf{x}_3, \mathbf{f}_W\big(\mathbf{x}_2, \mathbf{f}_W(\mathbf{x}_1, \mathbf{h}_0)\big)\Big)$$

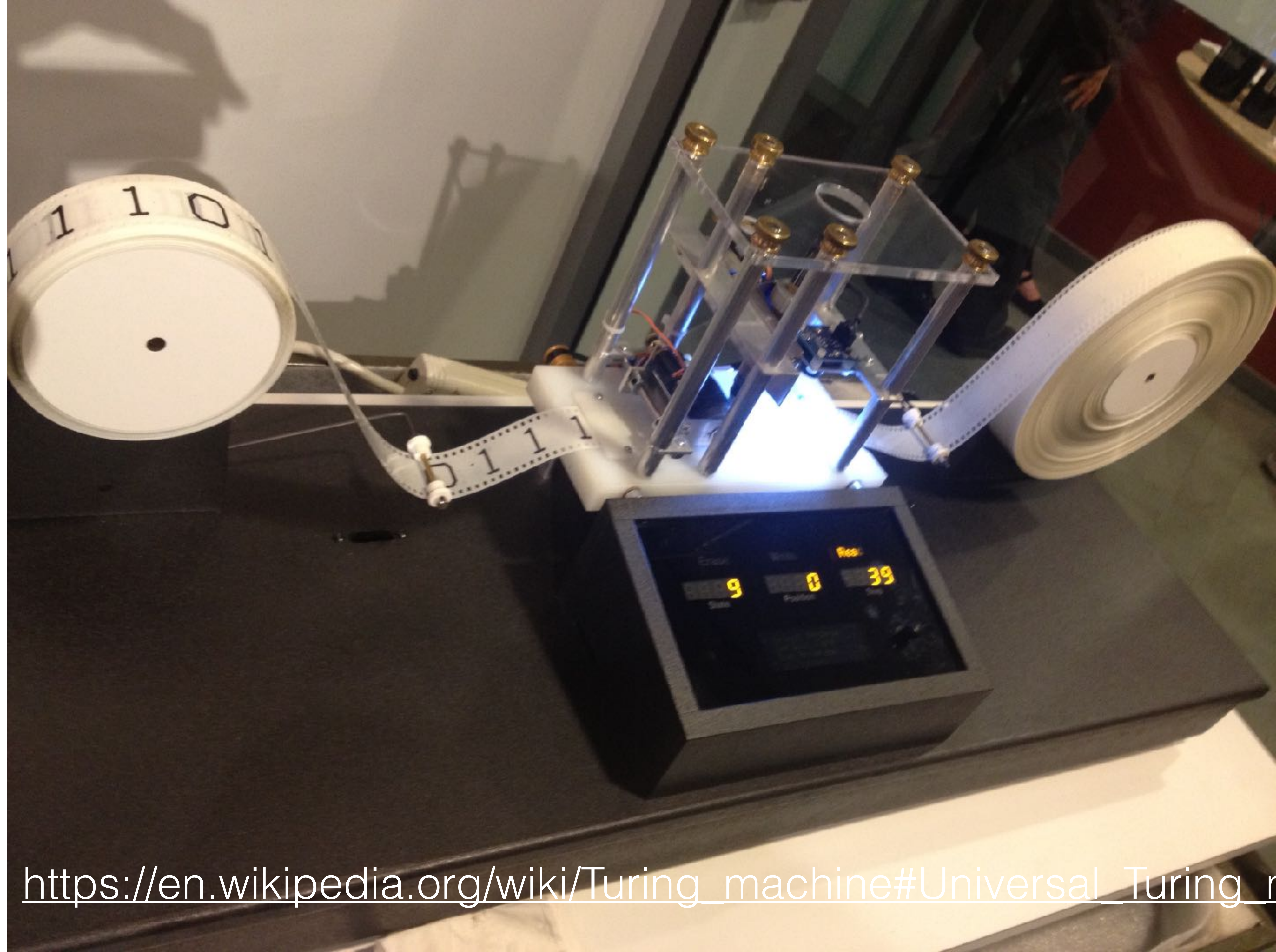
$$\frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\text{gt}})}{\partial W} = \frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\text{gt}})}{\partial \mathbf{y}_3} \frac{\partial \mathbf{y_3}}{\partial \mathbf{h}_3} \frac{\partial \mathbf{f}_W(\mathbf{x_3}, \mathbf{h_2})}{\partial W} + \frac{\partial \mathcal{L}(\mathbf{y_3}, \mathbf{y}_3^{\text{gt}})}{\partial \mathbf{y}_3} \frac{\partial \mathbf{y_3}}{\partial \mathbf{h}_3} \frac{\partial \mathbf{f}_W(\mathbf{x_3}, \mathbf{h_2})}{\partial \mathbf{h_2}} \frac{\partial \mathbf{f}_W(\mathbf{x_2}, \mathbf{h_1})}{\partial W} + \ldots$$

RNN vs feedforward network

recurrent (RNN): $\mathbf{h}_t = \mathbf{f}_{\mathtt{W}}(\mathbf{x}_t, \mathbf{f}_{\mathtt{W}}(\mathbf{x}_{t-1}, \ldots \mathbf{f}_{\mathtt{W}}(\mathbf{x}_1, \mathbf{h}_0))$

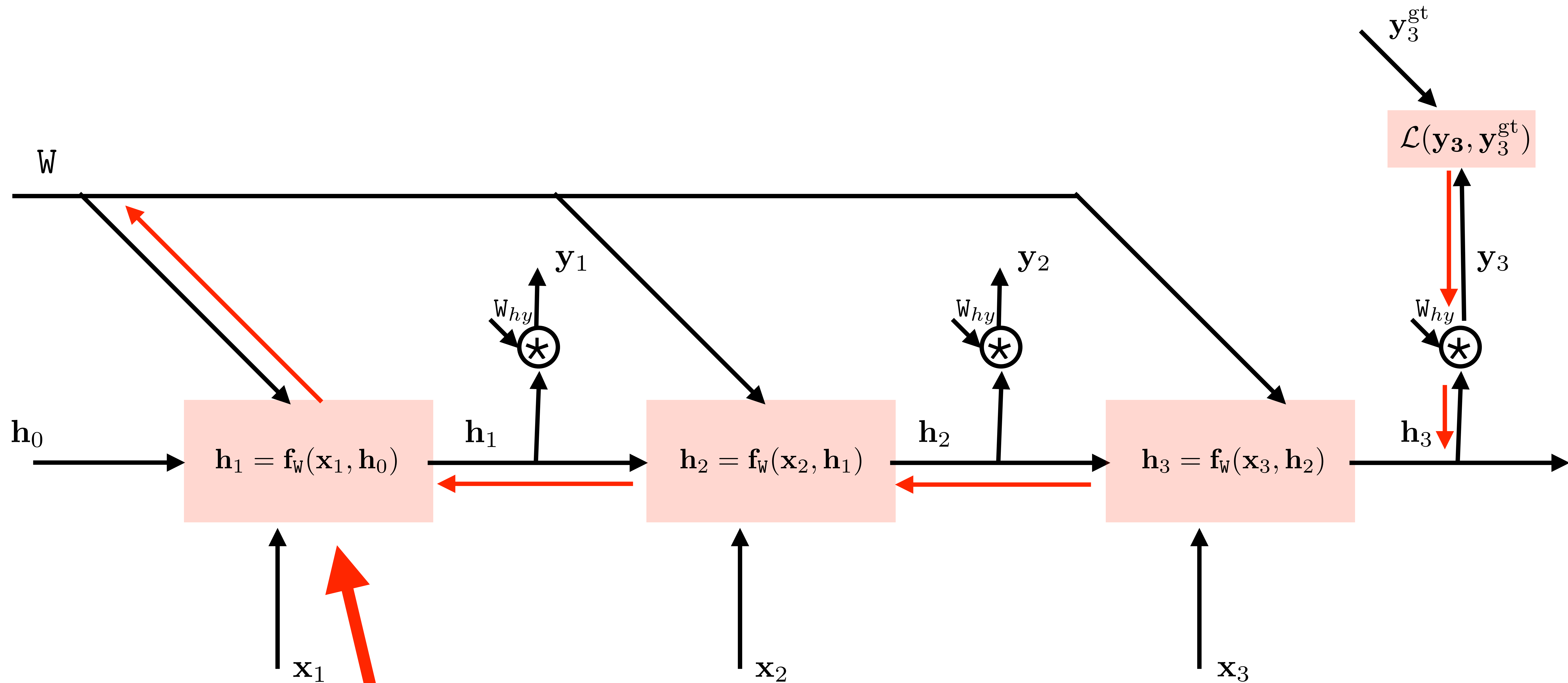
feedforward: $\mathbf{h}_t = \mathbf{g}(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots \mathbf{x}_1, \mathbf{w})$ (stacking sequence to long input vector)

- RNN works for different lengths of input sequences
- RNN share weights between different time instances (similarly as convolution on spatial domain).
- Some RNN are spatio-temporal convolutions [Hinton 1988]
- Memory is attention in time [Alex Graves 2020]
- RNN is universal (can compute any function computable by Turing machine)



https://en.wikipedia.org/wiki/Turing_machine#Universal_Turing_

Simple recurrent block - backward pass

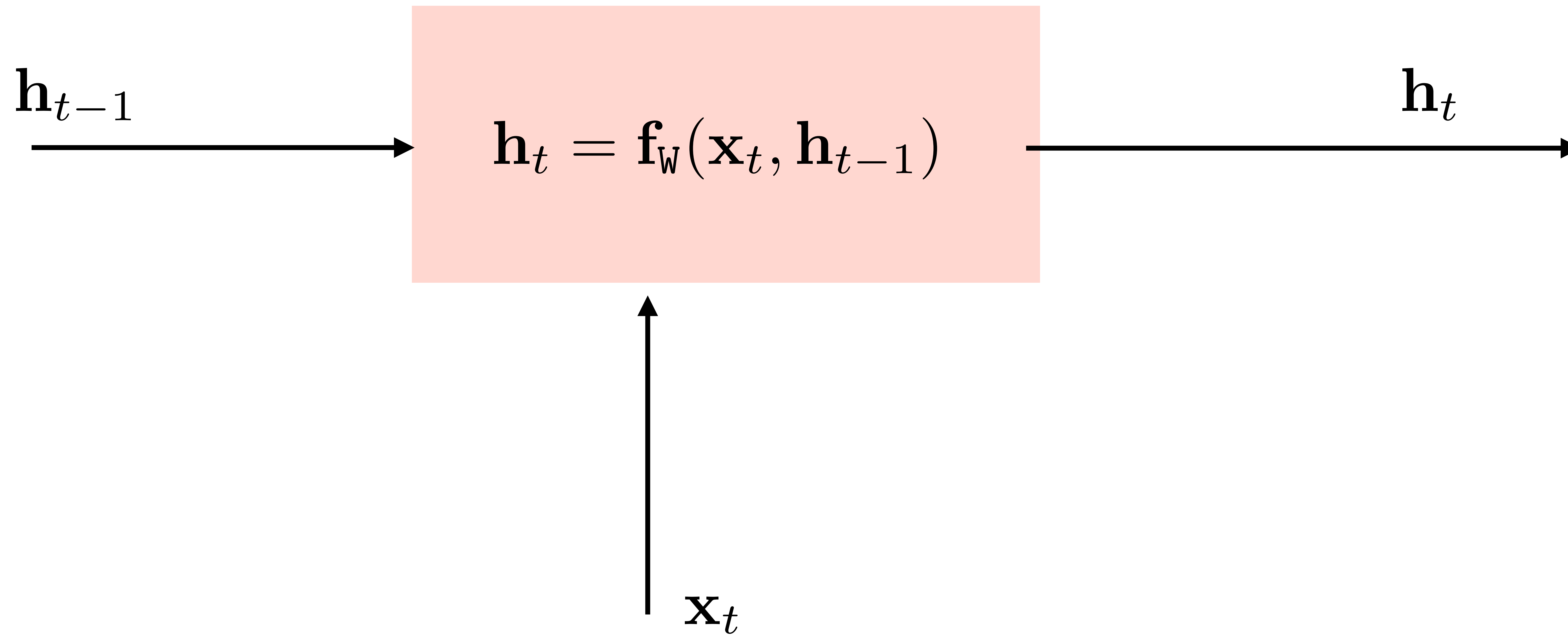


deep blocks often suffer from vanishing gradient

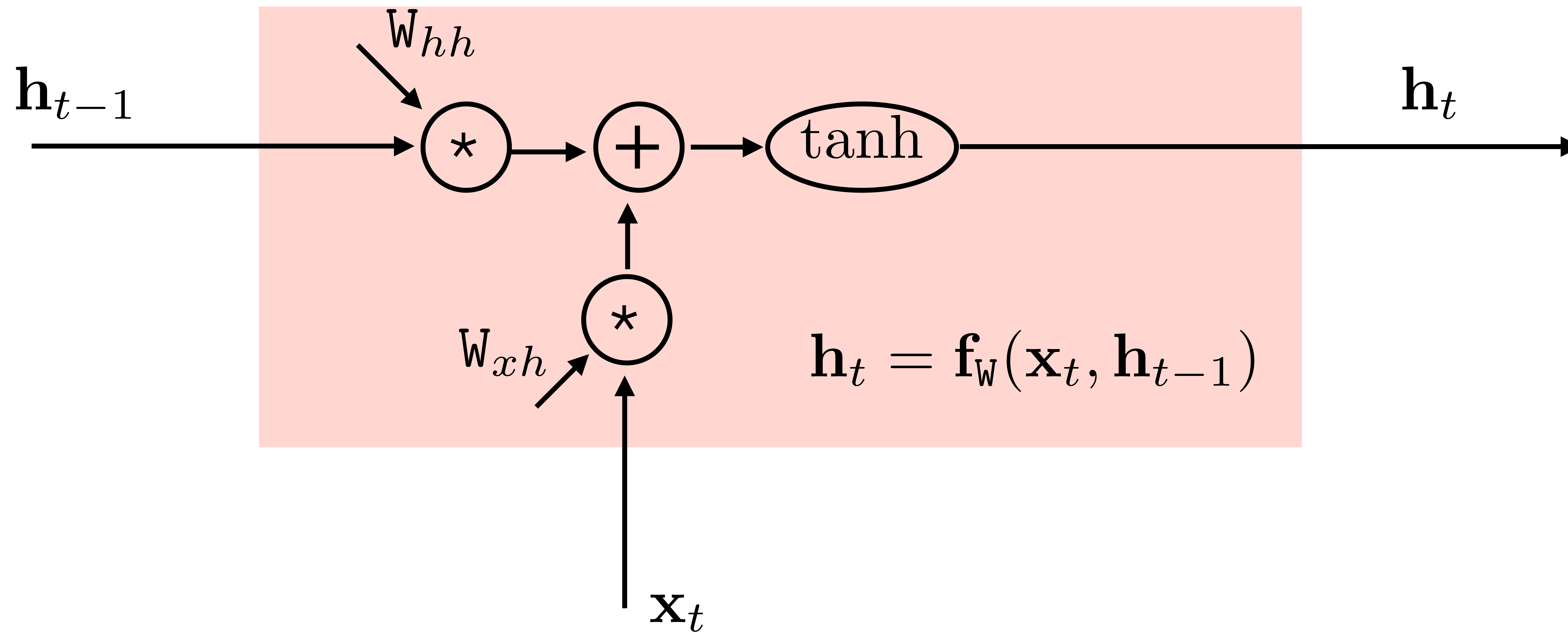
=> better structure needed

LSTM (kind of ResNet for recurrent networks)

