Summary / Review

LING 575K Deep Learning for NLP

Shane Steinert-Threlkeld

June 1 2022

Today's Plan

- Survey of what we covered in the class
  - Core progression
  - Guest lectures
  - Assignments
- Some pointers to what's next
- Question time

Learning Objectives

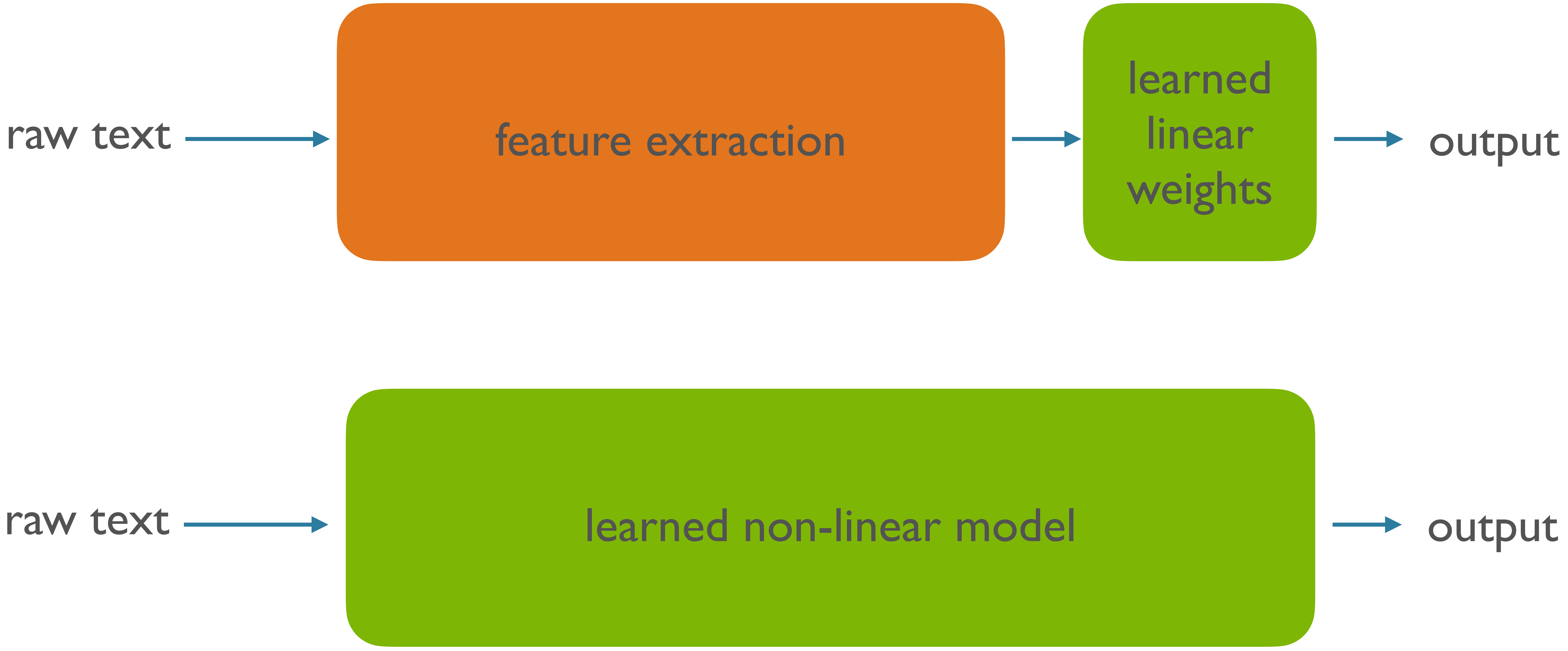
- Provide hands-on experience with building neural networks and using them for NLP tasks
- Theoretical understanding of building blocks
  - Computation graphs + gradient descent
  - Forward/backward API
    - Chain rule for computing gradients [backpropagation]
  - Various network architectures; their structure and biases

Topics Covered

Getting Started

- History
- Gradient descent optimization
  - Regularization, mini-batches, etc.
- Word vectors / word2vec
- Main tasks: classification (sentiment analysis), language modeling

Very potted history



The SGNS Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

The SGNS Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word
embedding

The SGNS Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word embedding

Context word embedding

The SGNS Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

Target word embedding

Similarity (dot-product)

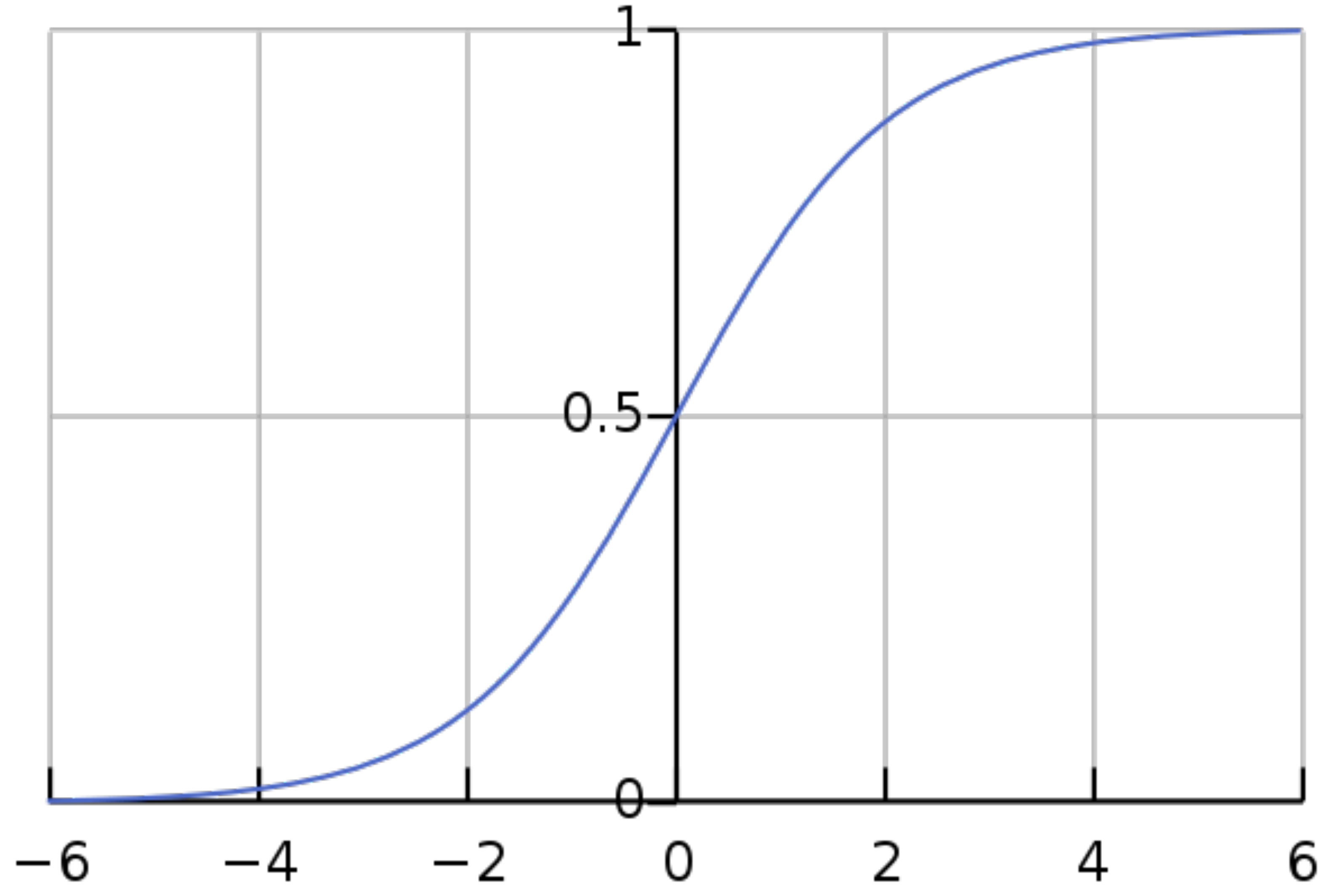
Context word embedding

The SGNS Model

$$P(1 \mid w, c) = \sigma\left(E_w \cdot C_c\right)$$

sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Target word embedding

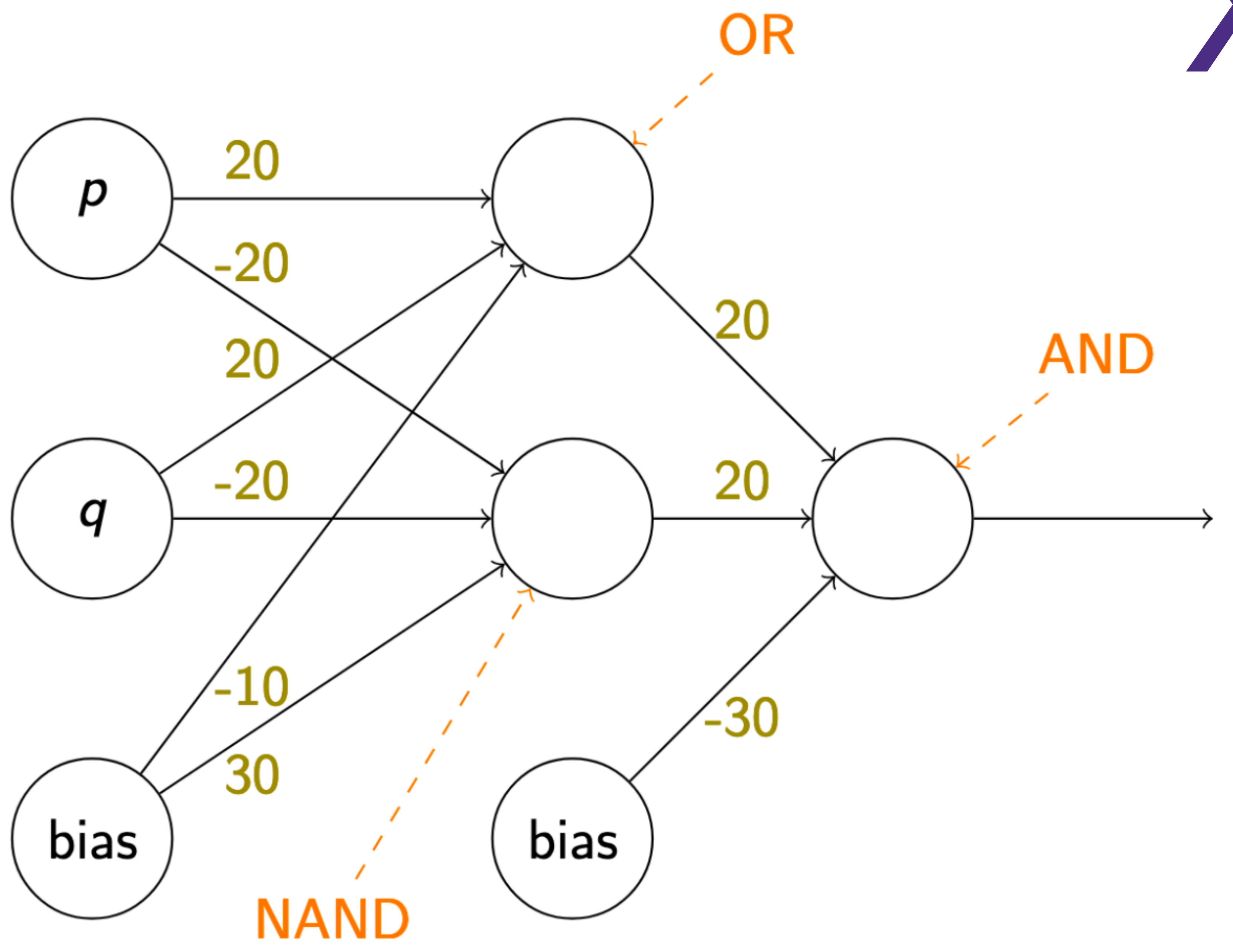
Similarity (dot-product)

Context word embedding

Neural Networks: Foundations

- Neural networks: intro
  - Expressive power / limitations
- Computation graph abstraction
- Backpropagation

XOR Network

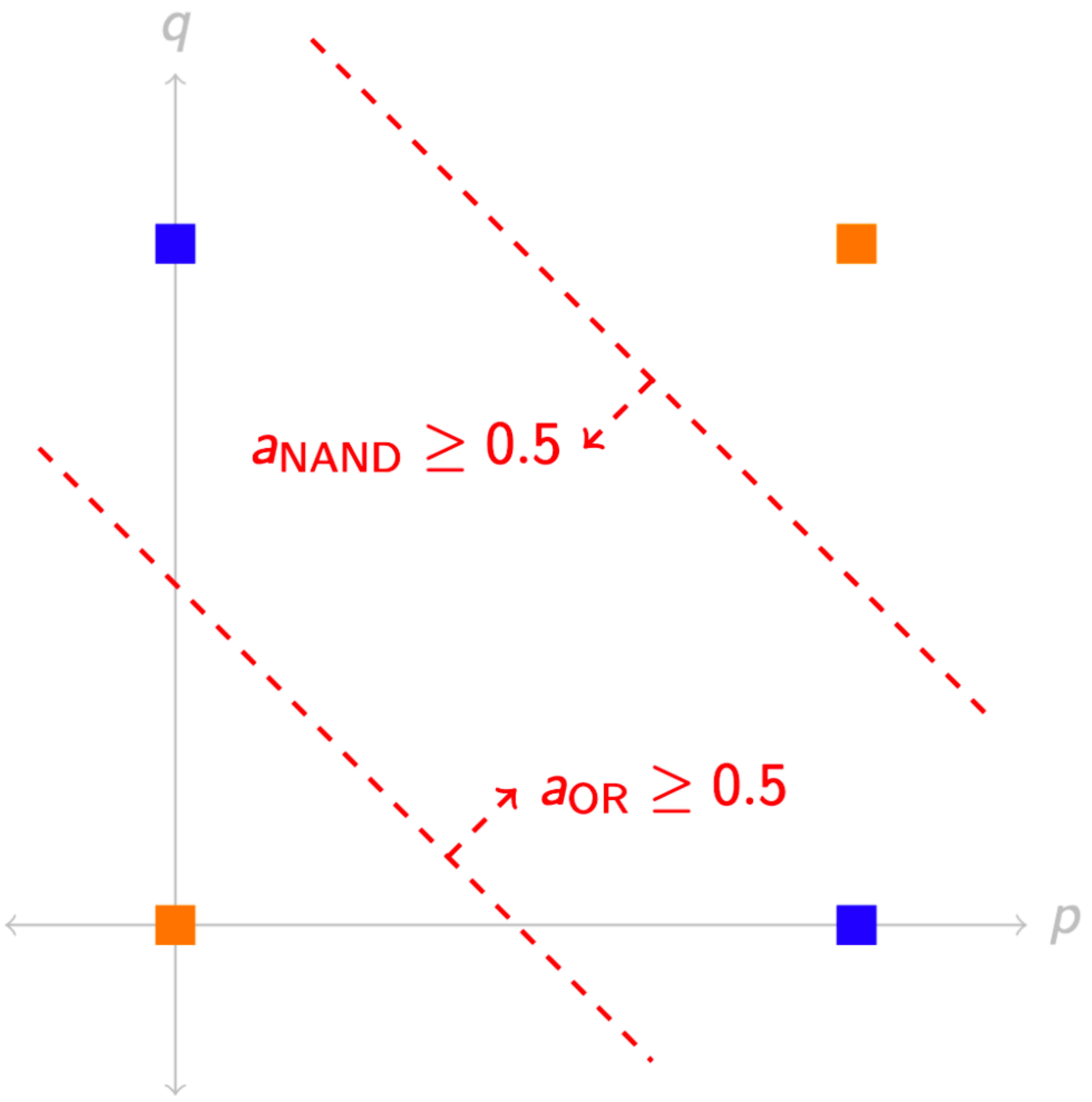


$$a_{\text{and}} = \sigma\left(w^{\text{and}}_{\text{or}} \cdot a_{\text{or}} + w^{\text{and}}_{\text{nand}} \cdot a_{\text{nand}} + b^{\text{and}}\right)$$

$$= \sigma\left(\begin{bmatrix} a_{\text{or}} & a_{\text{nand}} \end{bmatrix} \begin{bmatrix} w^{\text{and}}_{\text{or}} \\ w^{\text{and}}_{\text{nand}} \end{bmatrix} + b^{\text{and}}\right)$$

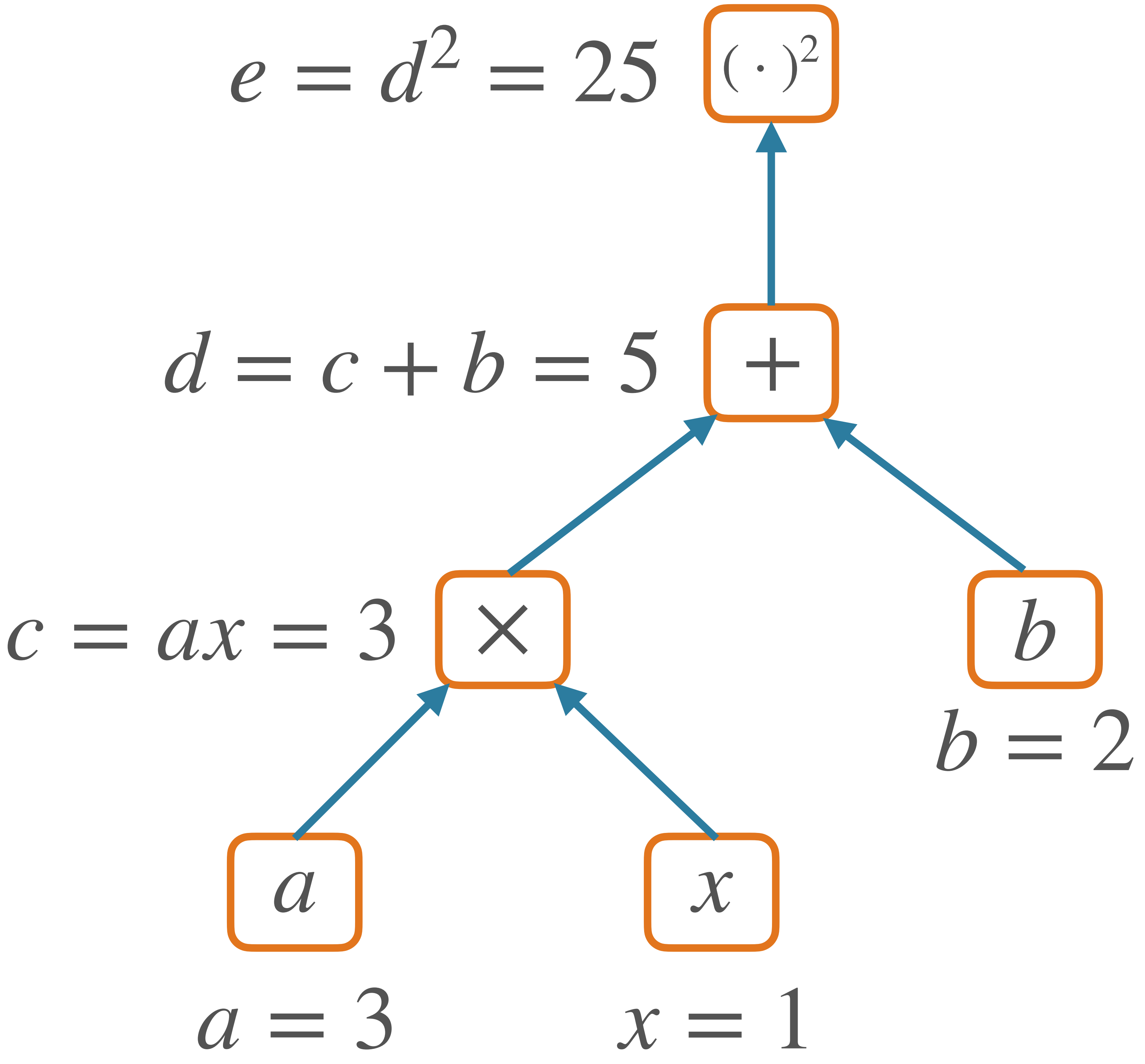
$$a_{\text{and}} = \sigma\left(\sigma\left(\begin{bmatrix} a_p & a_q \end{bmatrix} \begin{bmatrix} w^{\text{or}}_p & w^{\text{nand}}_p \\ w^{\text{or}}_q & w^{\text{nand}}_q \end{bmatrix} + \begin{bmatrix} b^{\text{or}} & b^{\text{nand}} \end{bmatrix}\right) \begin{bmatrix} w^{\text{and}}_{\text{or}} \\ w^{\text{and}}_{\text{nand}} \end{bmatrix} + b^{\text{and}}\right)$$

Computing XOR (not linearly-separable)



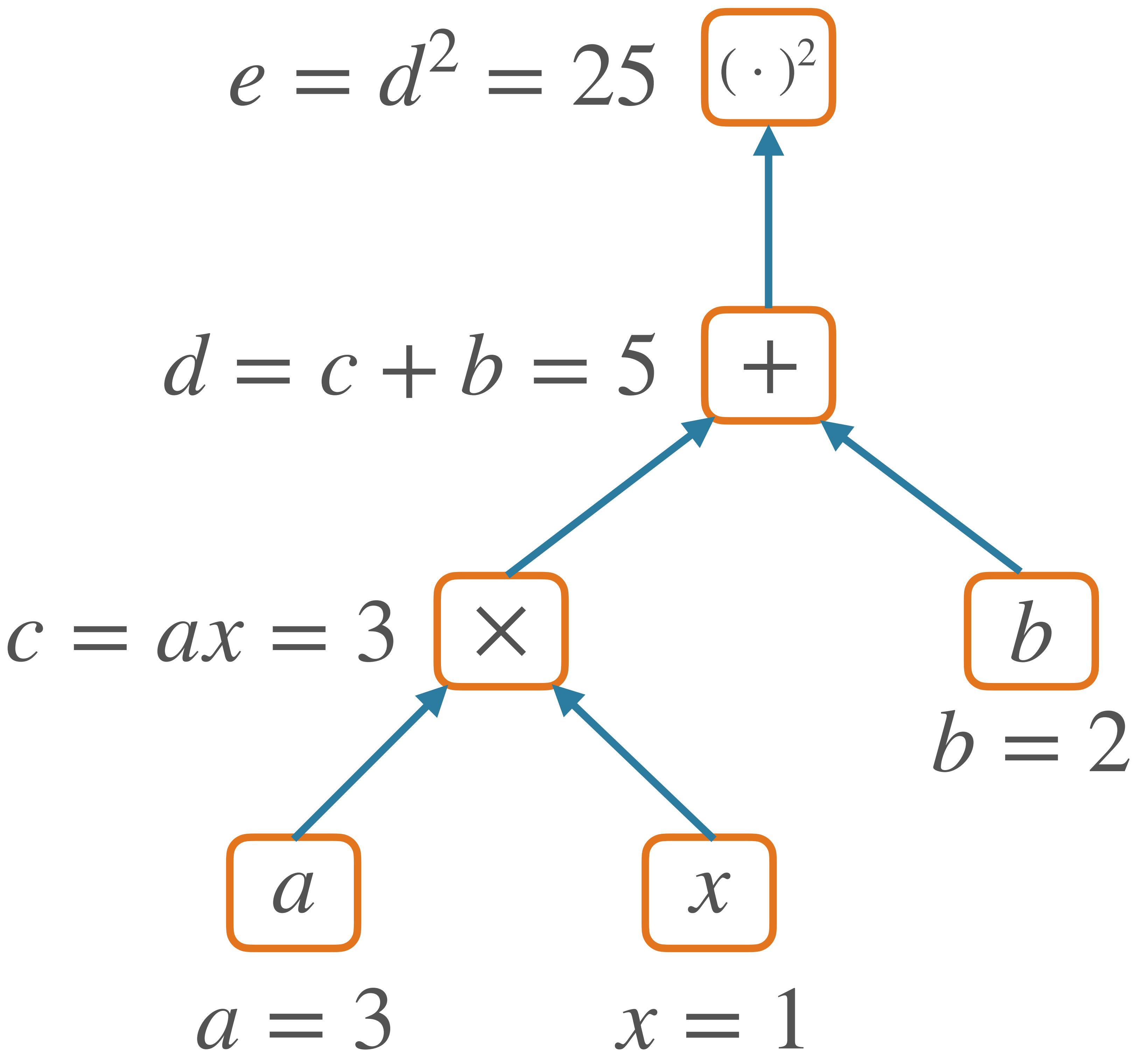
Backpropagation Example

$$f(x; a, b) = (ax + b)^2$$



Backpropagation Example

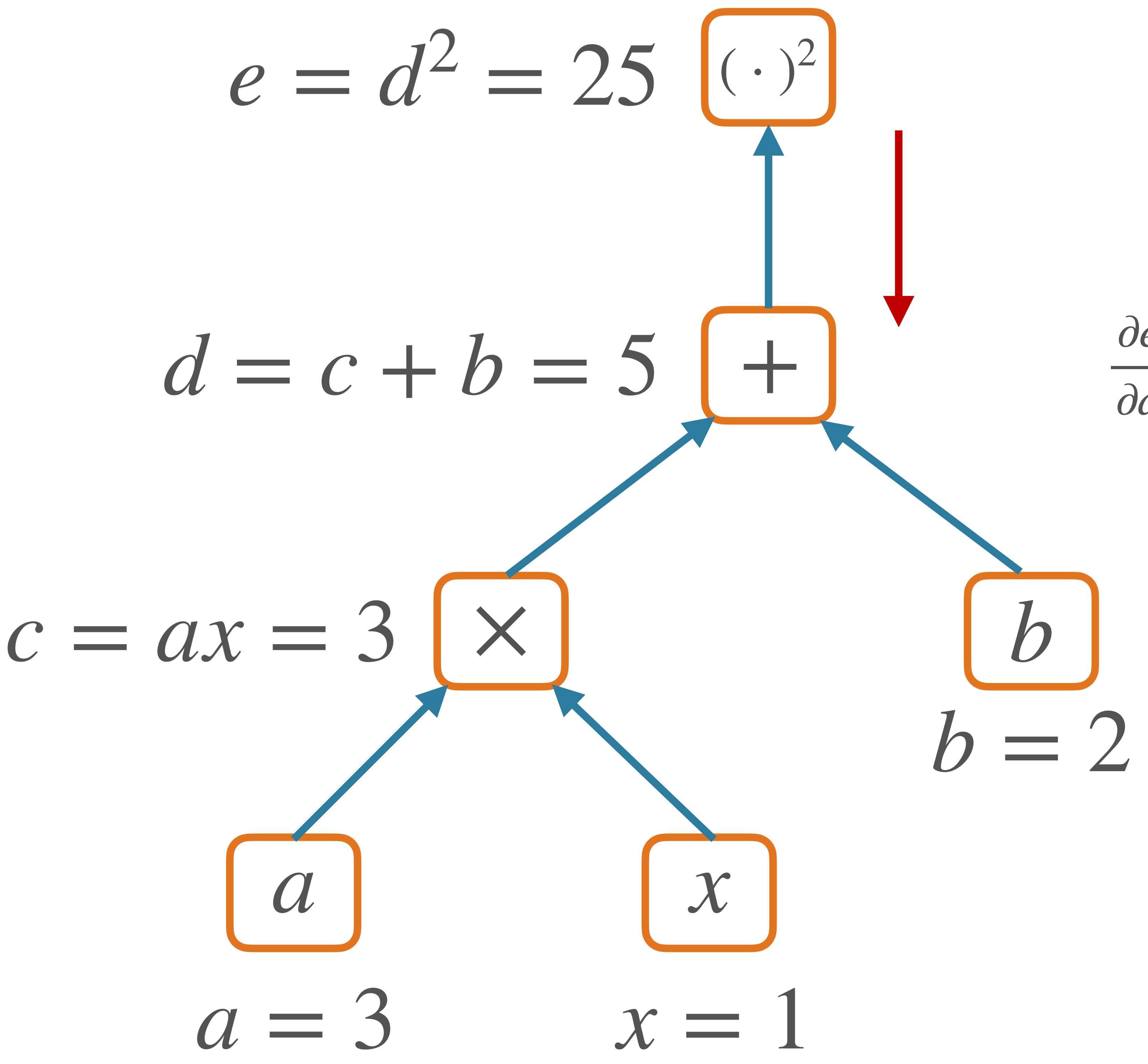
$$f(x; a, b) = (ax + b)^2$$



$$\frac{\partial e}{\partial e} = 1$$

Backpropagation Example

$$f(x; a, b) = (ax + b)^2$$

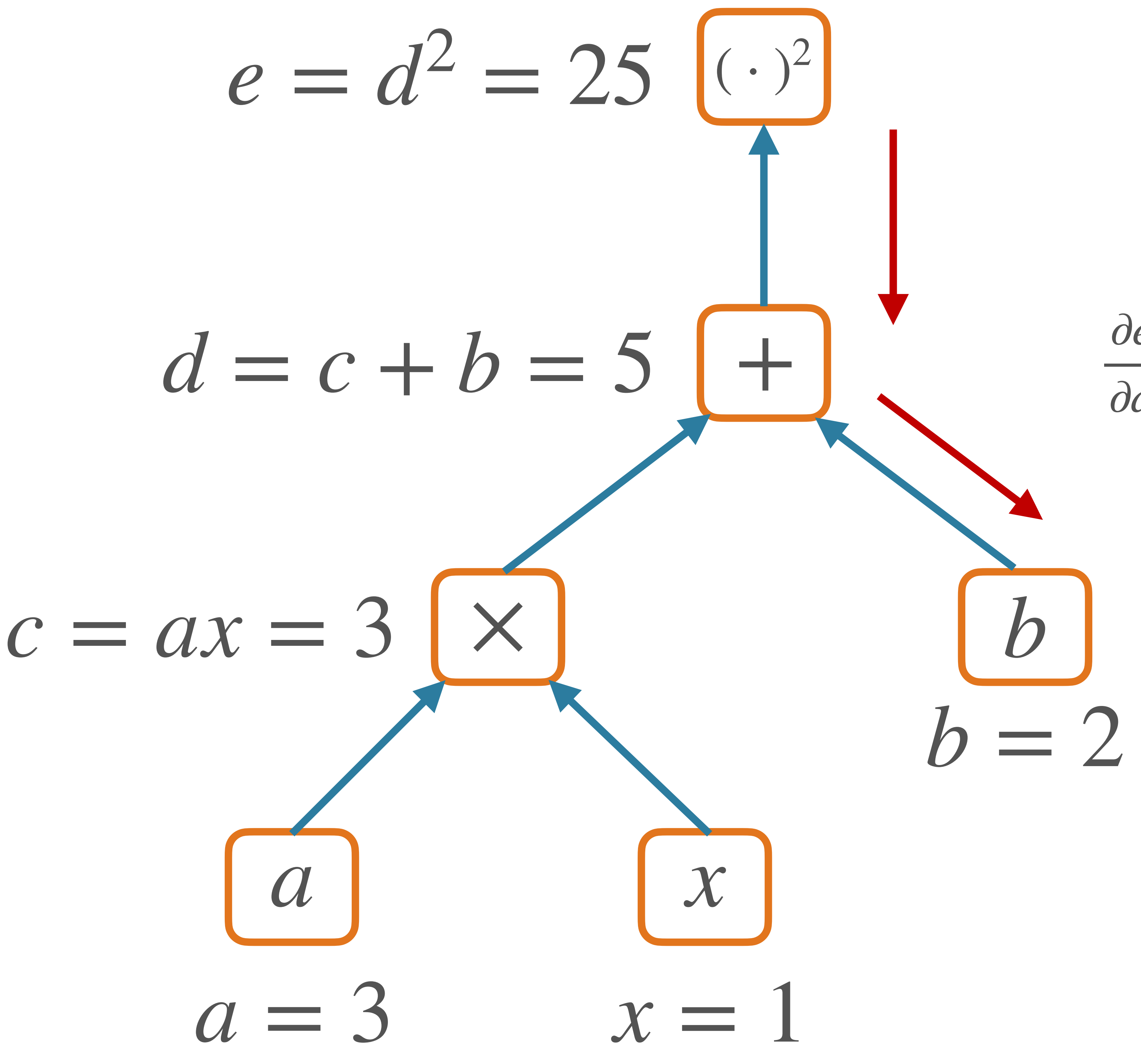


$$\frac{\partial e}{\partial e} = 1$$

$$\frac{\partial e}{\partial d} = 2d\frac{\partial e}{\partial e} = 10$$

Backpropagation Example

$$f(x; a, b) = (ax + b)^2$$



$$\frac{\partial e}{\partial e} = 1$$

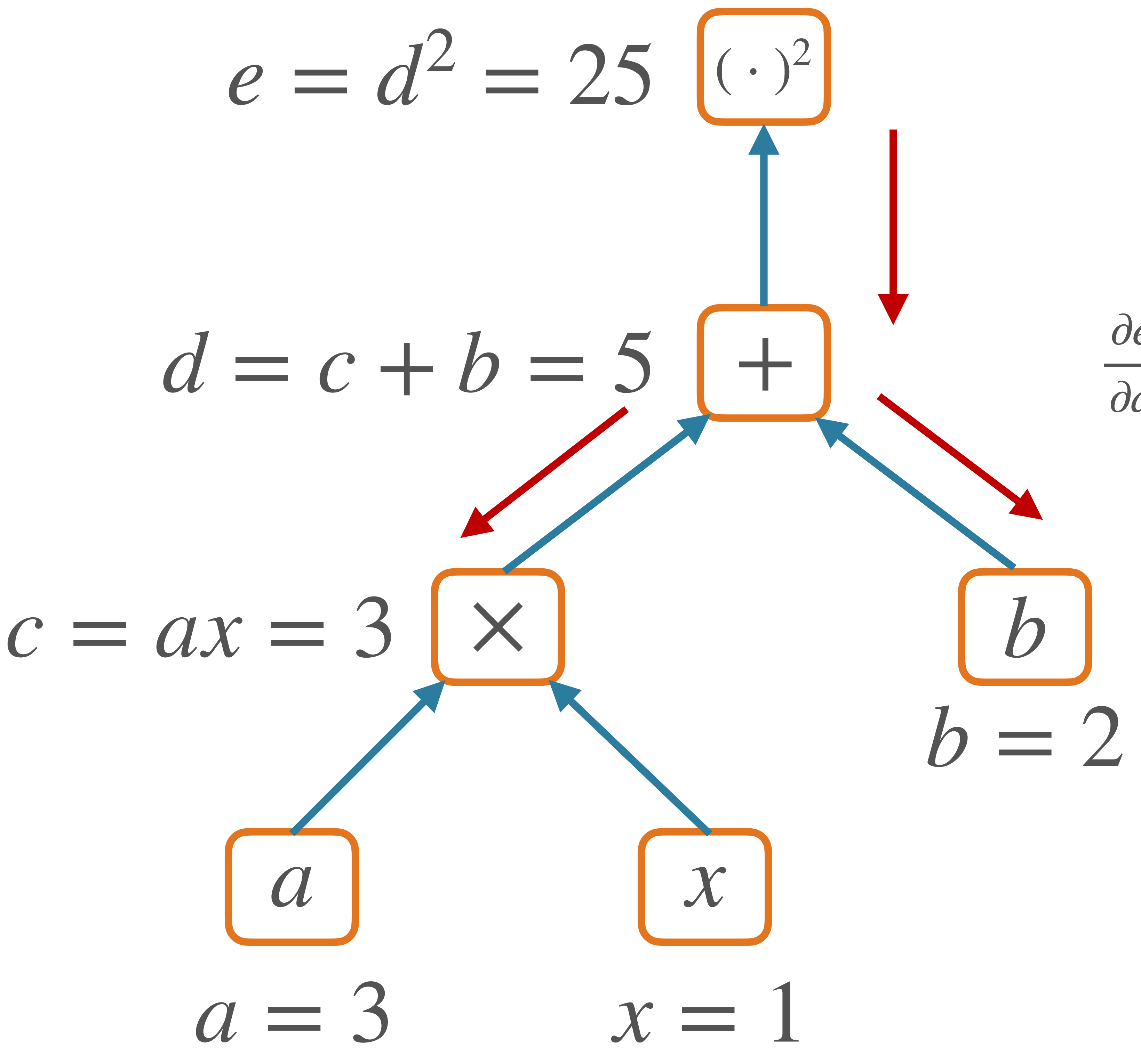
$$\frac{\partial e}{\partial d} = 2d\frac{\partial e}{\partial e} = 10$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 10\frac{\partial c + b}{\partial b} = 10$$