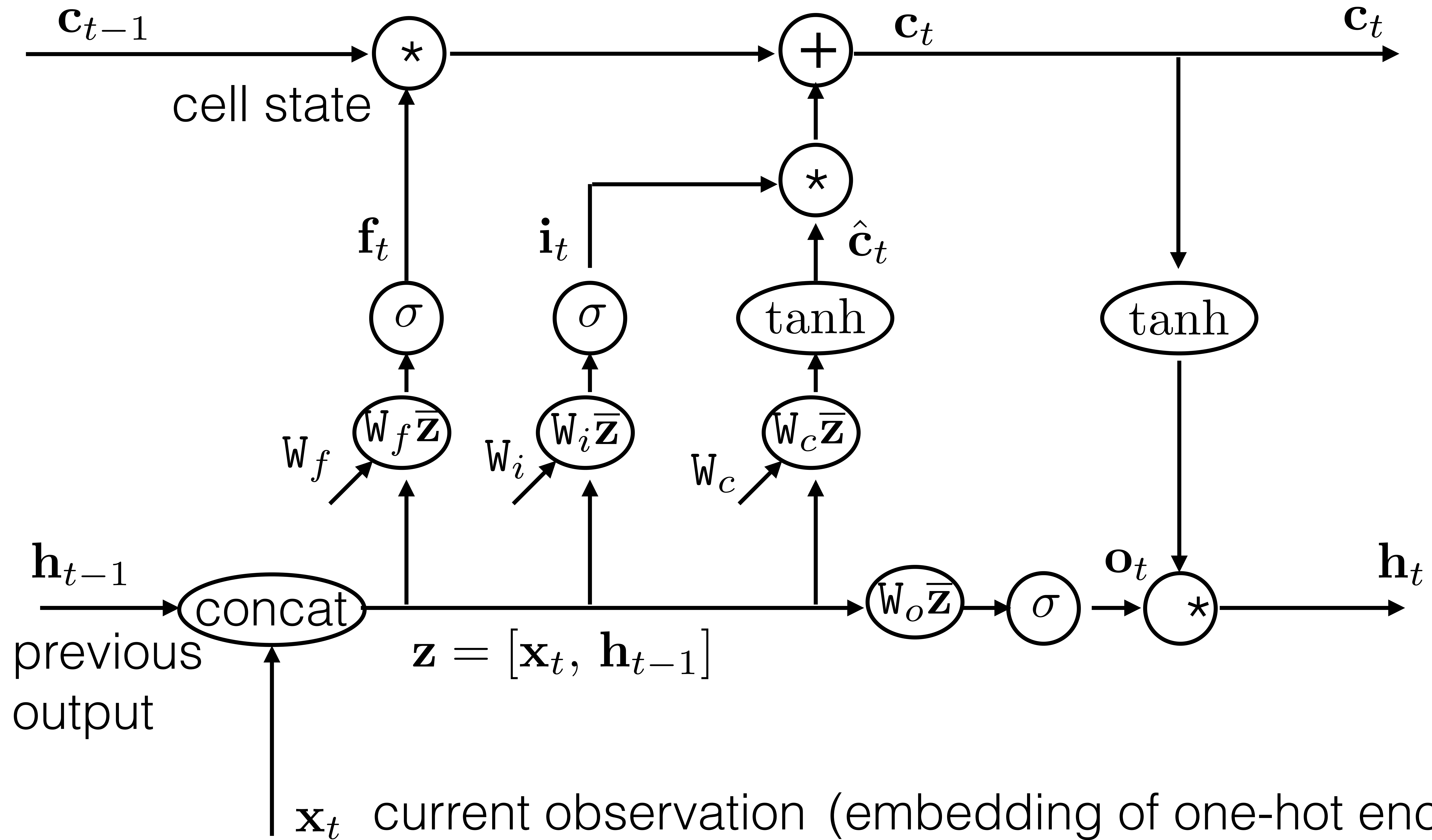
Simple recurrent block



Simple recurrent block

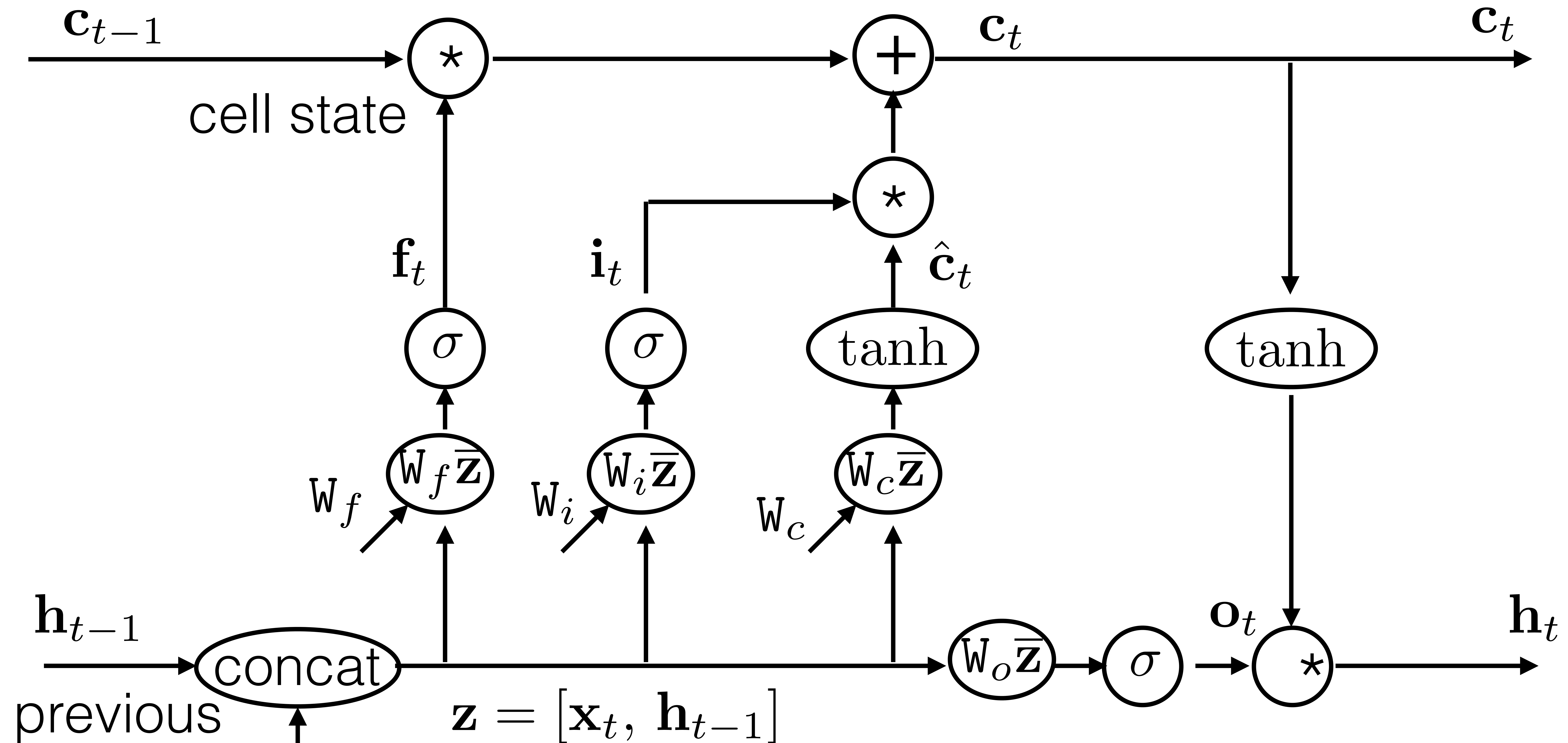


LSTM block



"I live with my parents, …"

LSTM block

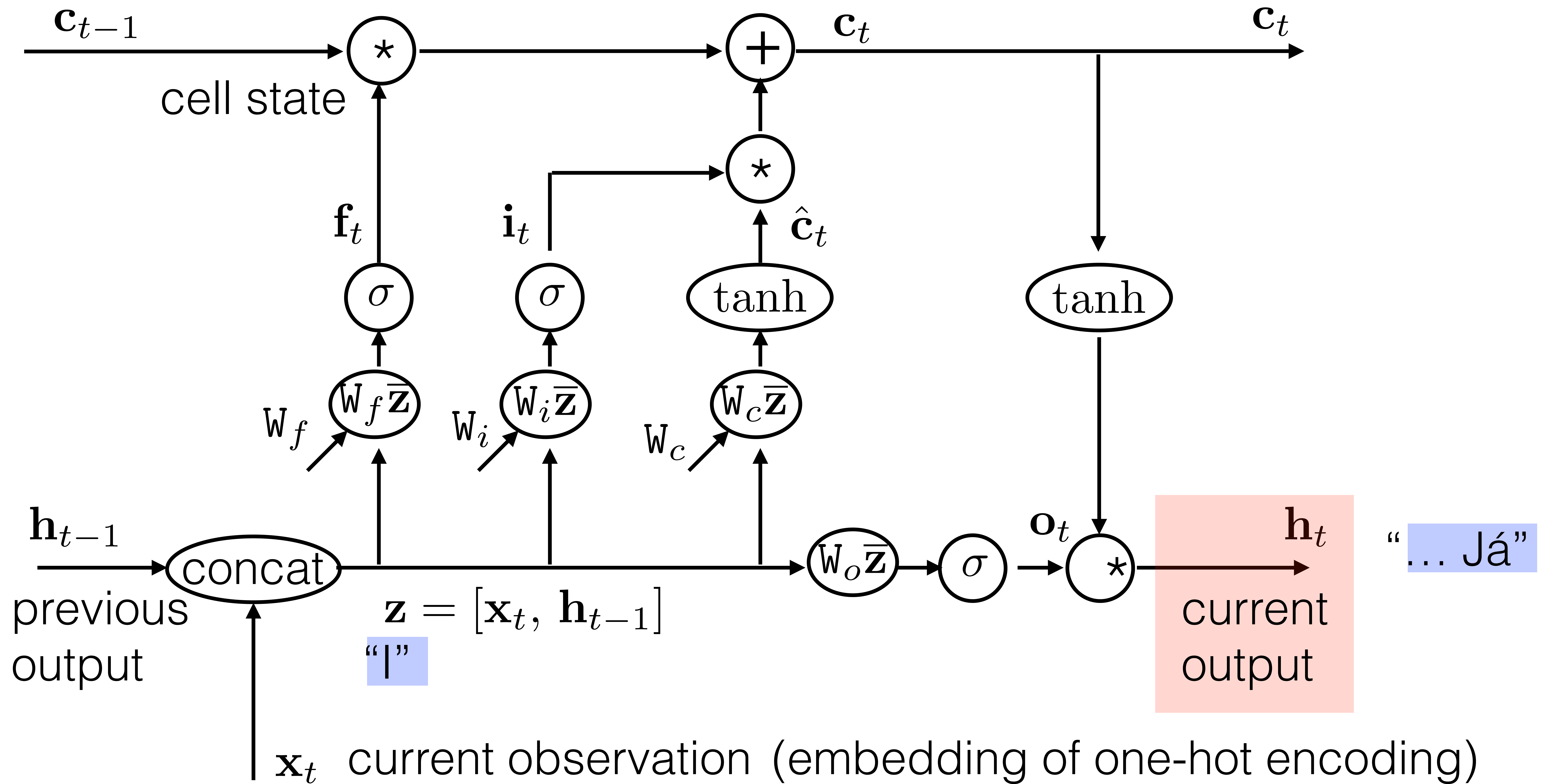


“I”

$\mathbf{x}_t$ current observation (embedding of one-hot encoding)

“I live with my parents, …”

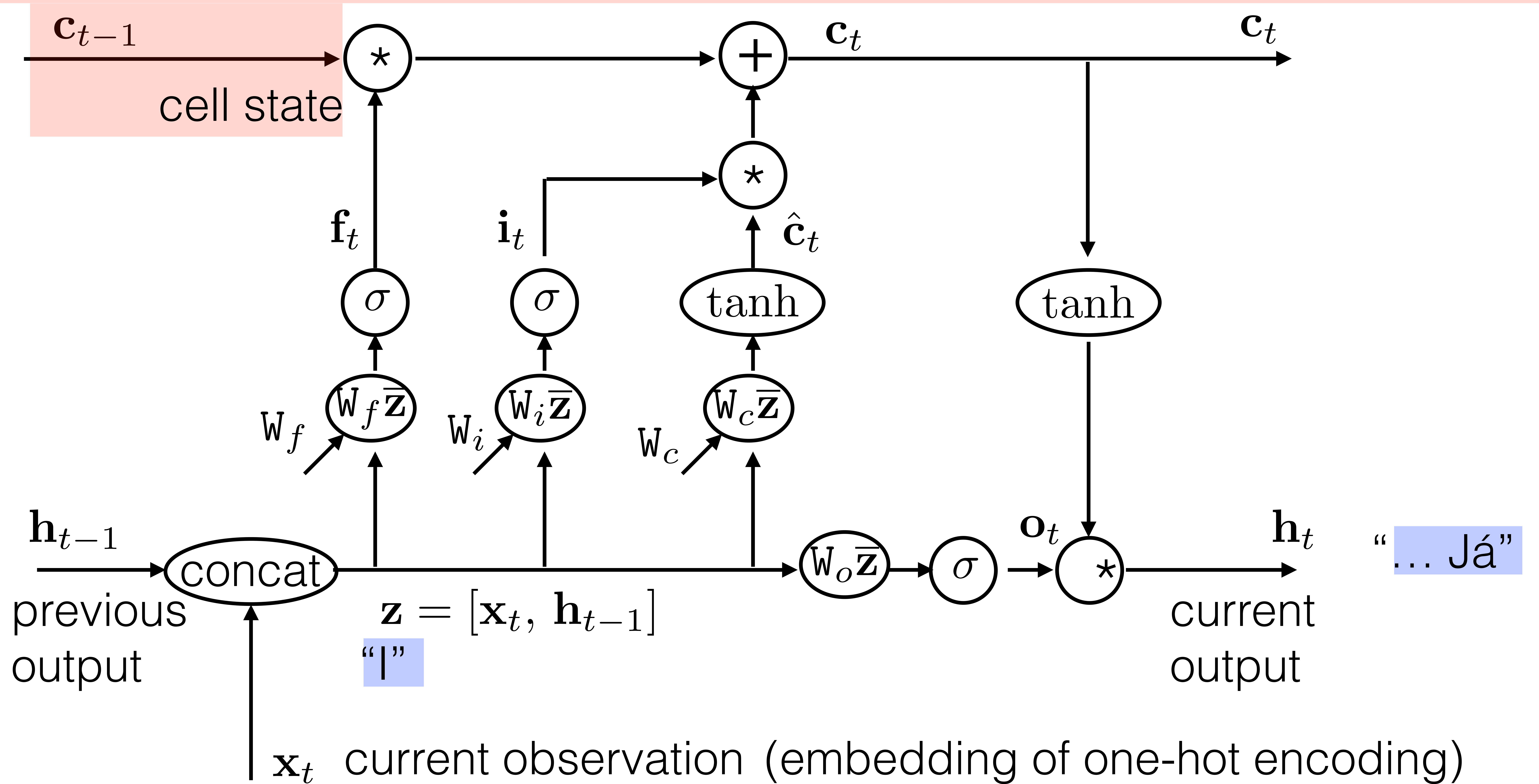
LSTM block



“I live with my parents, …”

LSTM block

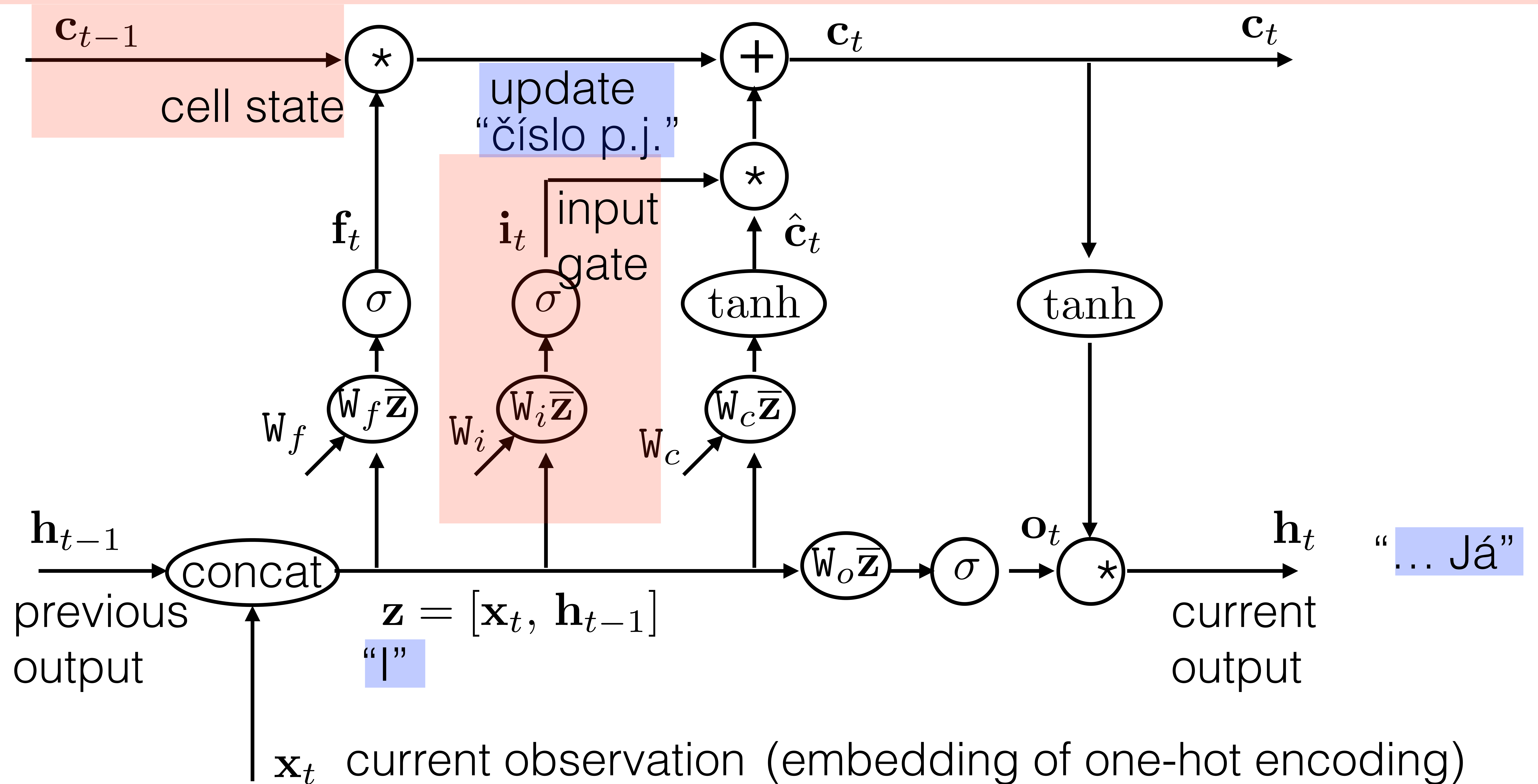
cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]

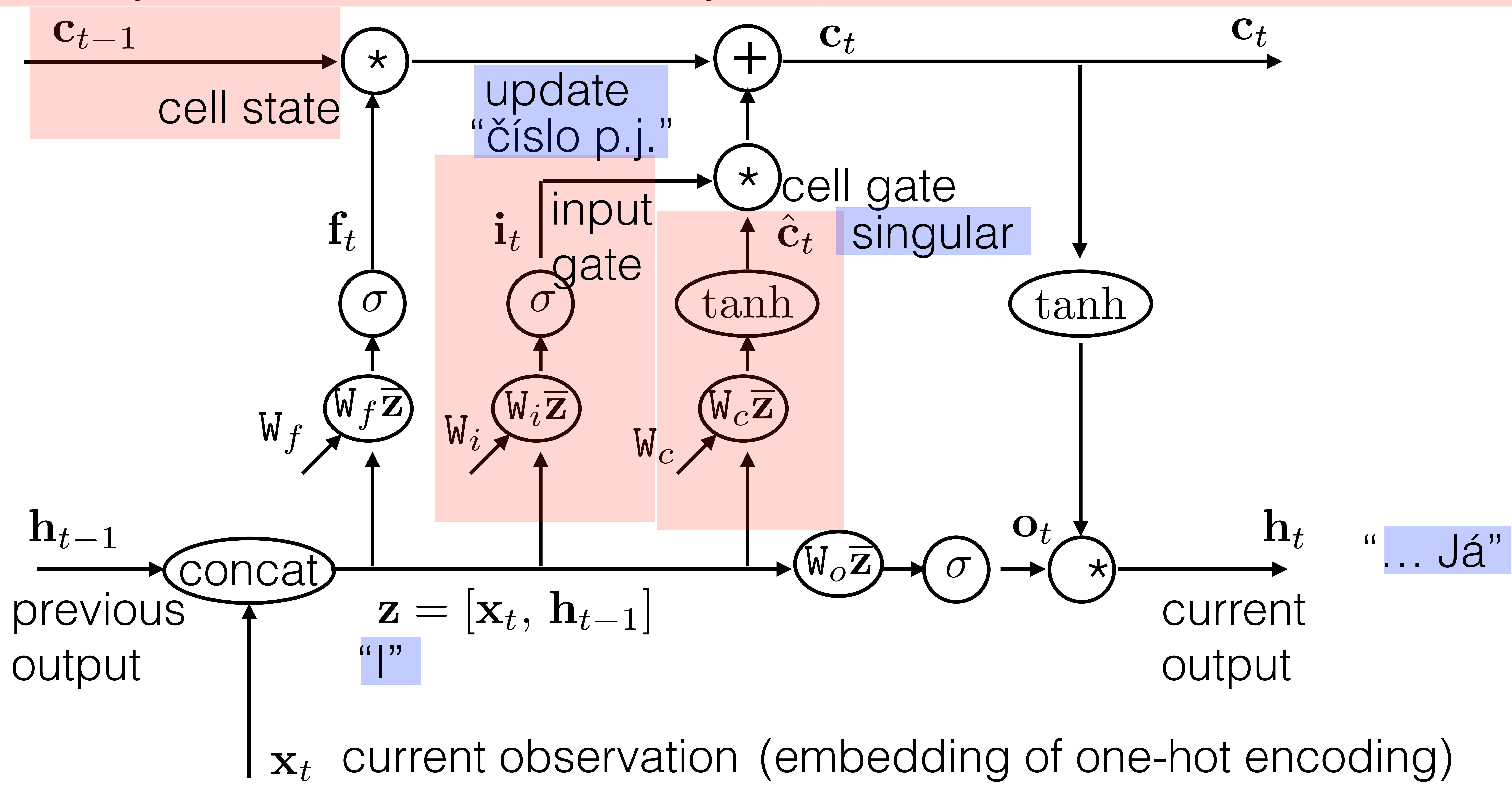


"I live with my parents, …"

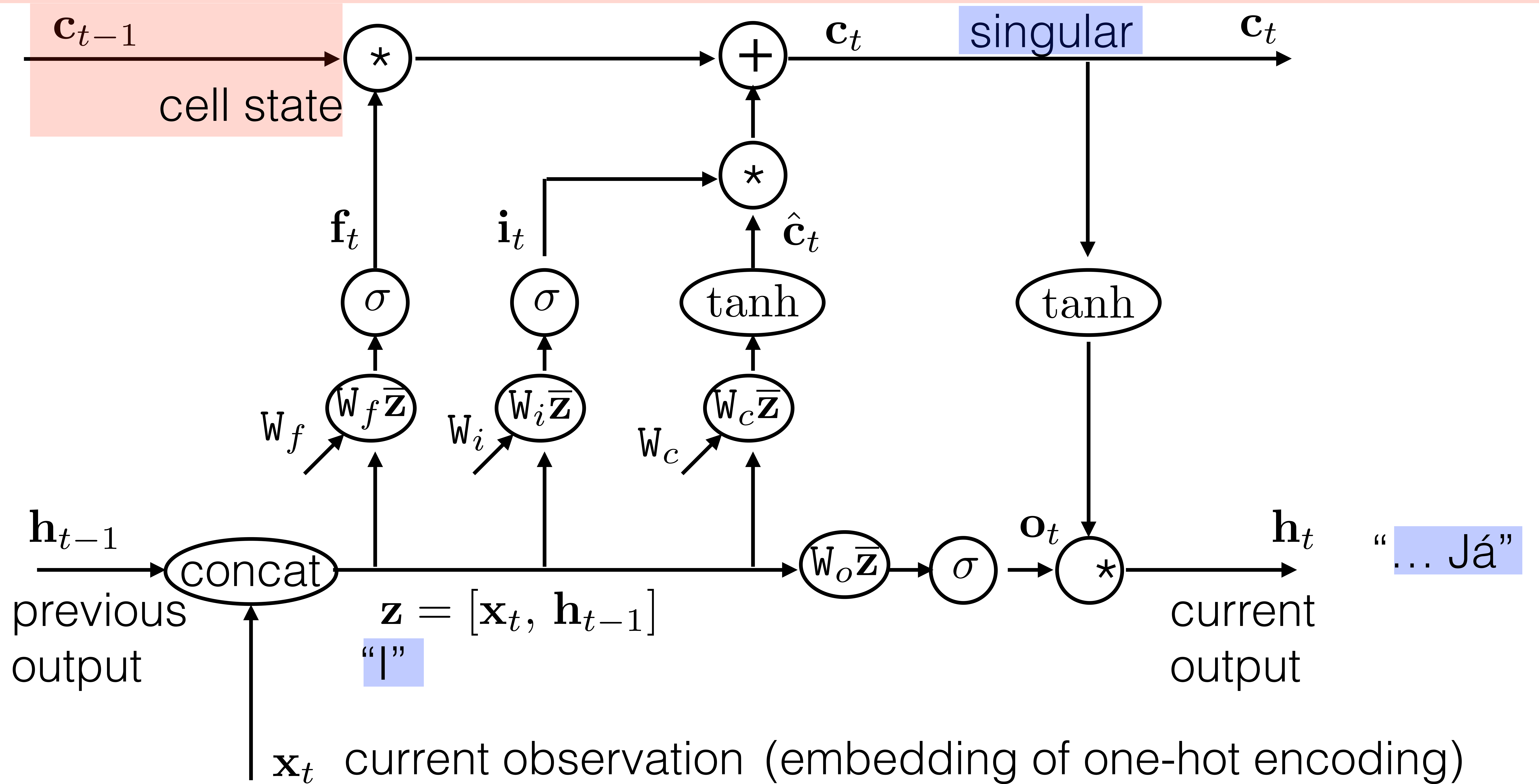
LSTM block

cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]



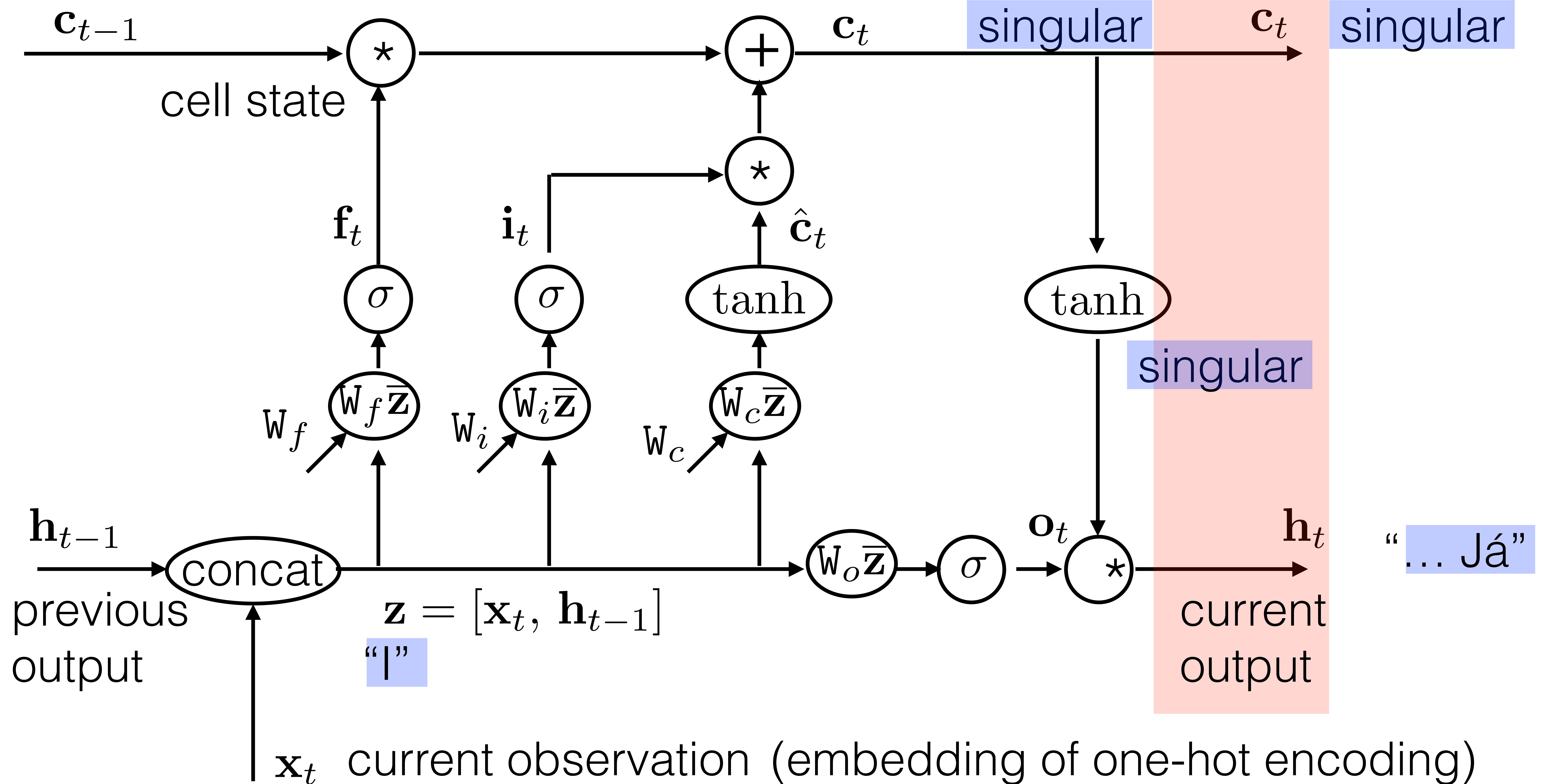


LSTM block
cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]
$\mathbf{c}_{t-1}$
cell state
$*$
update
"číslo p.j."
$+$
$\mathbf{c}_t$
$\mathbf{c}_t$
$*$
cell gate
$\mathbf{f}_t$
$\mathbf{i}_t$
input gate
$\hat{\mathbf{c}}_t$
singular
σ
σ
tanh
tanh
$\mathtt{W}_f\overline{\mathbf{z}}$
$\mathtt{W}_f$
$\mathtt{W}_i\overline{\mathbf{z}}$
$\mathtt{W}_i$
$\mathtt{W}_c\overline{\mathbf{z}}$
$\mathtt{W}_c$
$\mathbf{h}_{t-1}$
concat
previous output
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
"I"
$\mathtt{W}_o\overline{\mathbf{z}}$
σ
$\mathbf{o}_t$
$*$
$\mathbf{h}_t$
"… Já"
current output
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)
"I live with my parents, …"


LSTM block
cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]
$\mathbf{c}_{t-1}$
cell state
*
+
$\mathbf{c}_t$
singular
$\mathbf{c}_t$
*
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
σ
σ
tanh
tanh
$W_f\overline{\mathbf{z}}$
$W_i\overline{\mathbf{z}}$
$W_c\overline{\mathbf{z}}$
W_f
W_i
W_c
$\mathbf{h}_{t-1}$
concat
$W_o\overline{\mathbf{z}}$
σ
$\mathbf{o}_t$
*
$\mathbf{h}_t$
"… Já"
previous output
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
"I"
current output
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)
"I live with my parents, …"

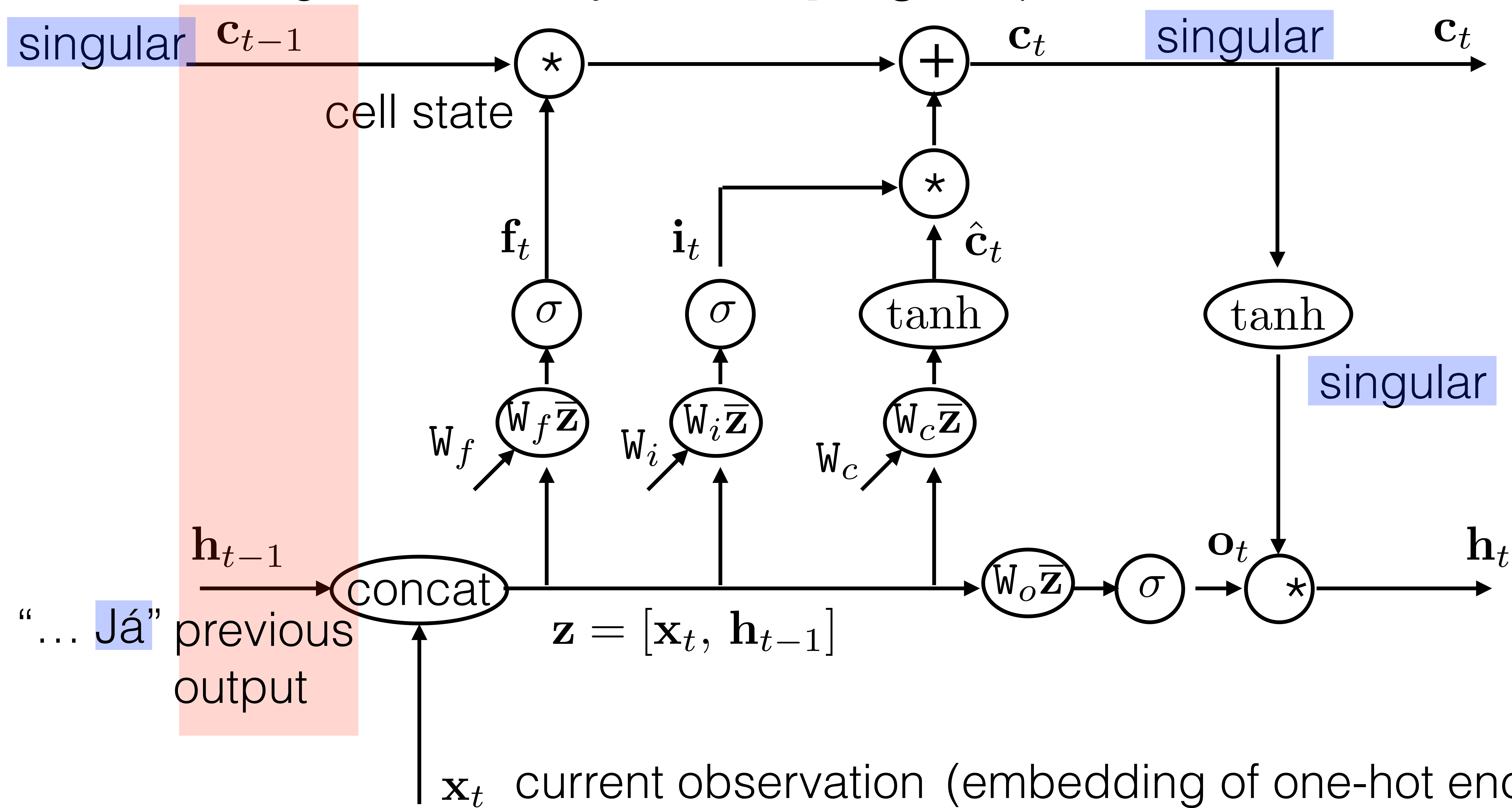
LSTM block

cell state is long term memory= vector [**singular**/plural, feminine/masculine, case, …]

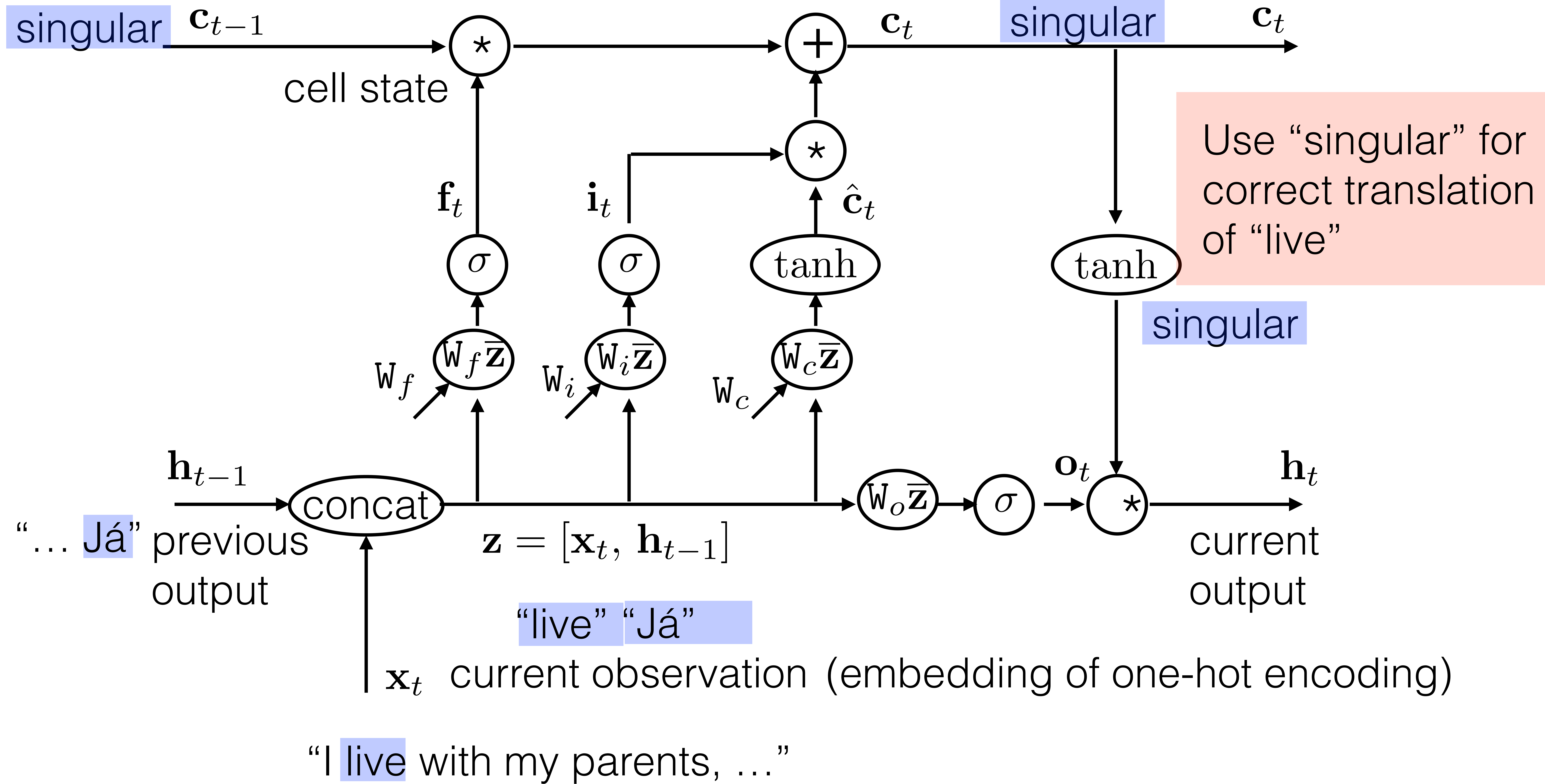


LSTM block

cell state is long term memory= vector [**singular**/plural, feminine/masculine, case, …]



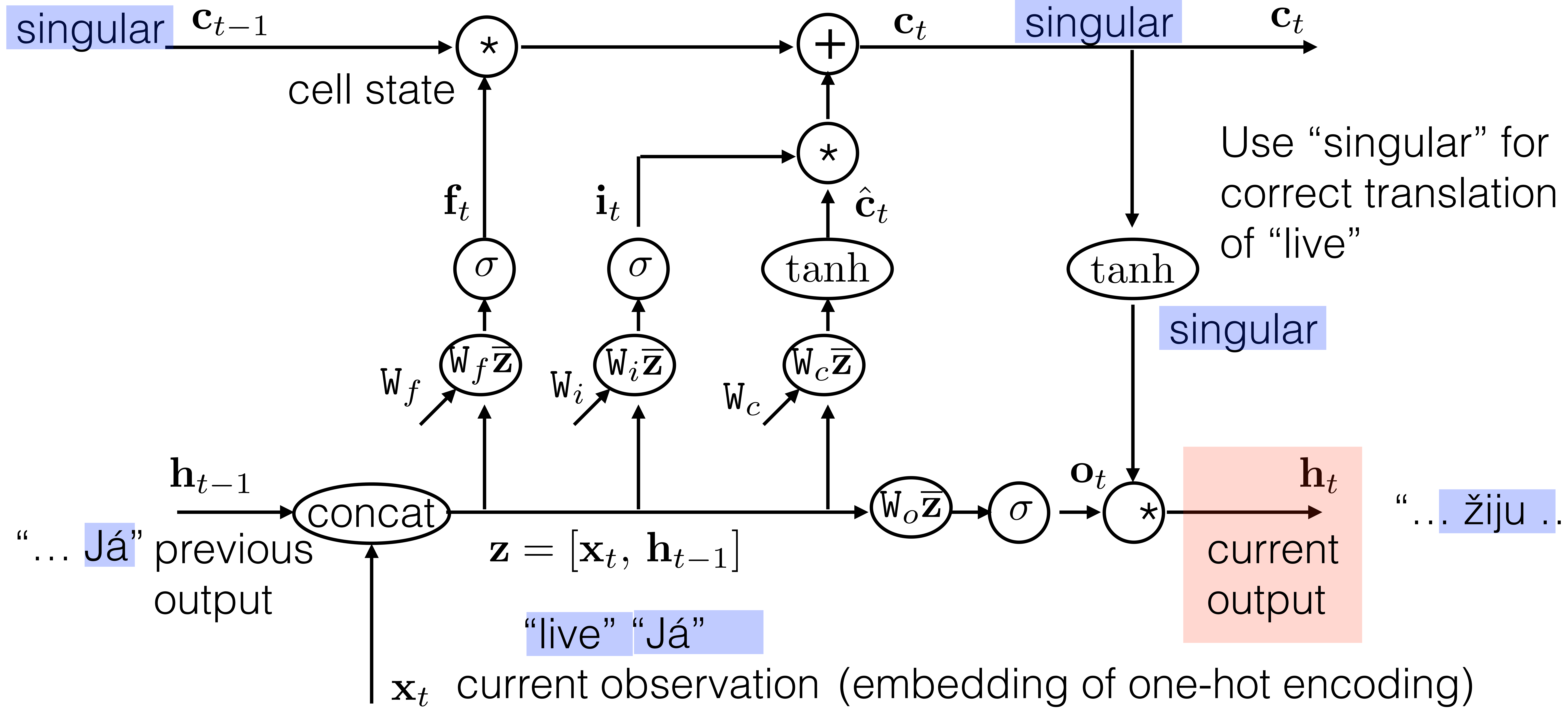
"I live with my parents, …"