Backpropagation Example

$$f(x; a, b) = (ax + b)^2$$

$$\frac{\partial e}{\partial e} = 1$$



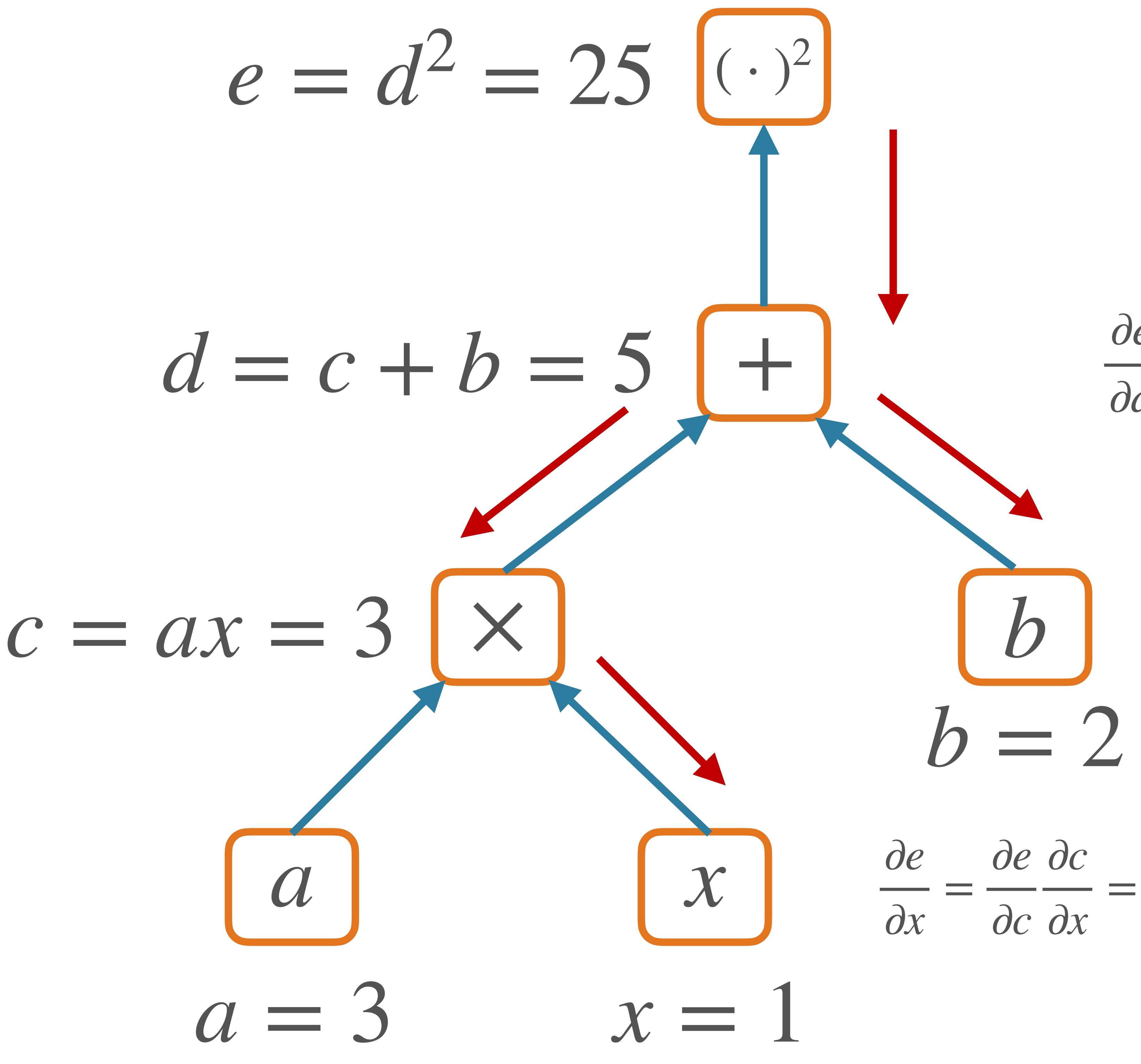
$$\frac{\partial e}{\partial d} = 2d\frac{\partial e}{\partial e} = 10$$

$$\frac{\partial e}{\partial c} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial c} = 10\frac{\partial c + b}{\partial c} = 10$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 10\frac{\partial c + b}{\partial b} = 10$$

Backpropagation Example

$$f(x; a, b) = (ax + b)^2$$



$$\frac{\partial e}{\partial e} = 1$$

$$\frac{\partial e}{\partial d} = 2d\frac{\partial e}{\partial e} = 10$$

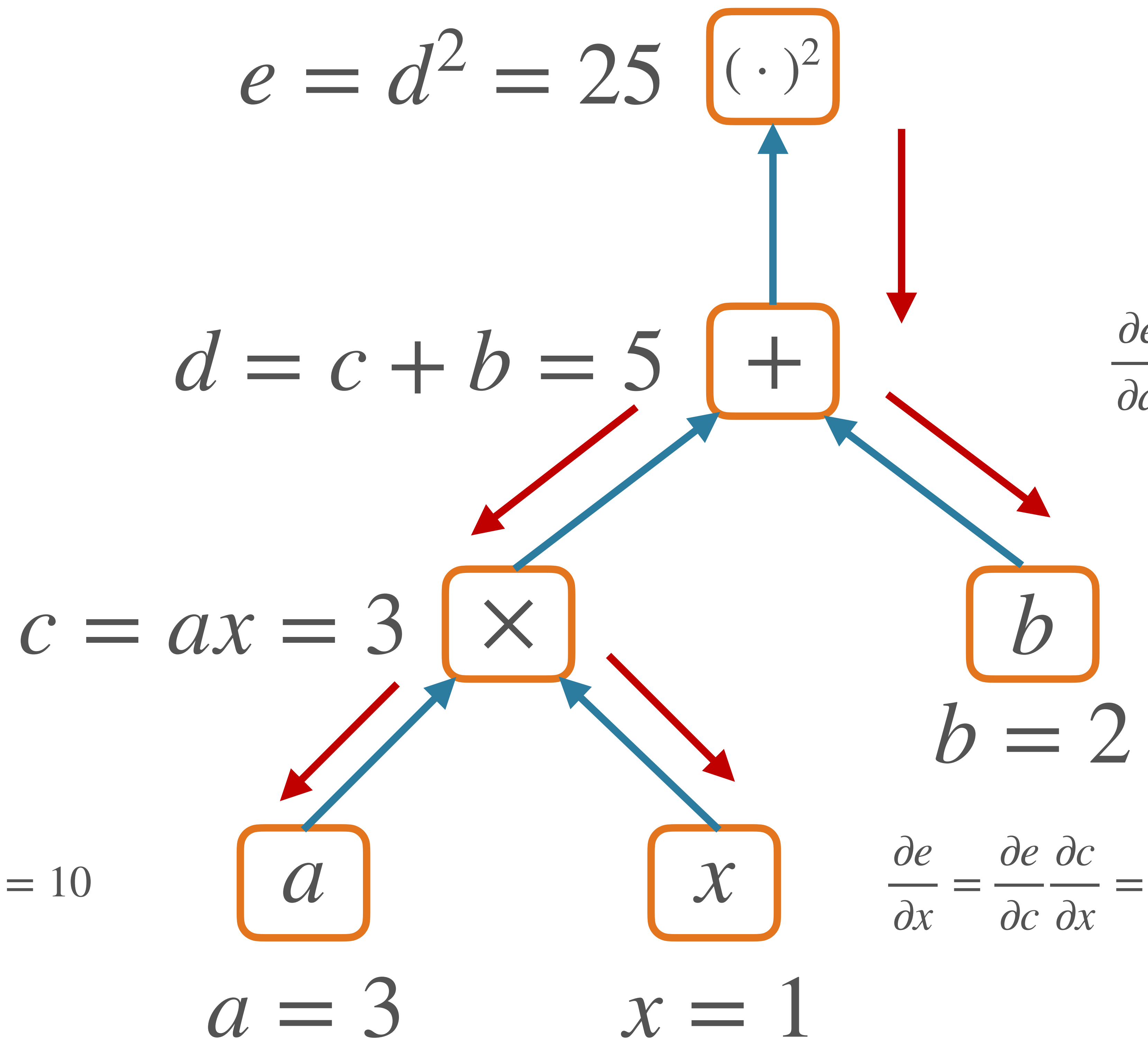
$$\frac{\partial e}{\partial c} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial c} = 10\frac{\partial c + b}{\partial c} = 10$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 10\frac{\partial c + b}{\partial b} = 10$$

$$\frac{\partial e}{\partial x} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial x} = 10a = 30$$

Backpropagation Example

$$f(x; a, b) = (ax + b)^2$$



$$\frac{\partial e}{\partial e} = 1$$

$$\frac{\partial e}{\partial d} = 2d\frac{\partial e}{\partial e} = 10$$

$$\frac{\partial e}{\partial c} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial c} = 10\frac{\partial c + b}{\partial c} = 10$$

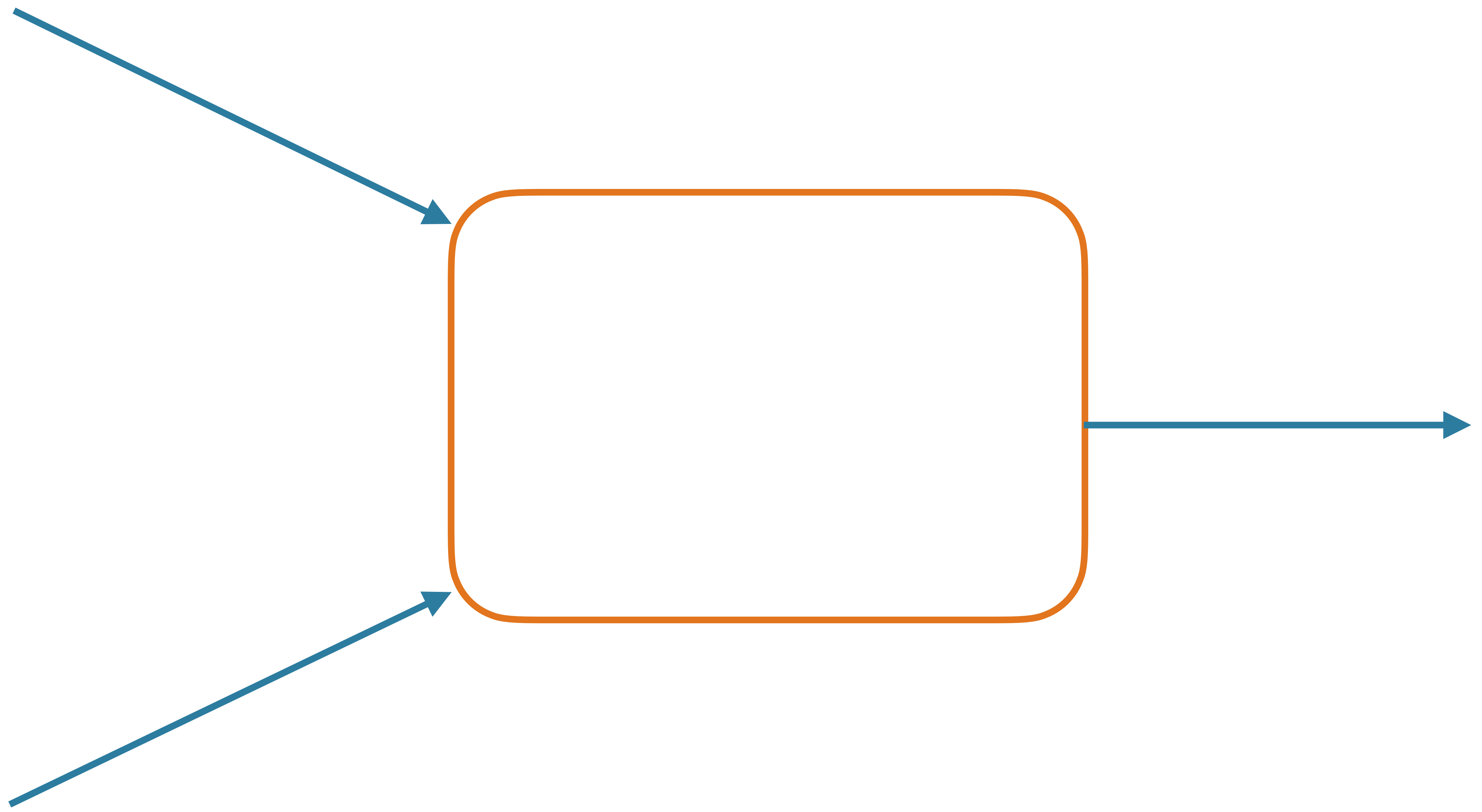
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 10\frac{\partial c + b}{\partial b} = 10$$

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial a} = 10x = 10$$

$$\frac{\partial e}{\partial x} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial x} = 10a = 30$$

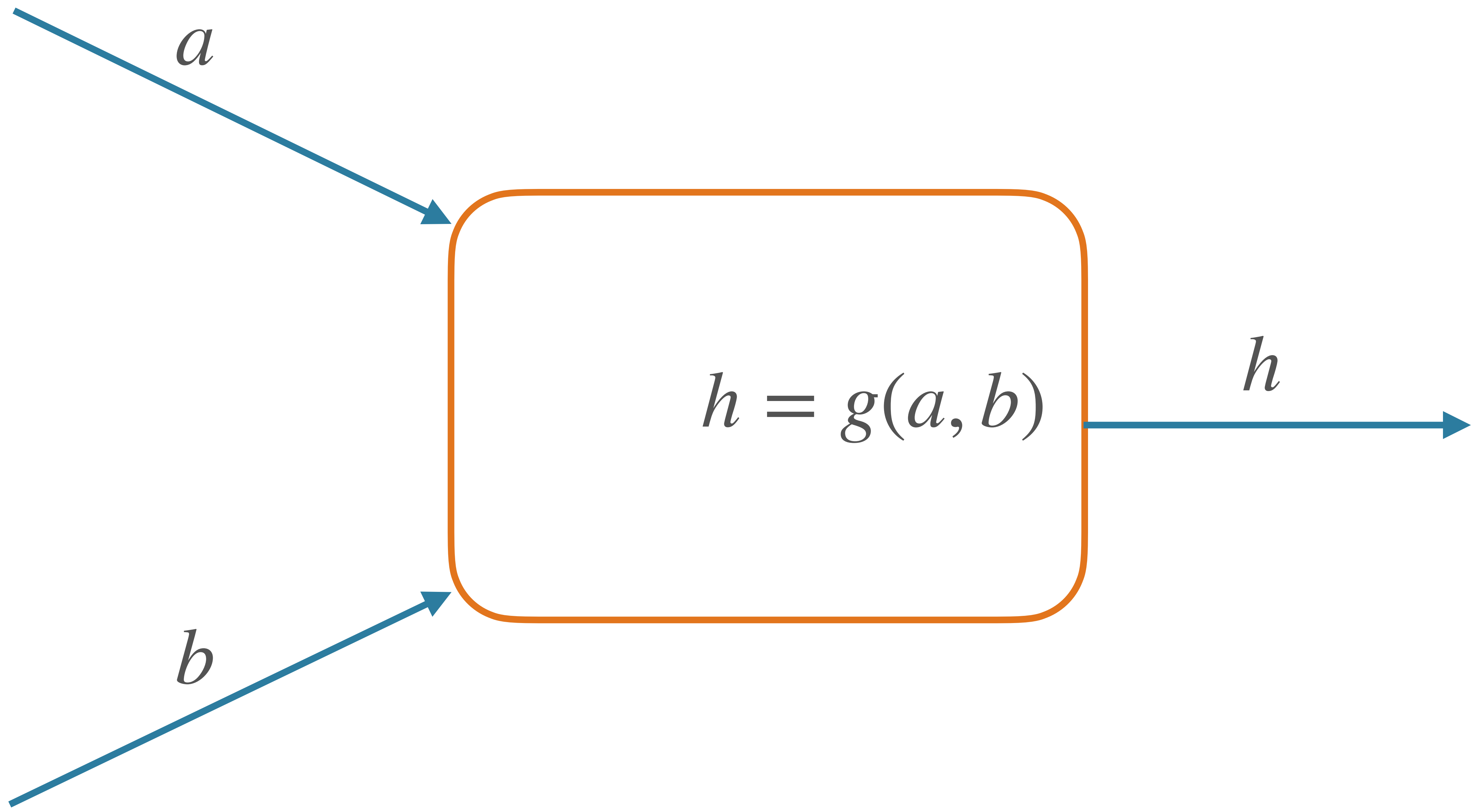
Nodes in Computational Graph

- Forward pass:
  - Compute value given parents' values
- Backward pass:
  - Compute parents' gradients given children's



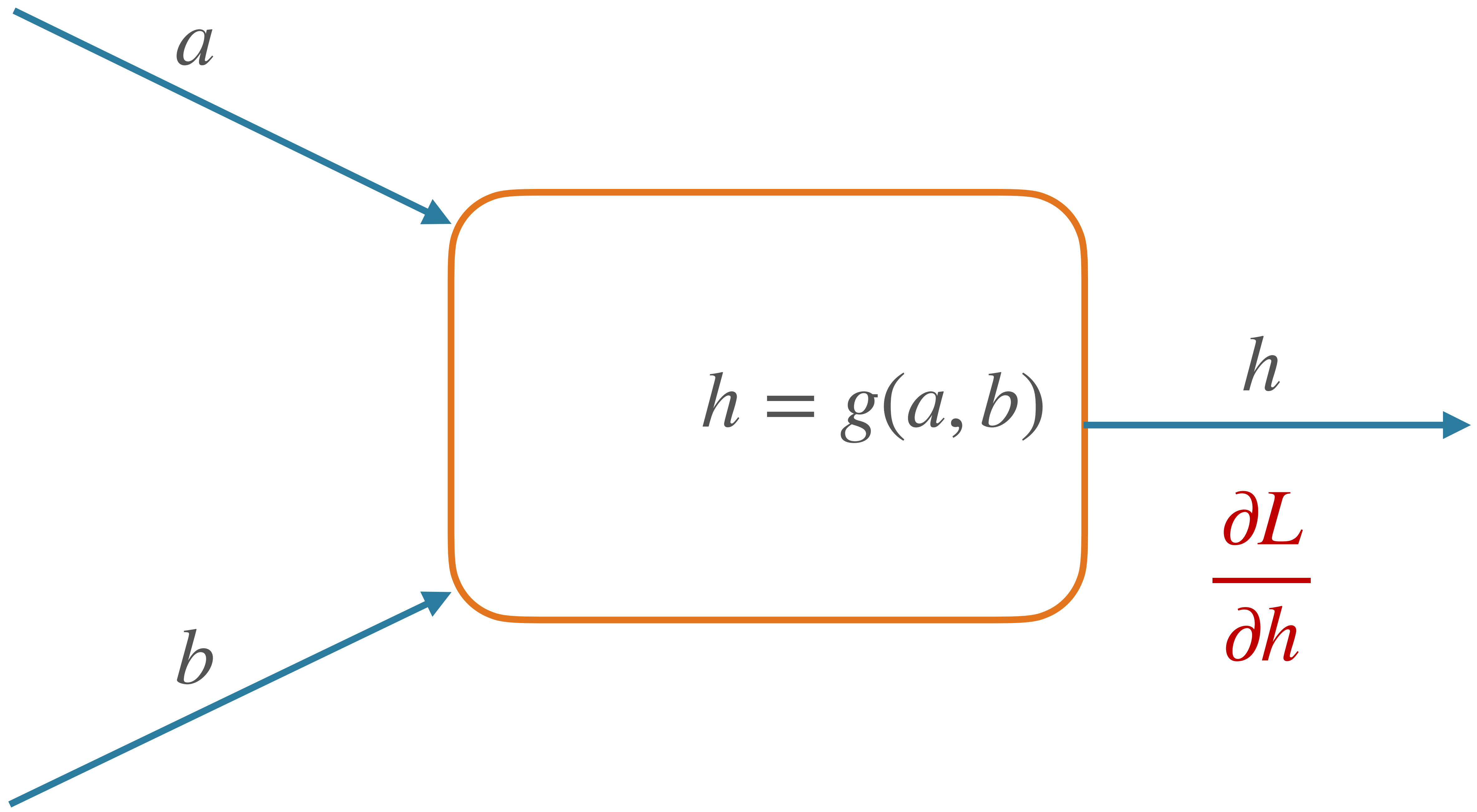
Nodes in Computational Graph

- Forward pass:
  - Compute value given parents' values
- Backward pass:
  - Compute parents' gradients given children's



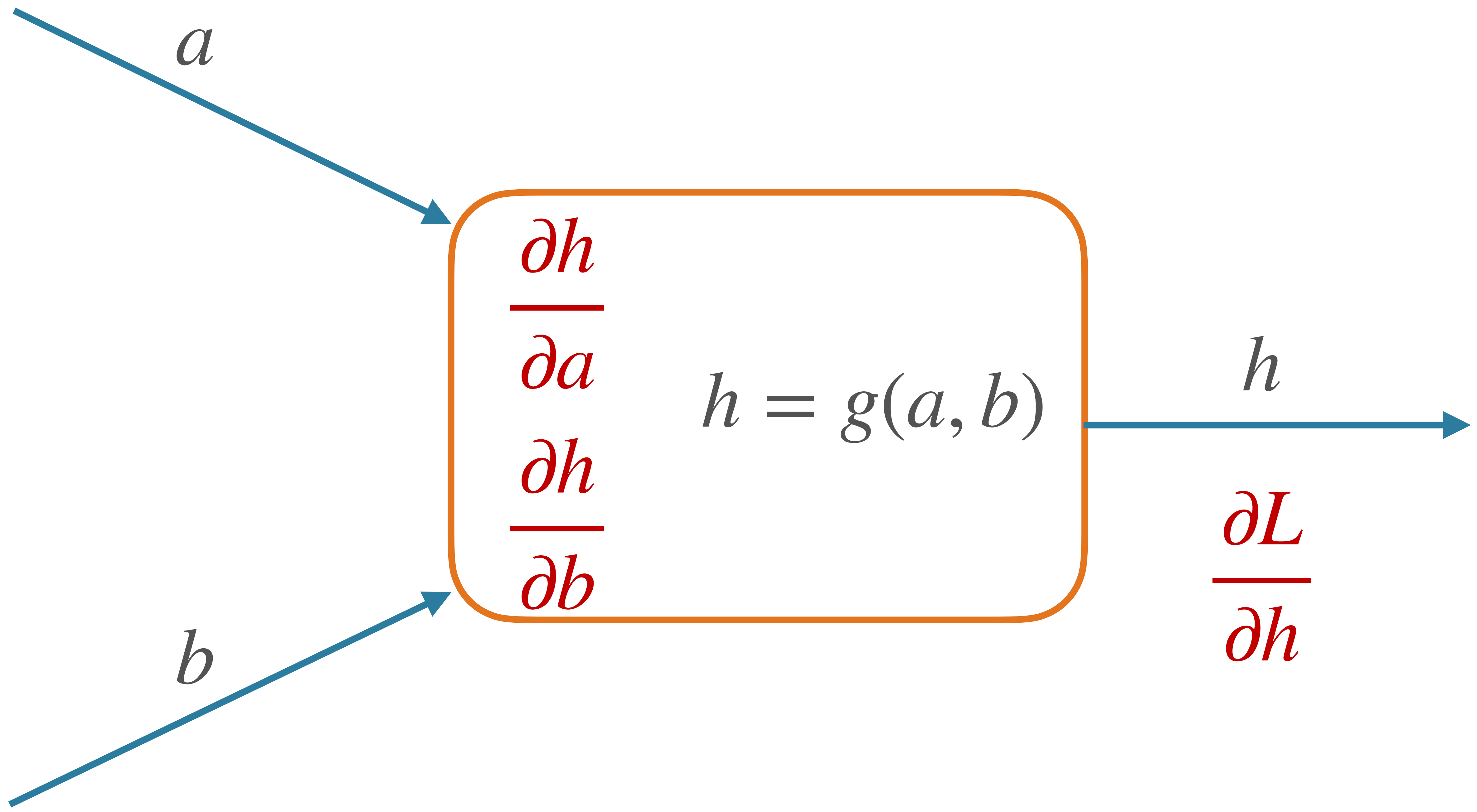
Nodes in Computational Graph

- Forward pass:
  - Compute value given parents' values
- Backward pass:
  - Compute parents' gradients given children's



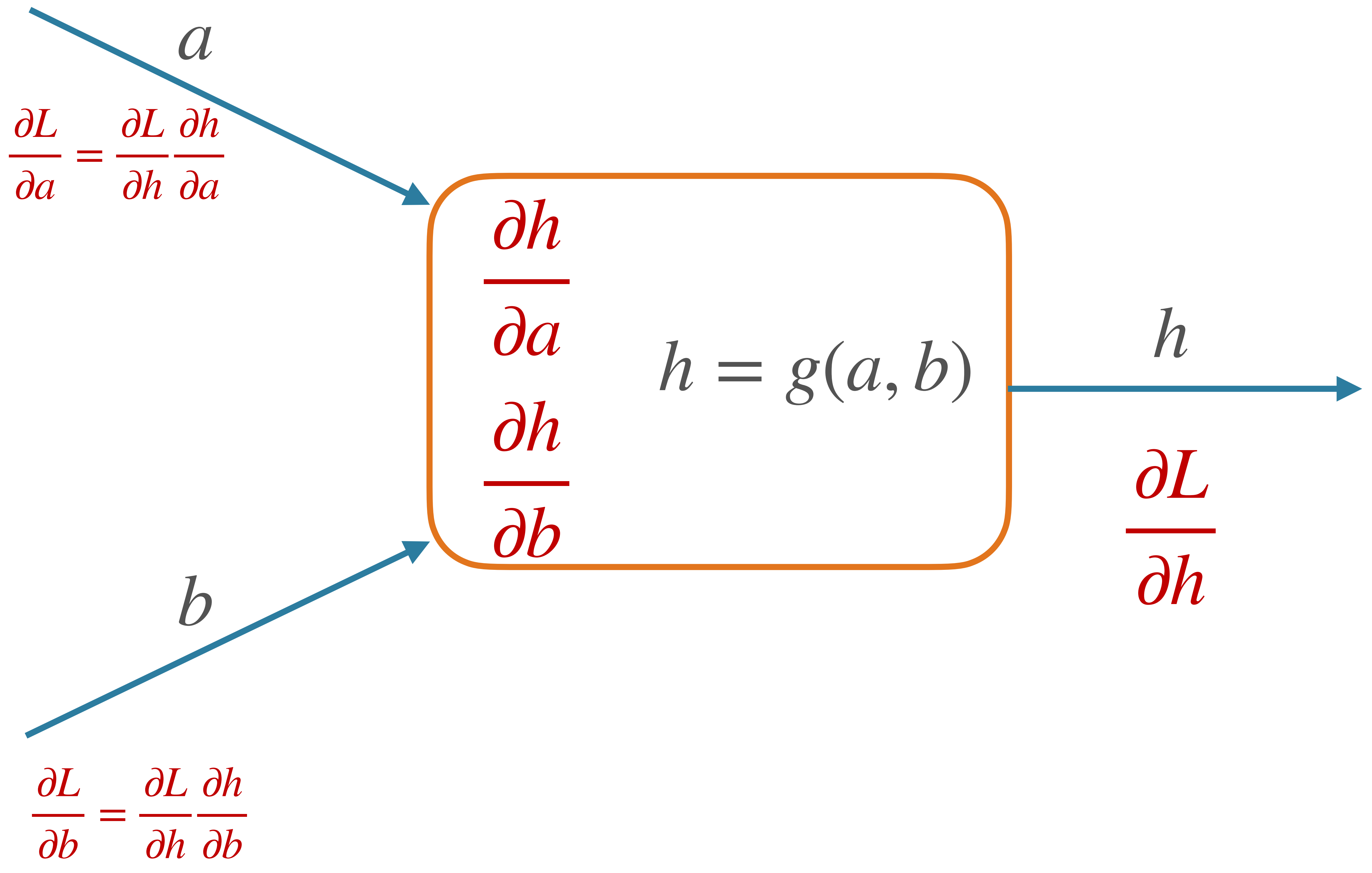
Nodes in Computational Graph

- Forward pass:
  - Compute value given parents' values
- Backward pass:
  - Compute parents' gradients given children's



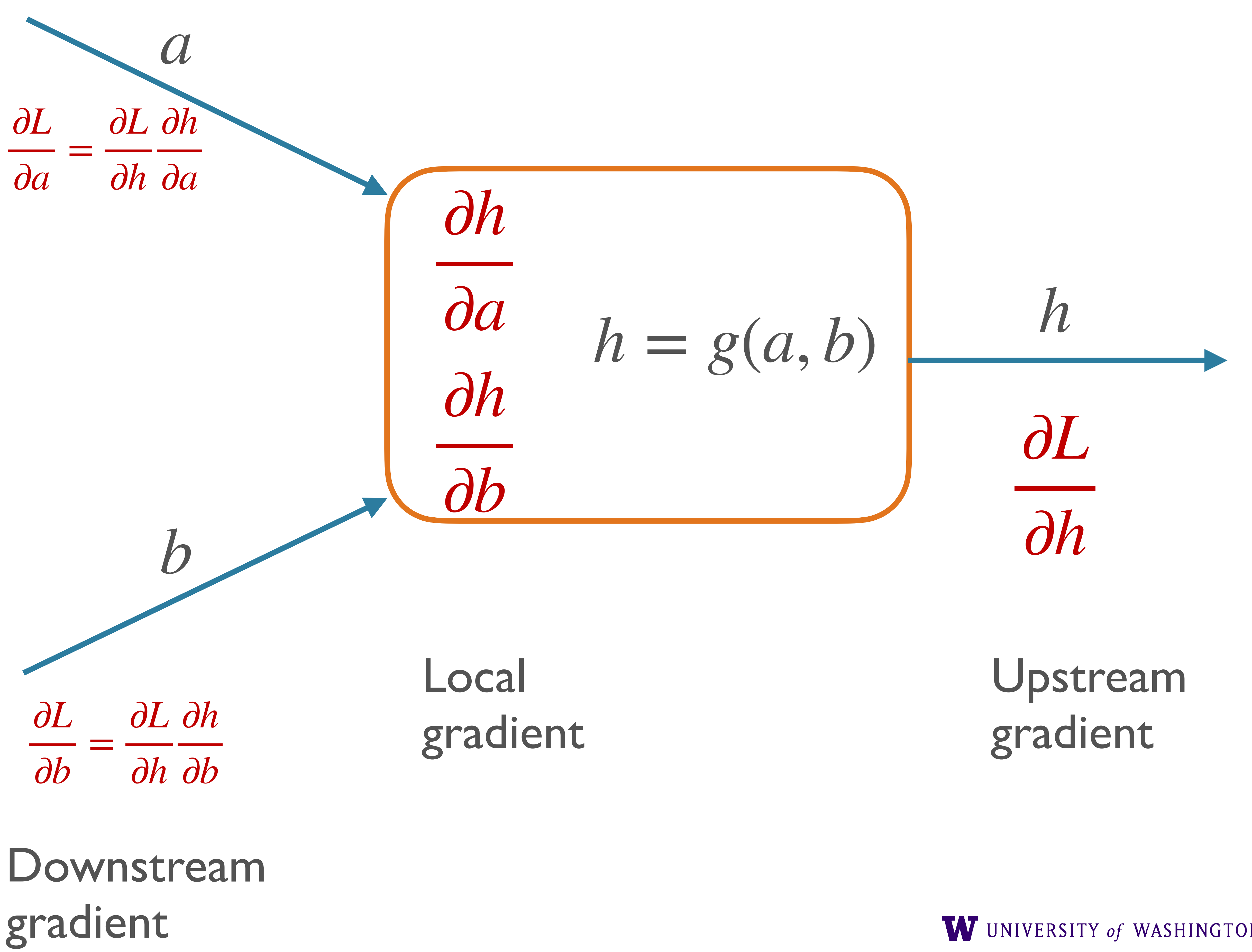
Nodes in Computational Graph

- Forward pass:
  - Compute value given parents' values
- Backward pass:
  - Compute parents' gradients given children's



Nodes in Computational Graph

- Forward pass:
  - Compute value given parents' values
- Backward pass:
  - Compute parents' gradients given children's



Example: ReLU

```
@tensor_op
class relu(Operation):
    @staticmethod
    def forward(ctx, value):
        new_val = np.maximum(0, value)
        ctx.append(new_val)
        return new_val

    @staticmethod
    def backward(ctx, grad_output):
        value = ctx[-1]
        return [(value > 0).astype(float) * grad_output]
```

Example: ReLU

```
@tensor_op
class relu(Operation):
    @staticmethod
    def forward(ctx, value):
        new_val = np.maximum(0, value)
        ctx.append(new_val)
        return new_val

    @staticmethod
    def backward(ctx, grad_output):
        value = ctx[-1]
        return [(value > 0).astype(float) * grad_output]
```

Save and retrieve the input value!