LSTM block
cell state is long term memory= vector [**singular**/plural, feminine/masculine, case, …]
singular
$\mathbf{c}_{t-1}$
cell state
$\mathbf{c}_t$
singular
$\mathbf{c}_t$
Use "singular" for correct translation of "live"
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
σ
σ
tanh
tanh
singular
W_f
$W_f\overline{\mathbf{z}}$
W_i
$W_i\overline{\mathbf{z}}$
W_c
$W_c\overline{\mathbf{z}}$
$\mathbf{h}_{t-1}$
concat
$\mathbf{o}_t$
$\mathbf{h}_t$
$W_o\overline{\mathbf{z}}$
σ
"… Já" previous output
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
current output
"live" "Já"
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)
"I live with my parents, …"

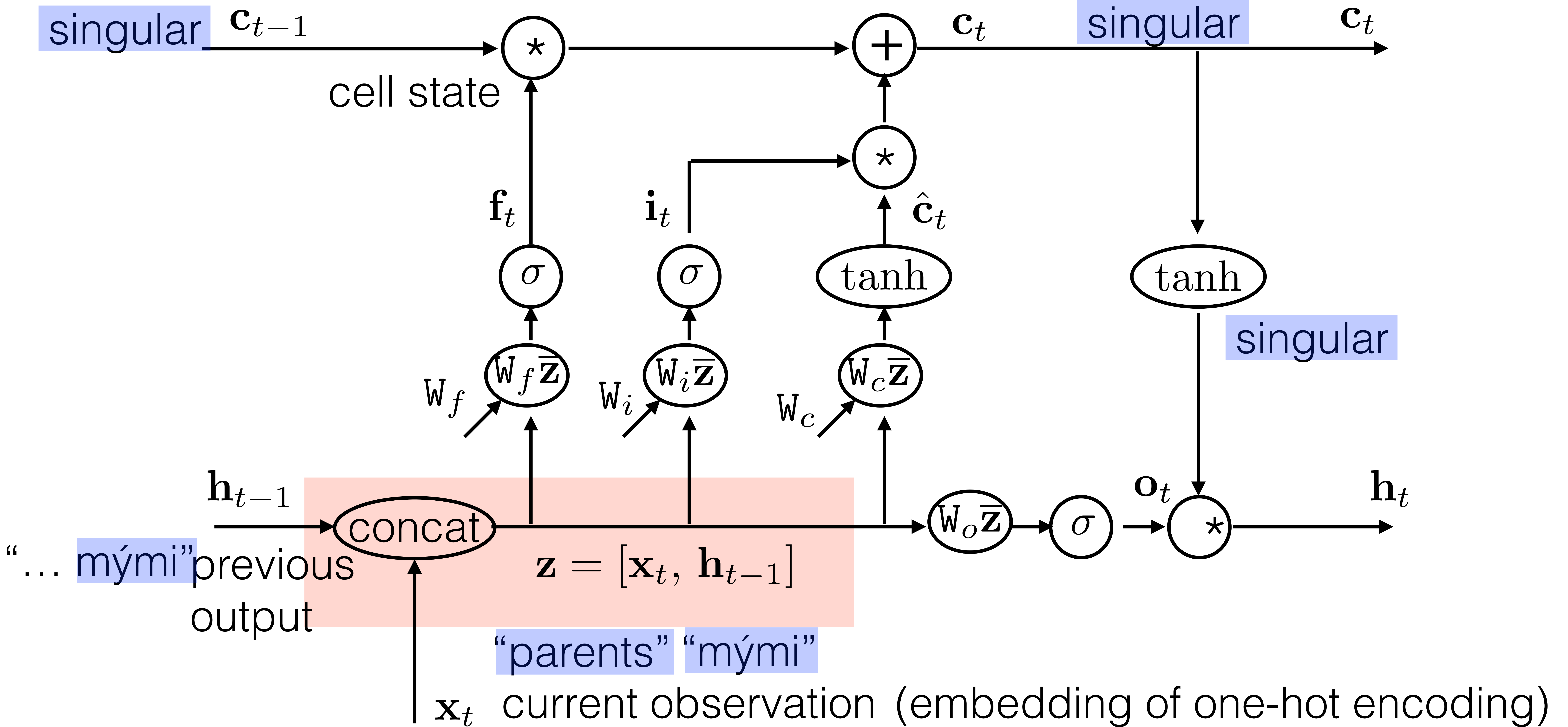
LSTM block

cell state is long term memory= vector [**singular**/plural, feminine/masculine, case, …]



LSTM block

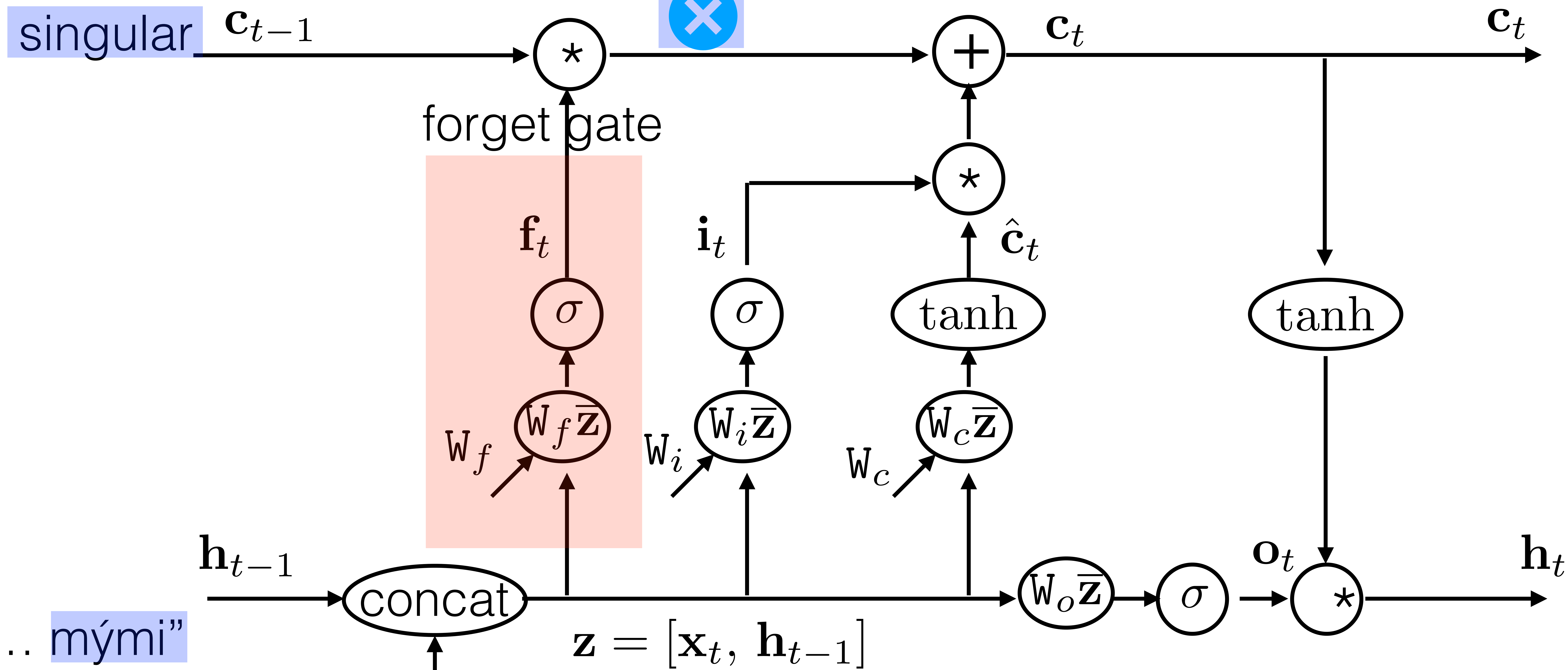
cell state is long term memory= vector [**singular**/plural, feminine/masculine, case, …]



"I live with my parents, …"

LSTM block

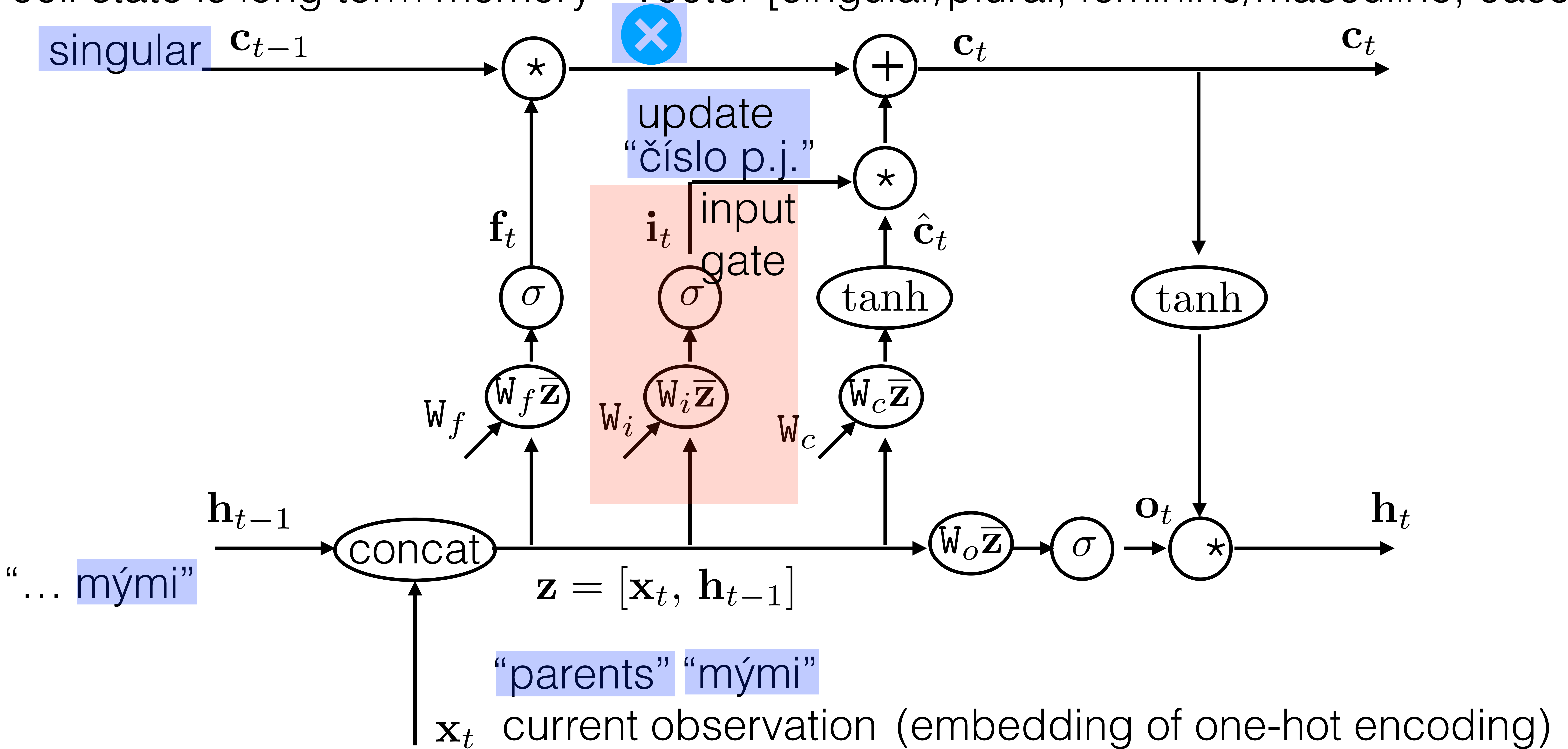
cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]



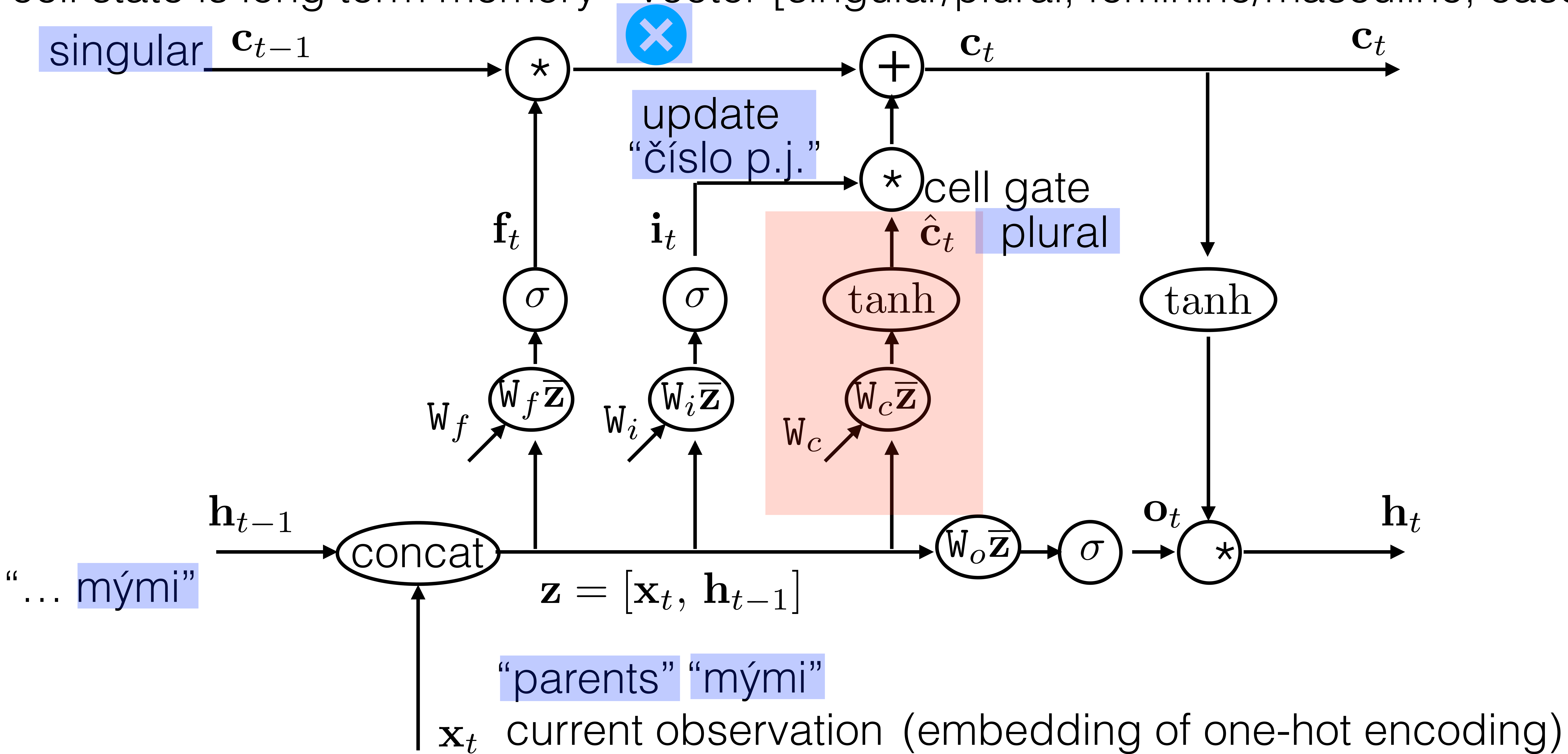
“parents” “mými”

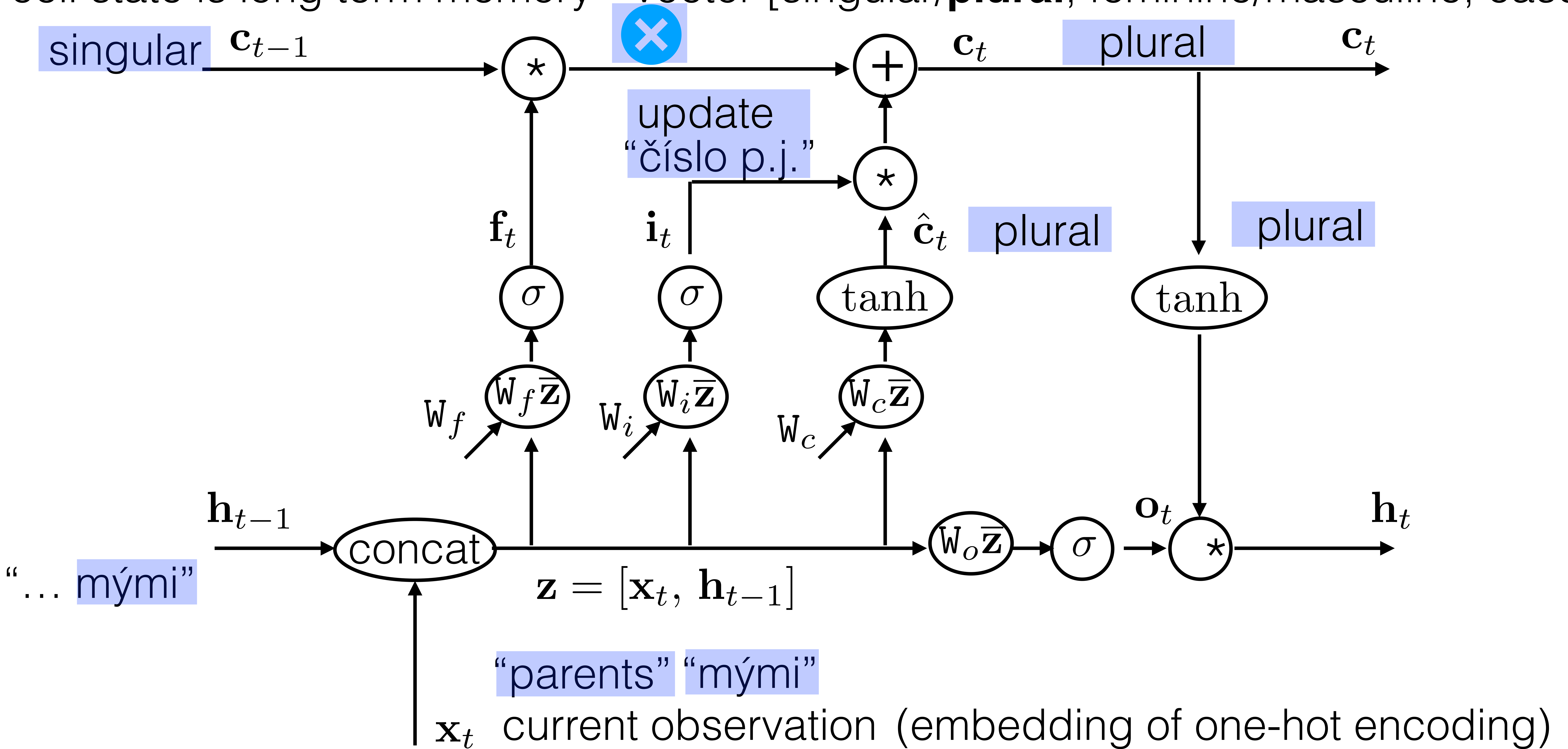
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)

“I live with my parents, …”



LSTM block
cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]
singular
$\mathbf{c}_{t-1}$
$*$
$+$
$\mathbf{c}_t$
$\mathbf{c}_t$
update
“číslo p.j.”
$*$
input
gate
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
σ
σ
tanh
tanh
$W_f\overline{\mathbf{z}}$
$W_i\overline{\mathbf{z}}$
$W_c\overline{\mathbf{z}}$
W_f
W_i
W_c
$\mathbf{h}_{t-1}$
concat
$W_o\overline{\mathbf{z}}$
σ
$\mathbf{o}_t$
$*$
$\mathbf{h}_t$
“… mými”
$\mathbf{z} = [\mathbf{x}_t,\ \mathbf{h}_{t-1}]$
“parents” “mými”
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)
“I live with my parents, …”

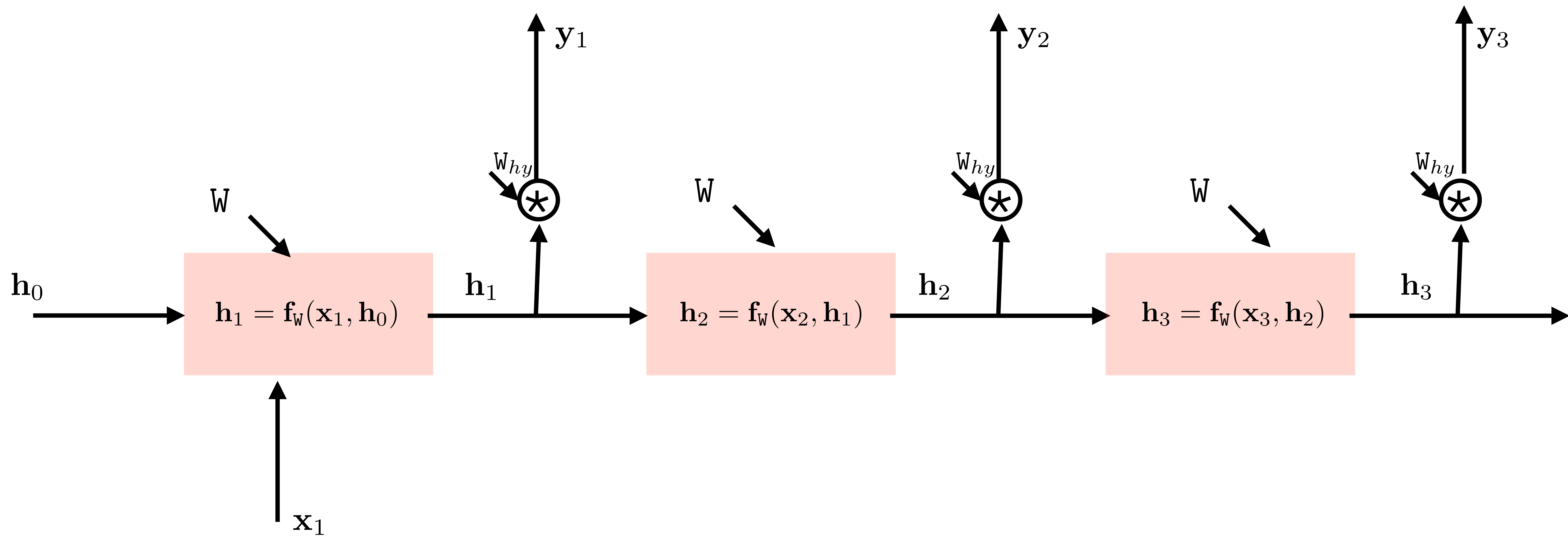

LSTM block
cell state is long term memory= vector [singular/plural, feminine/masculine, case, …]
singular
$\mathbf{c}_{t-1}$
$\mathbf{c}_t$
$\mathbf{c}_t$
update
“číslo p.j.”
cell gate
plural
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
tanh
tanh
$\mathtt{W}_f \overline{\mathbf{z}}$
$\mathtt{W}_i \overline{\mathbf{z}}$
$\mathtt{W}_c \overline{\mathbf{z}}$
$\mathtt{W}_f$
$\mathtt{W}_i$
$\mathtt{W}_c$
$\mathbf{h}_{t-1}$
concat
$\mathtt{W}_o \overline{\mathbf{z}}$
$\mathbf{o}_t$
$\mathbf{h}_t$
“… mými”
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
“parents” “mými”
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)
“I live with my parents, …”


LSTM block
cell state is long term memory= vector [singular/**plural**, feminine/masculine, case, …]
singular
$\mathbf{c}_{t-1}$
*
+
$\mathbf{c}_t$
plural
$\mathbf{c}_t$
update
"číslo p.j."
*
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
plural
plural
σ
σ
tanh
tanh
$W_f\bar{\mathbf{z}}$
$W_i\bar{\mathbf{z}}$
$W_c\bar{\mathbf{z}}$
W_f
W_i
W_c
$\mathbf{h}_{t-1}$
"… mými"
concat
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
$W_o\bar{\mathbf{z}}$
σ
$\mathbf{o}_t$
*
$\mathbf{h}_t$
"parents" "mými"
$\mathbf{x}_t$ current observation (embedding of one-hot encoding)
"I live with my parents, …"

LSTM block
cell state is long term memory= vector [singular/**plural**, feminine/masculine, case, …]
singular
$\mathbf{c}_{t-1}$
*
+
$\mathbf{c}_t$
plural
$\mathbf{c}_t$
update
“číslo p.j.”
*
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
plural
plural
σ
σ
tanh
tanh
$W_f\bar{\mathbf{z}}$
$W_i\bar{\mathbf{z}}$
$W_c\bar{\mathbf{z}}$
W_f
W_i
W_c
output gate
$\mathbf{h}_{t-1}$
concat
$\mathbf{o}_t$
$\mathbf{h}_t$
$W_o\bar{\mathbf{z}}$
σ
*
“… mými”
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
“… rodiči …”
“parents” “mými”
(embedding of one-hot encoding)
$\mathbf{x}_t$ current observation
“I live with my parents, …”

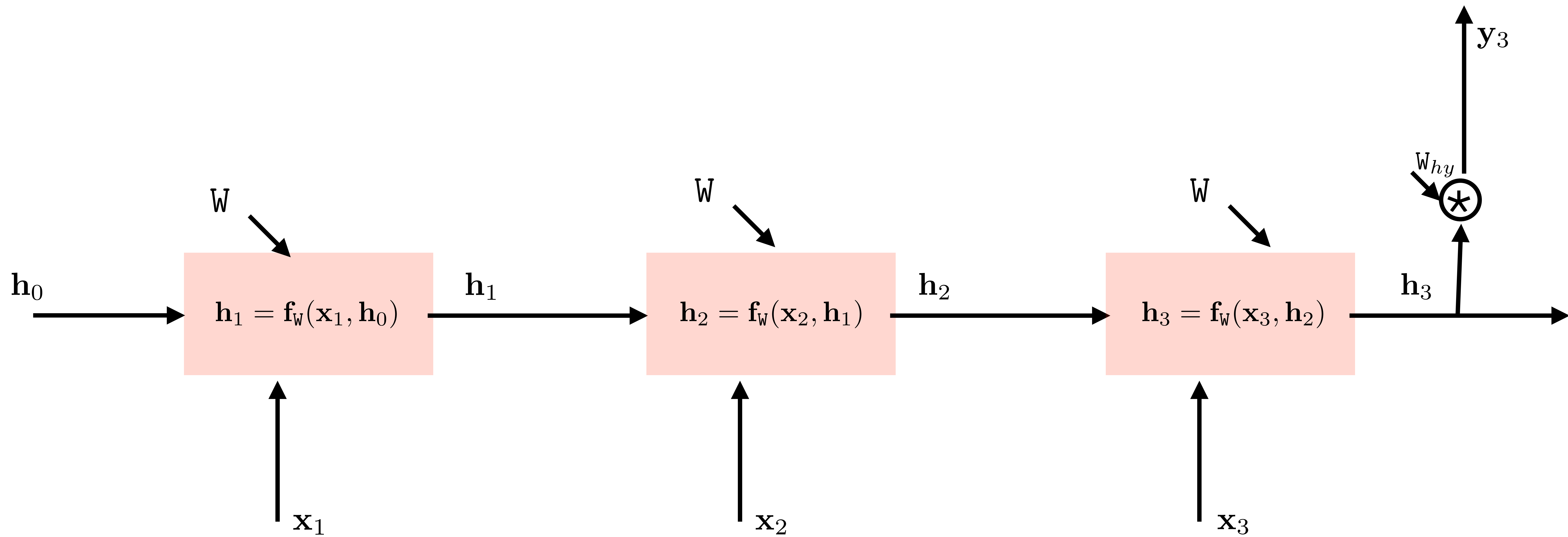
LSTM block
torch.nn.LSTM(input_size, hidden_dim, n_layers)
$\mathbf{c}_{t-1}$
$\mathbf{c}_t$
$\mathbf{c}_t$
gradient highway
$\mathbf{f}_t$
$\mathbf{i}_t$
$\hat{\mathbf{c}}_t$
σ
σ
tanh
tanh
$\mathtt{W}_f\overline{\mathbf{z}}$
$\mathtt{W}_i\overline{\mathbf{z}}$
$\mathtt{W}_c\overline{\mathbf{z}}$
$\mathtt{W}_f$
$\mathtt{W}_i$
$\mathtt{W}_c$
$\mathbf{h}_{t-1}$
concat
$\mathtt{W}_o\overline{\mathbf{z}}$
σ
$\mathbf{o}_t$
$\mathbf{h}_t$
$\mathbf{z} = [\mathbf{x}_t, \mathbf{h}_{t-1}]$
$\mathbf{x}_t$

RNN architectures: one-to-many



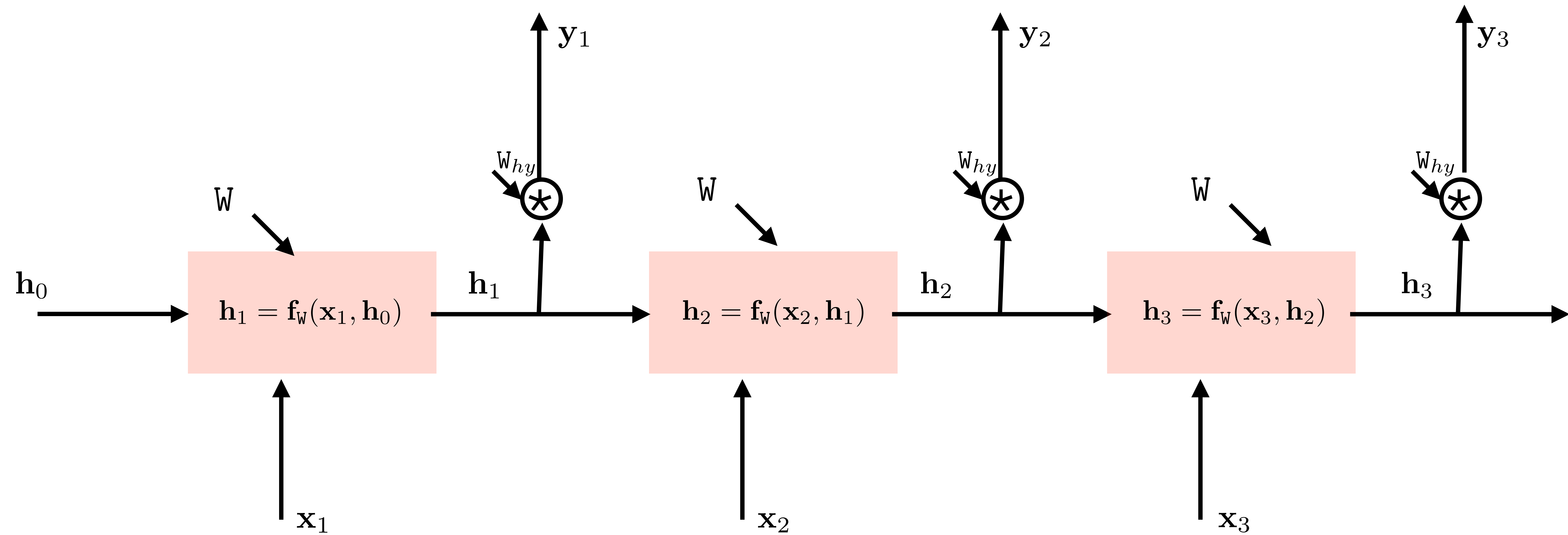
- Music generation
- Image captioning

RNN architectures: one-to-many



- Sentiment classification
- Action recognition

RNN architectures: one-to-many



- Coloring/enhancing video sequences
- Named-entity recognition
- Speech recognition

RNN architectures: one-to-many

- Machine translation
- Question answering

RNN architectures: many-to-many

output: variable-size sequence

input: variable-size sequence

context: fixed-size semantic summary of input sequence

RNN architectures: Machine translation

output: Czech sentence

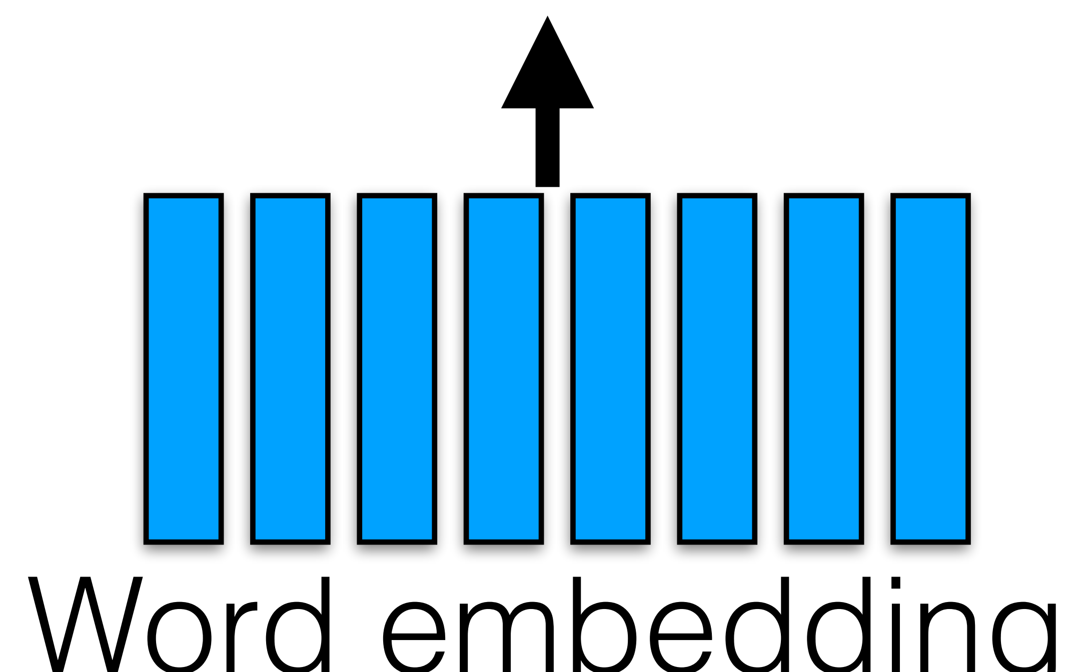
input: English sentence

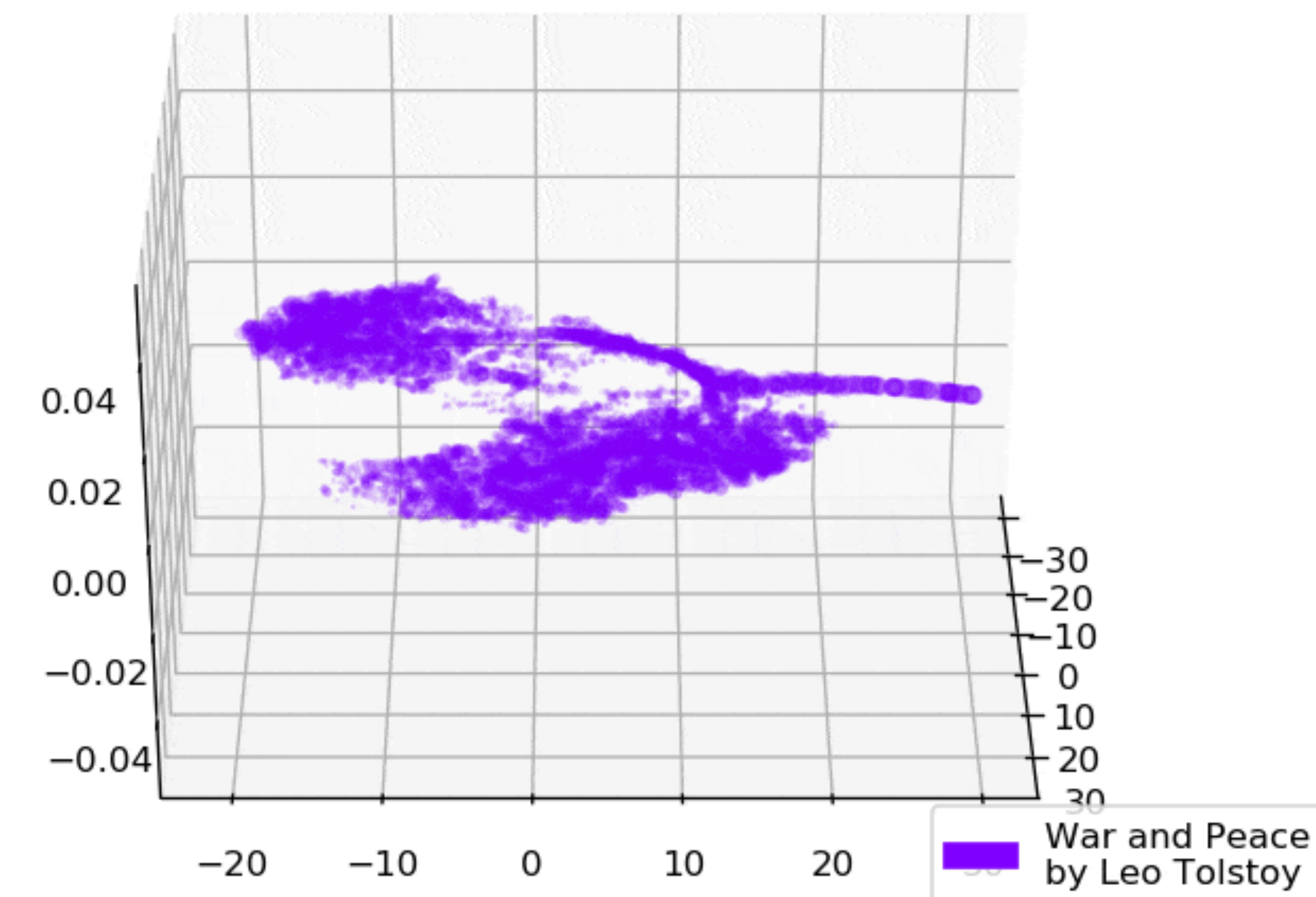
RNN can theoretically **remember everything** important but in practice it suffers from **catastrophic forgetting** (important relations could be very far).

Let's process the whole sentence at once through **transformer**!