Example: ReLU

```
@tensor_op
class relu(Operation):
    @staticmethod
    def forward(ctx, value):
        new_val = np.maximum(0, value)
        ctx.append(new_val)
        return new_val

    @staticmethod
    def backward(ctx, grad_output):
        value = ctx[-1]
        return [(value > 0).astype(float) * grad_output]
```

Save and retrieve the input value!

local gradient times upstream gradient

Example: ReLU

```
@tensor_op
class relu(Operation):
    @staticmethod
    def forward(ctx, value):
        new_val = np.maximum(0, value)
        ctx.append(new_val)
        return new_val

    @staticmethod
    def backward(ctx, grad_output):
        value = ctx[-1]
        return [(value > 0).astype(float) * grad_output]
```

Save and retrieve the input value!

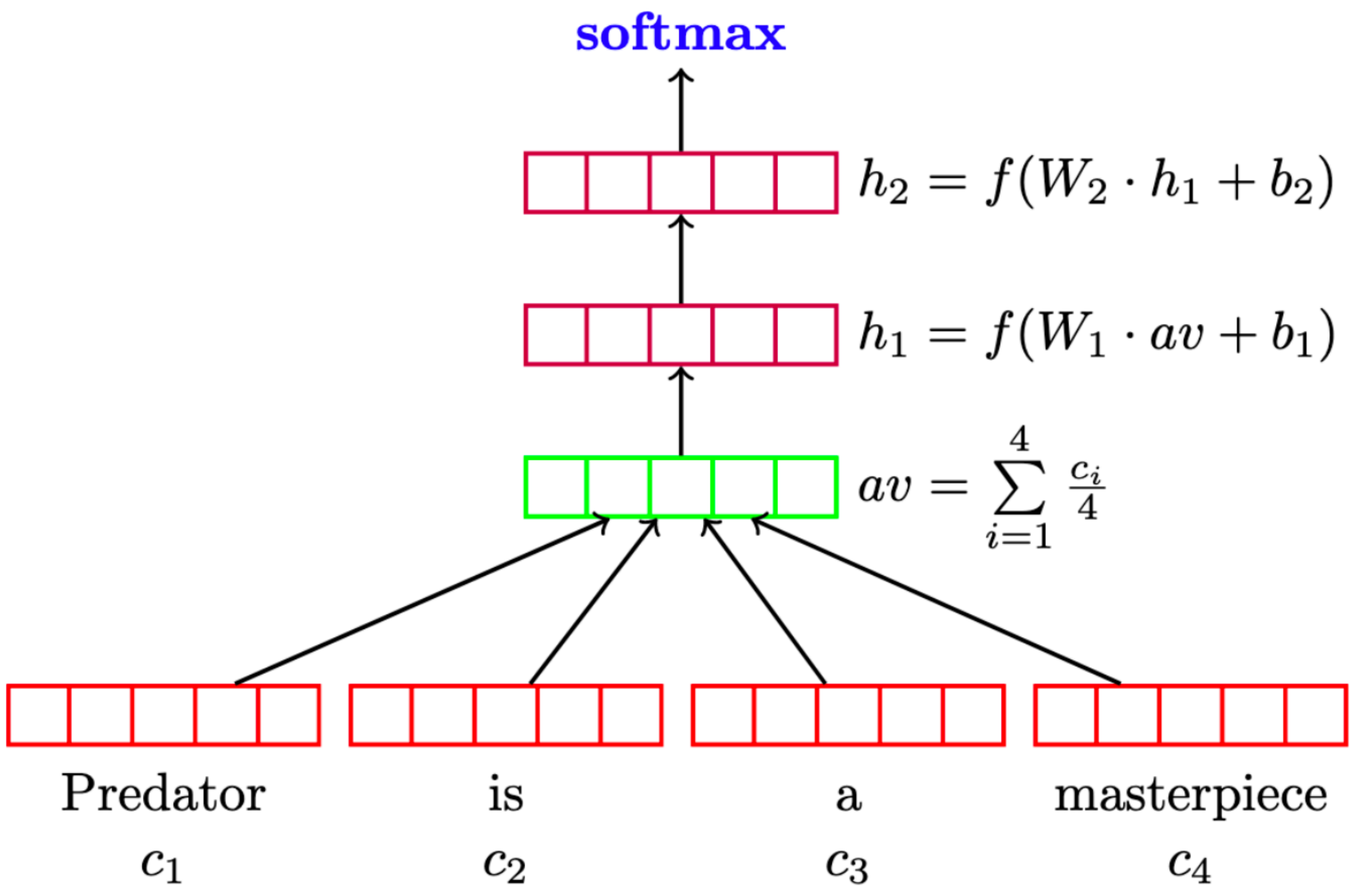
NB: list, one downstream gradient per input (in this case, one)

local gradient times upstream gradient

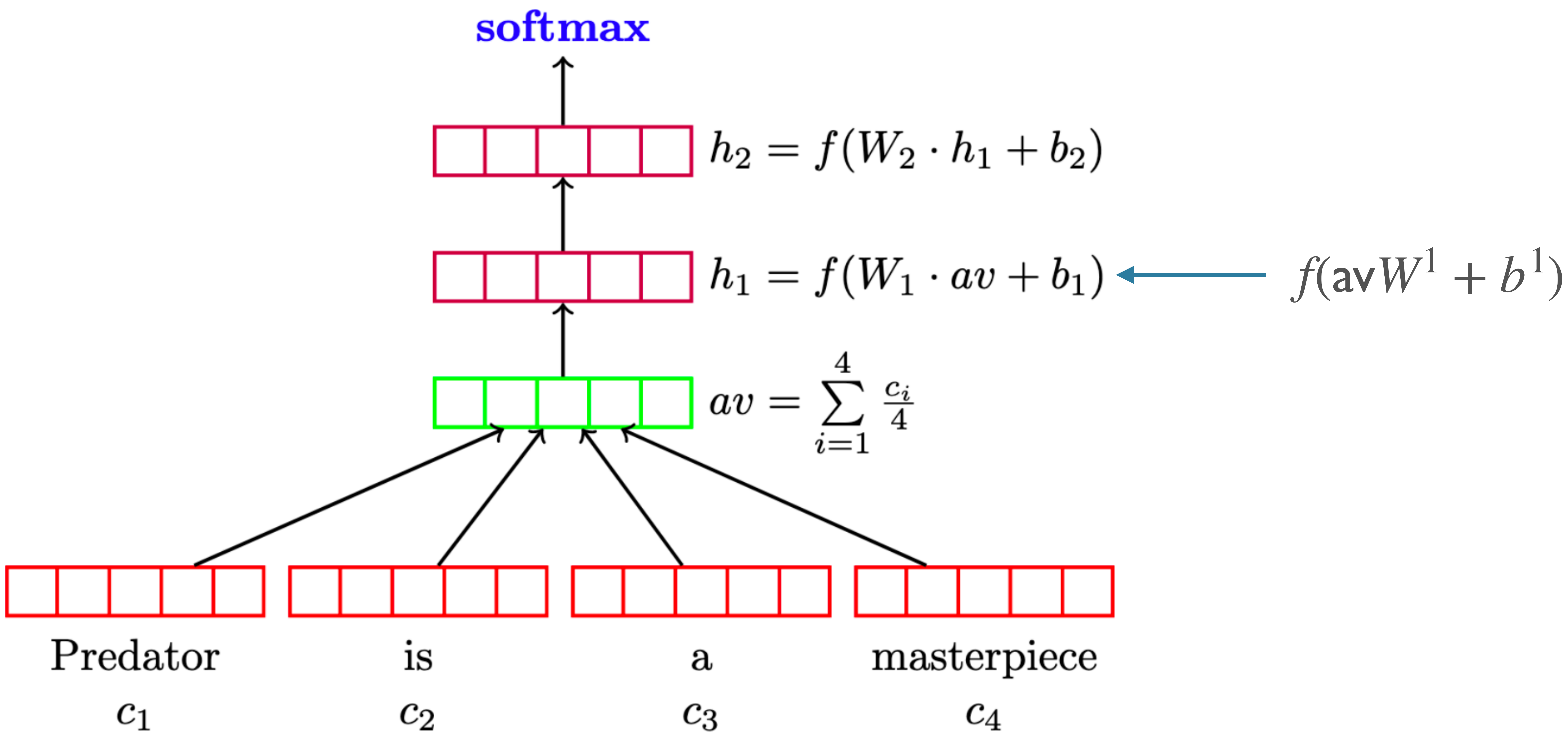
Neural Networks, I

- Feed-forward networks
  - Fixed size: average, fixed window of prep tokens
- Recurrent neural networks: sequence processors
  - Vanishing gradients, gated variants (LSTM)
  - Encoder-decoder / seq2seq architecture and tasks
    - Attention mechanism

Model Architecture, One Input

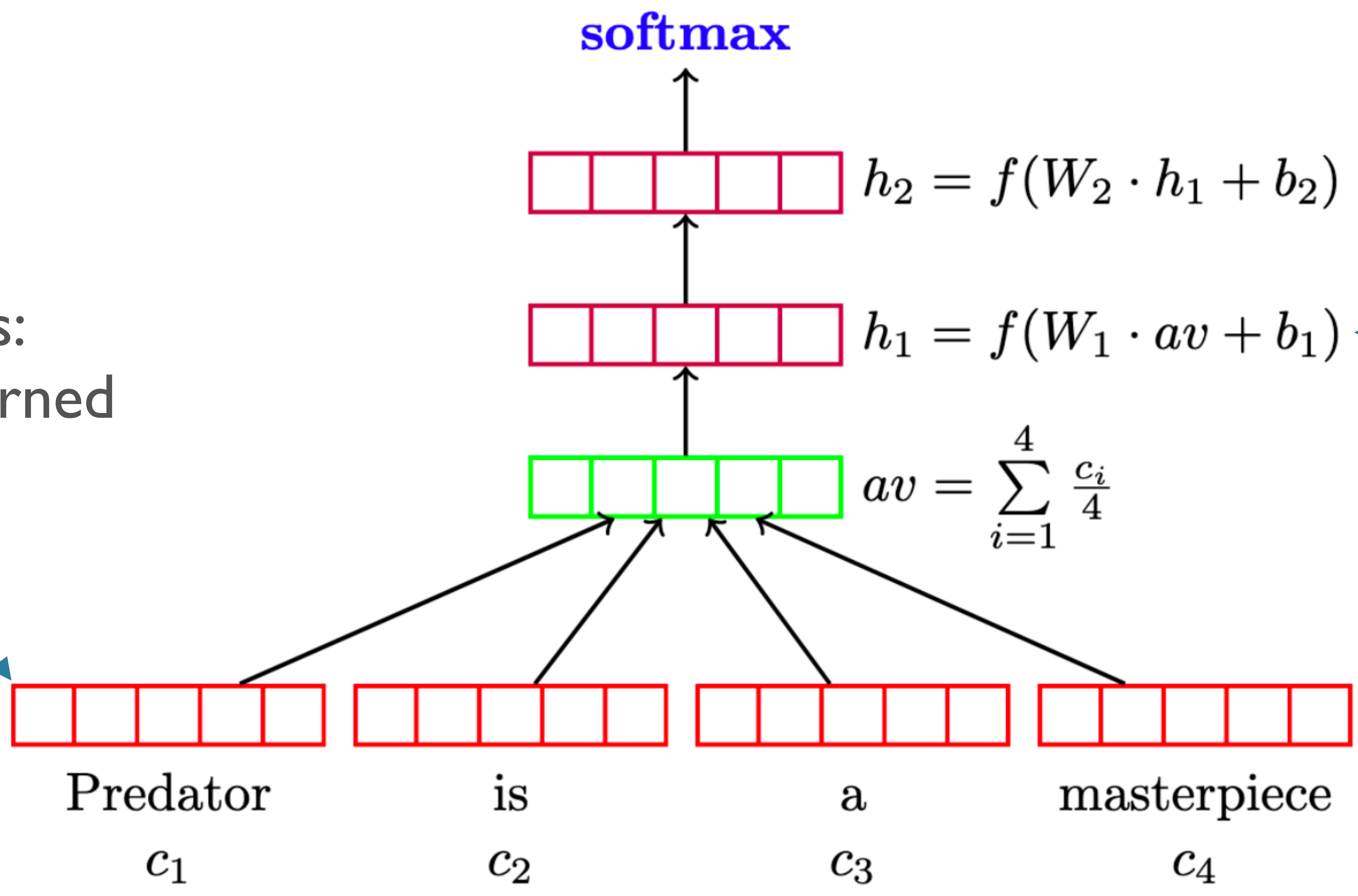


Model Architecture, One Input



Model Architecture, One Input

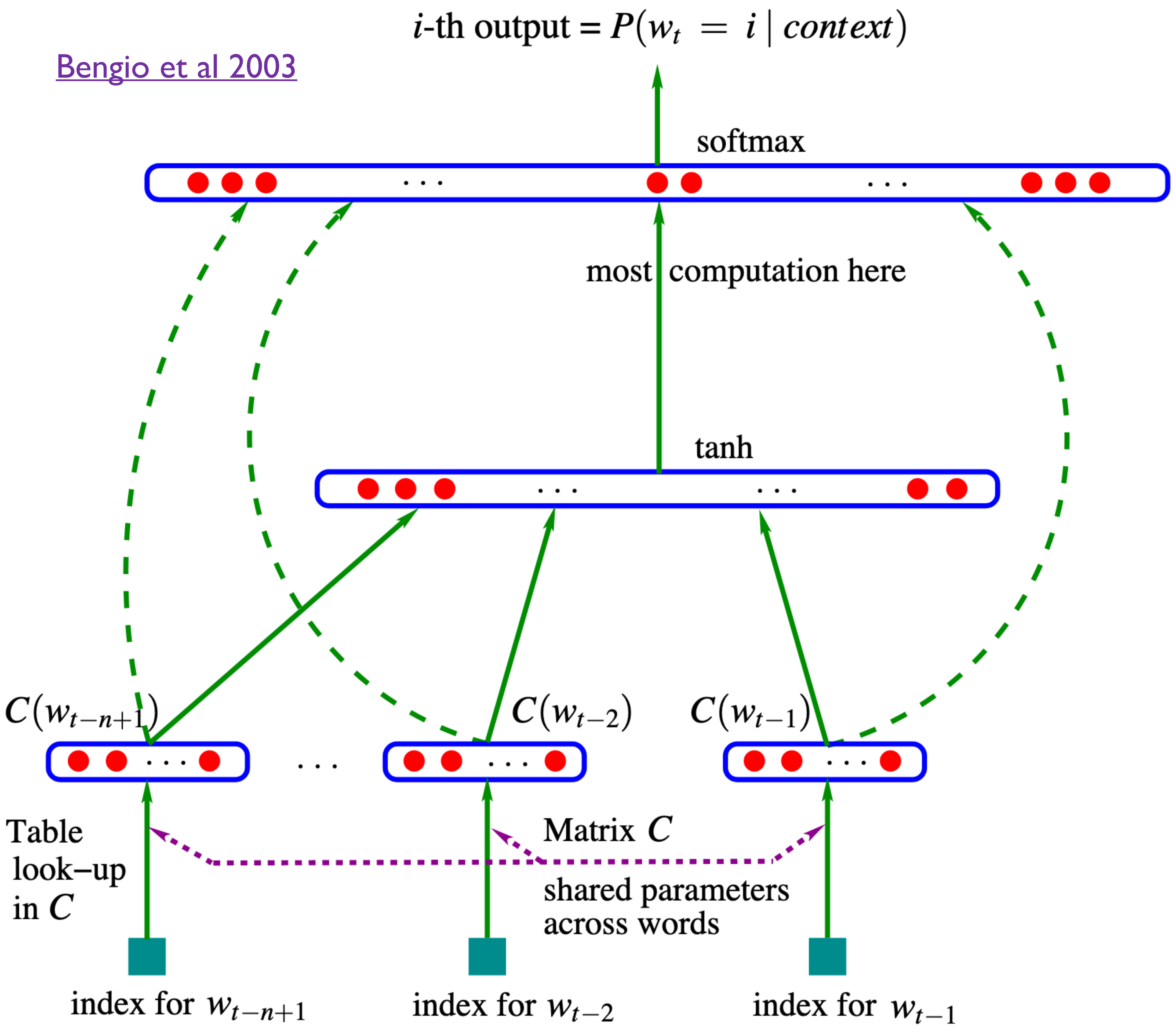
Word embeddings:
Pre-trained or learned



$f(\text{av}W^1 + b^1)$

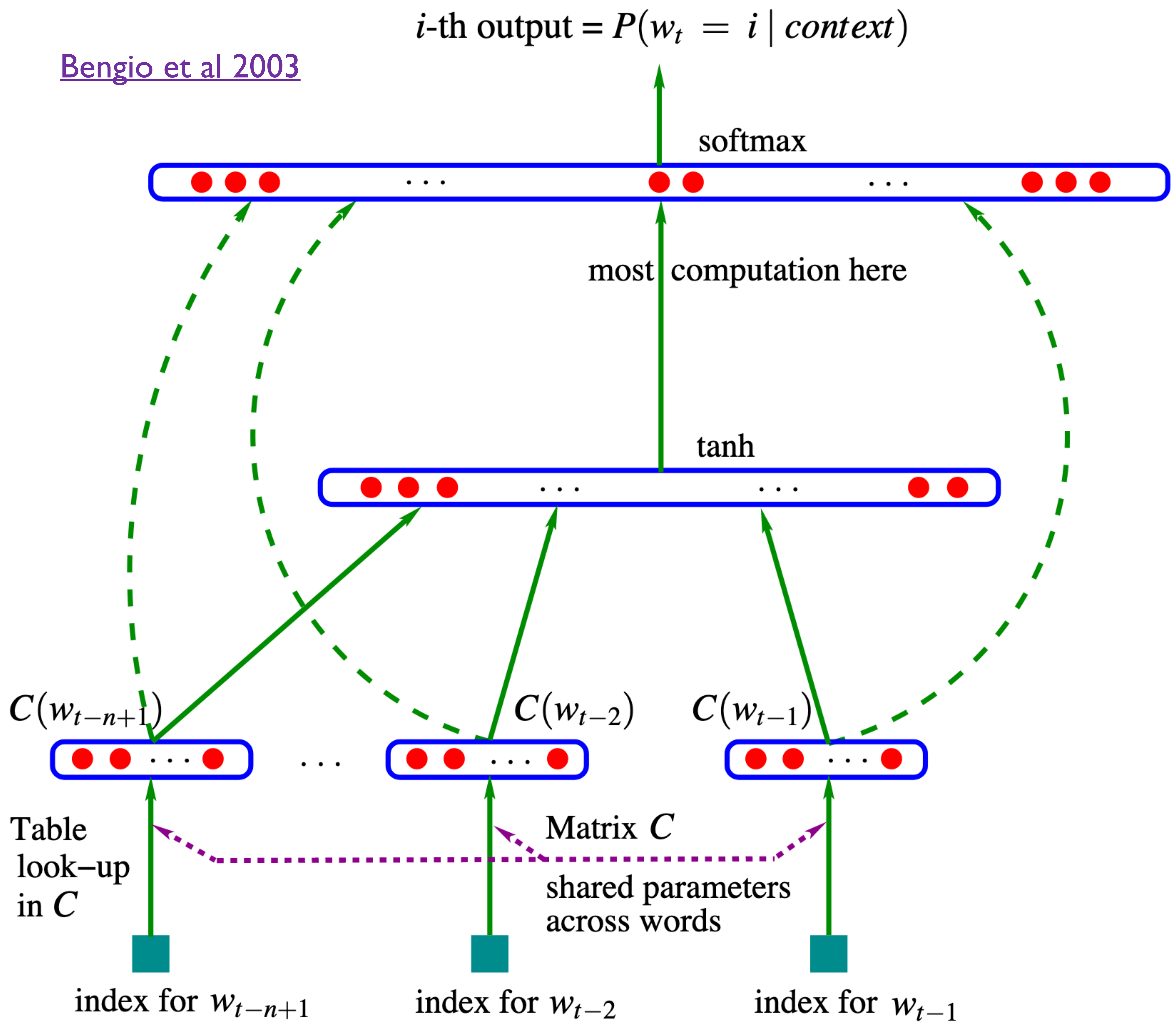
Neural LM Architecture

Bengio et al 2003



Neural LM Architecture

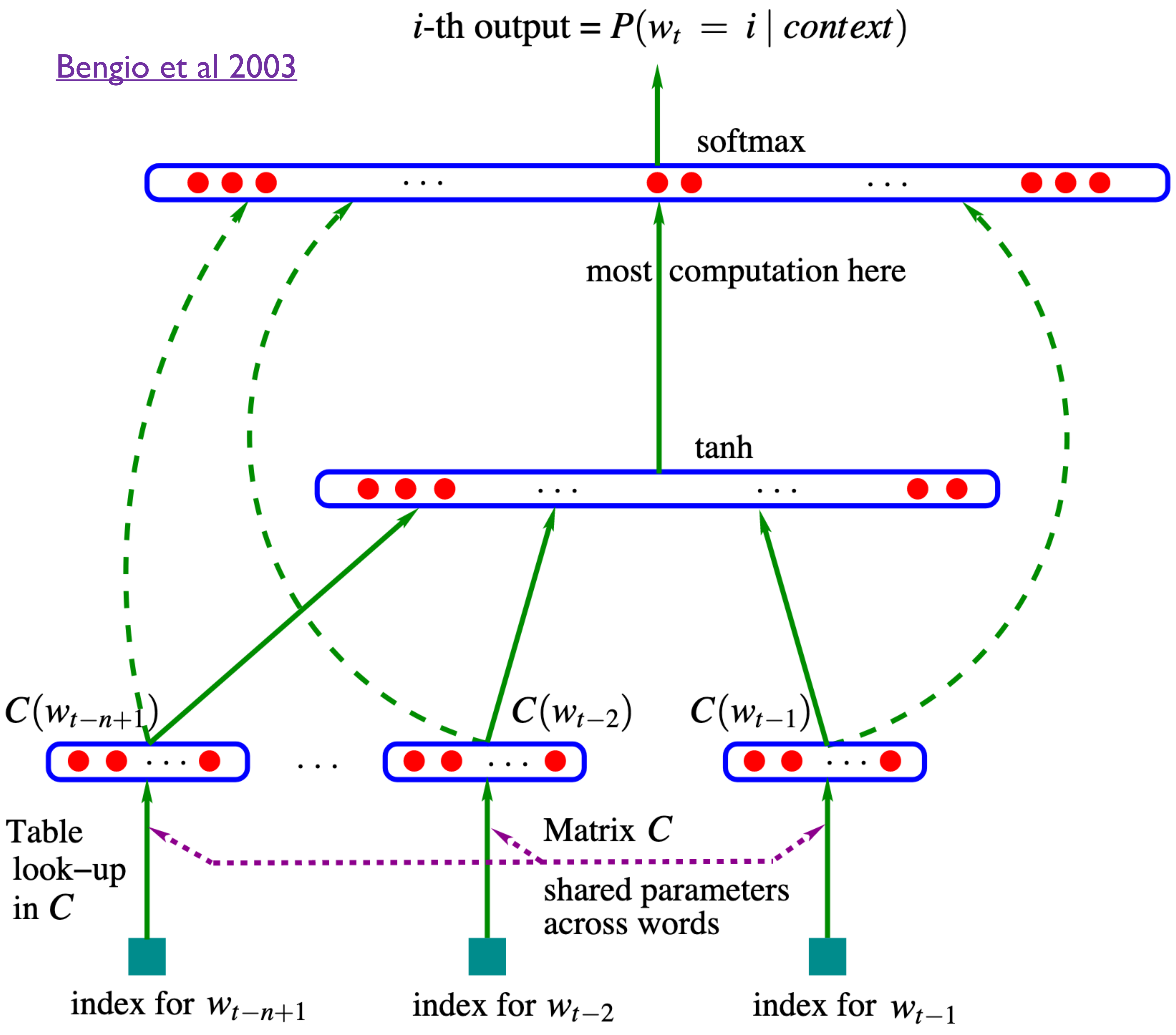
Bengio et al 2003



w_t: one-hot vector

Neural LM Architecture

Bengio et al 2003

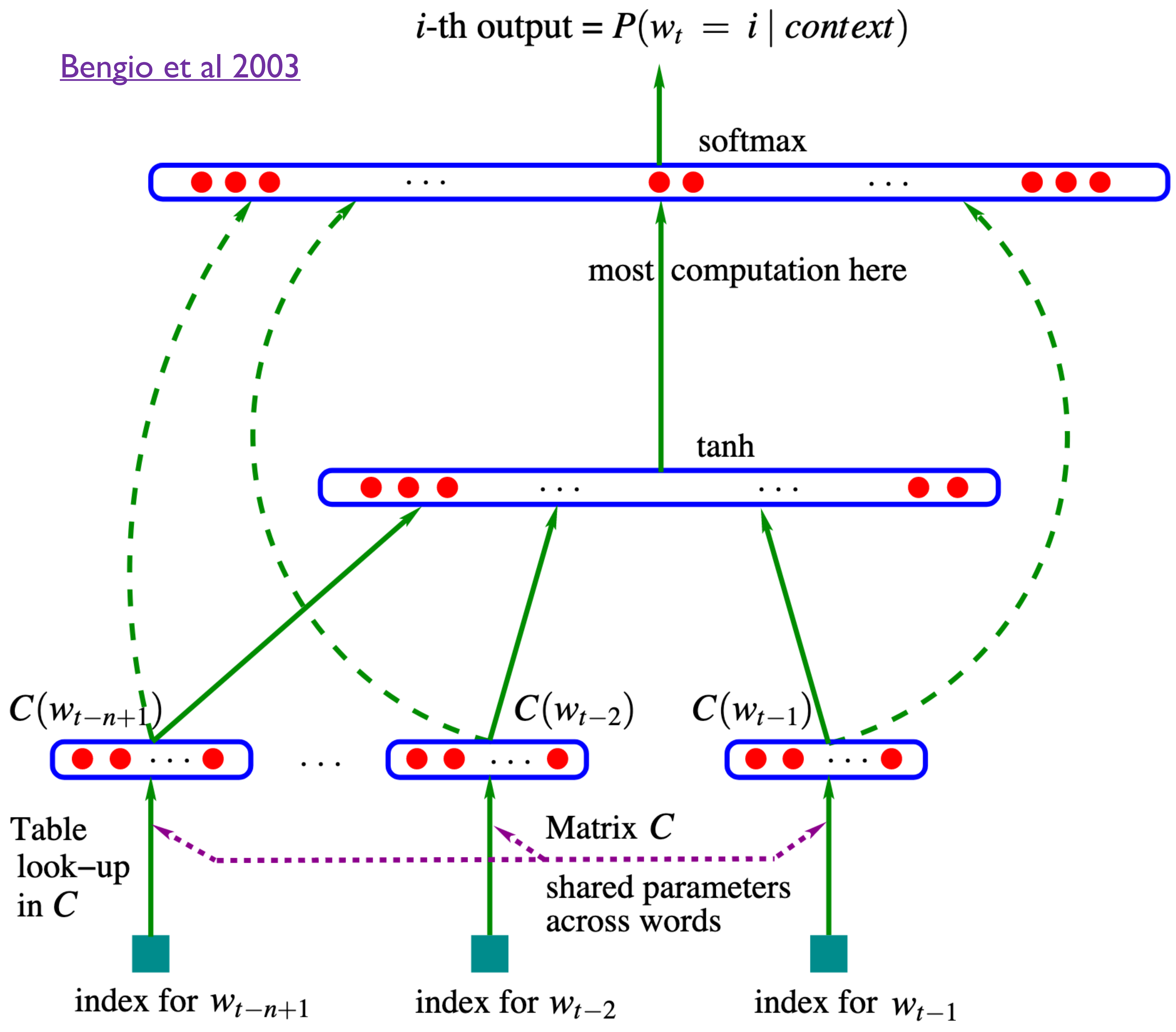


$$\text{embeddings} = \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$$

w_t: one-hot vector

Neural LM Architecture

[Bengio et al 2003]