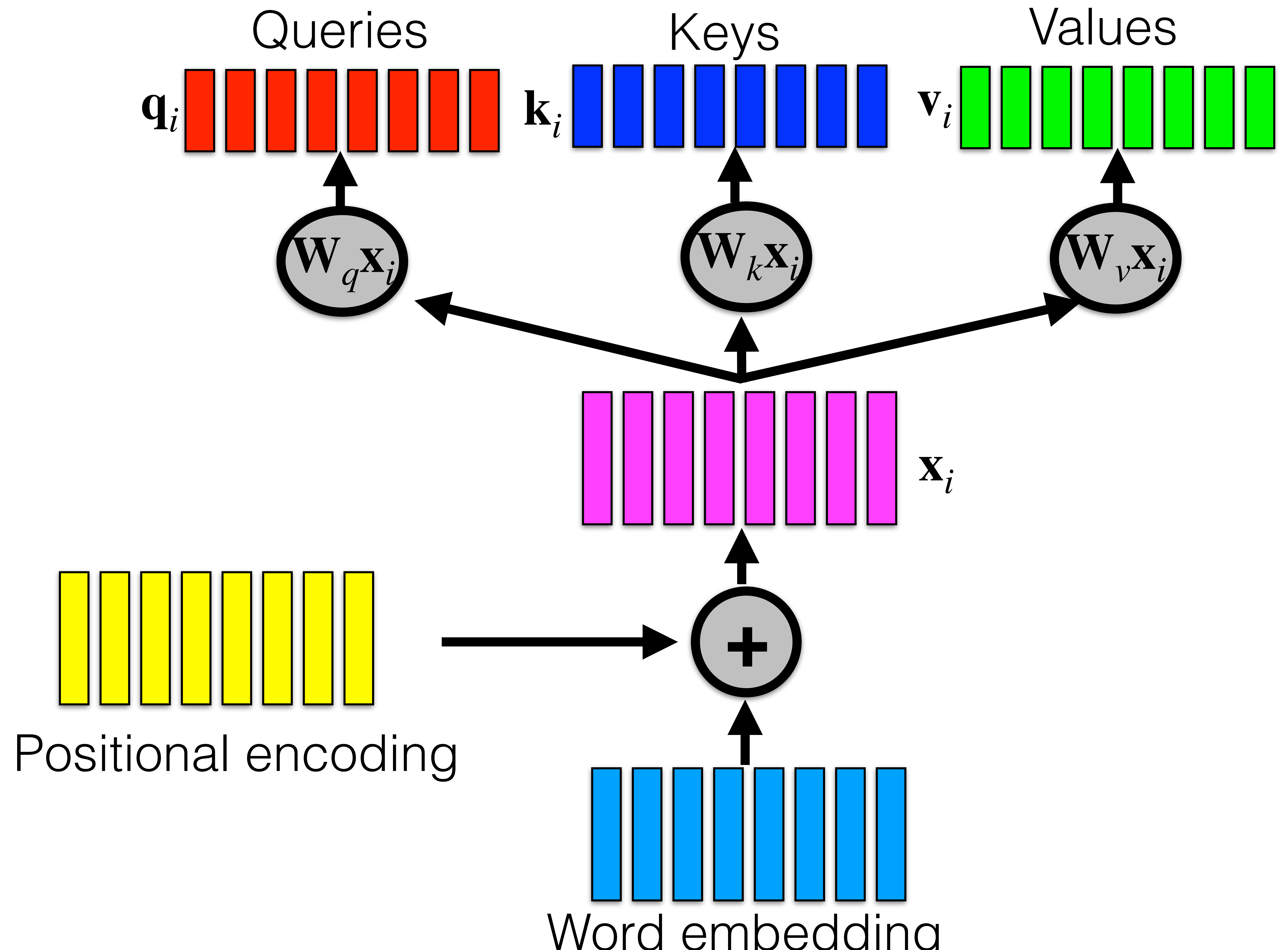
encoder

[Mikolov NIPS 2013]

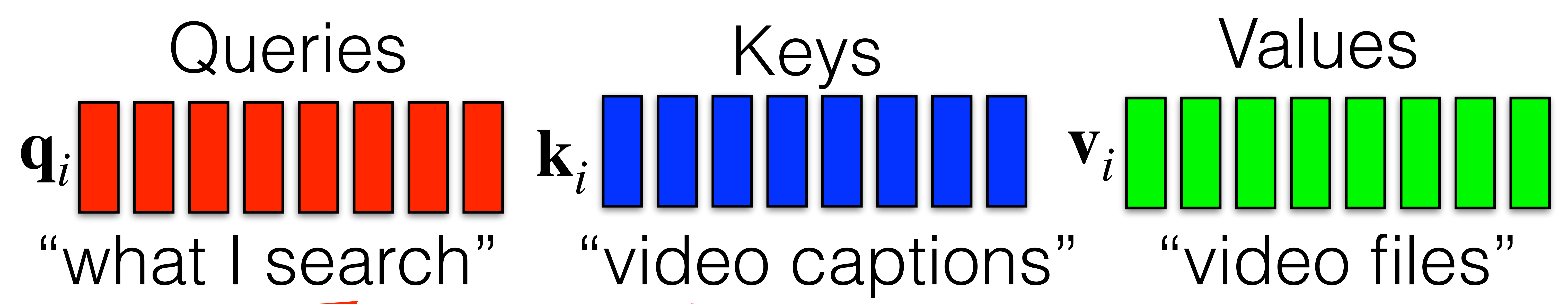




encoder



encoder



Scalar product measures similarity between vectors.

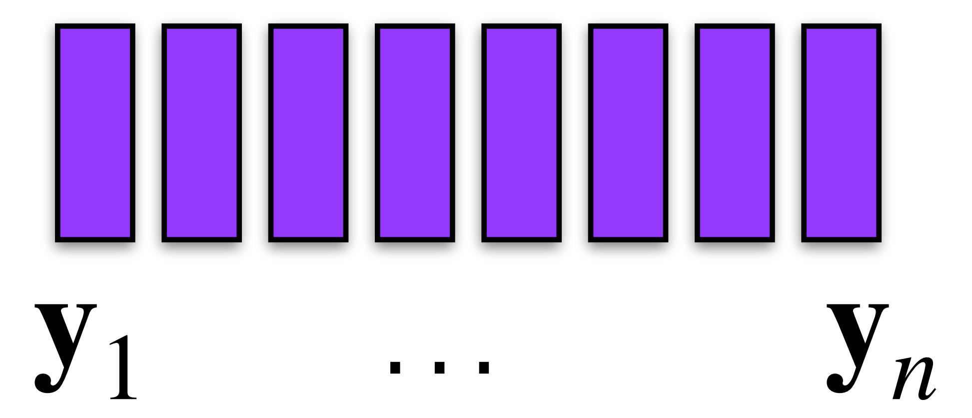
Get attention weights

$$\mathbf{s}\left(\mathbf{q}_i^\top \times \mathbf{k}_1 \dots \mathbf{k}_n\right) = \mathbf{a}_1 \dots \mathbf{a}_n$$

Attention-weighted sum of values

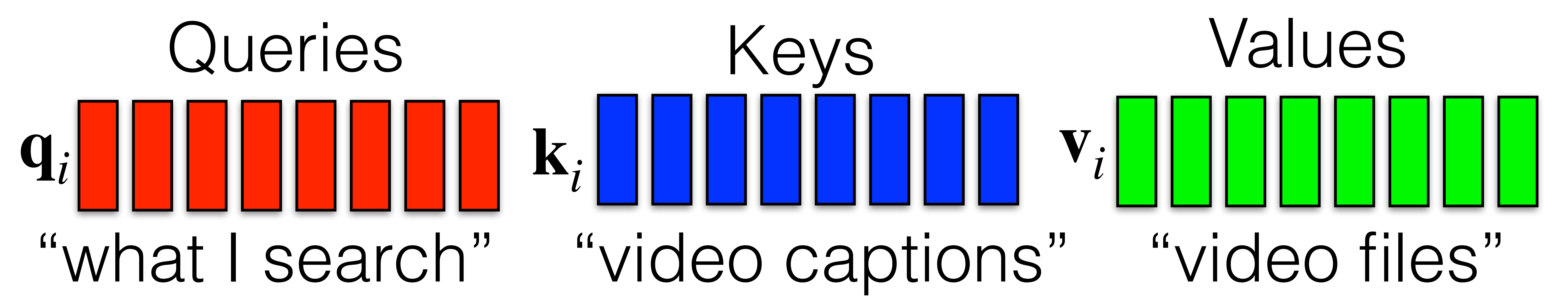
$$\mathbf{a}_1 \dots \mathbf{a}_n \quad \mathbf{v}_1 \vdots \mathbf{v}_n = \dots = \mathbf{y}_1$$

Outputs:

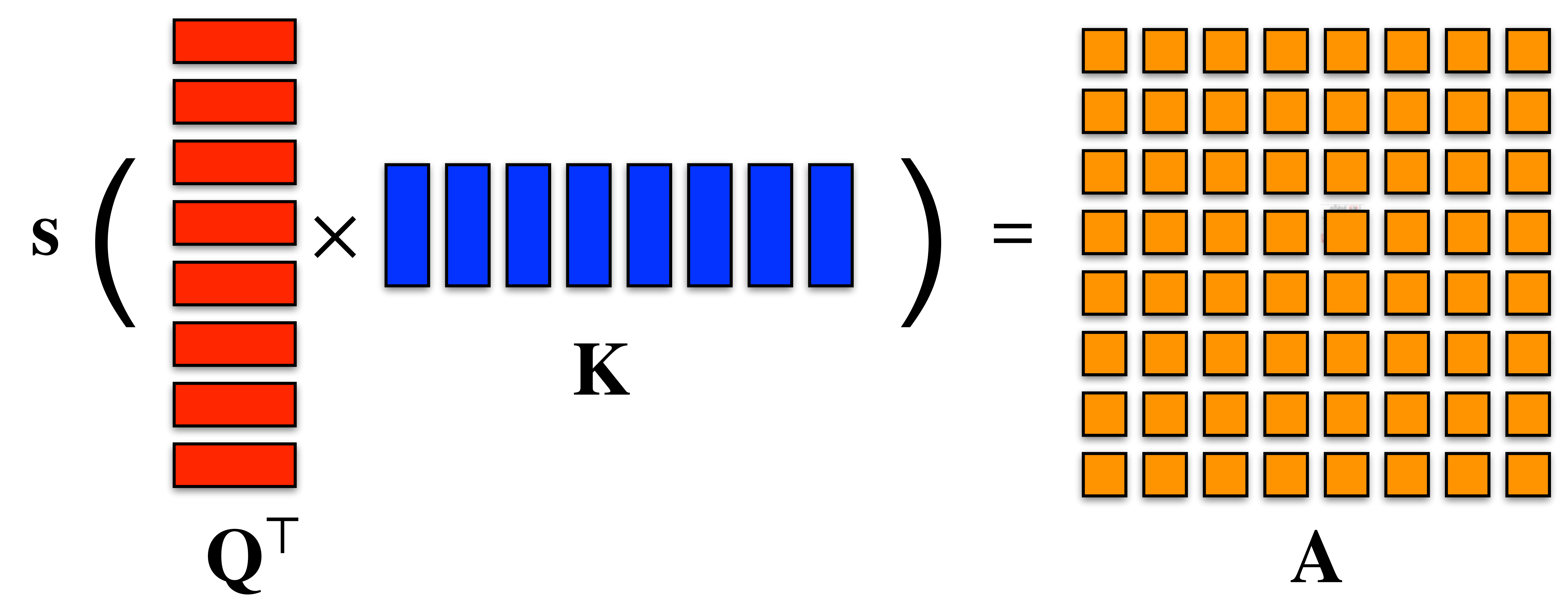


Avoid for-loop by smart matrix multiplication

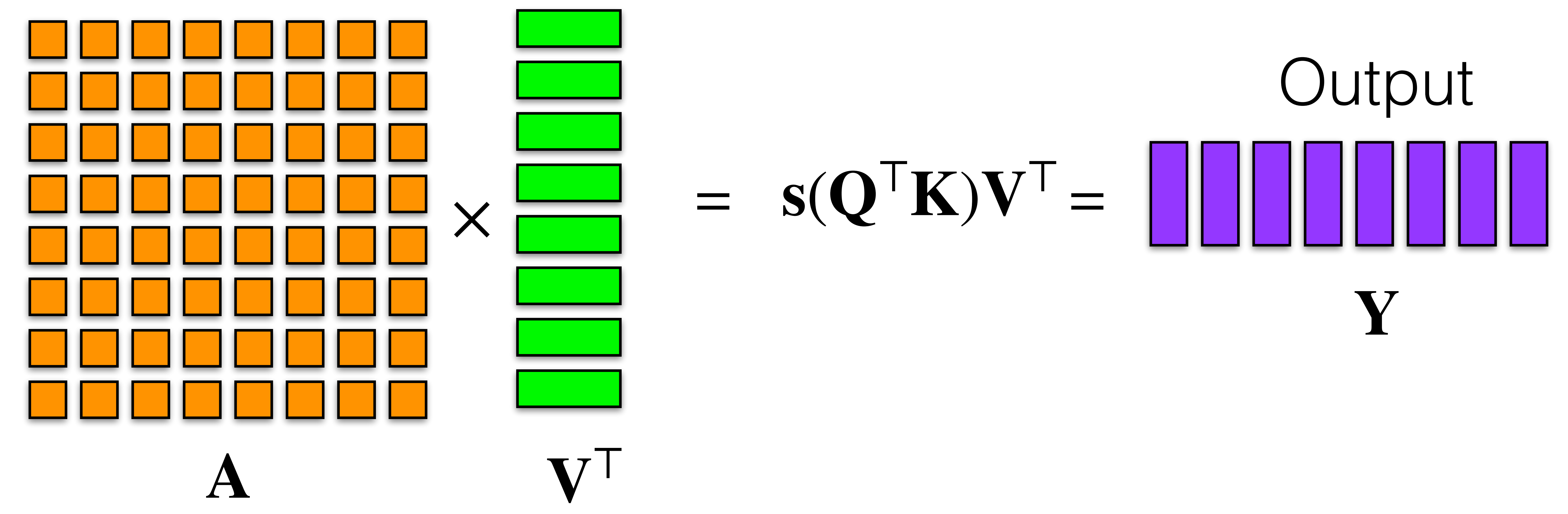
encoder



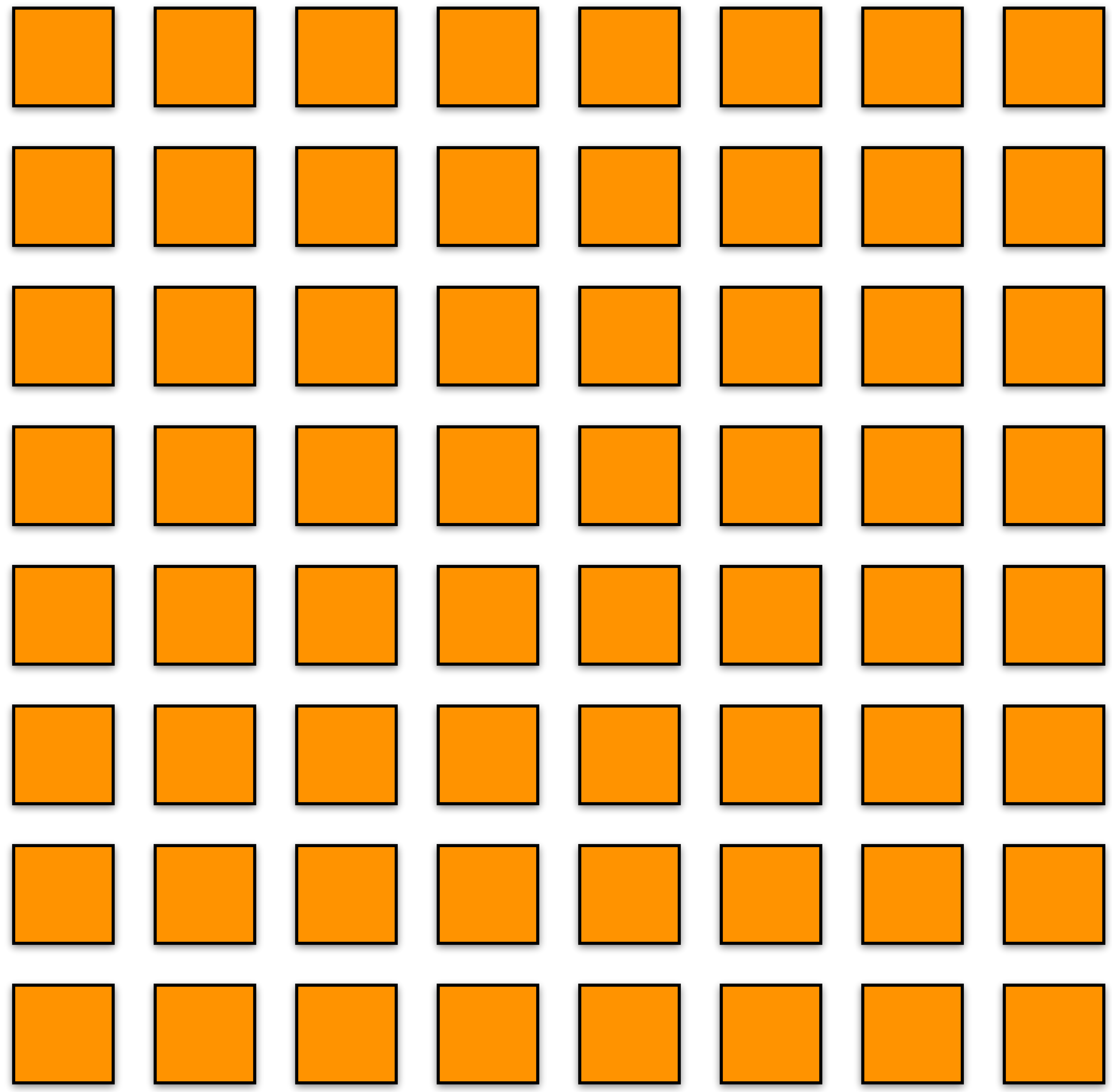
Get attention weights



Attention-weighted sum of values

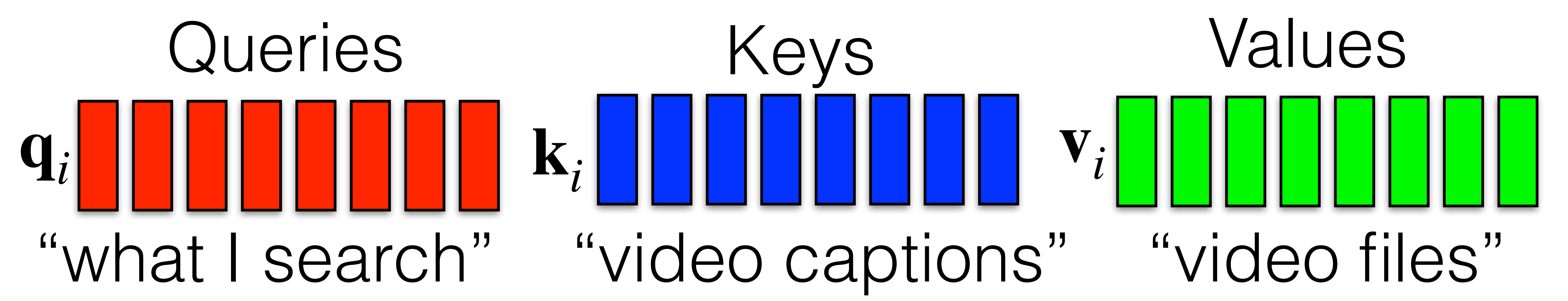


encoder

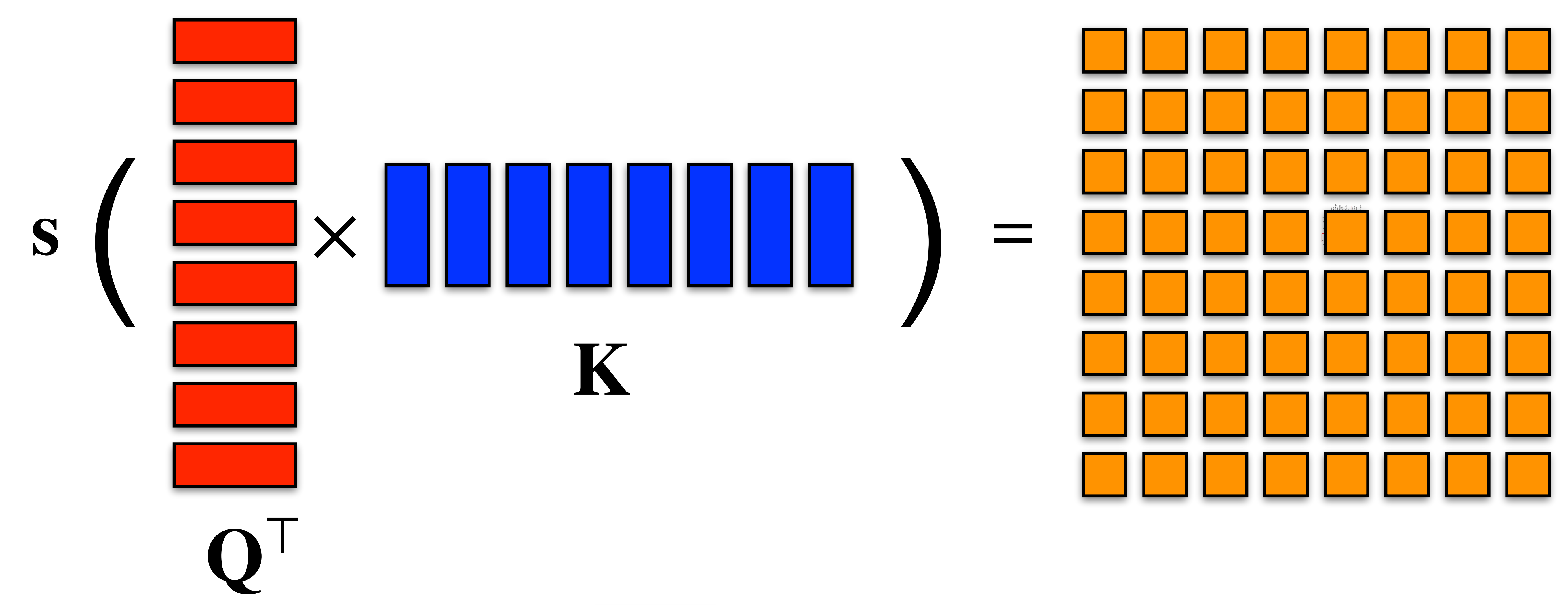


=

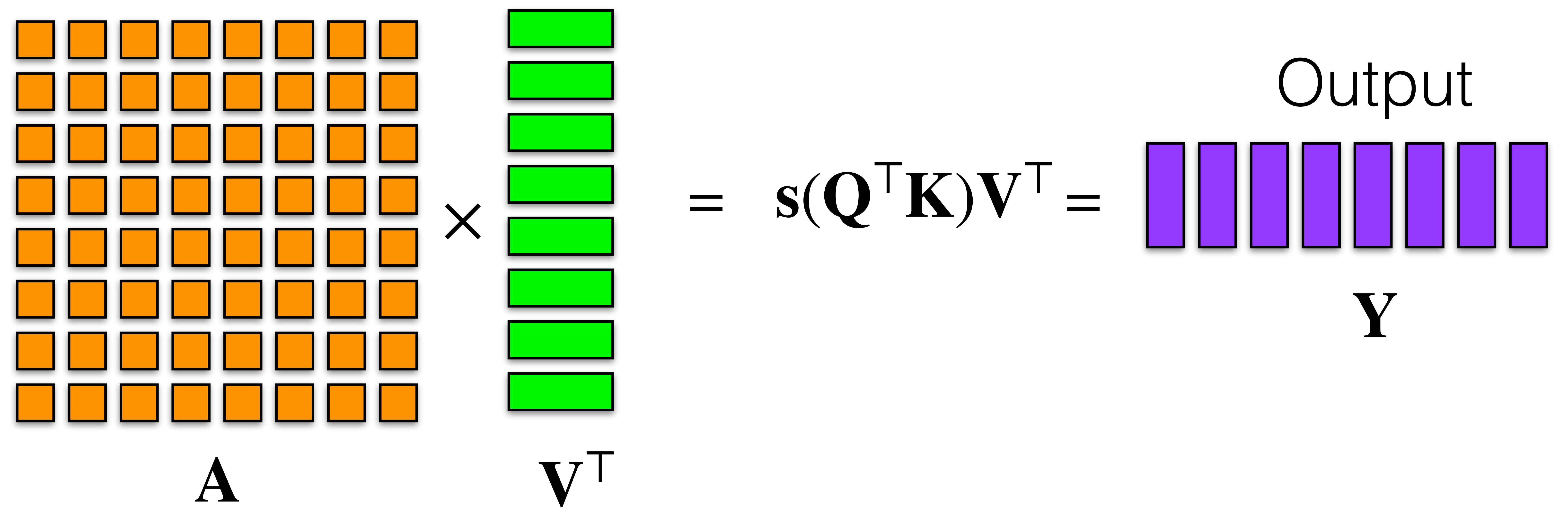
encoder



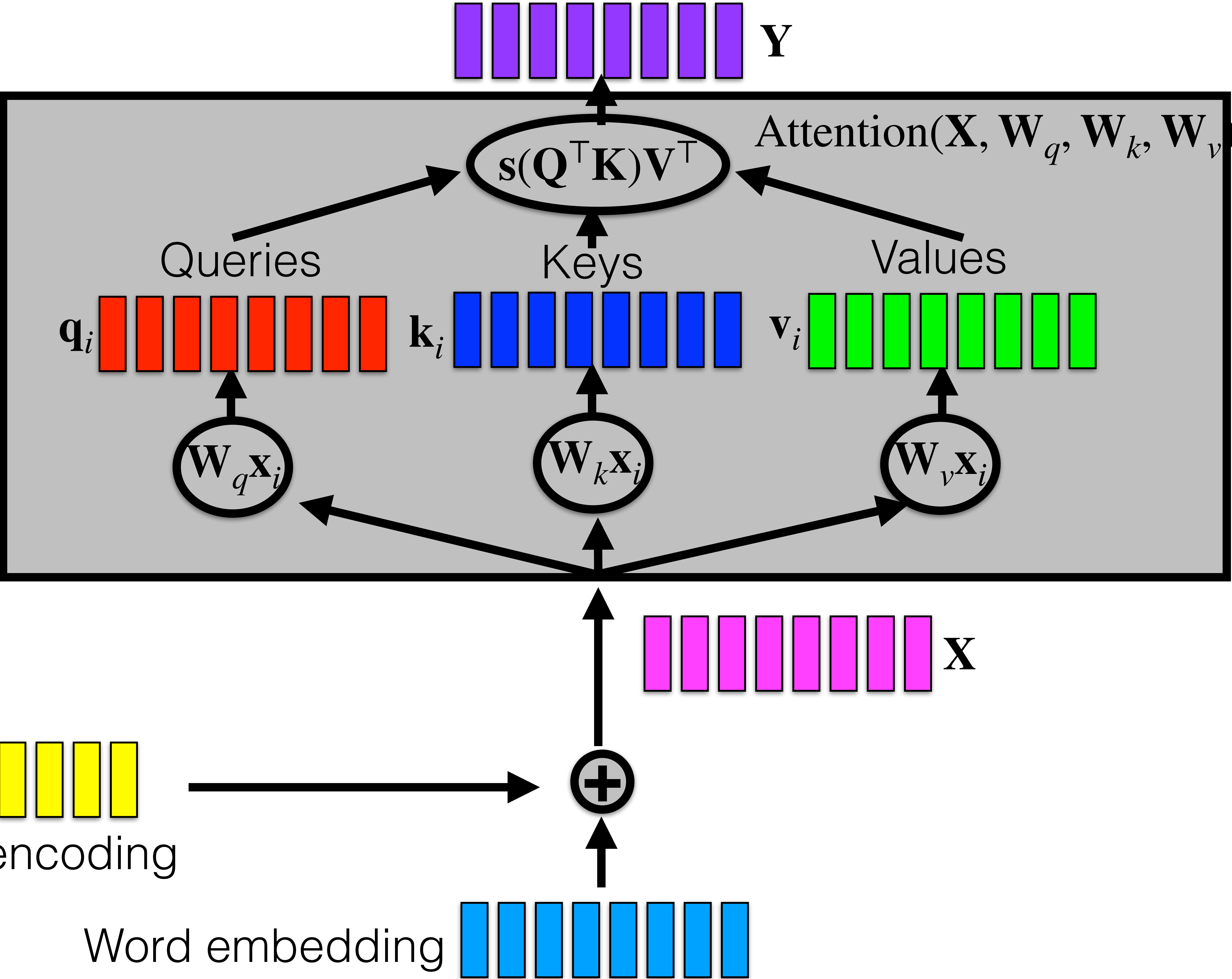
Get attention weights

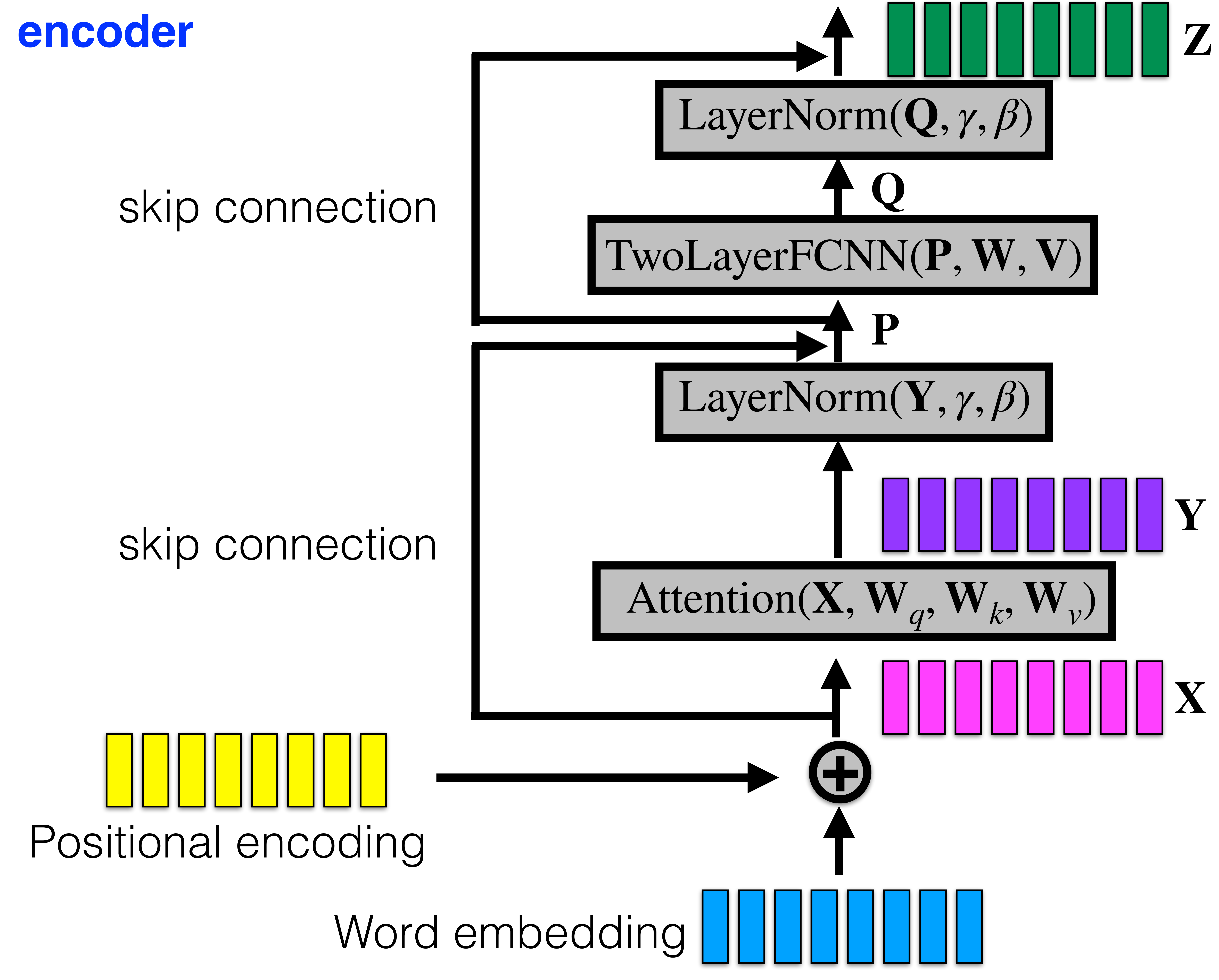


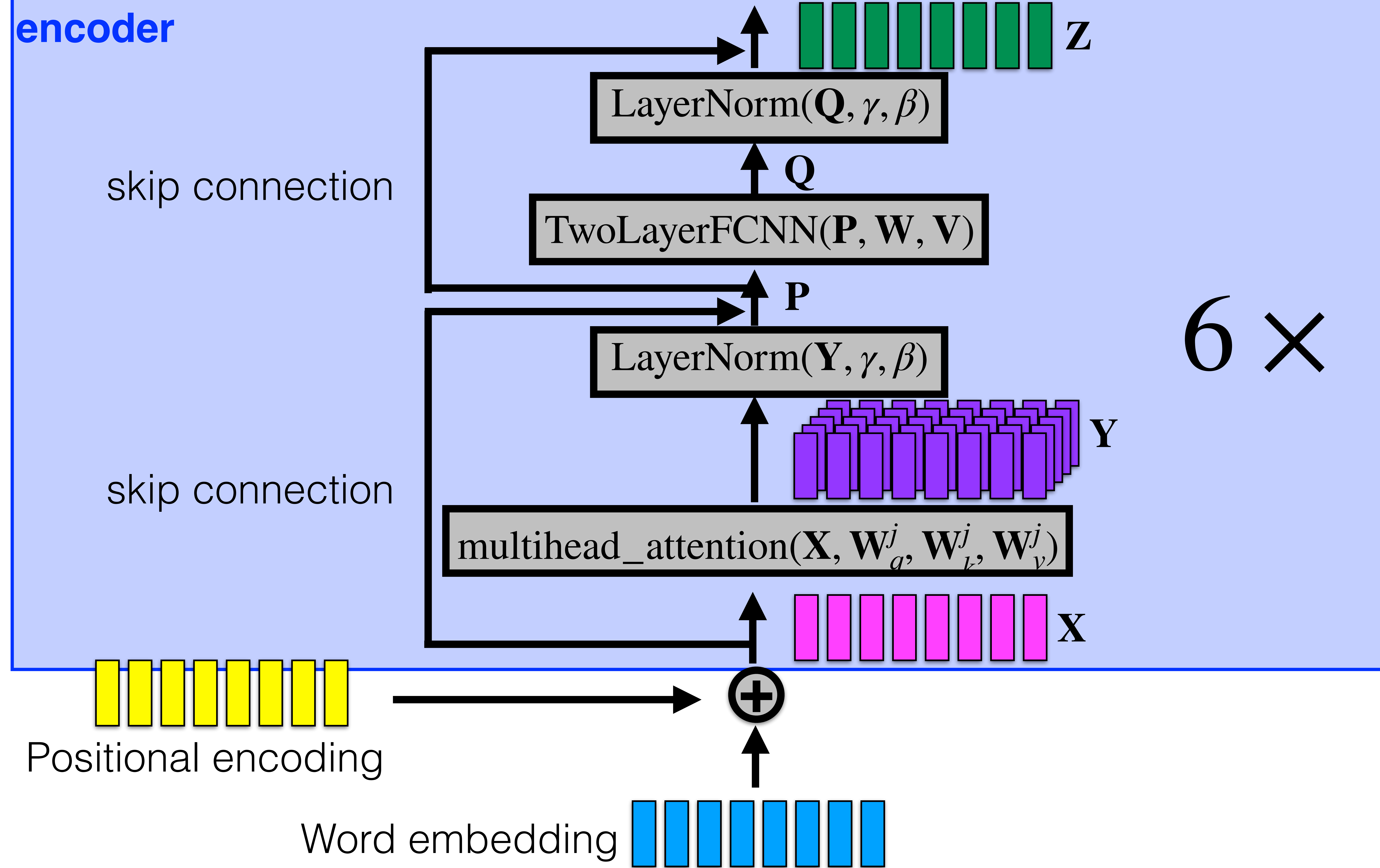
Attention-weighted sum of values



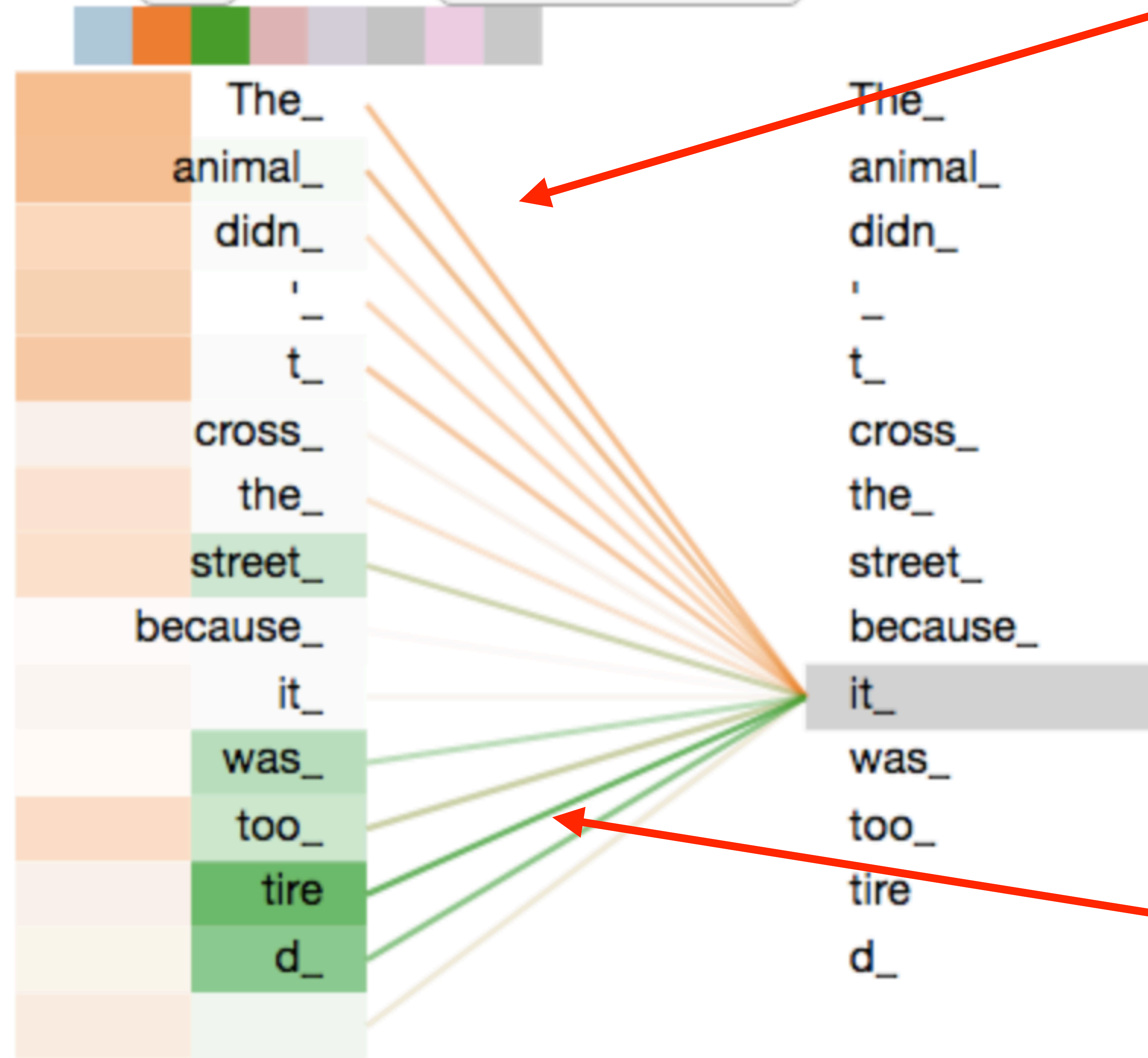
encoder




encoder
Z
LayerNorm(Q, γ, β)
skip connection
Q
TwoLayerFCNN(P, W, V)
P
LayerNorm(Y, γ, β)
Y
skip connection
Attention(X, W_q, W_k, W_v)
X
Positional encoding
Word embedding


encoder
Z
LayerNorm(Q, γ, β)
Q
skip connection
TwoLayerFCNN(P, W, V)
P
6 ×
LayerNorm(Y, γ, β)
Y
skip connection
multihead_attention(X, W_q^j, W_k^j, W_v^j)
X
Positional encoding
Word embedding