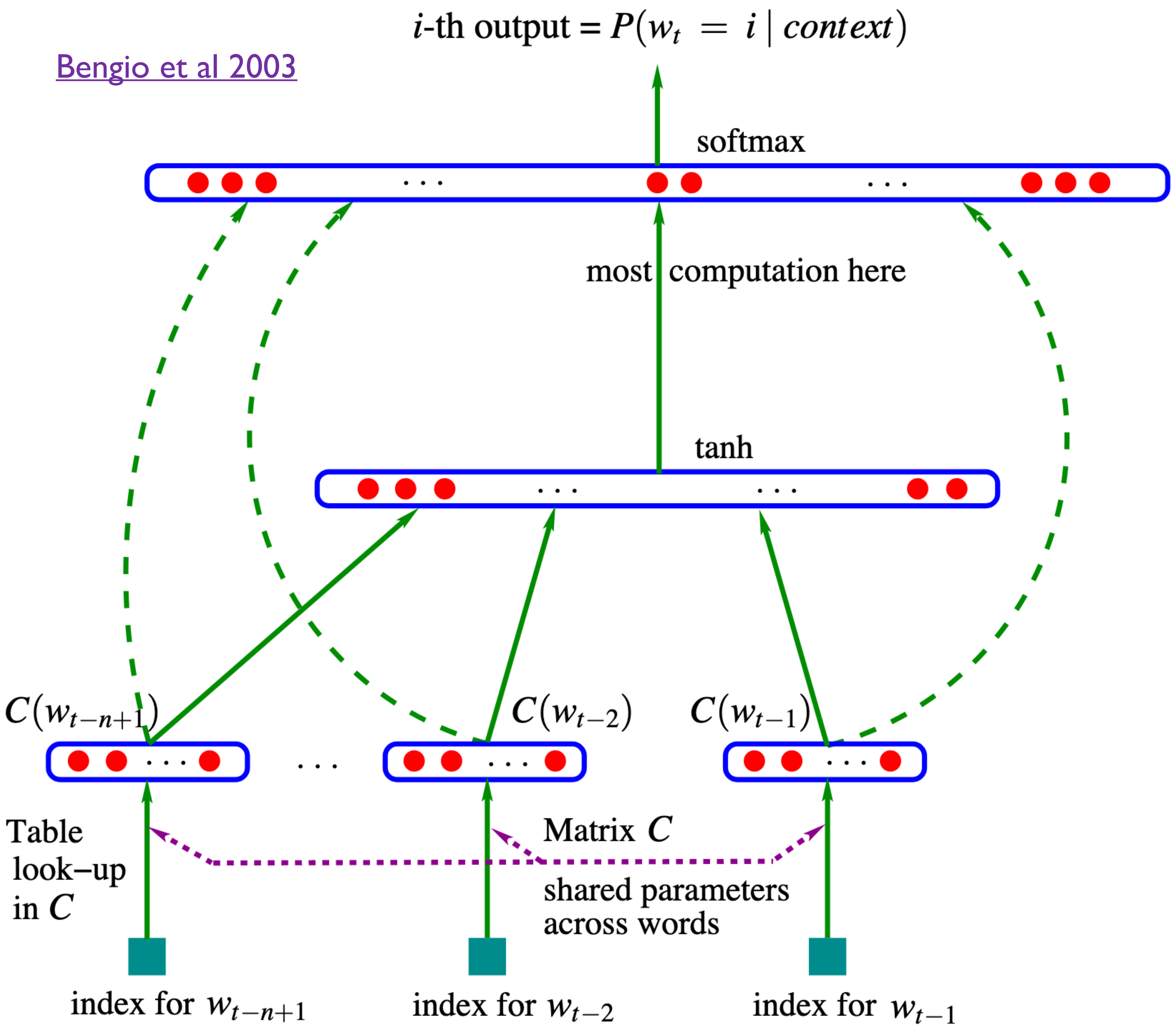
$$\text{hidden} = \tanh(\text{embeddings} W^1 + b^1)$$

$$\text{embeddings} = \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$$

w_t: one-hot vector

Neural LM Architecture

Bengio et al 2003



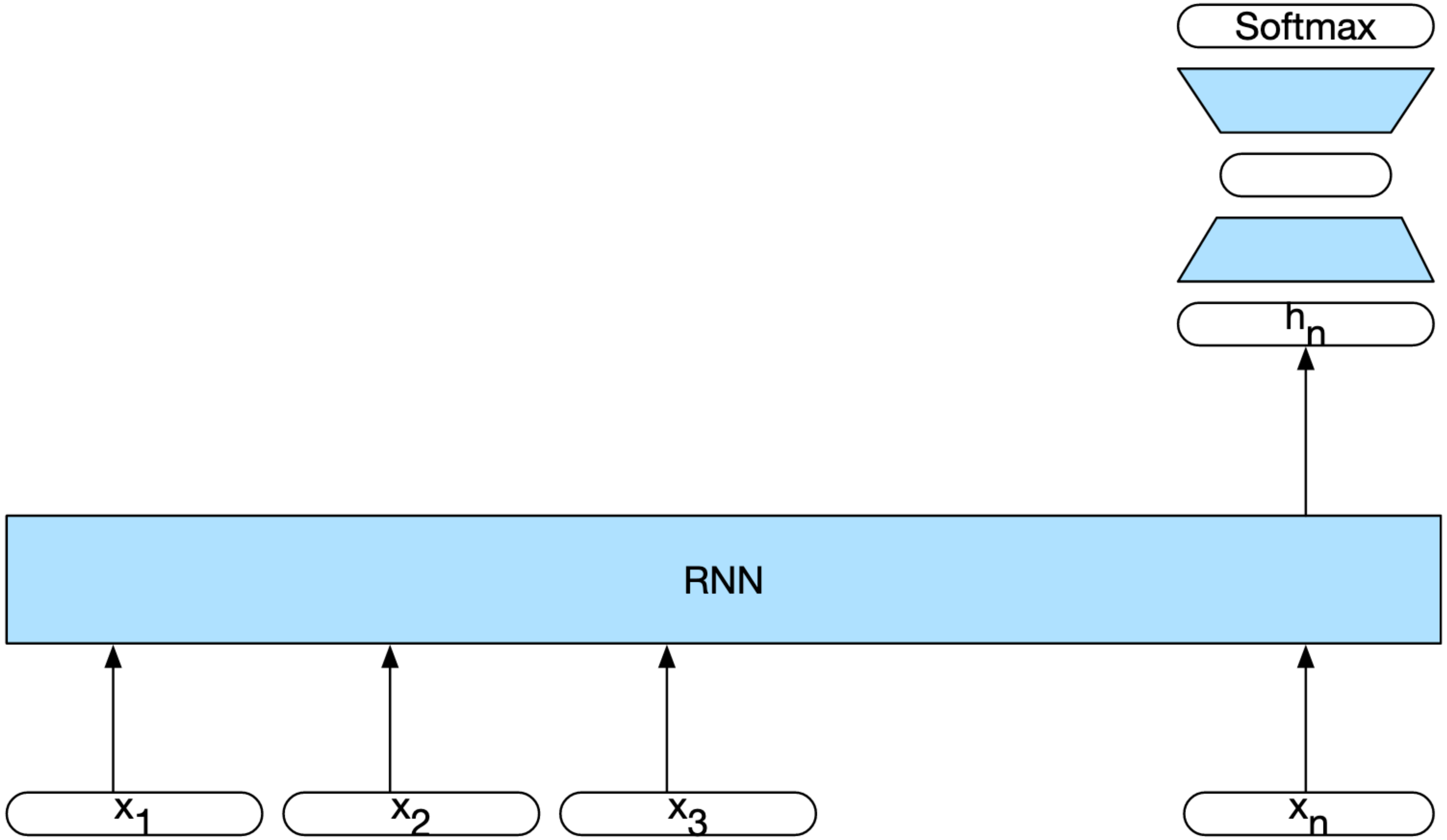
$$\text{probabilities} = \text{softmax}(\text{hidden}W^2 + b^2)$$

$$\text{hidden} = \tanh(\text{embeddings}W^1 + b^1)$$

$$\text{embeddings} = \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$$

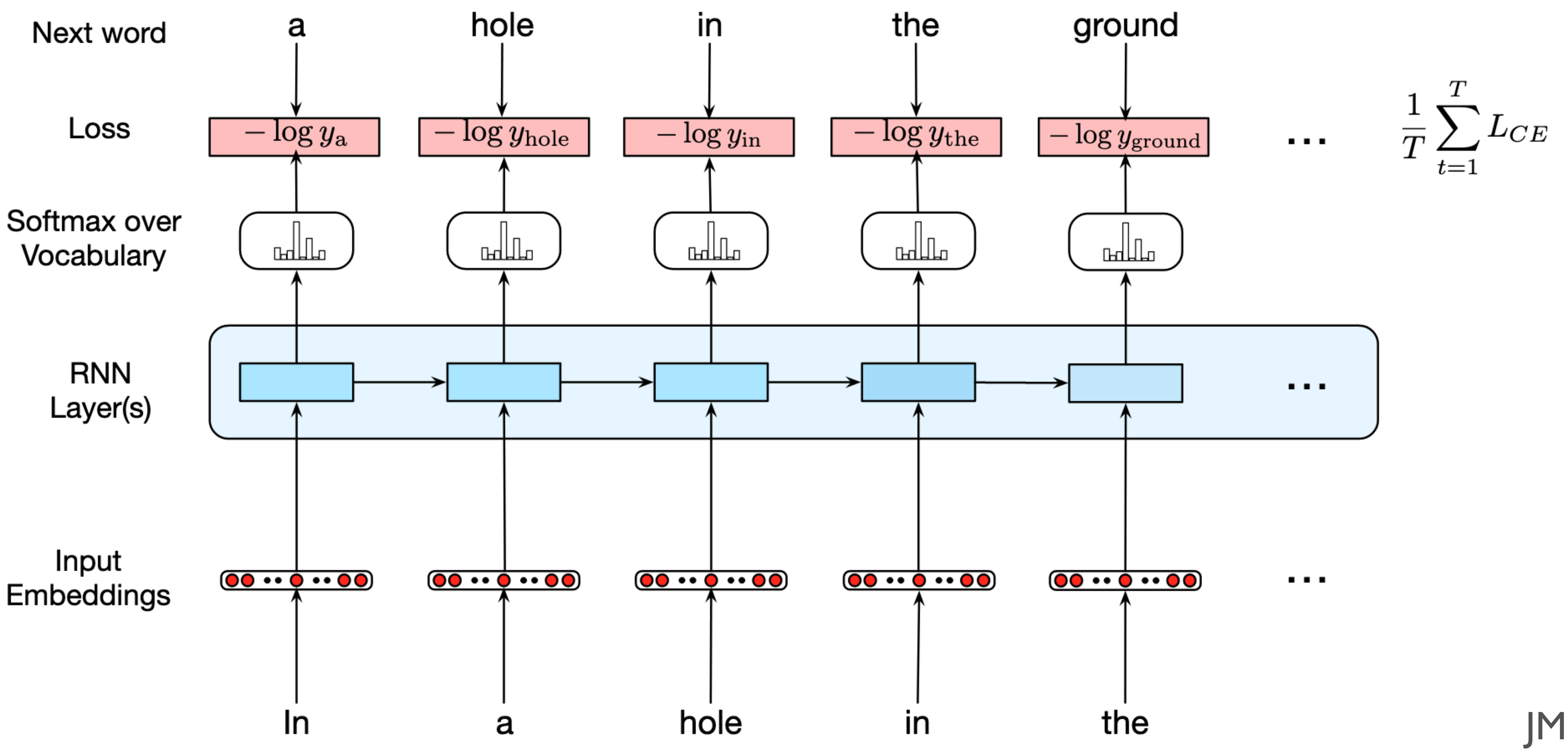
w_t: one-hot vector

RNN for Text Classification



JM sec 9.2.5

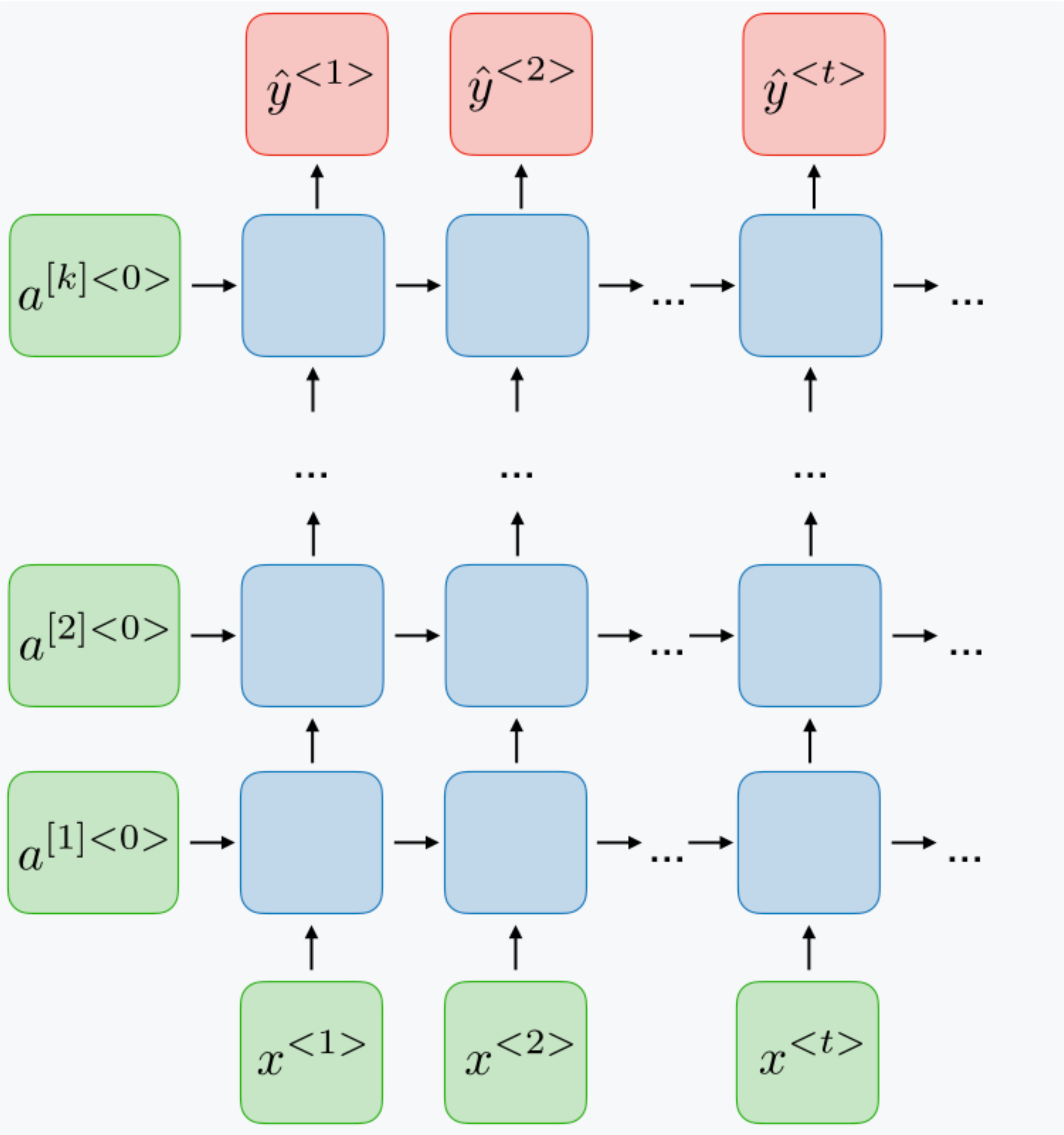
RNNs for Language Modeling



JM sec 9.2.3

Two Extensions

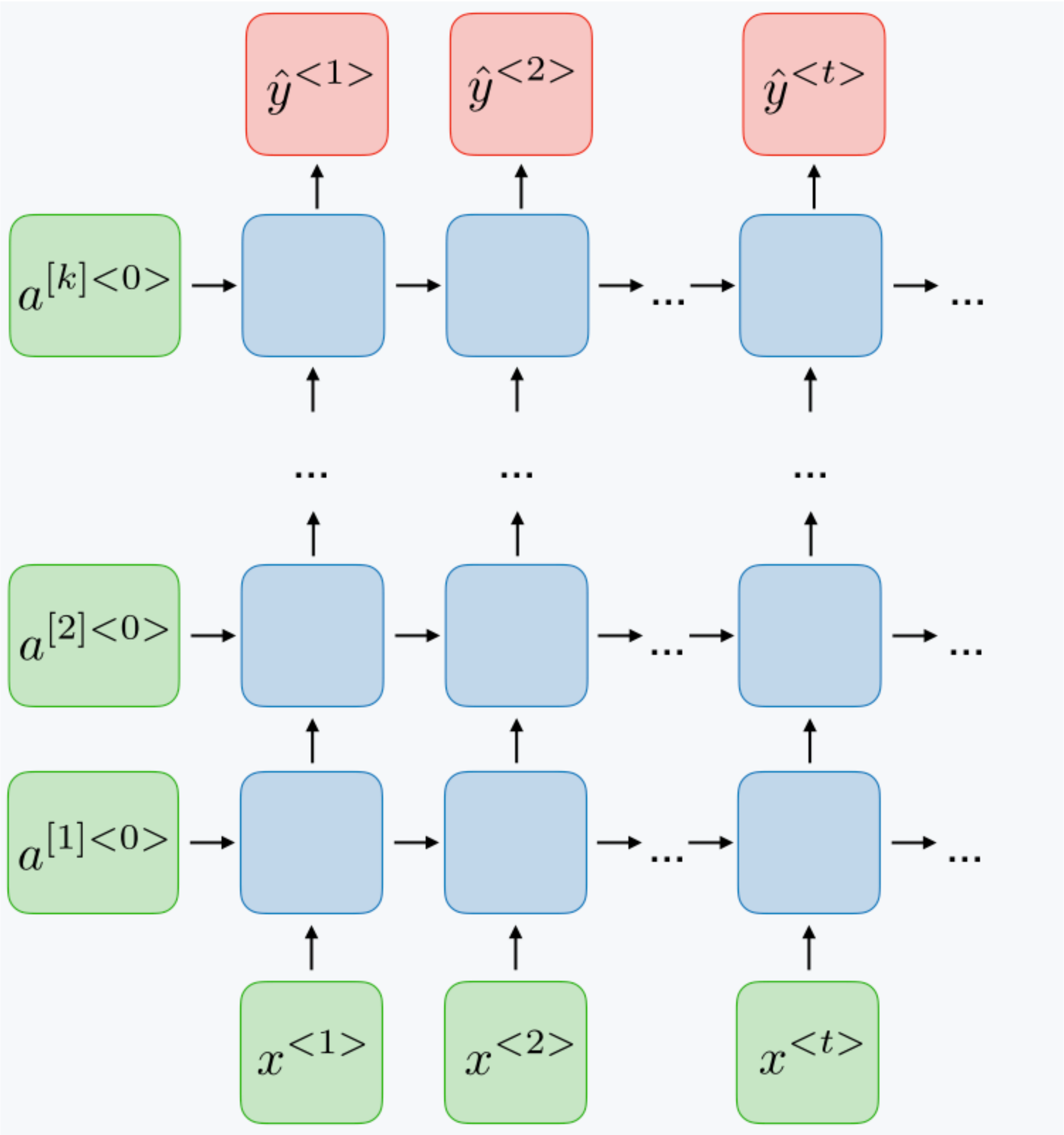
- Deep RNNs:



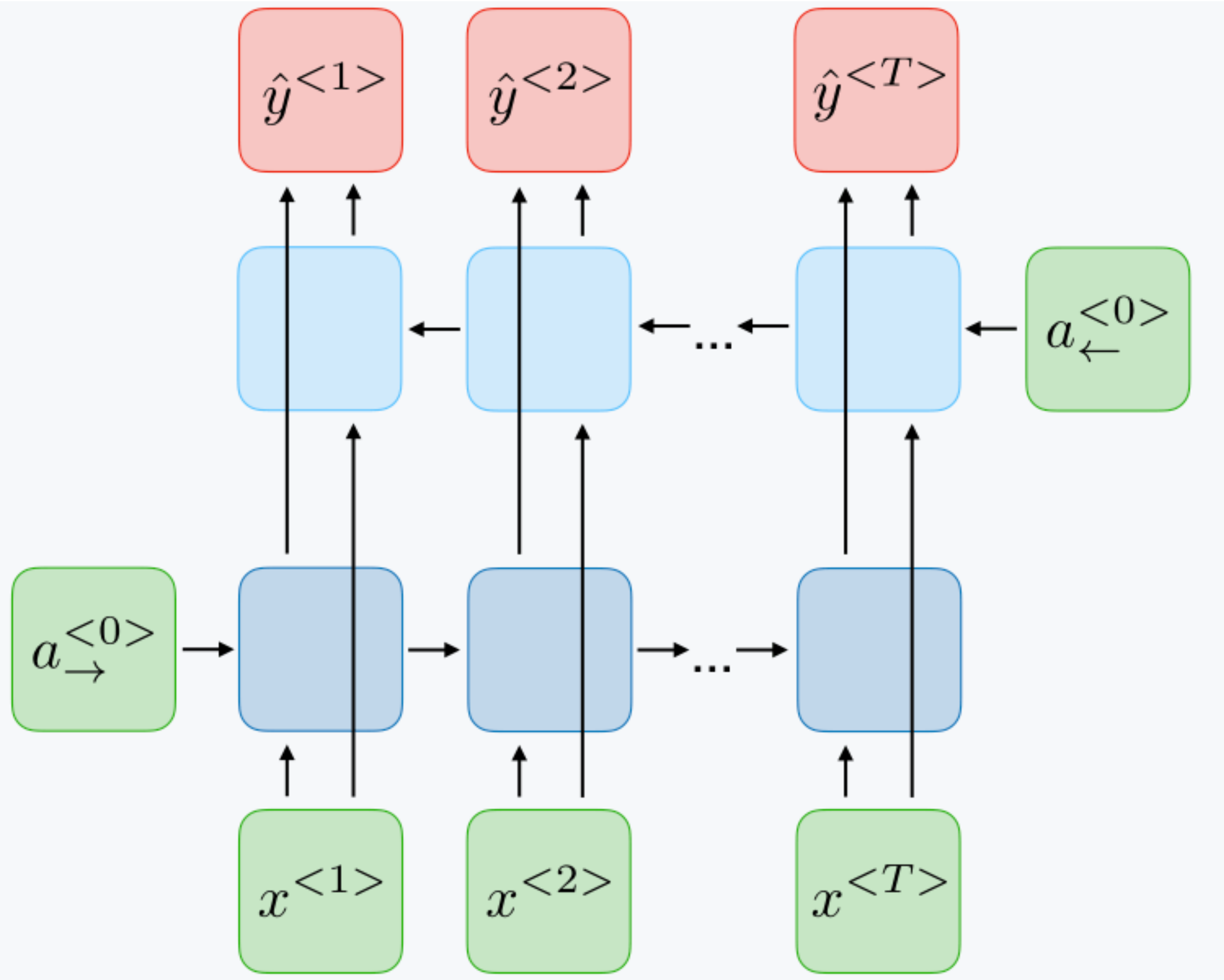
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



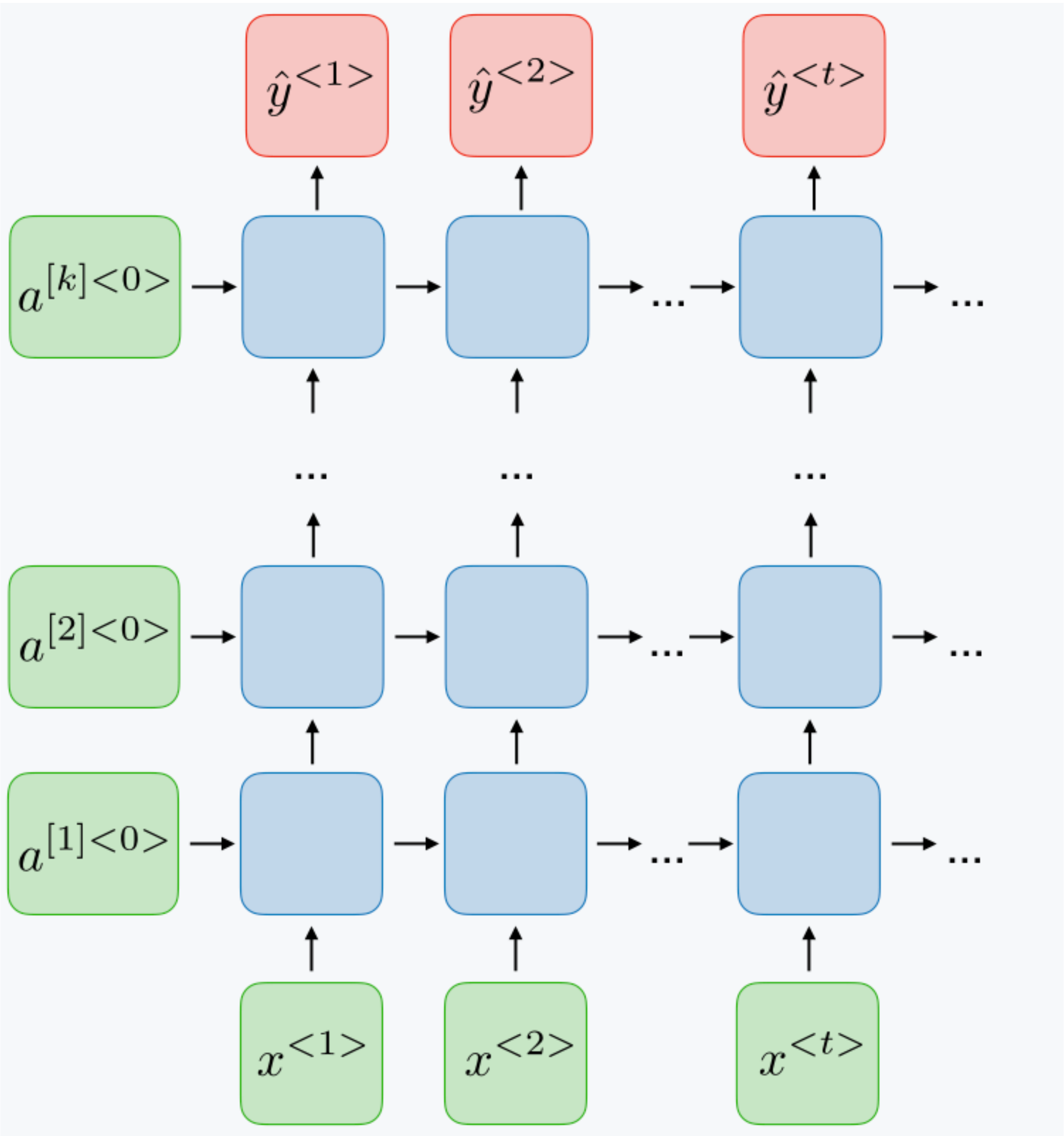
- Bidirectional RNNs:



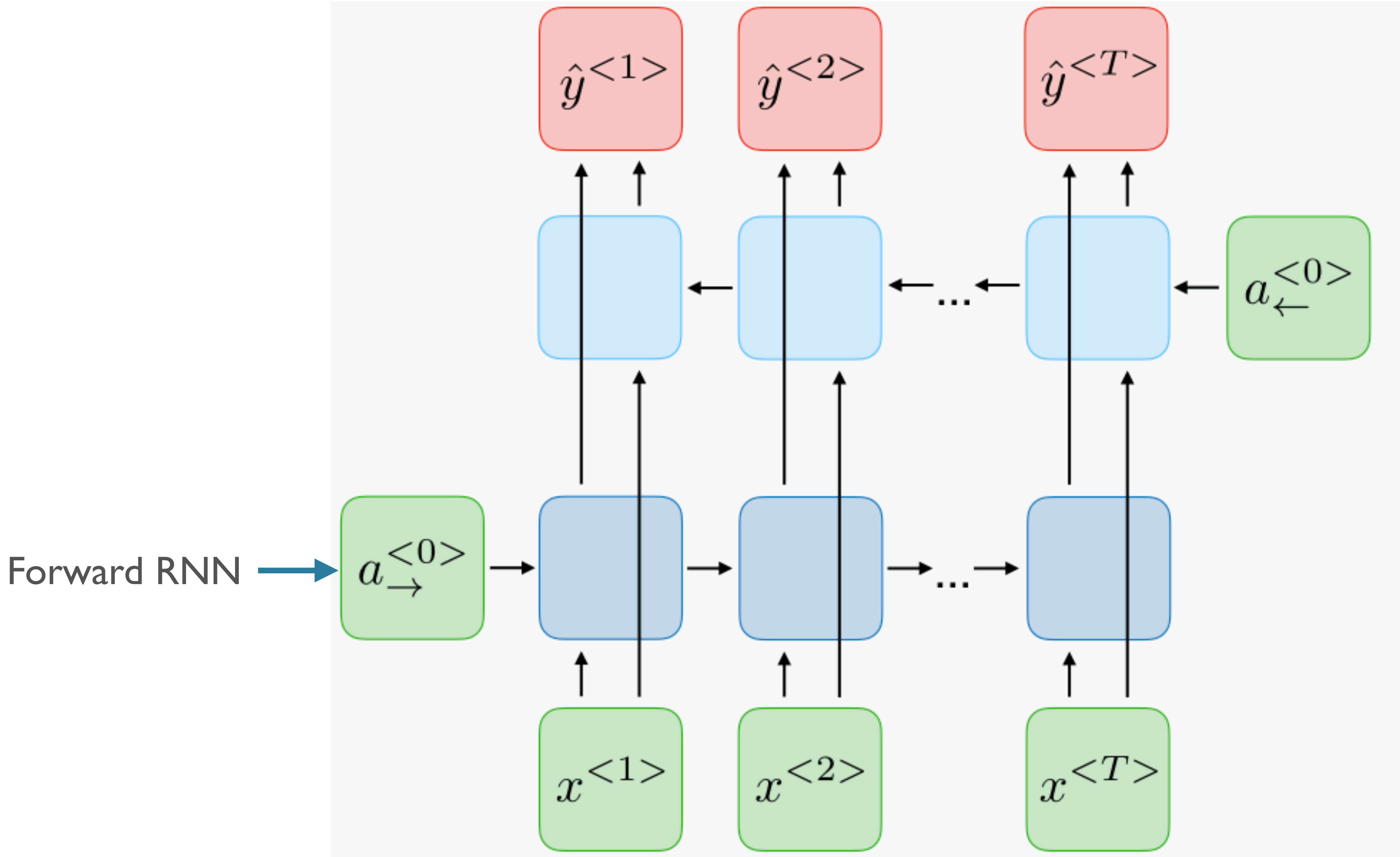
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



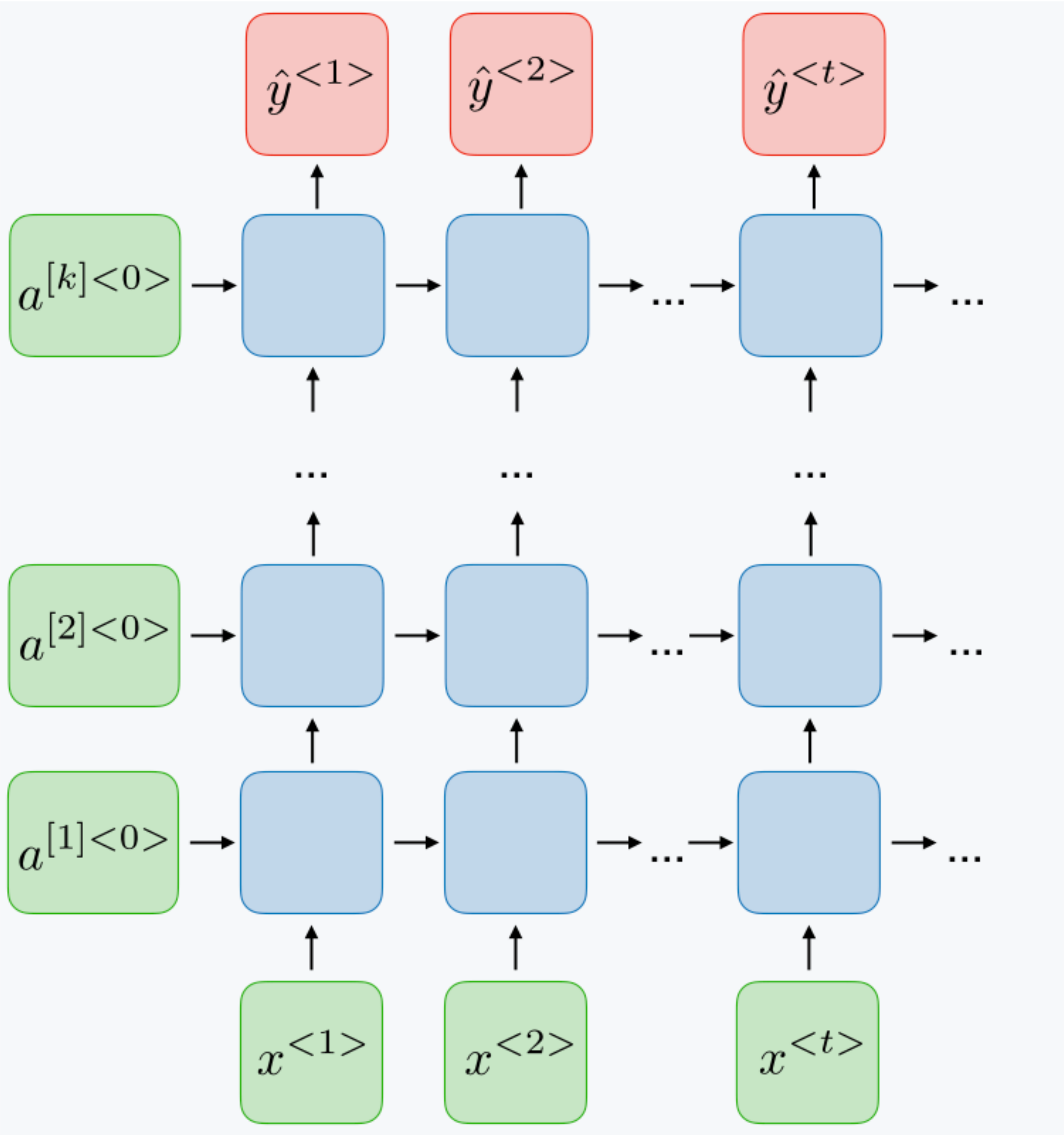
- Bidirectional RNNs:



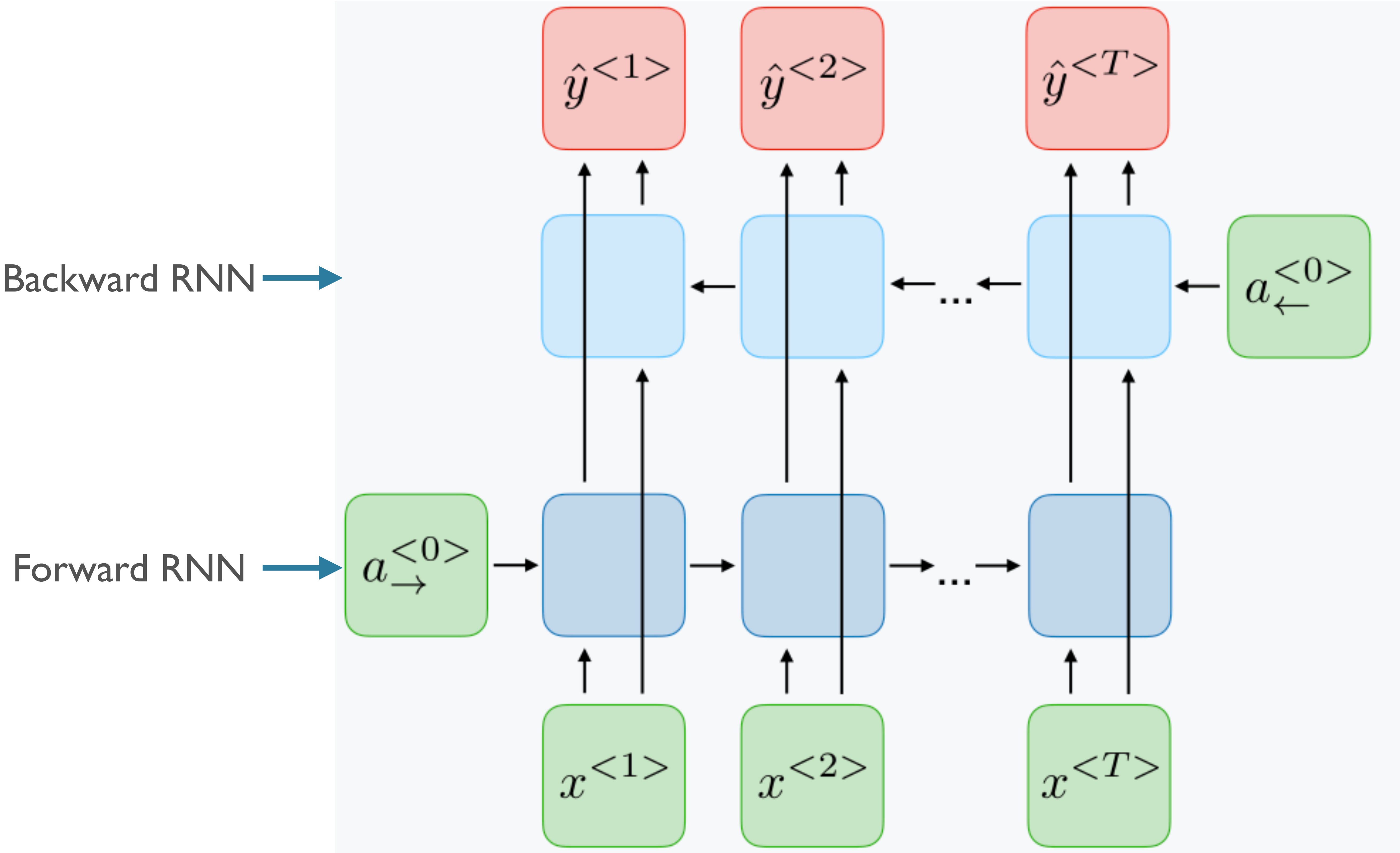
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



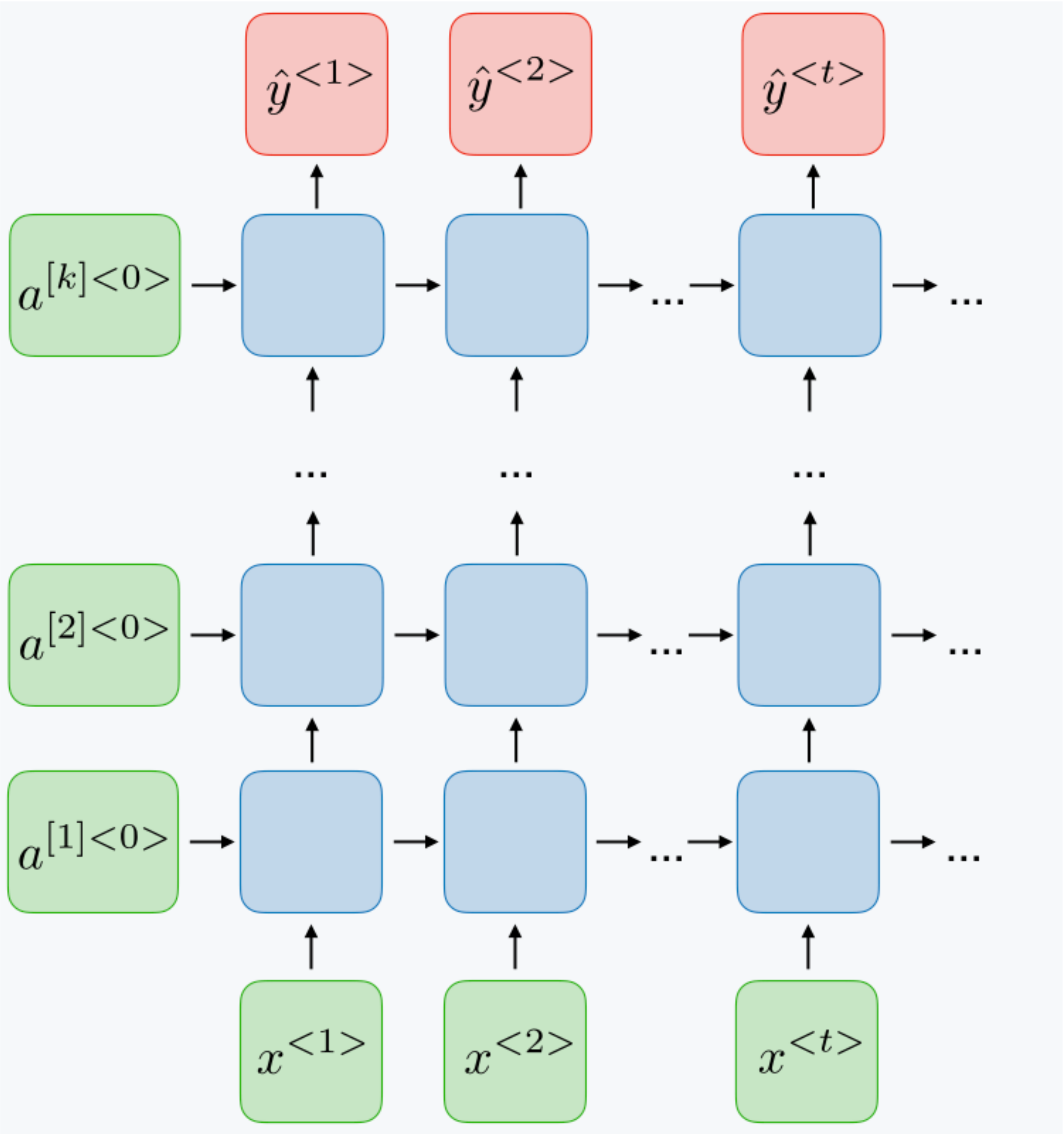
- Bidirectional RNNs:



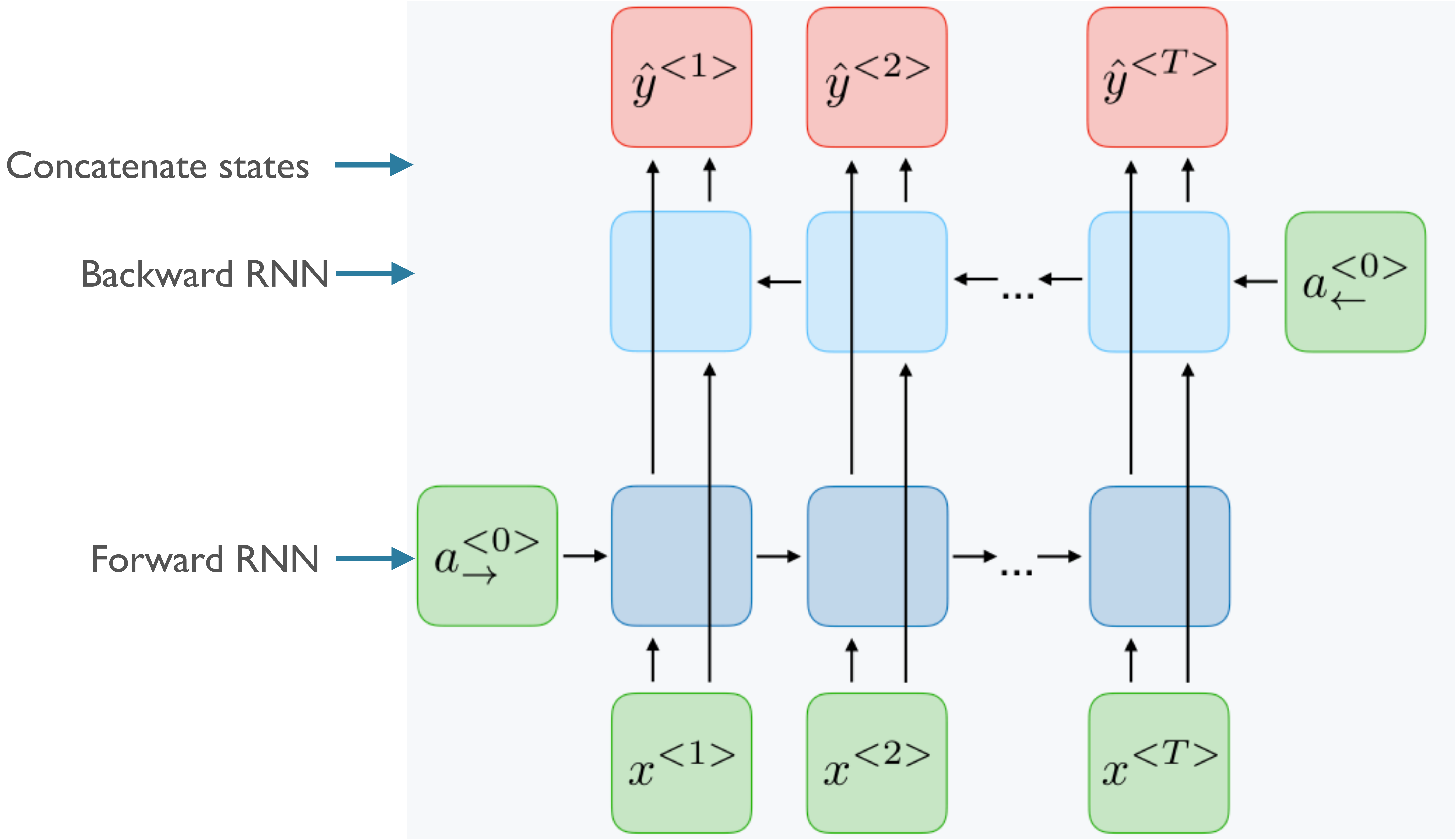
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:

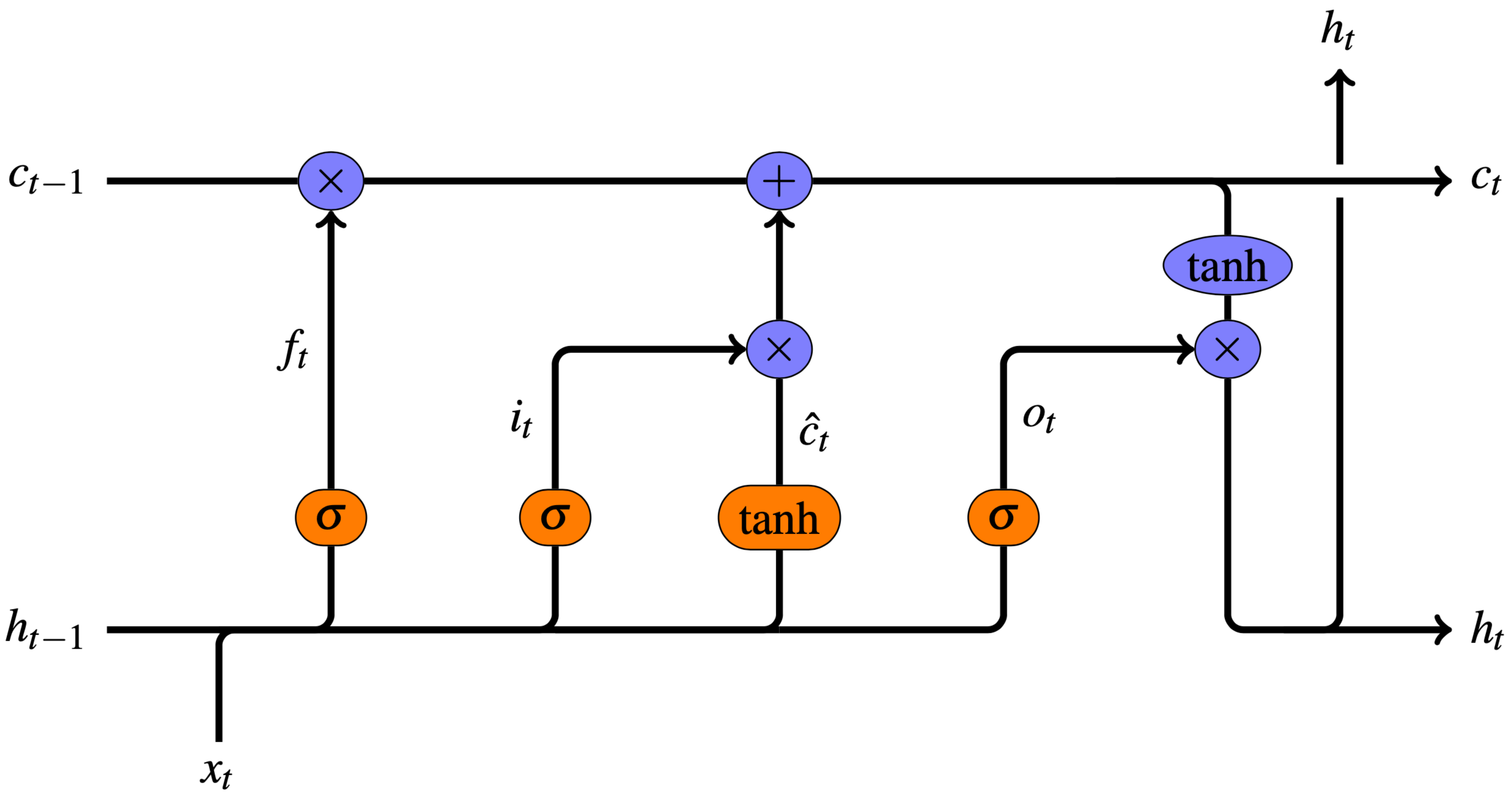


- Bidirectional RNNs:



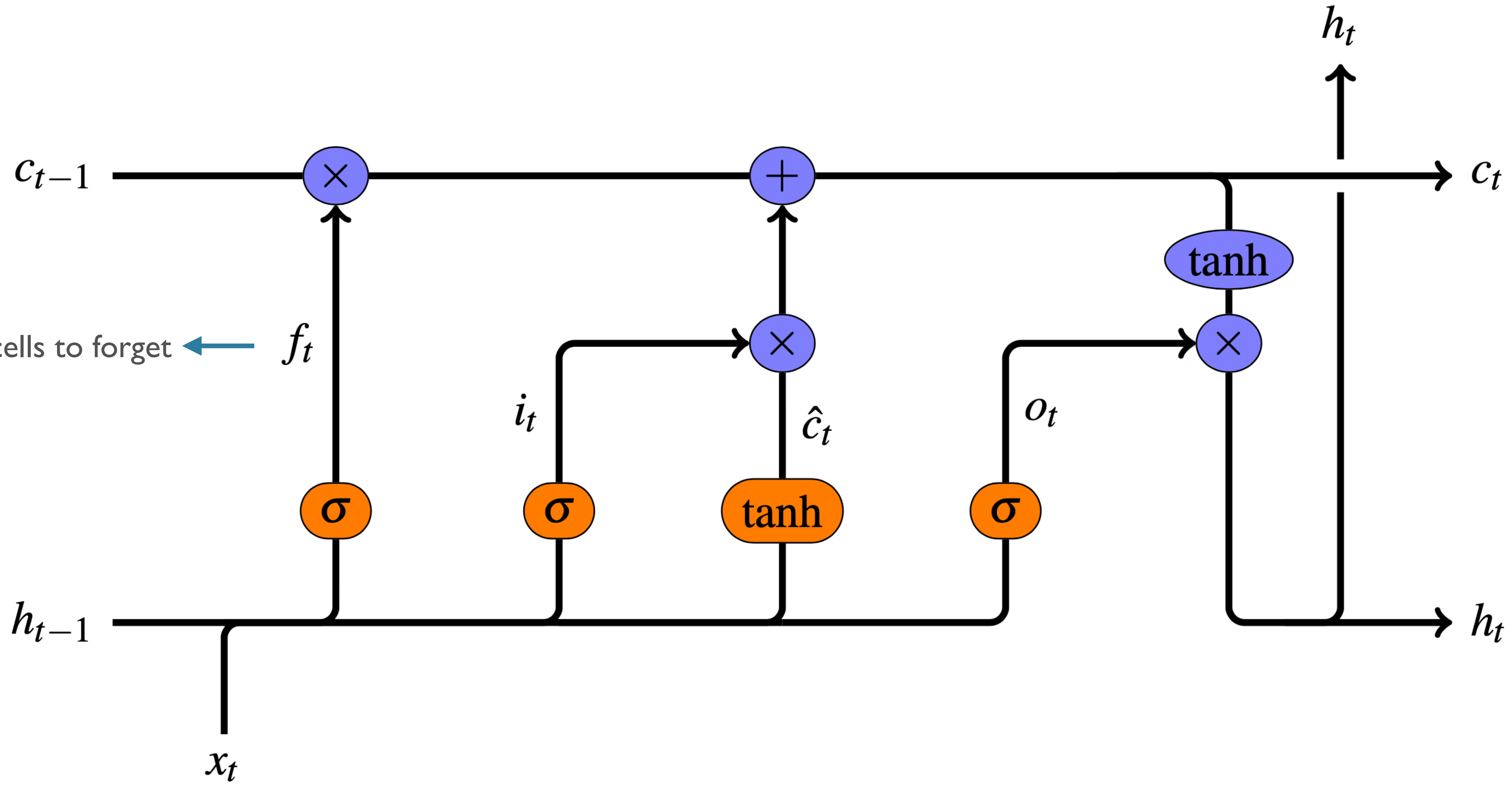
Source: RNN cheat sheet

LSTMs

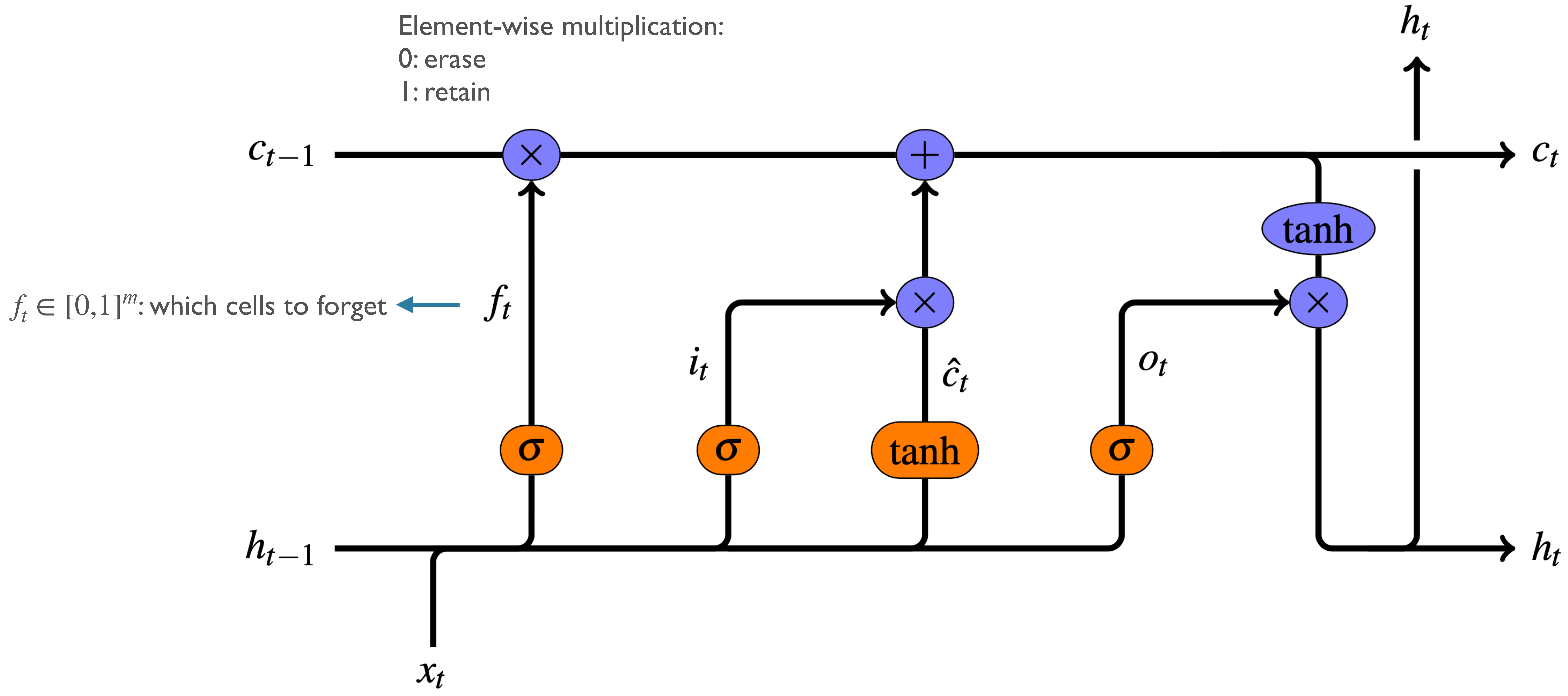


LSTMs

$f_t \in [0,1]^m$: which cells to forget

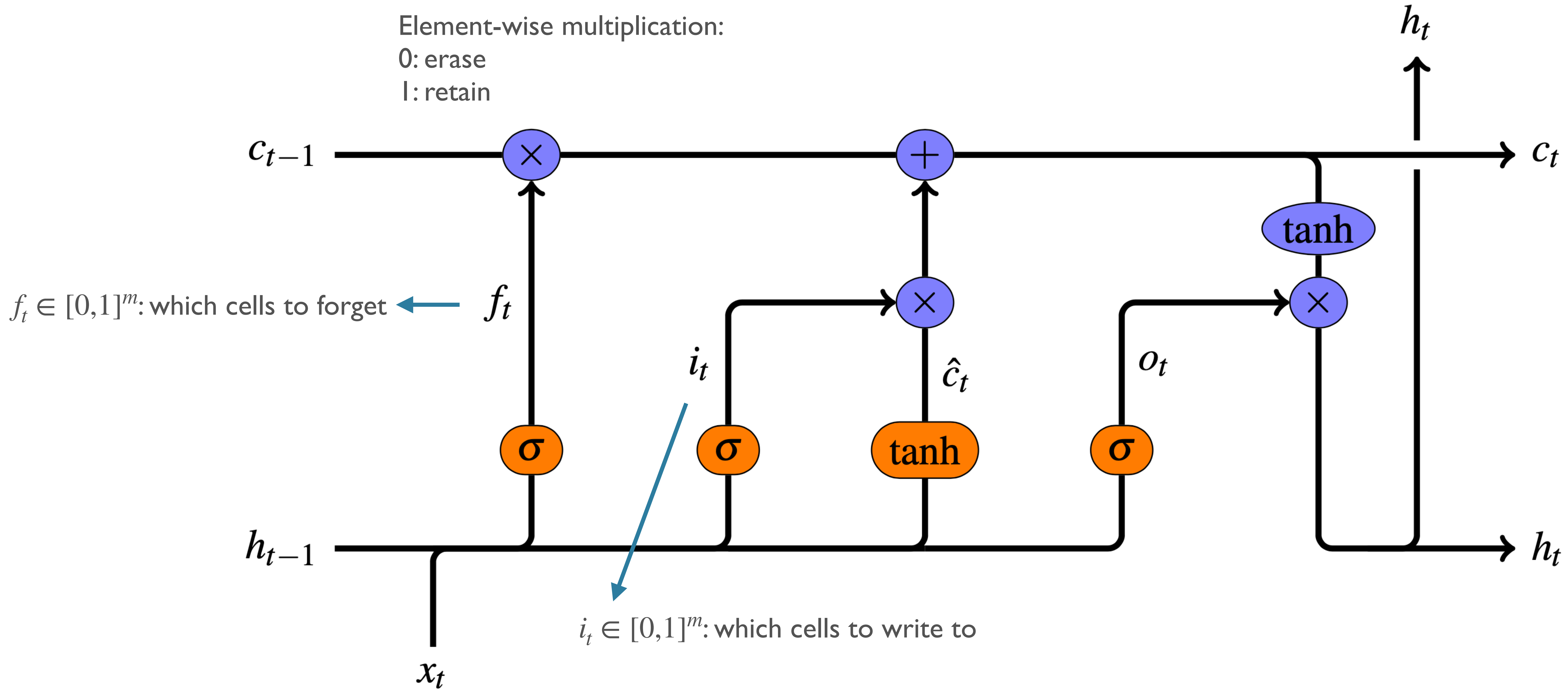


LSTMs



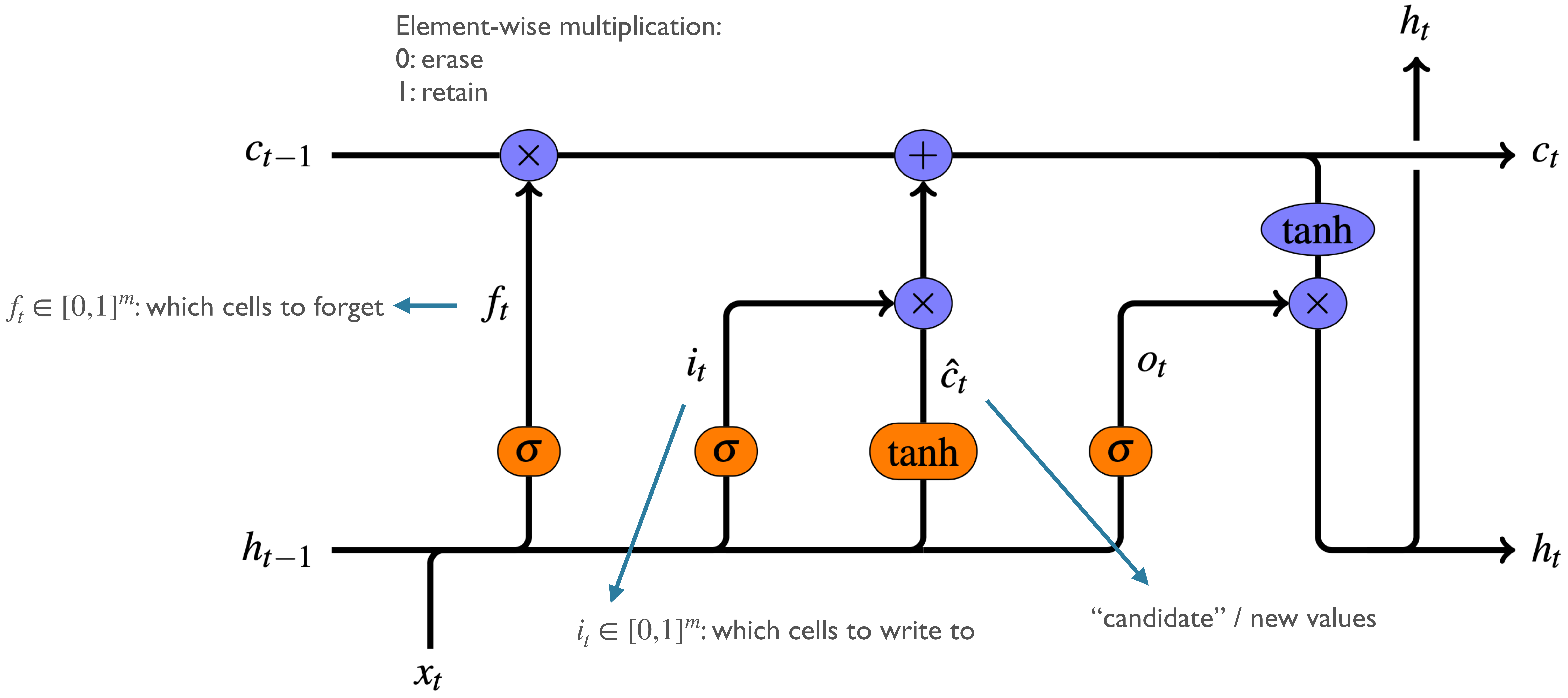
Element-wise multiplication:
0: erase
1: retain
$f_t \in [0,1]^m$: which cells to forget
c_{t-1}
c_t
h_{t-1}
h_t
h_t
x_t
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
tanh

LSTMs



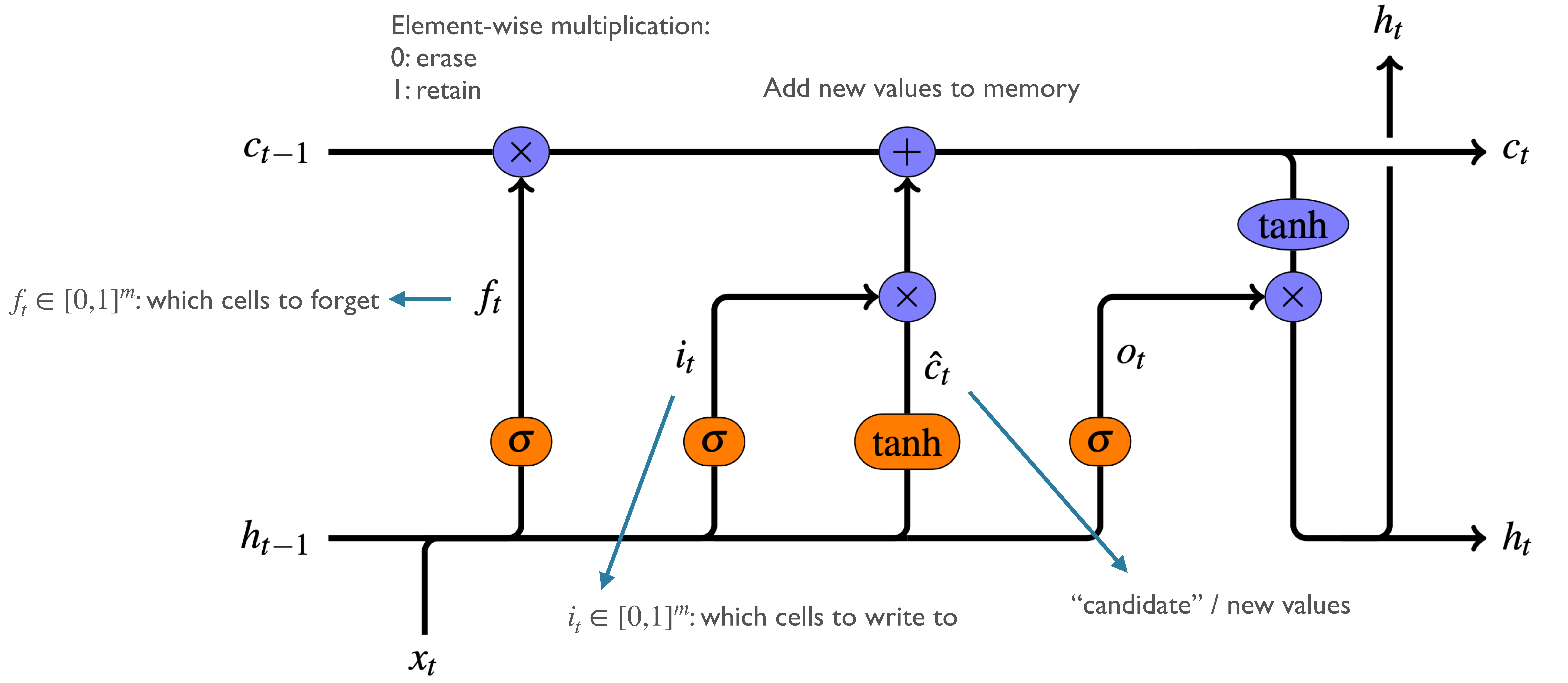
Element-wise multiplication:
0: erase
1: retain
c_{t-1}
c_t
h_t
h_{t-1}
h_t
x_t
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
tanh
$f_t \in [0,1]^m$: which cells to forget
$i_t \in [0,1]^m$: which cells to write to

LSTMs

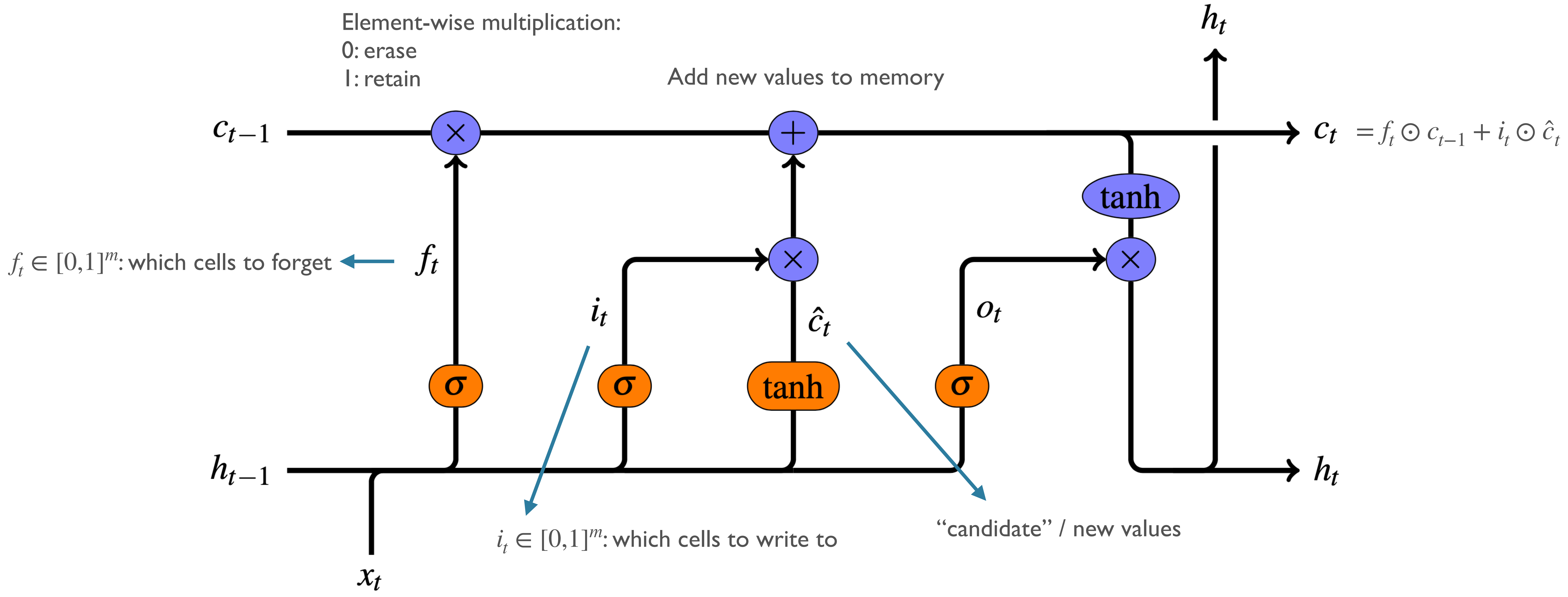


Element-wise multiplication:
0: erase
1: retain
$f_t \in [0,1]^m$: which cells to forget
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values
c_{t-1}
c_t
h_{t-1}
h_t
h_t
x_t
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
σ
tanh
tanh