BertViz



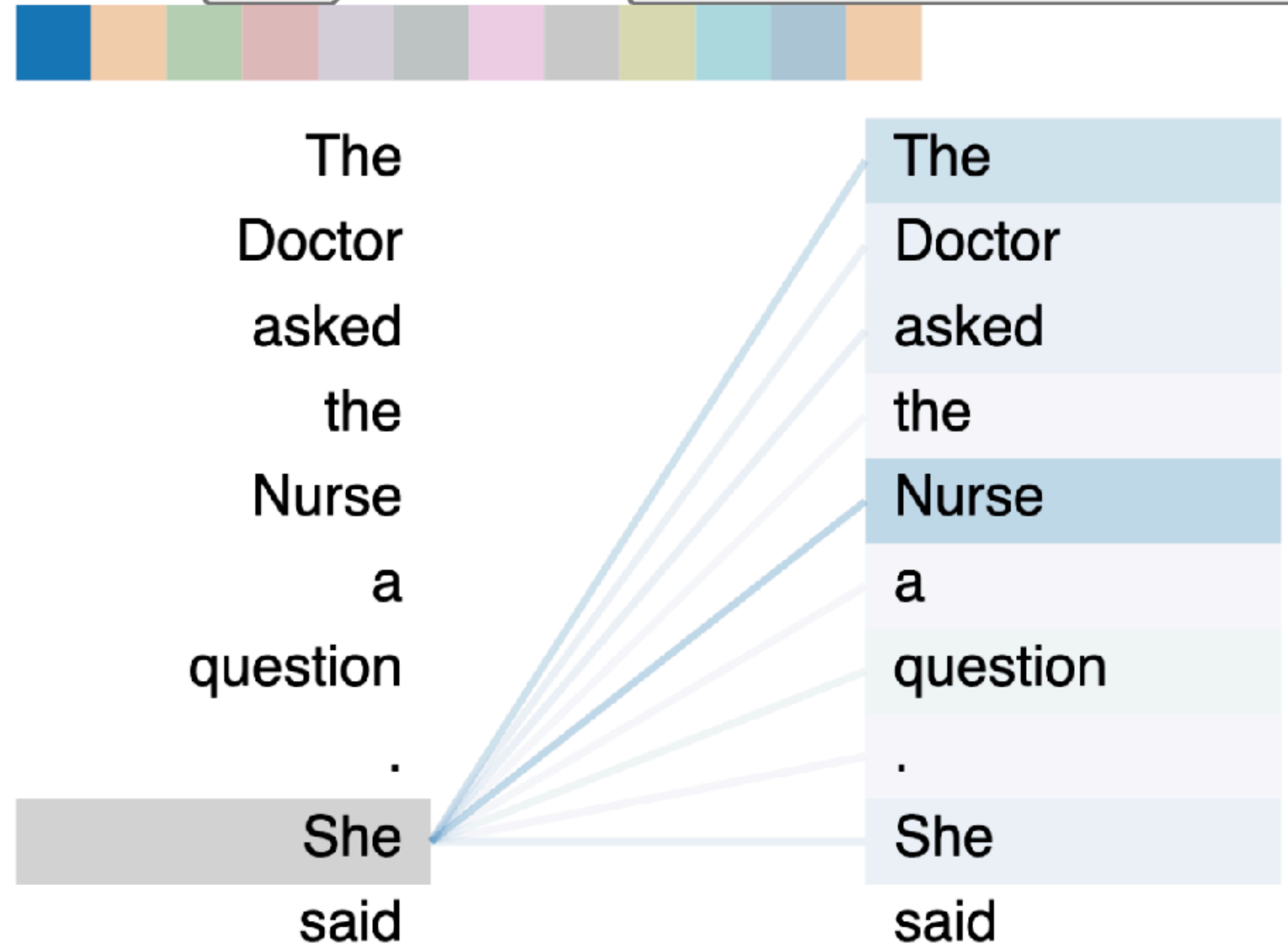
attention weights of **orange** attention head

"it=animal" vs "it=street"??

attention weights of **green** attention head

https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

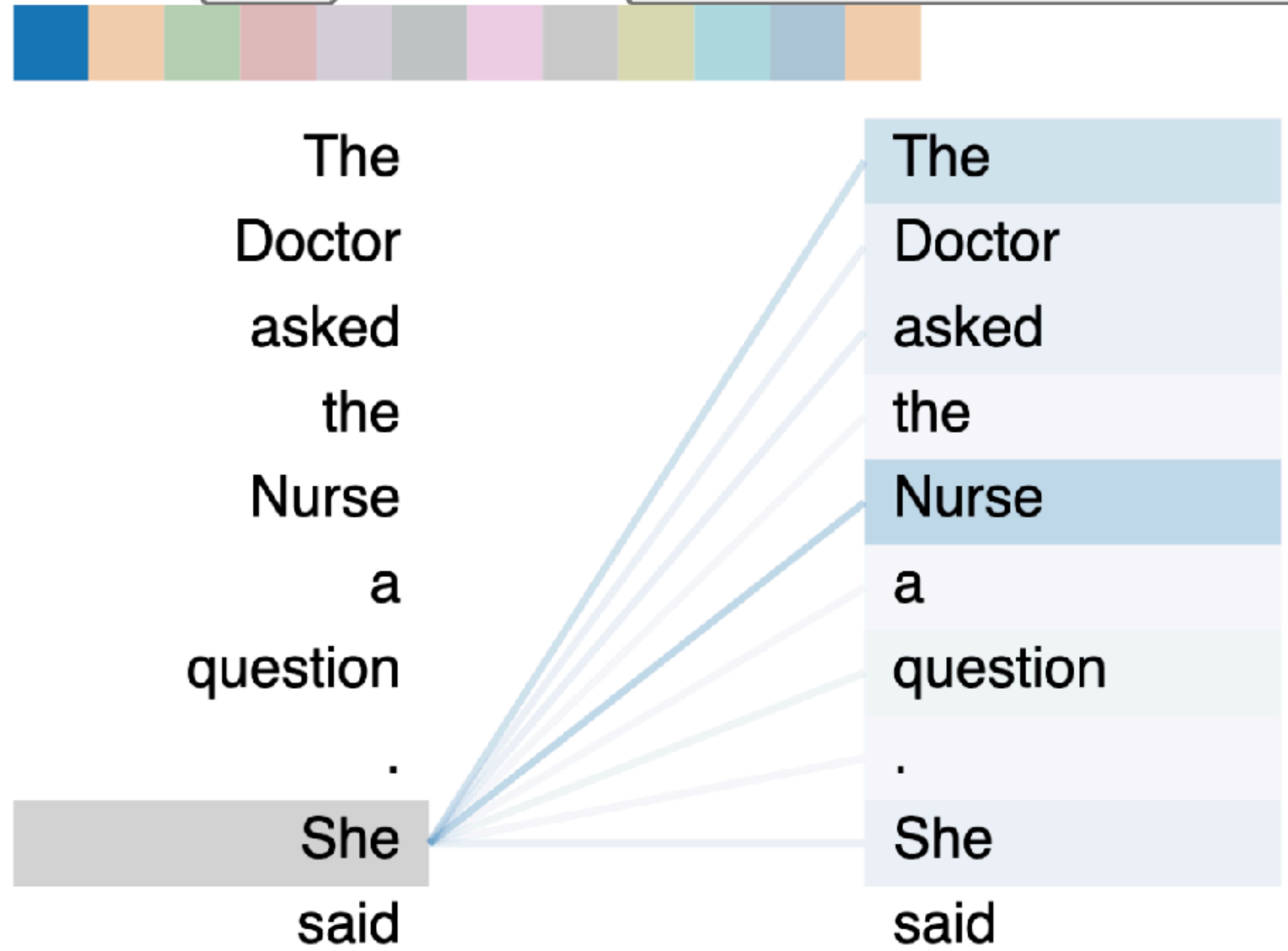
BertViz



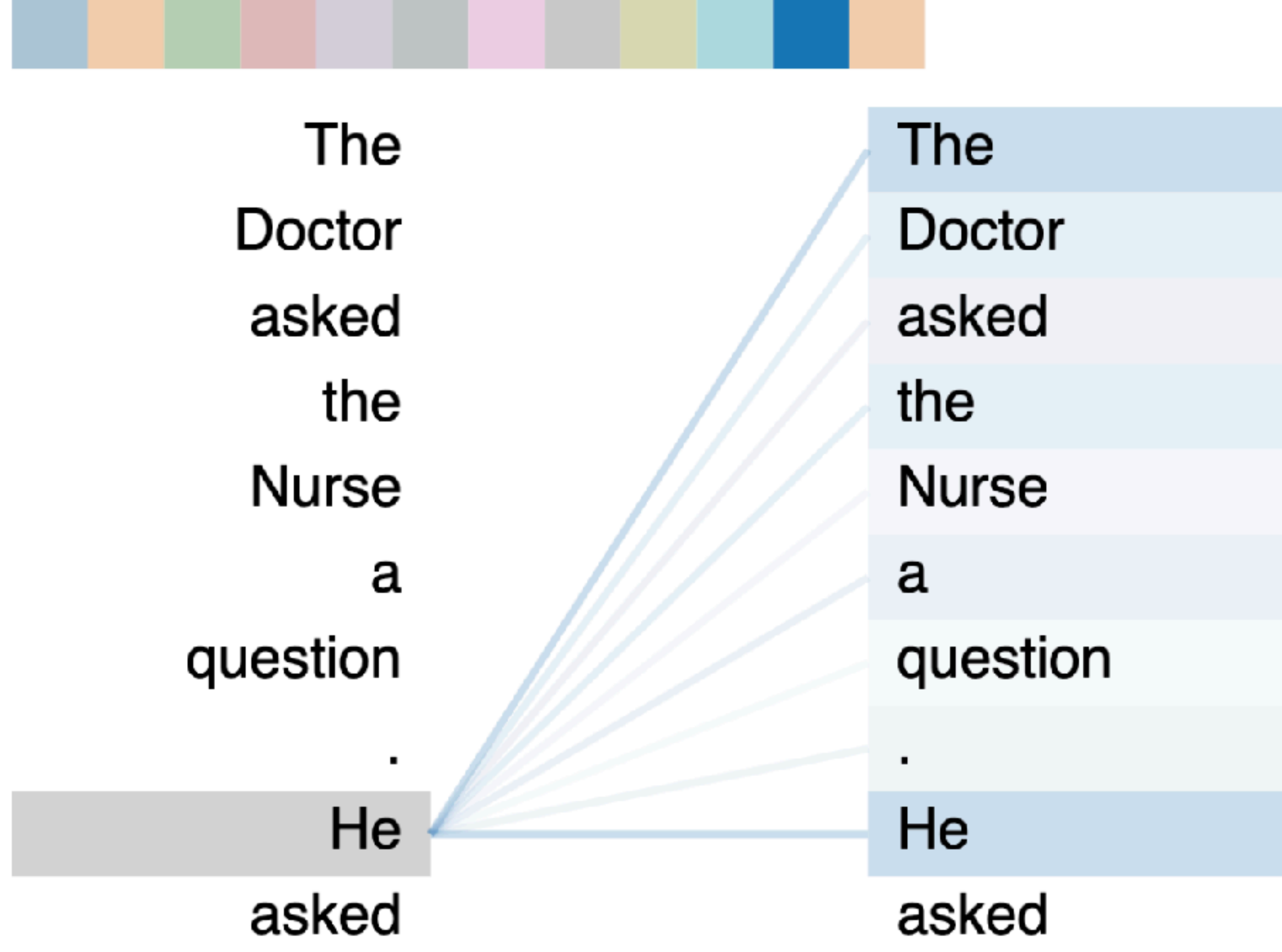
Model assumes “she=nurse”

https://www.comet.com/site/blog/explainable-ai-for-transformers/

BertViz (GPT2 model)



Model assumes "she=nurse"

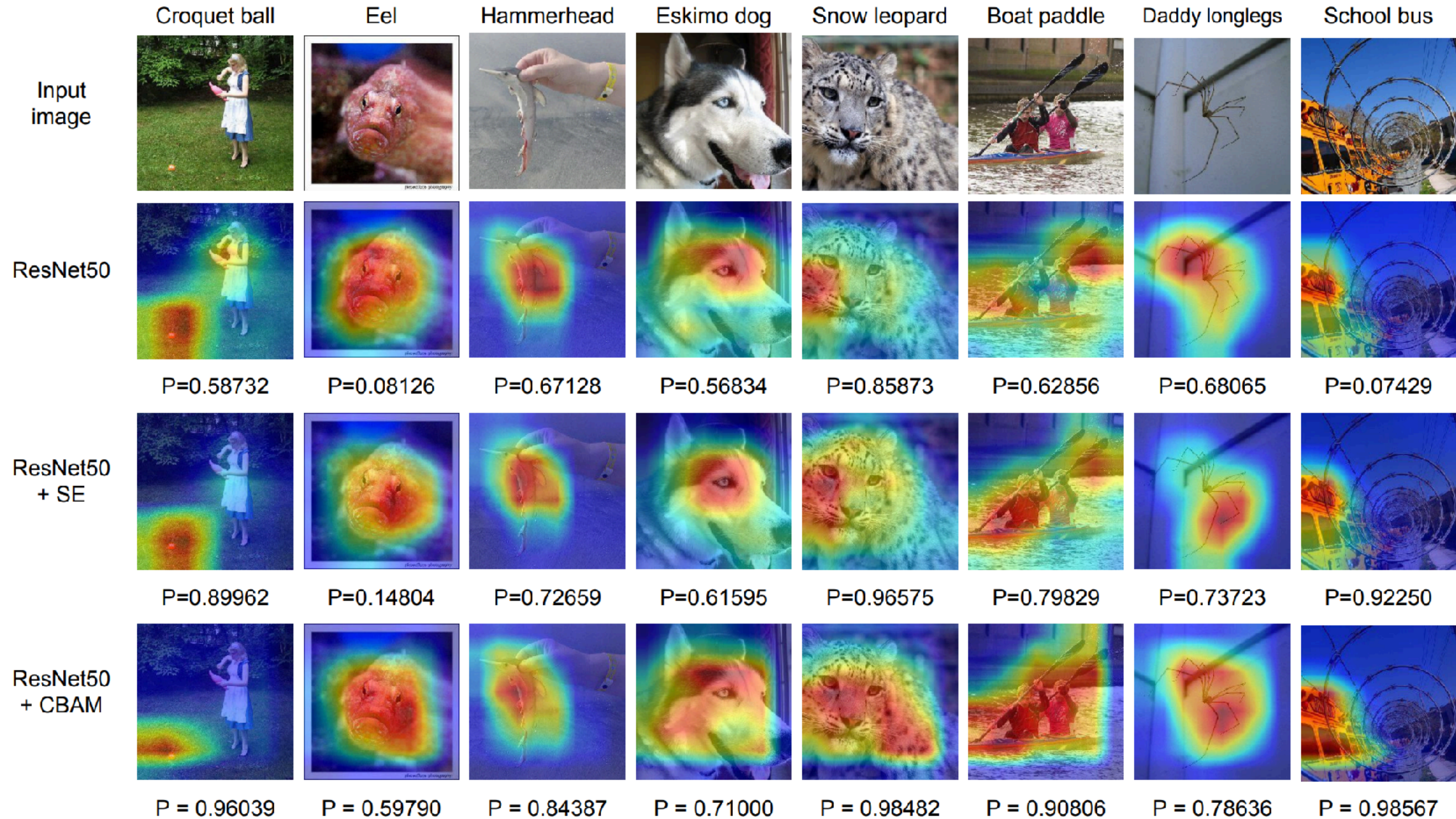


Model assumes "he=doctor"

https://www.comet.com/site/blog/explainable-ai-for-transformers/

Transformers and Attention in images

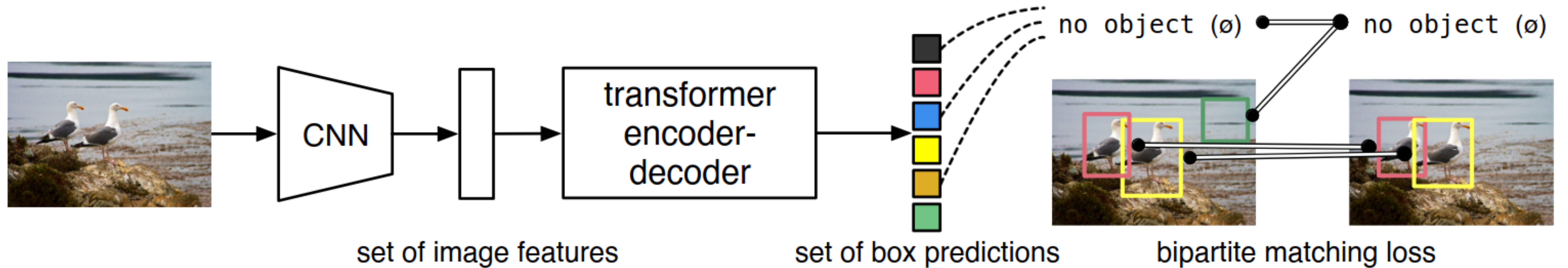
Attention modules [Woo et al,ECCV, 2018]
https://arxiv.org/pdf/1807.06521v2.pdf



Z
LayerNorm($\mathbf{Q}, \gamma, \beta$)
Q
skip connection
TwoLayerFCNN($\mathbf{P}, \mathbf{W}, \mathbf{V}$)
DEIT
[Touvron, Facebook AI 2021]
https://arxiv.org/pdf/2012.12877
P
LayerNorm($\mathbf{Y}, \gamma, \beta$)
Y
image
skip connection
multihead_attention($\mathbf{X}, \mathbf{W}_q^j, \mathbf{W}_k^j, \mathbf{W}_v^j$)
ConvNet
X
Positional encoding
feature map
Image embedding

DETR: transformers for object detection

https://arxiv.org/abs/2005.12872



- No anchors
- Output set of N bbs (ordering does not matter)
- Matching loss matches each predicted bb with ground truth bb, or enforces ∅

Attention in RL

Value function V(x)

Advantage function A(x,u)

Summary

- self-attention overfits (requires large dataset) => combining with hard explicit attention may work better
- memory is attention through time [Alex Graves 2020]
- pyTorch library: https://github.com/The-AI-Summer/self-attention-cv

```
model = MultiHeadSelfAttention(dim=64)
```

Future?

- Most of predictions were wrong
  - 1954 IBM predicted that natural language processing will be solved in 3 years
  - 1965 Herbert Simon: machines will replace humans in all manual works
  - 1970 Marvin Minsky: machines will have general AI comparable with humans
  - 2014 Rei Kurzweil: the same for for 2029, now talks about 2045

- Rodney Brooks prediction score card:
https://rodneybrooks.com/predictions-scorecard-2021-january-01/

- False generalization
  - AI is better in solving particular instances (image processing, stabilization)
  - Rather carefully isolated successes than exponentially growing general AI