LSTMs

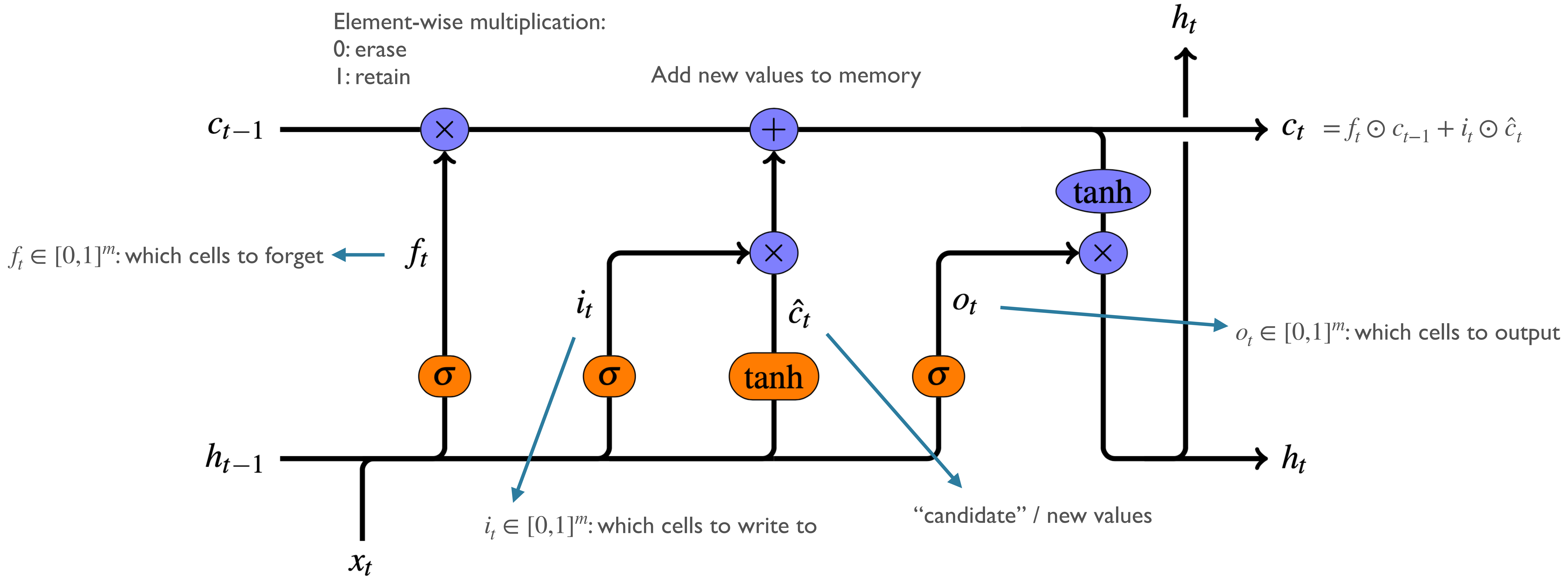

Element-wise multiplication:
0: erase
1: retain
Add new values to memory
h_t
c_{t-1}
c_t
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values

LSTMs



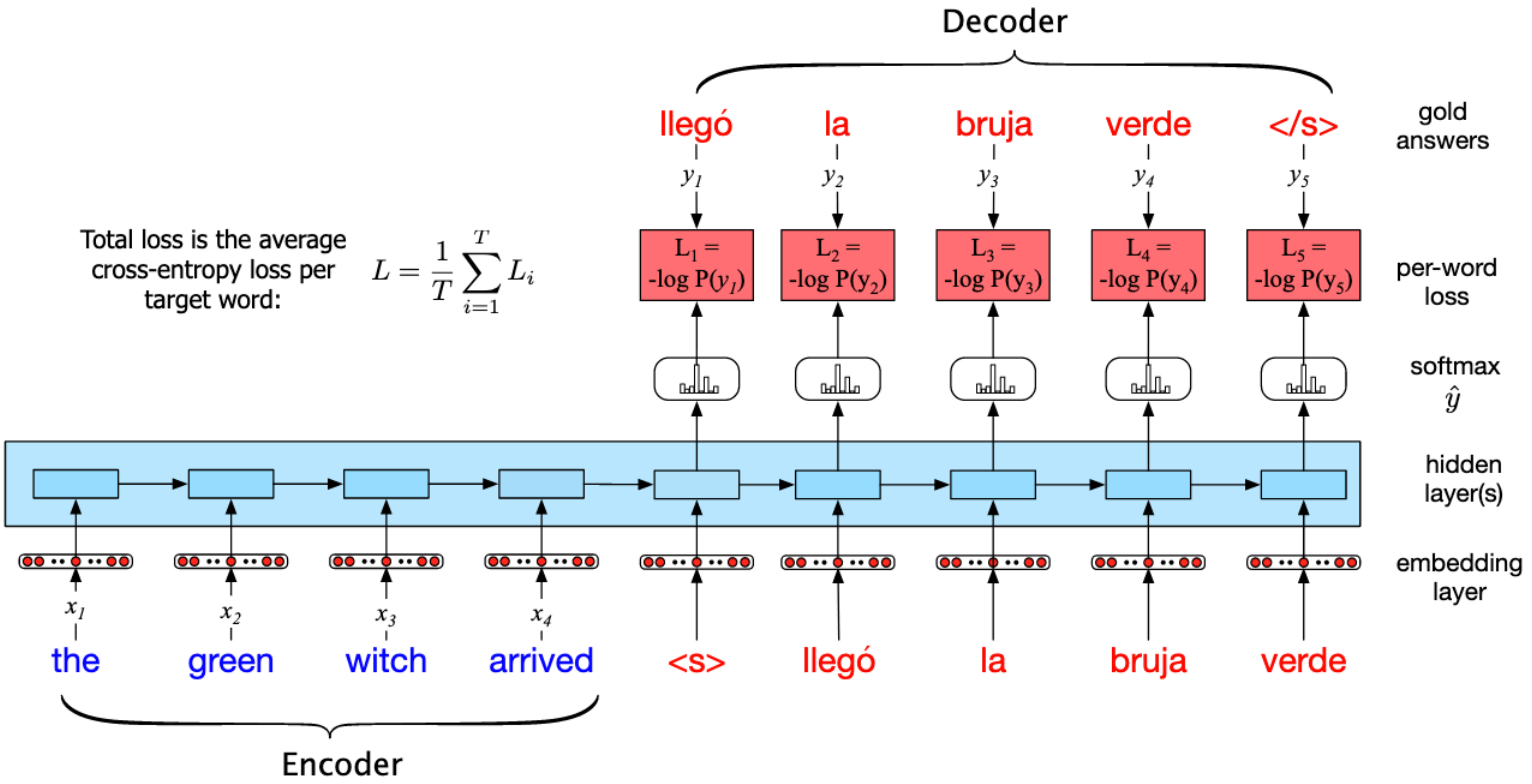
Element-wise multiplication:
0: erase
1: retain
Add new values to memory
h_t
c_{t-1}
c_t $= f_t \odot c_{t-1} + i_t \odot \hat{c}_t$
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values
x_t

LSTMs


Element-wise multiplication:
0: erase
1: retain
Add new values to memory
h_t
c_{t-1}
c_t $= f_t \odot c_{t-1} + i_t \odot \hat{c}_t$
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
$o_t \in [0,1]^m$: which cells to output
σ
σ
tanh
σ
h_{t-1}
h_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values
x_t

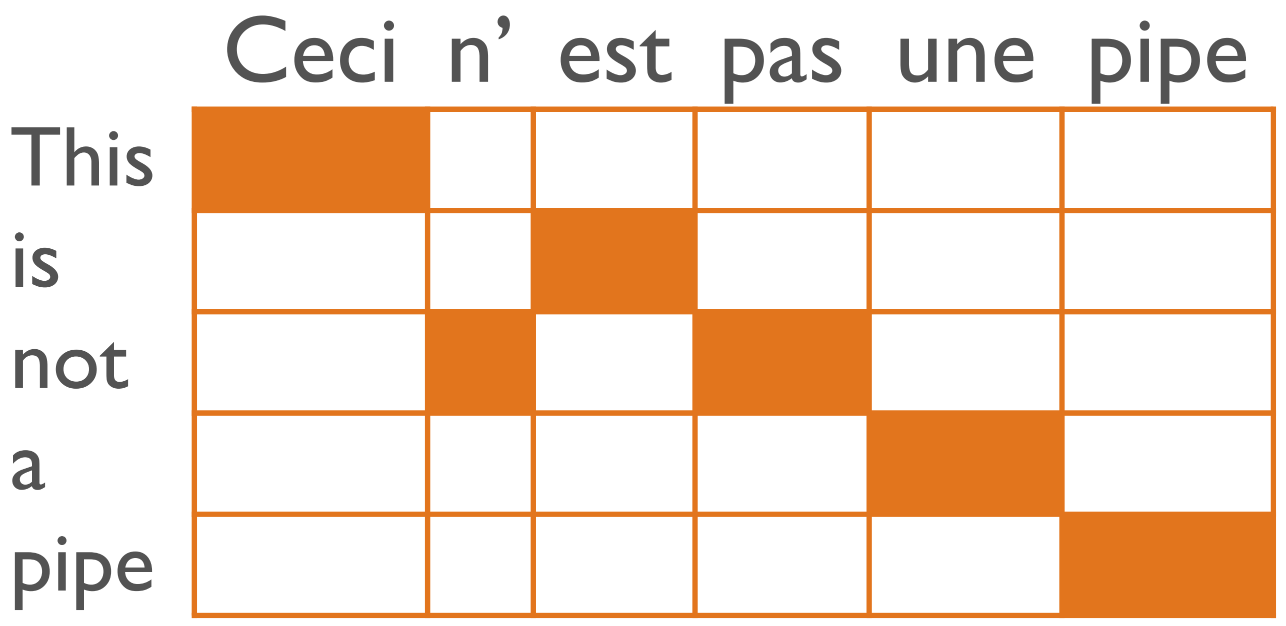
Training an encoder-decoder RNN

Total loss is the average cross-entropy loss per target word: $L = \frac{1}{T}\sum_{i=1}^{T} L_i$



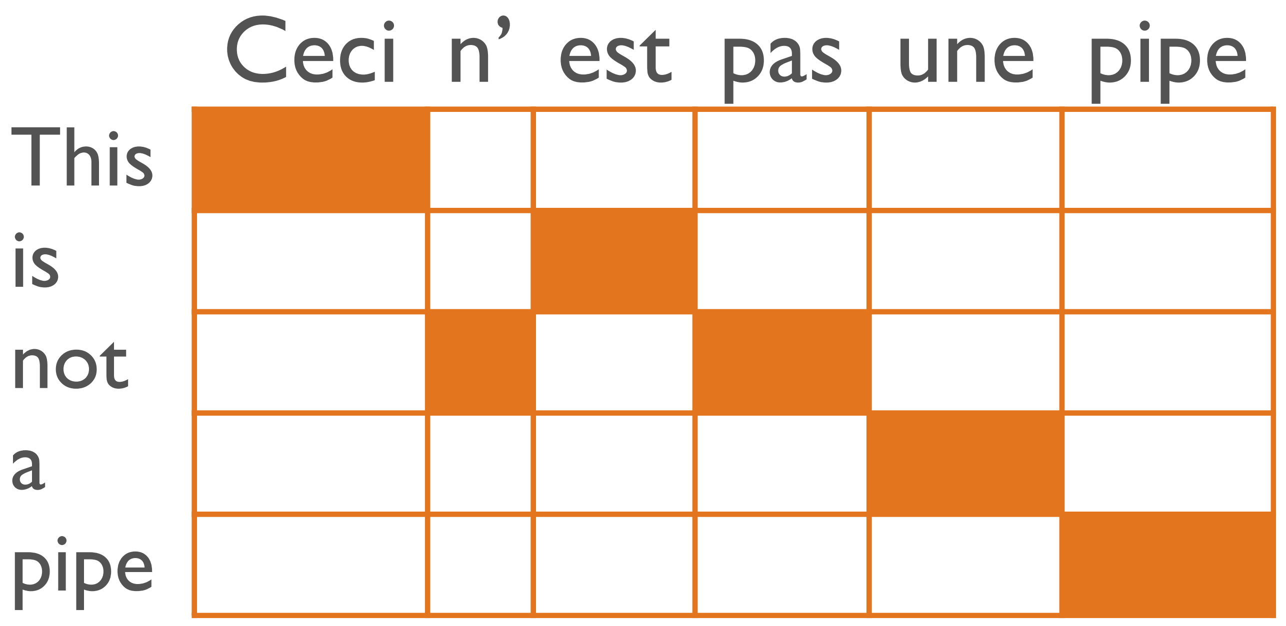
JM 11.3.1

Alignment, example



Alignment, example

Ceci n’ est pas une pipe



Alignment, example

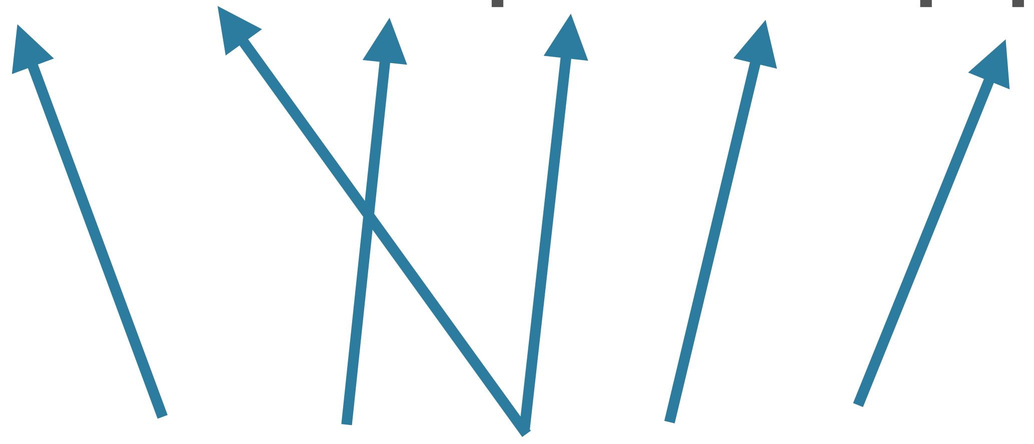
Ceci n’ est pas une pipe

This is not a pipe

	Ceci	n’	est	pas	une	pipe
This	■					
is			■			
not		■		■		
a					■	
pipe						■

Alignment, example

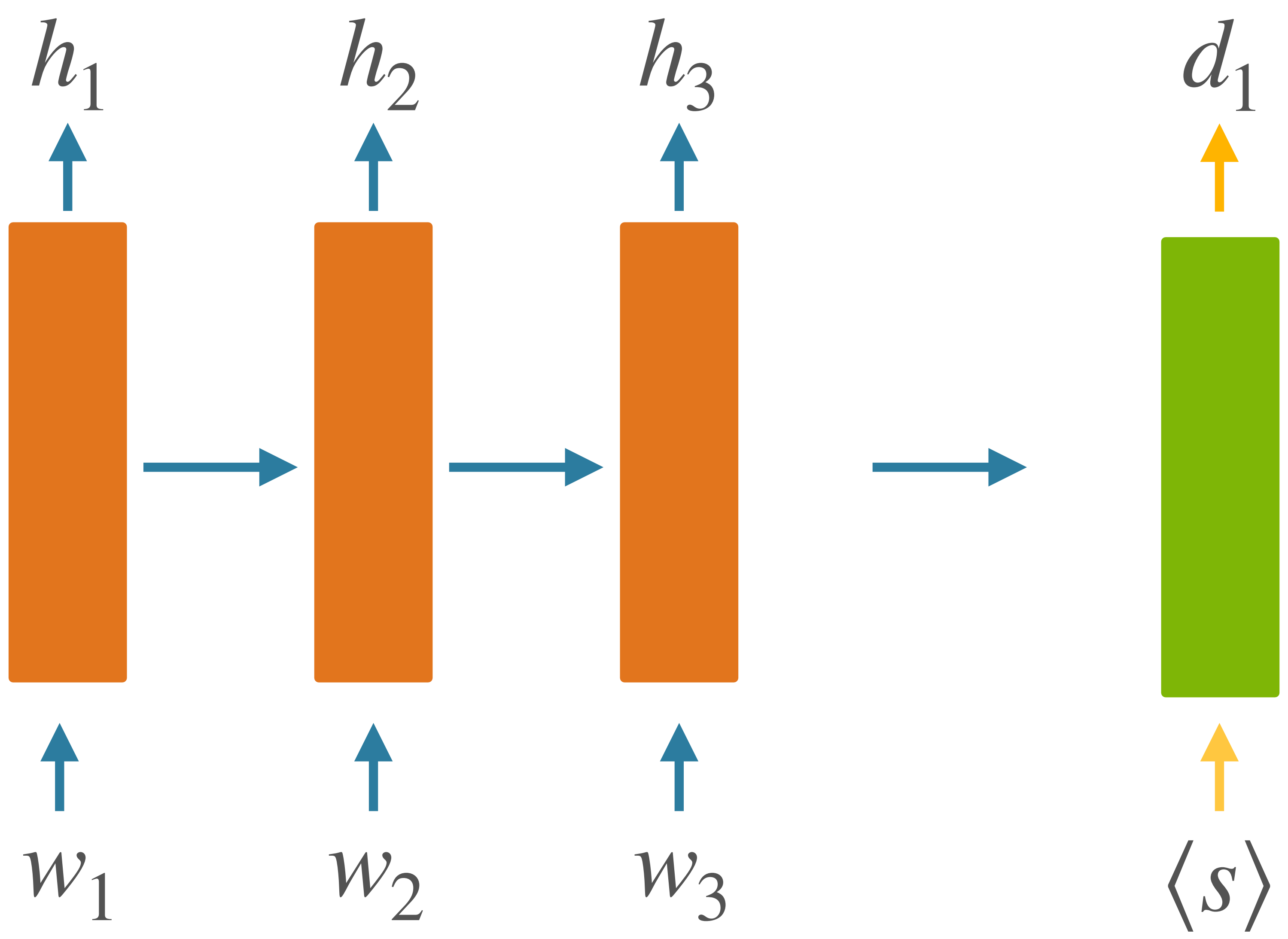
Ceci n’ est pas une pipe



This is not a pipe

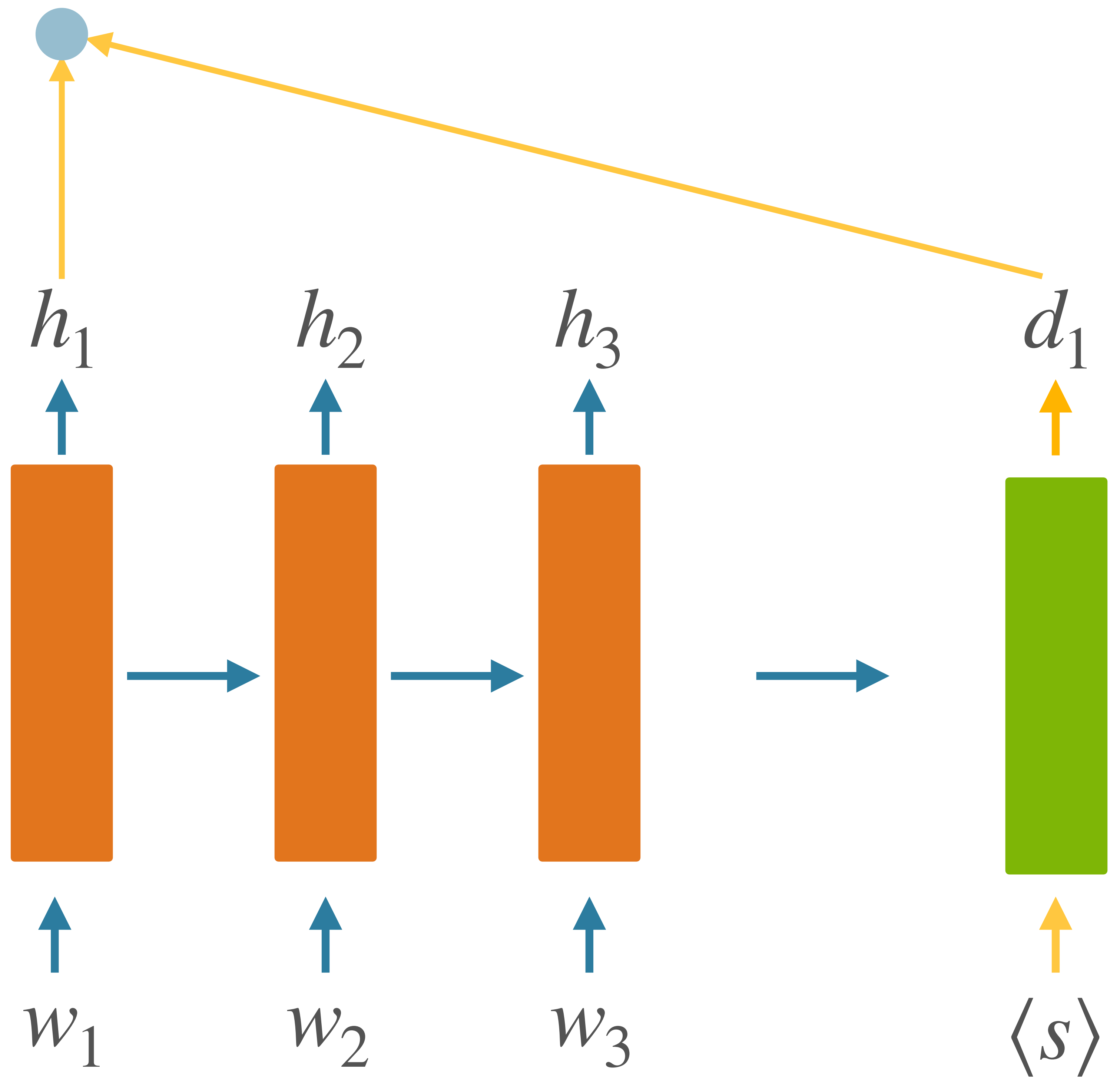
	Ceci	n’	est	pas	une	pipe
This	■					
is			■			
not		■		■		
a					■	
pipe						■

Adding Attention



[Badhanau et al 2014]

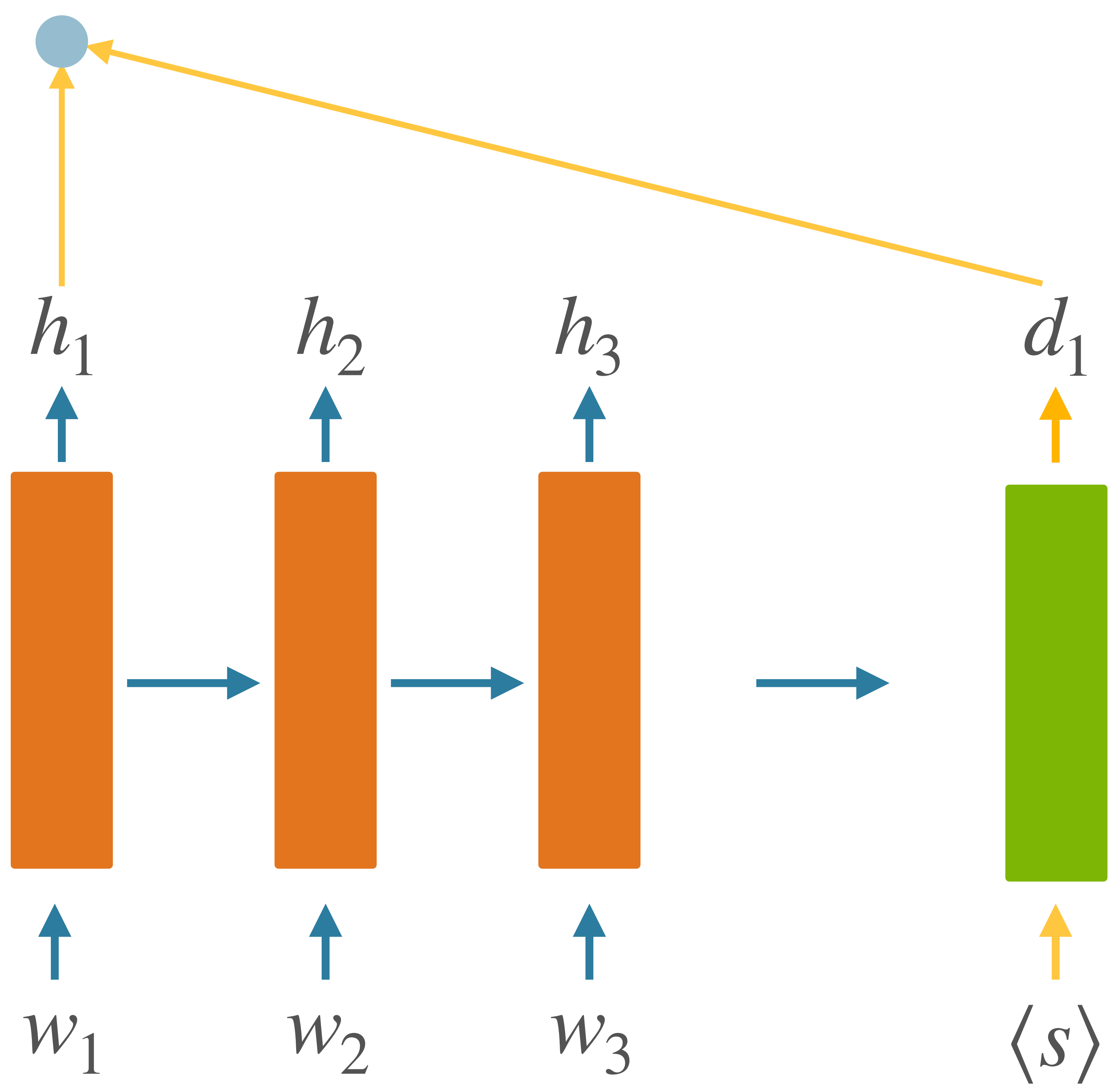
Adding Attention



Badhanau et al 2014

Adding Attention

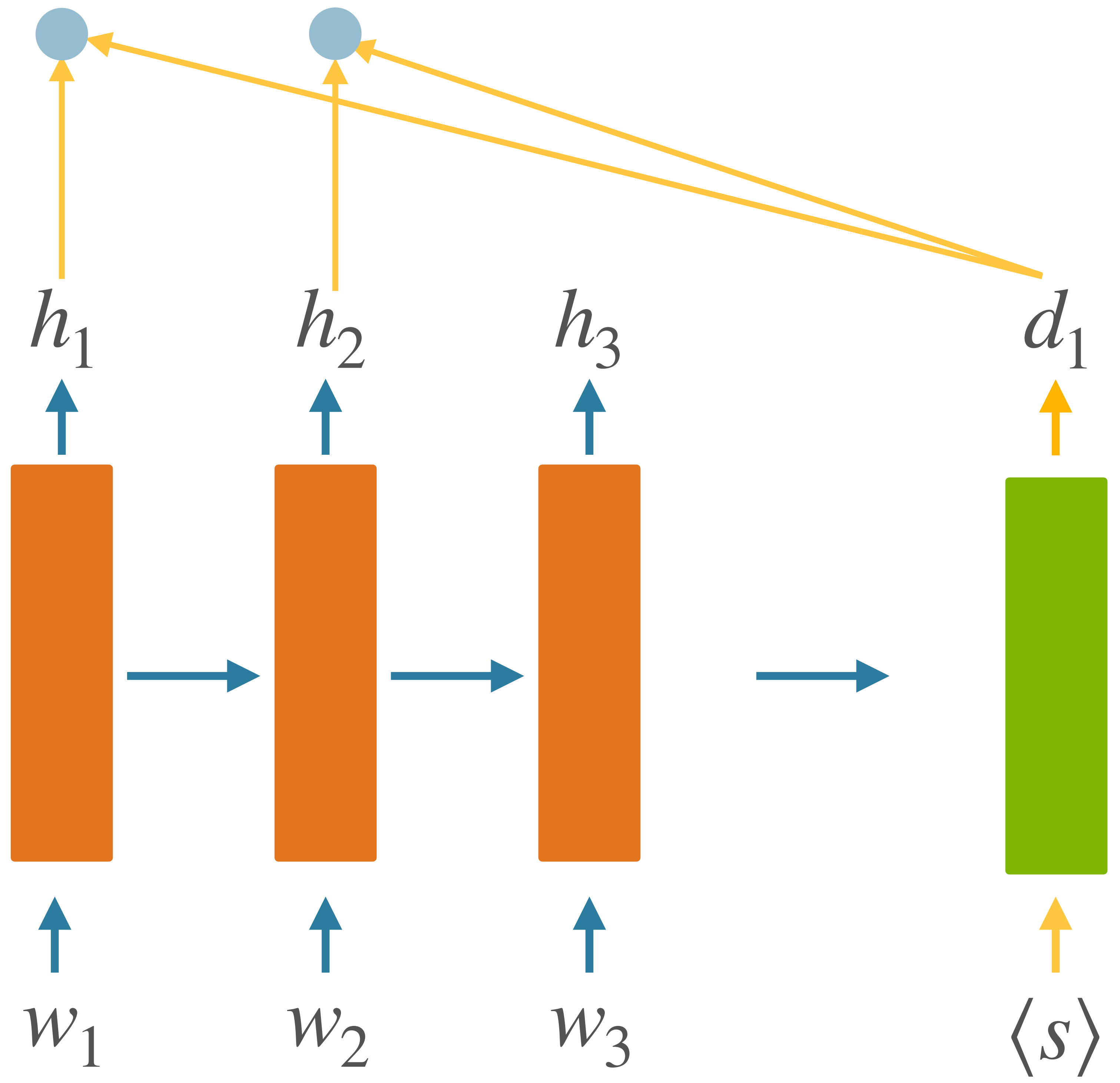
$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



Badhanau et al 2014

Adding Attention

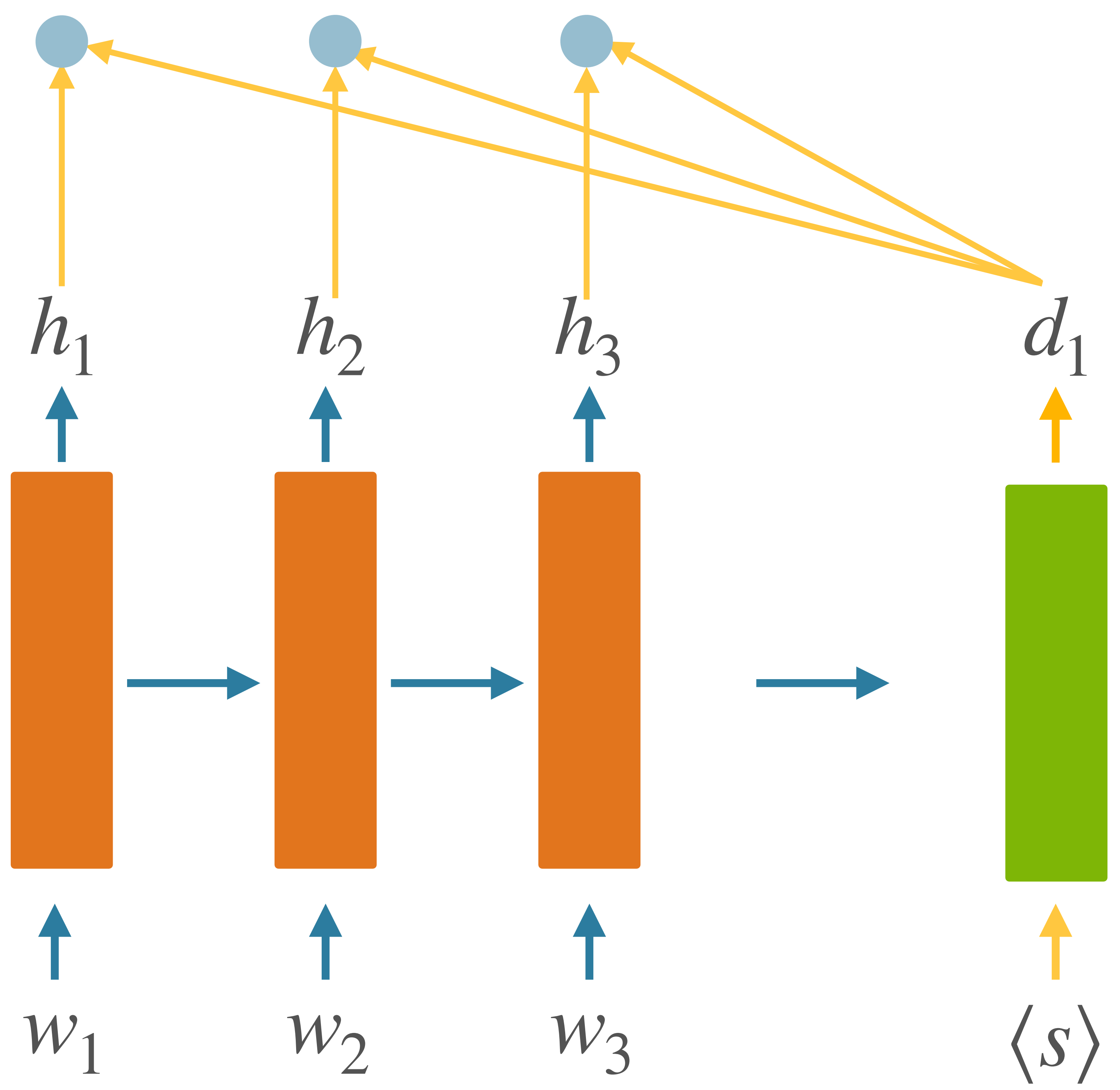
$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



[Badhanau et al 2014](#)

Adding Attention

$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)

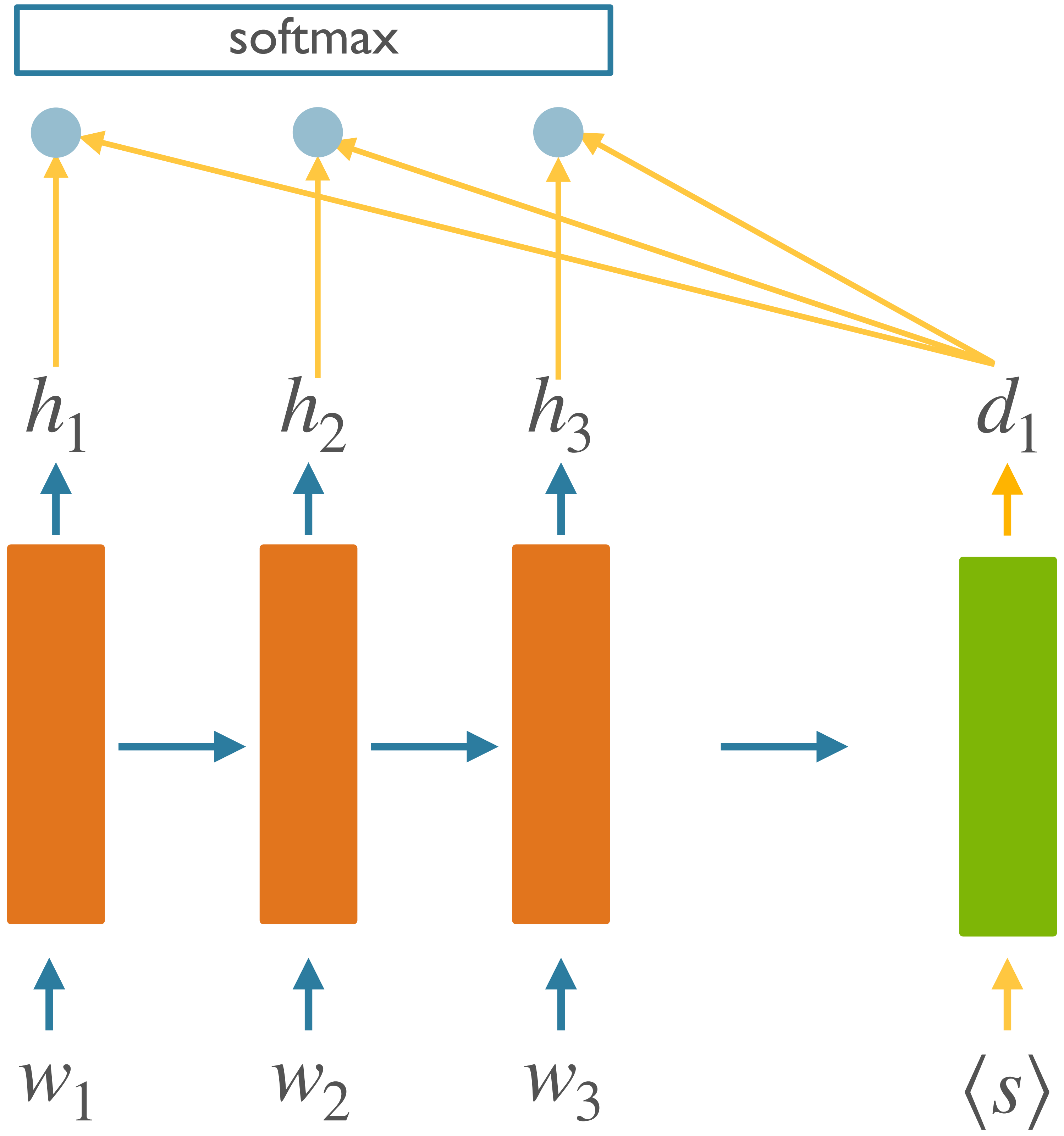


Badhanau et al 2014

Adding Attention

$e_{ij} = \text{softmax}(\alpha)_j$

$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



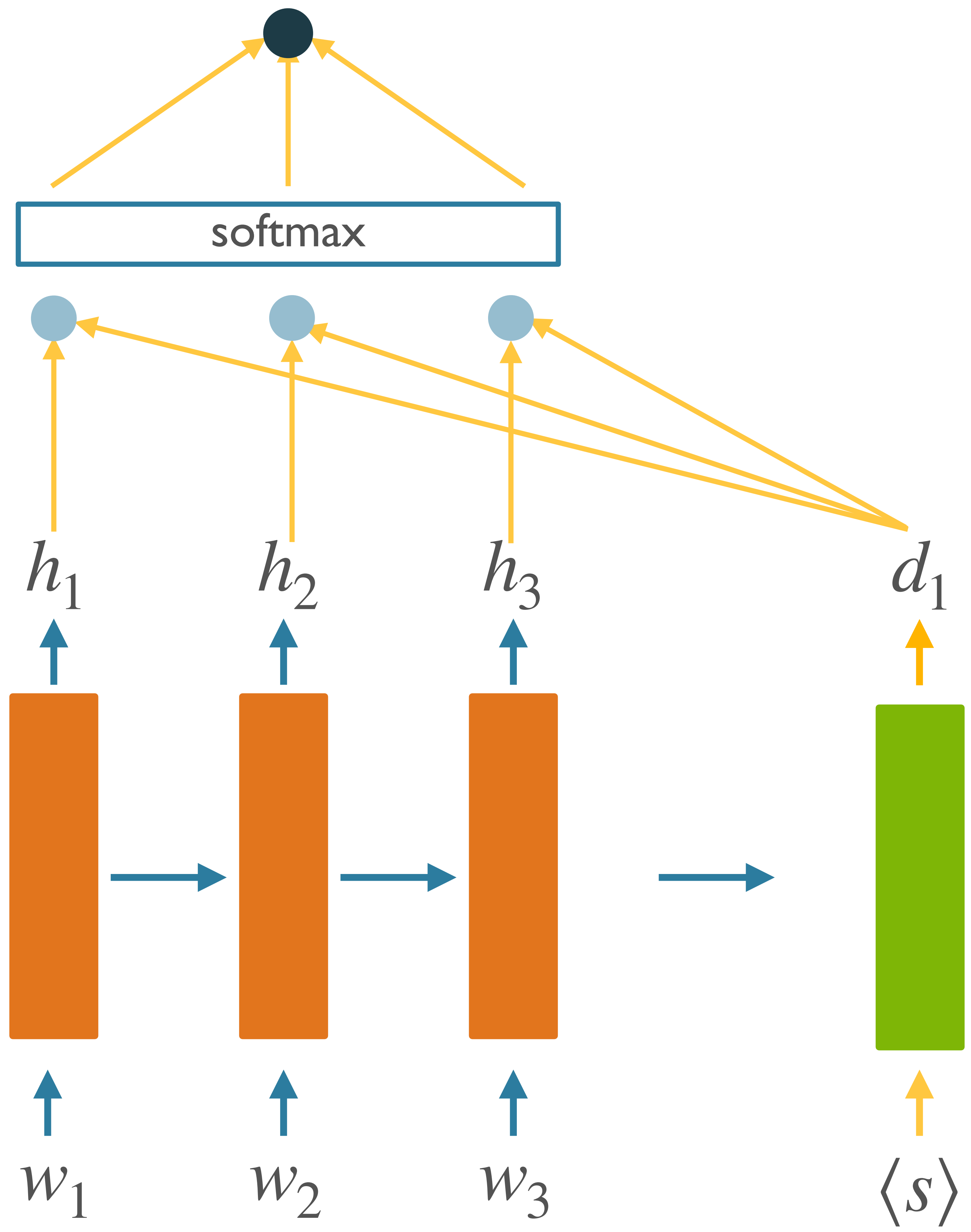
Badhanau et al 2014

Adding Attention

$c_i = \Sigma_j e_{ij} h_j$

$e_{ij} = \text{softmax}(\alpha)_j$

$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



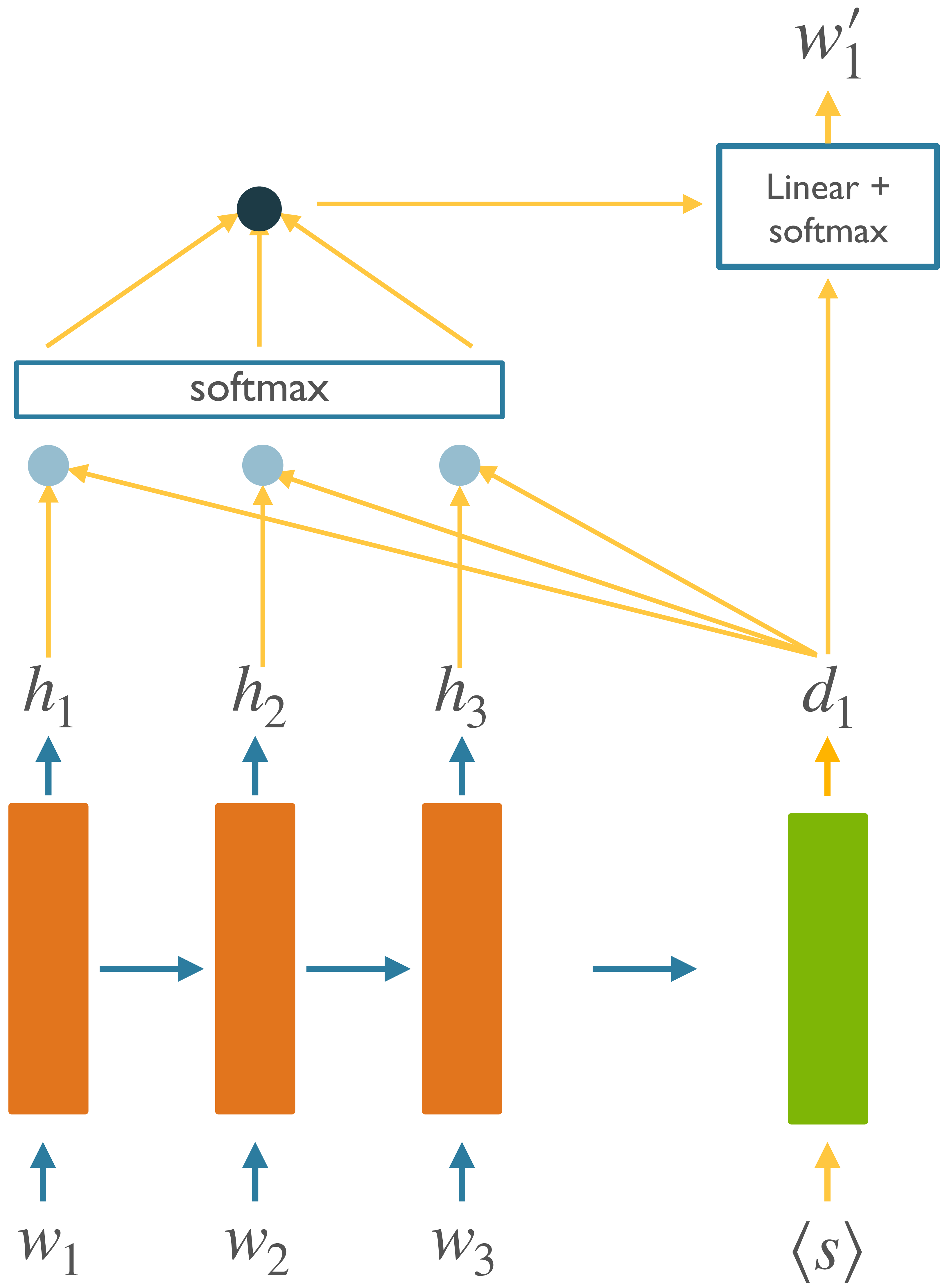
Badhanau et al 2014

Adding Attention

$c_i = \Sigma_j e_{ij} h_j$

$e_{ij} = \text{softmax}(\alpha)_j$

$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



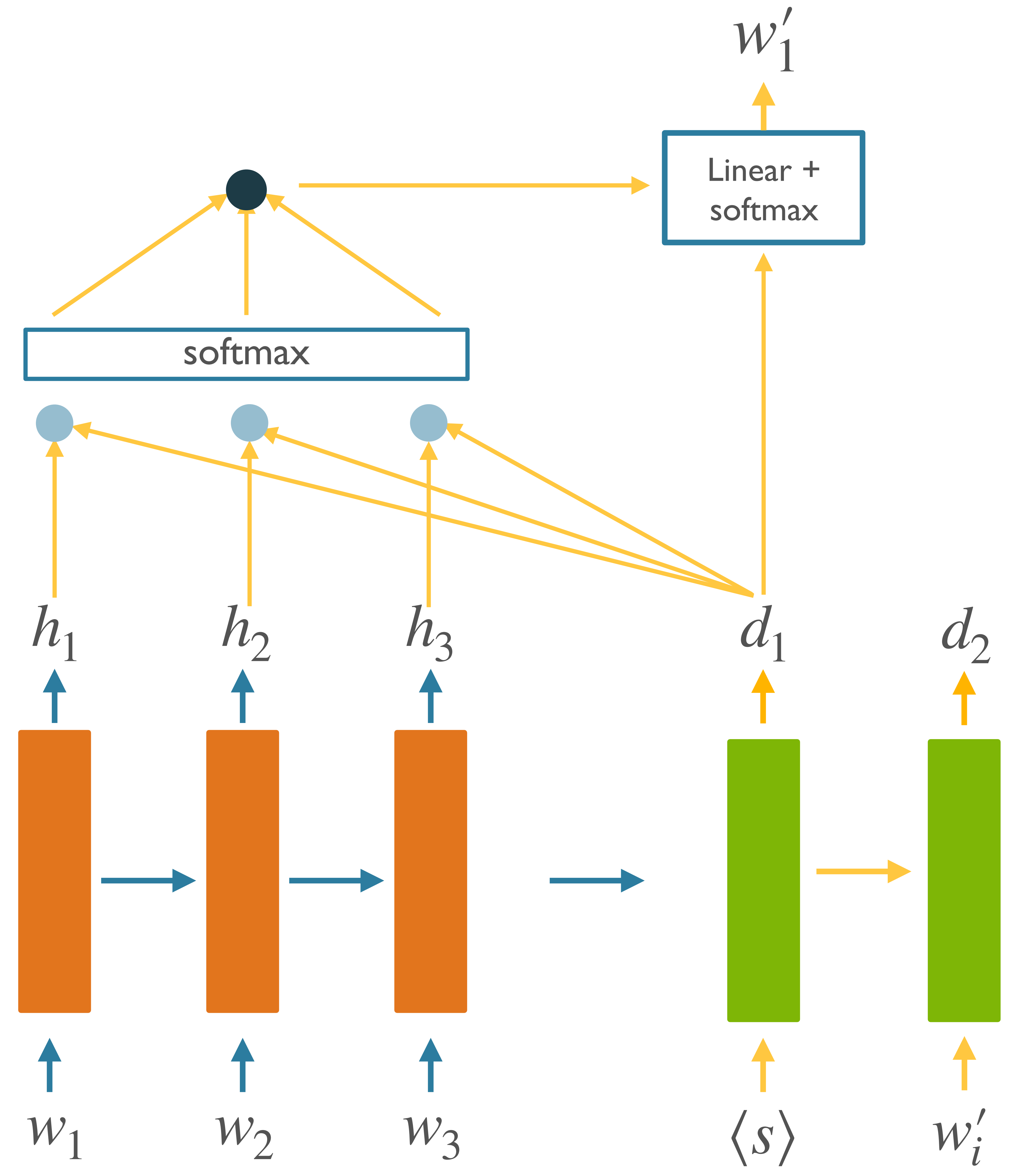
Badhanau et al 2014

Adding Attention

$c_i = \Sigma_j e_{ij} h_j$

$e_{ij} = \text{softmax}(\alpha)_j$

$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



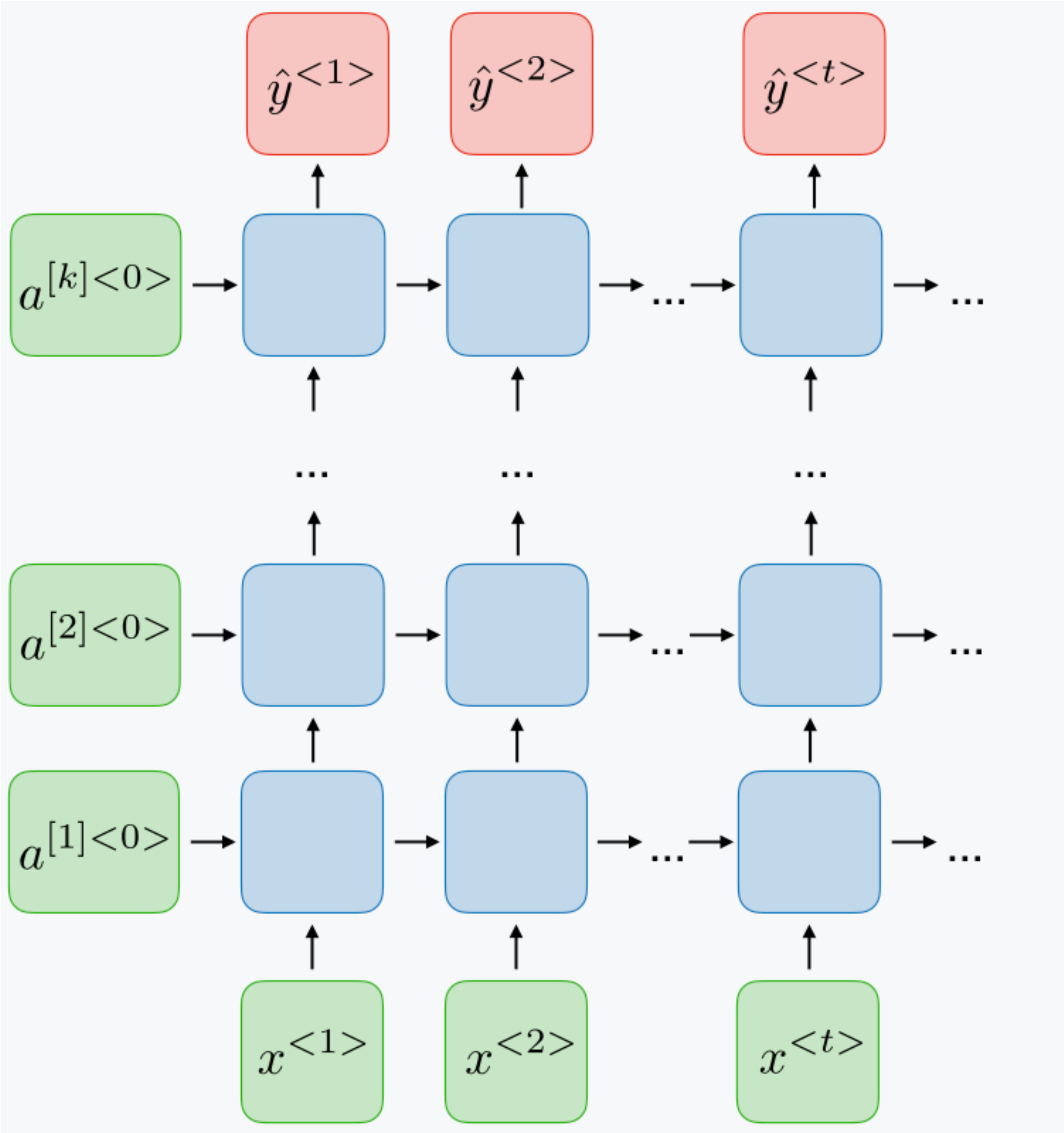
Badhanau et al 2014

Neural Networks, II

- Transformers
  - Core architecture
  - Pre-training + Fine-tuning Paradigm
- Interpretability / analysis

Lack of Parallelizability

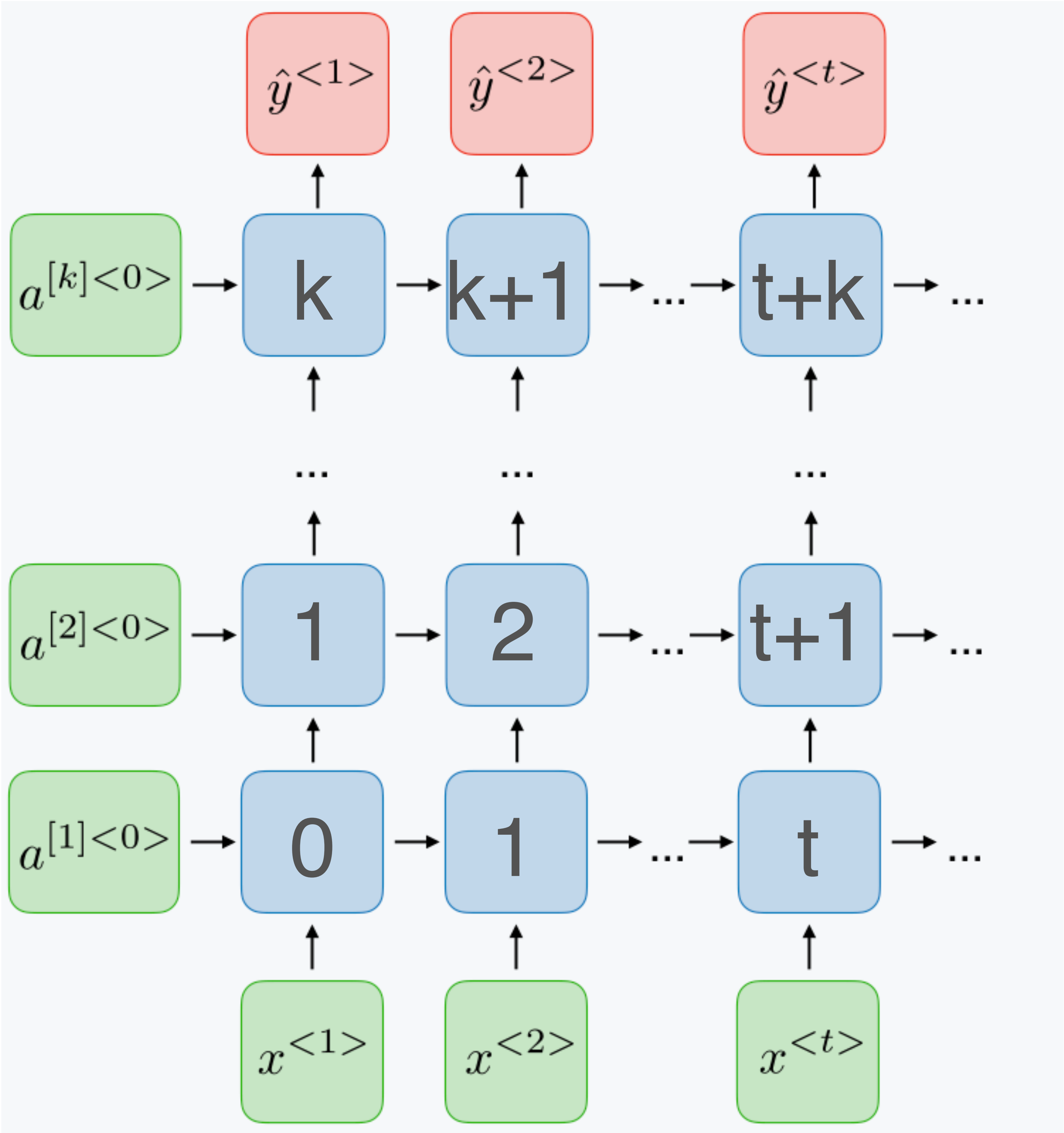
- Modern hardware (e.g. GPUs) are very good at doing *independent* computations in parallel
- RNNs are inherently serial:
  - Cannot compute future time steps without the past
- Bottleneck that makes scaling up difficult



Students who … enjoy

Lack of Parallelizability

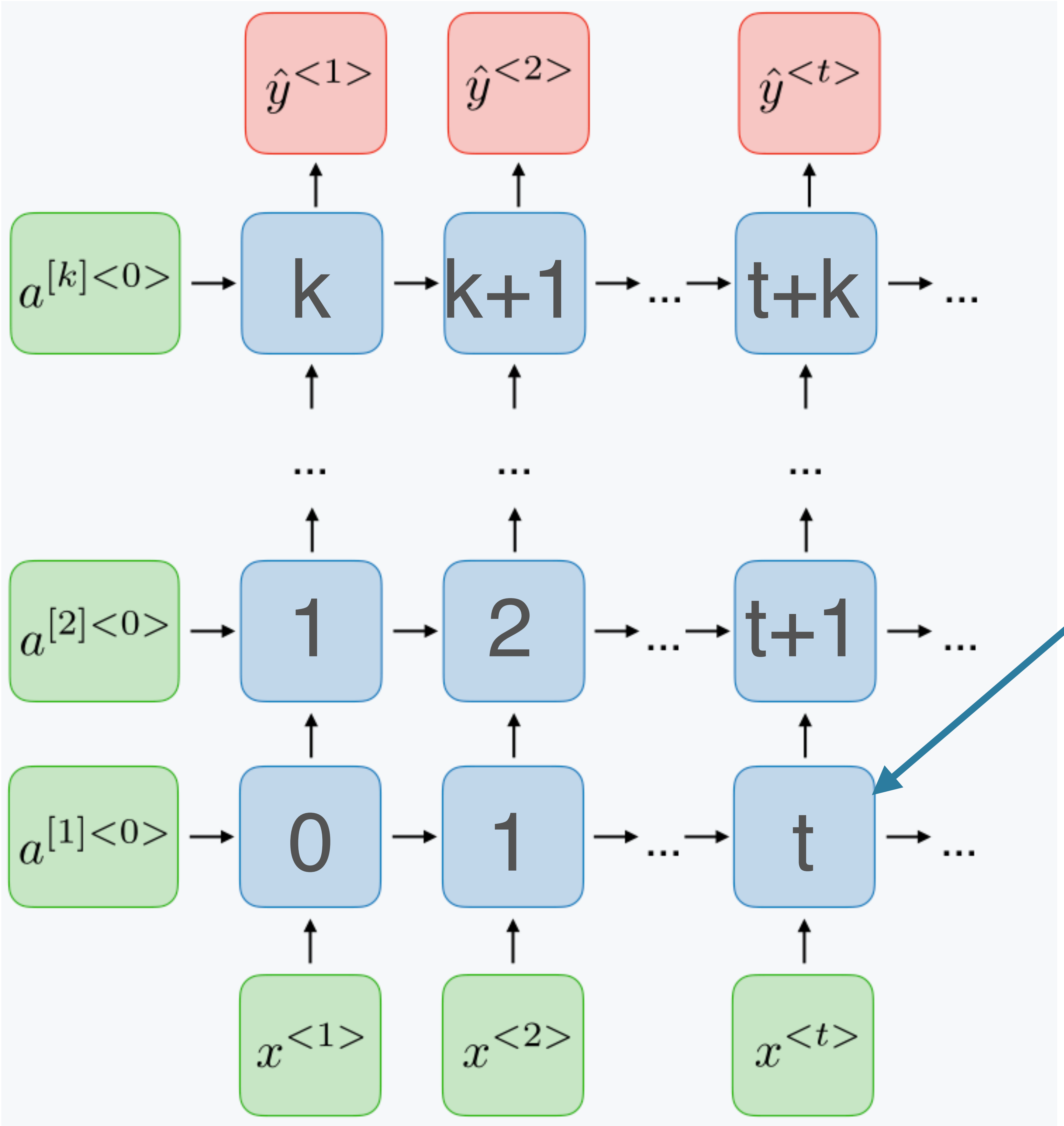
- Modern hardware (e.g. GPUs) are very good at doing *independent* computations in parallel
- RNNs are inherently serial:
  - Cannot compute future time steps without the past
- Bottleneck that makes scaling up difficult



Students who … enjoy

Lack of Parallelizability

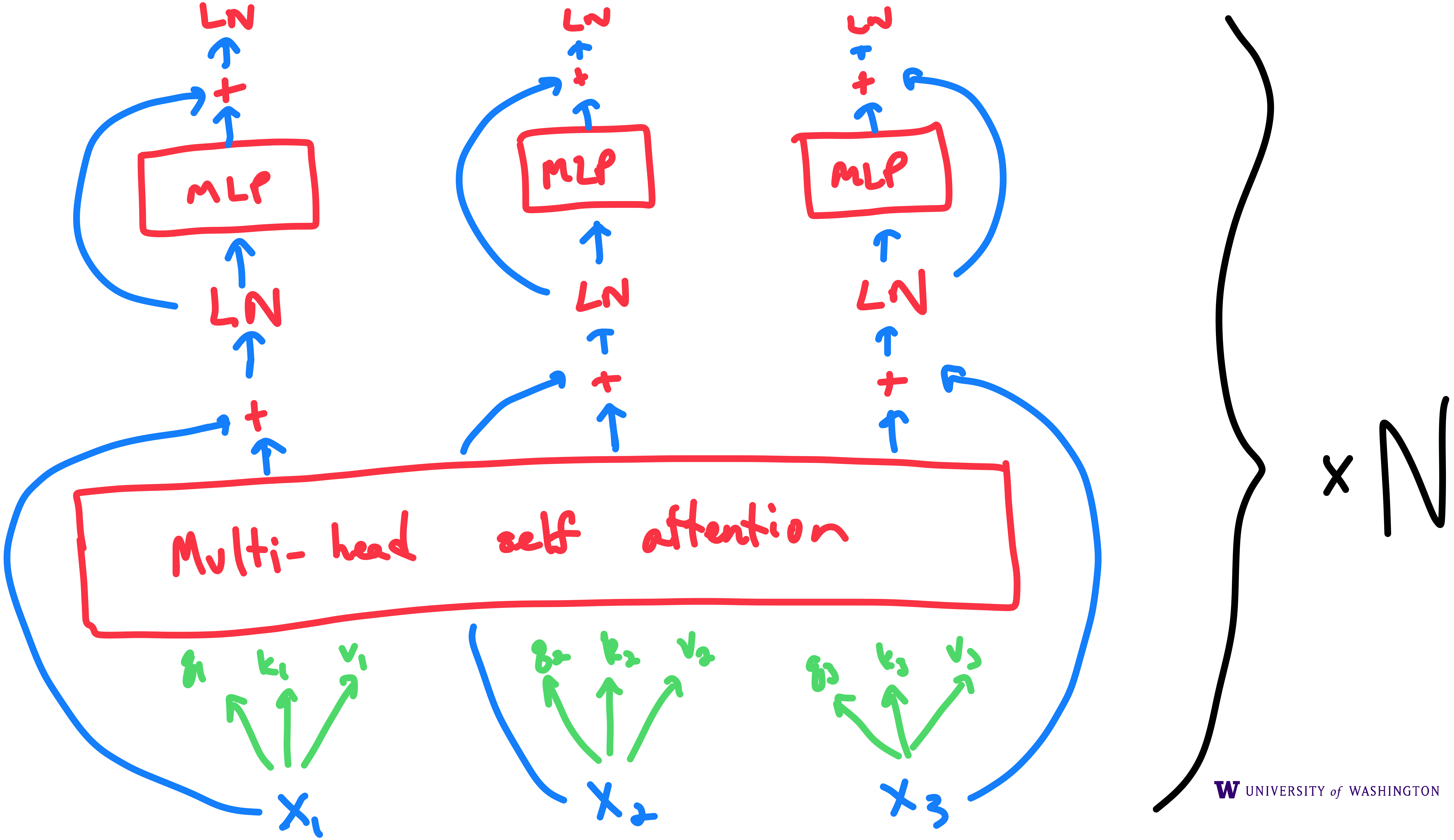
- Modern hardware (e.g. GPUs) are very good at doing *independent* computations in parallel
- RNNs are inherently serial:
  - Cannot compute future time steps without the past
- Bottleneck that makes scaling up difficult



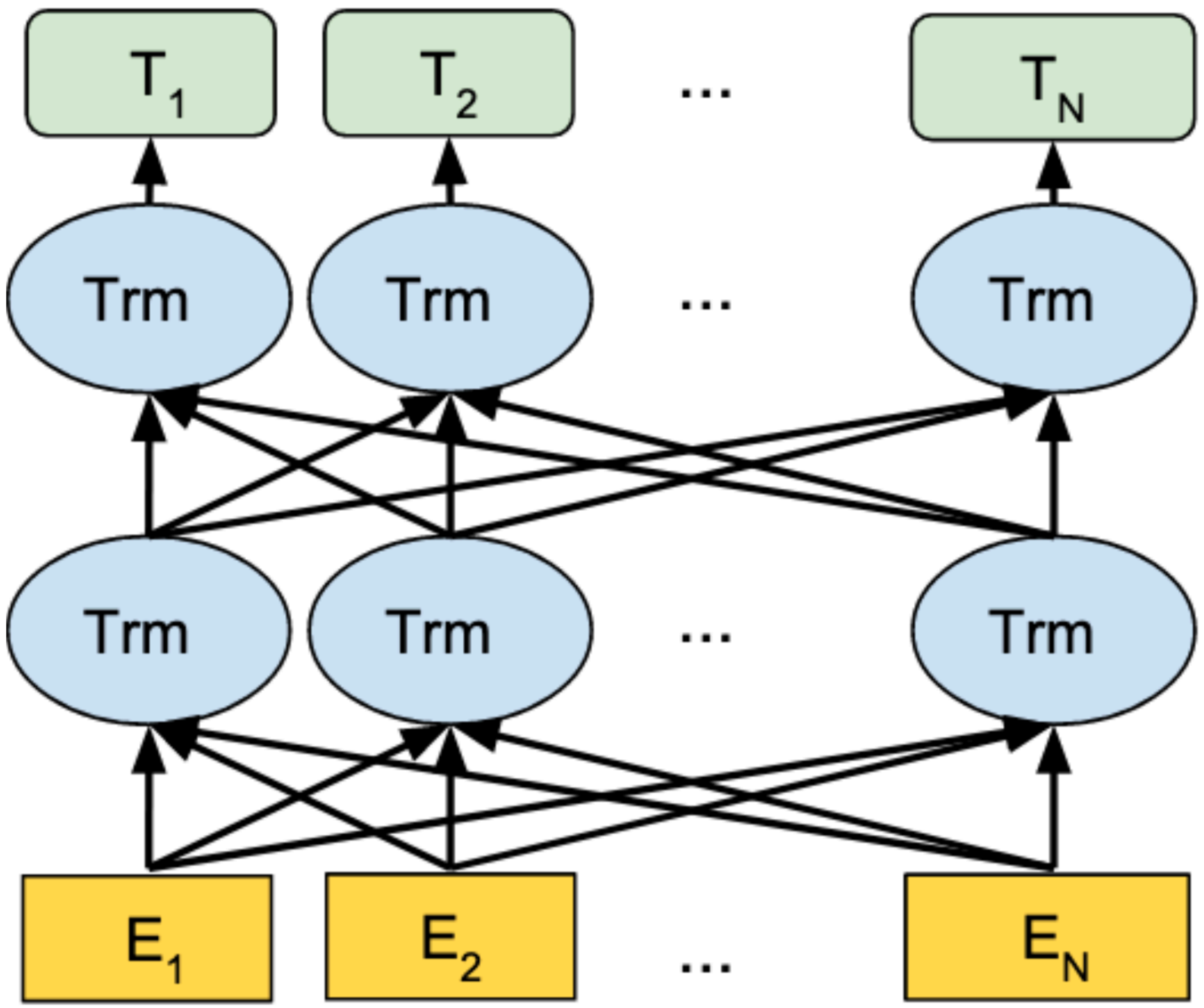
Students who … enjoy

Number of computation steps required: linear in sequence length

Full Transformer Encoder Block

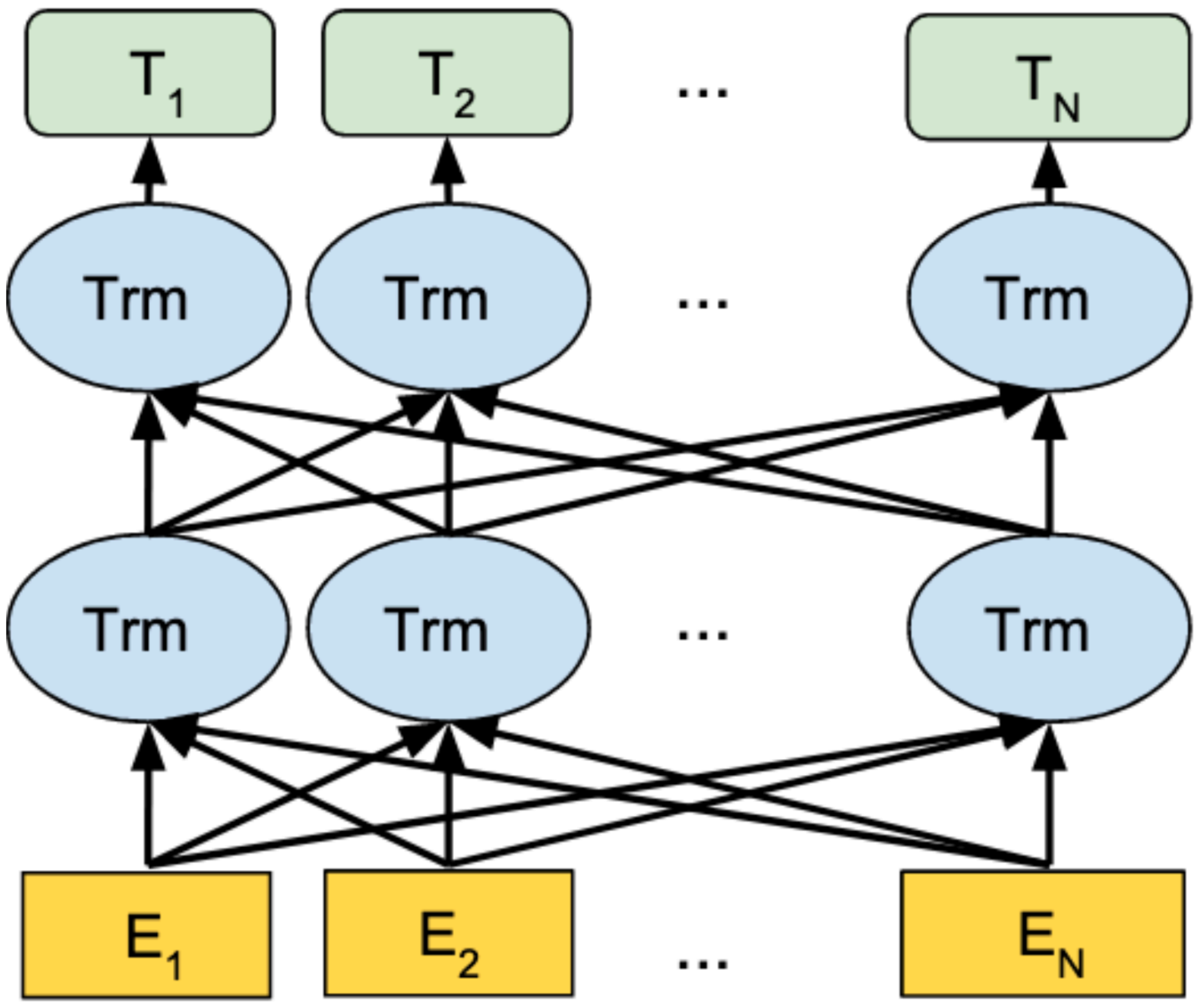


Transformer: Path Lengths + Parallelism



source (BERT paper)

Transformer: Path Lengths + Parallelism



Path lengths between tokens: 1
[constant, not linear]

source (BERT paper)

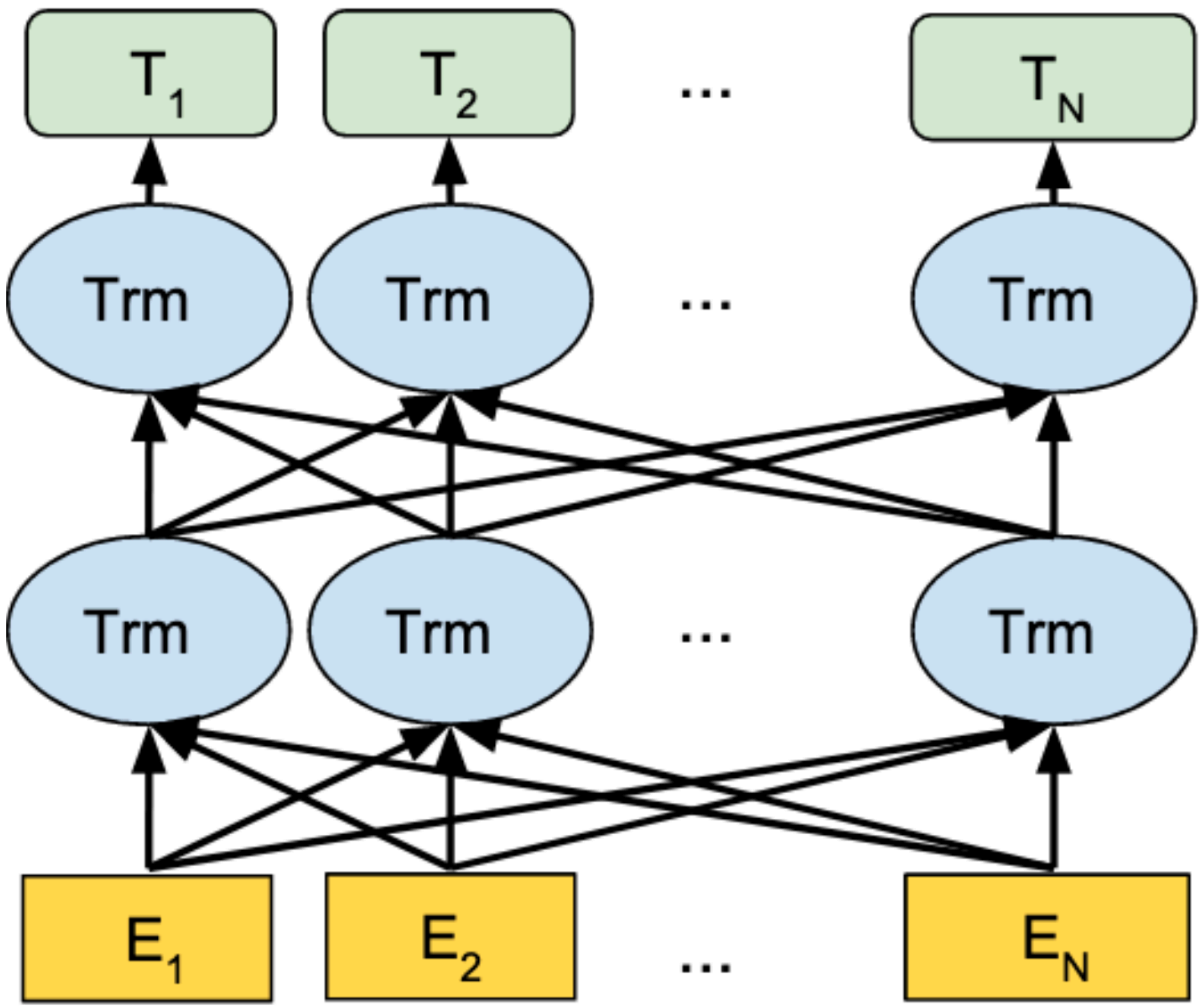
Transformer: Path Lengths + Parallelism

Computation order:

Entire second layer: 1

Entire first layer: 0

Also not linear in sequence length! Can be parallelized.



Path lengths between tokens: 1
[constant, not linear]

source (BERT paper)

Decoder: Masking Out the Future

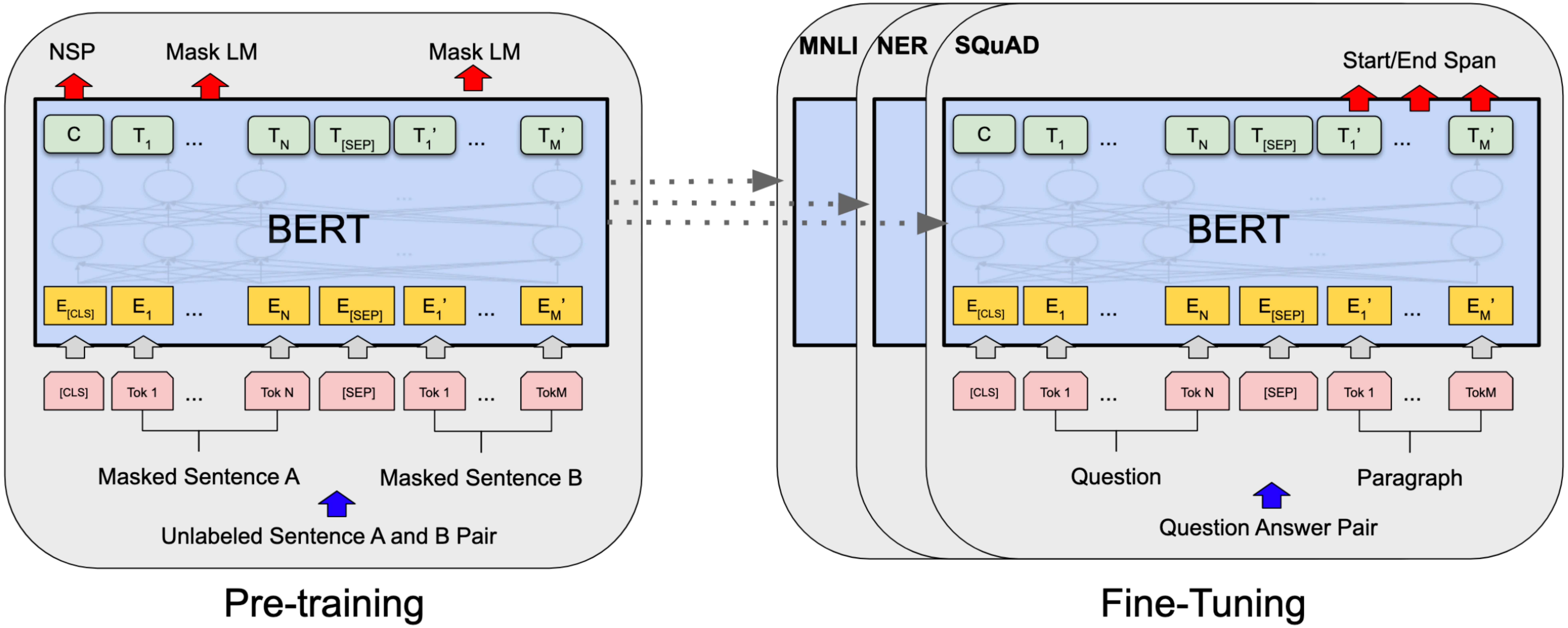
QK^T: total attention scores

$$\text{mask}_{ij} = \begin{cases} -\infty & j > i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \text{mask}\right)V$$

	<S>	Ceci	n’	est	pas	une	pipe
<S>	0	-inf	-inf	-inf	-inf	-inf	-inf
Ceci	0	0	-inf	-inf	-inf	-inf	-inf
n’	0	0	0	-inf	-inf	-inf	-inf
est	0	0	0	0	-inf	-inf	-inf
pas	0	0	0	0	0	-inf	-inf
une	0	0	0	0	0	0	-inf
pipe	0	0	0	0	0	0	0

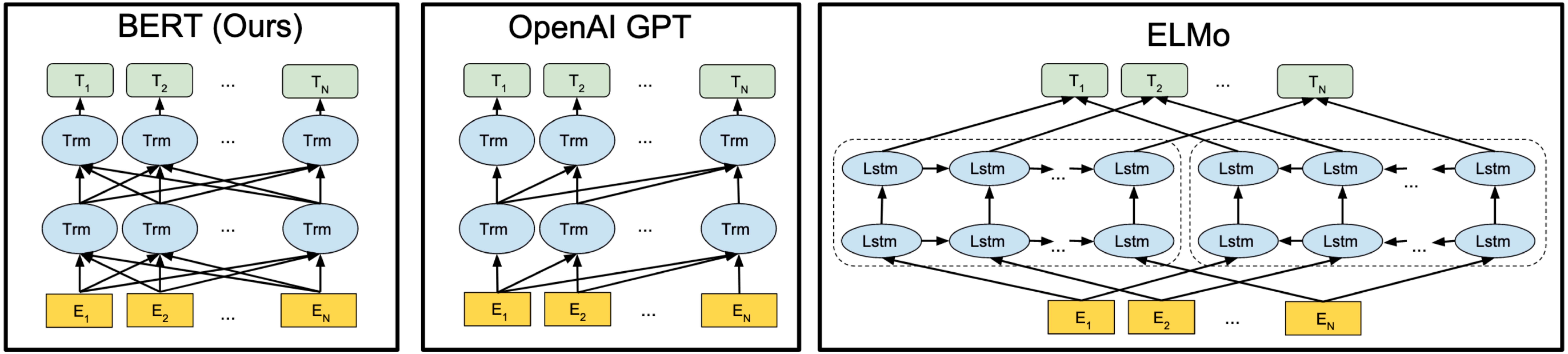
Schematically



Initial Results

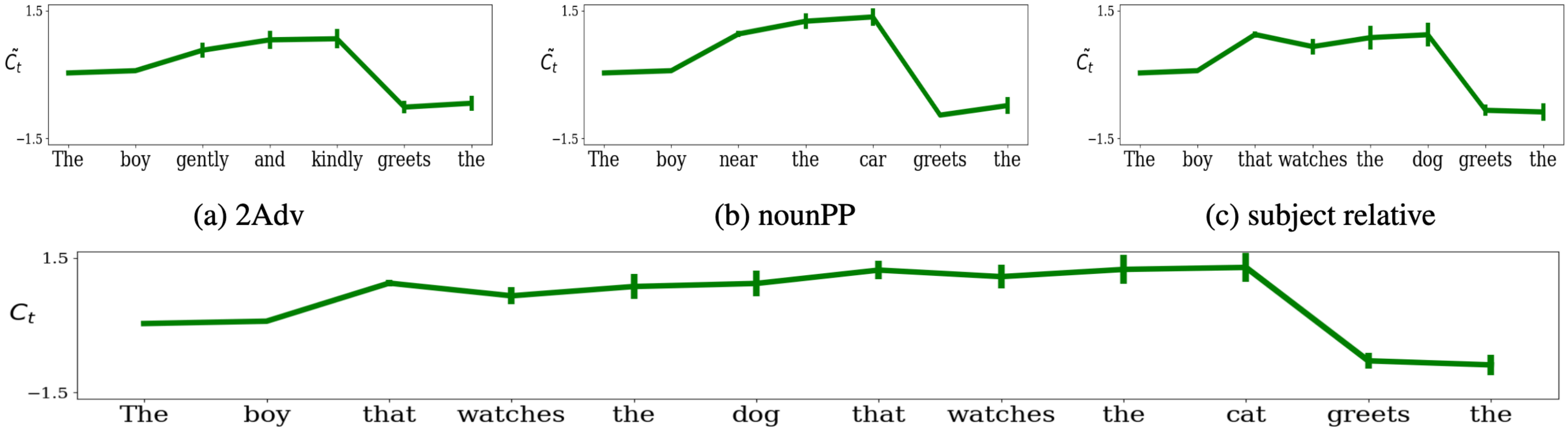
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Comparison



Source: BERT paper

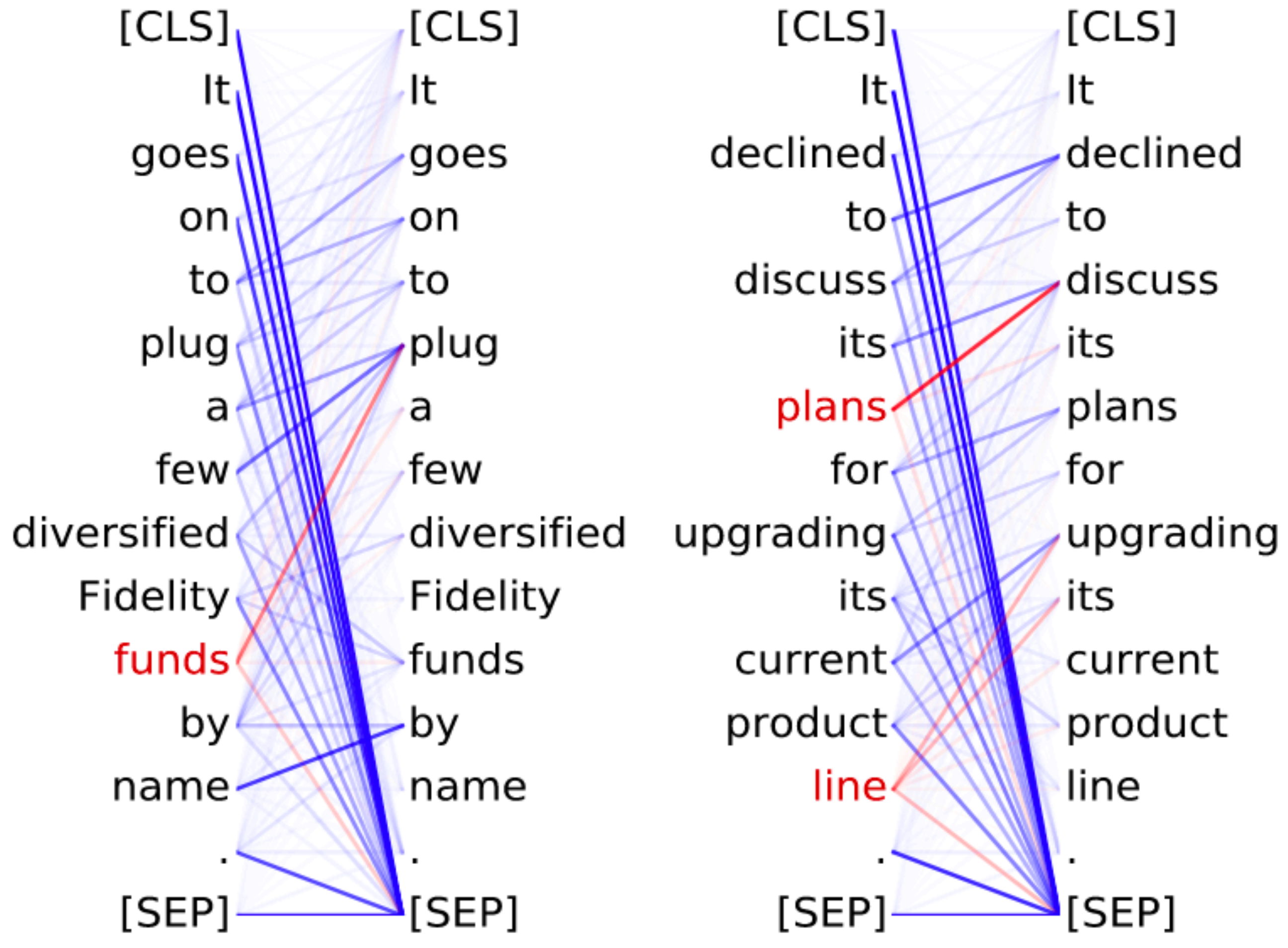
Cell dynamics for a syntax unit



Examples

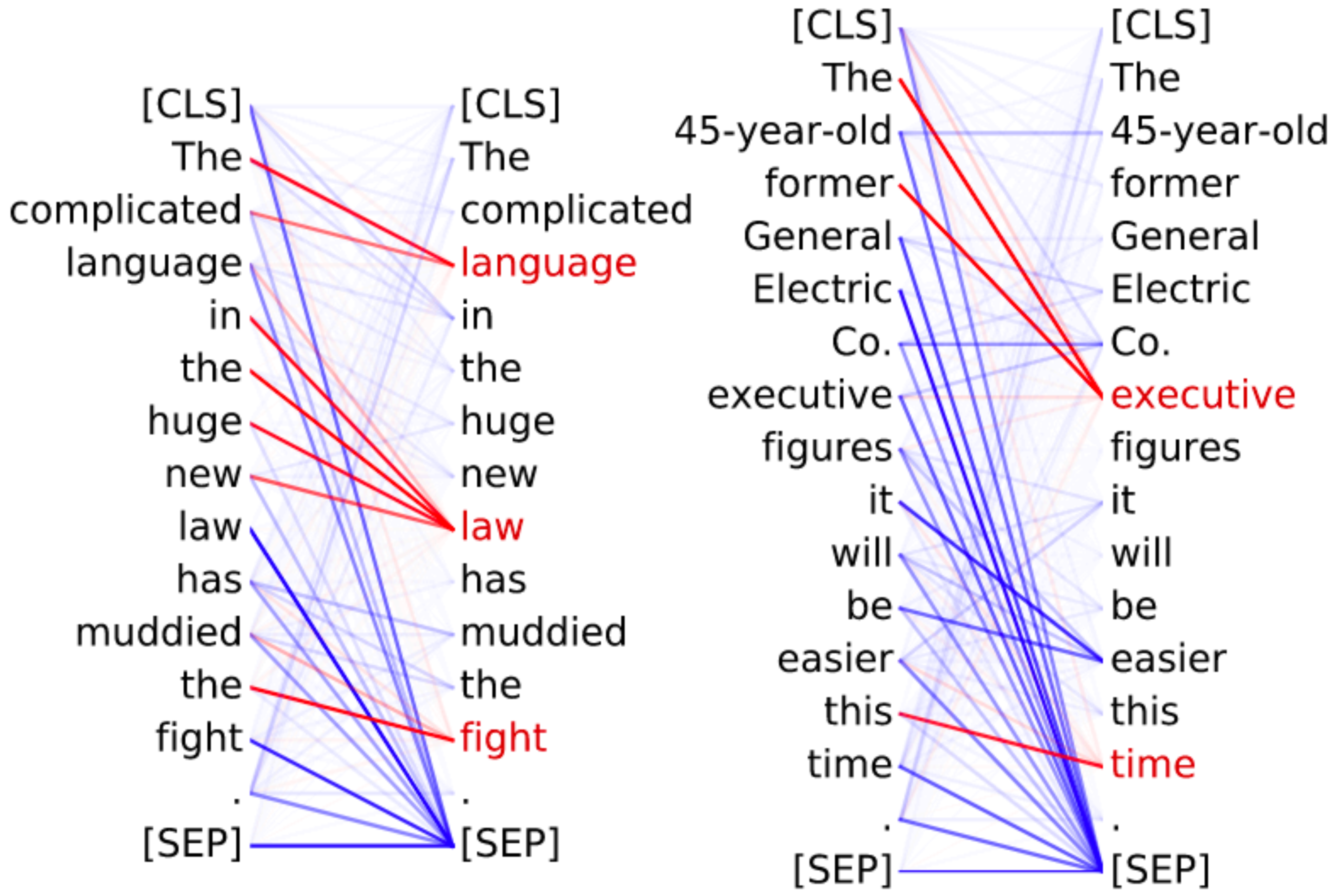
Head 8-10

- **Direct objects** attend to their verbs
- 86.8% accuracy at the `dobj` relation



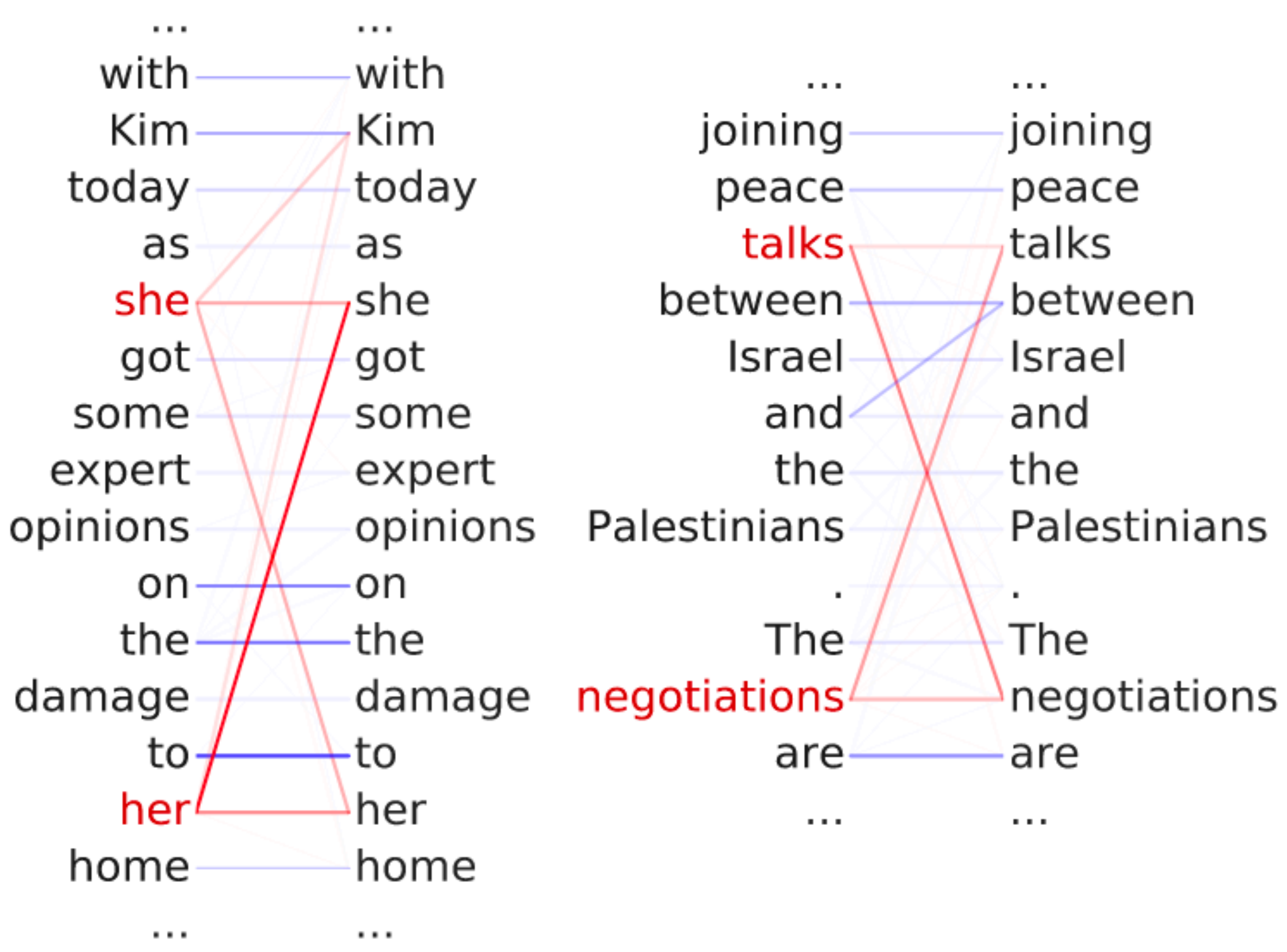
Head 8-11

- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the `det` relation

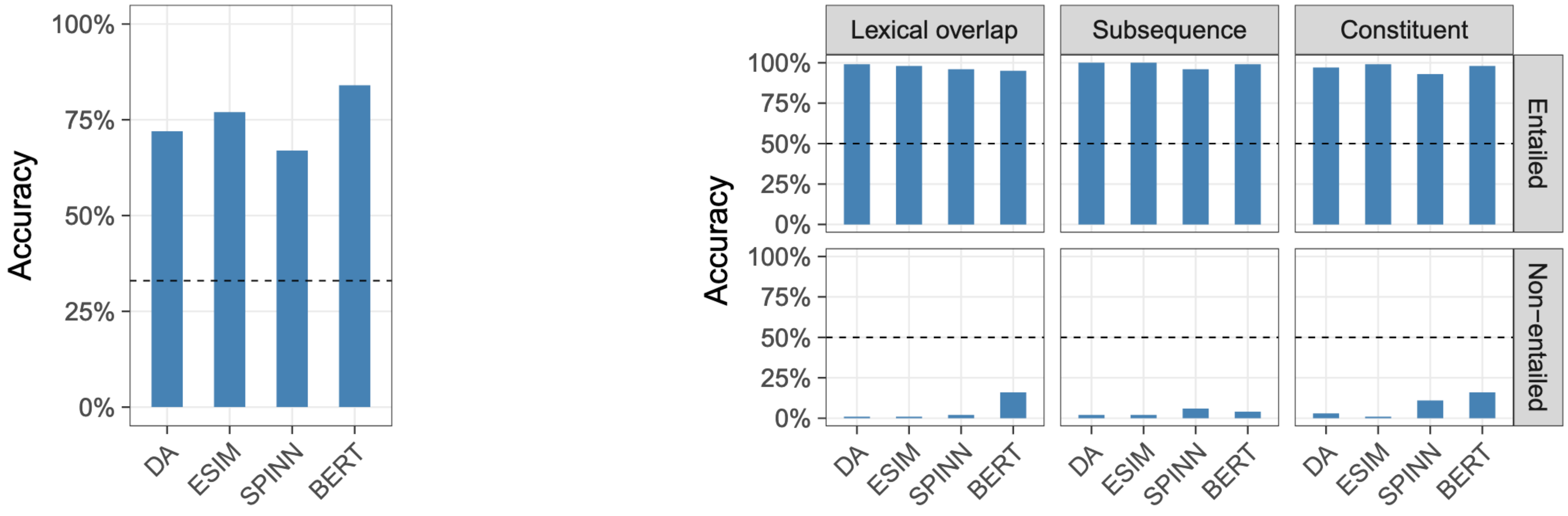


Head 5-4

- **Coreferent** mentions attend to their antecedents
- 65.1% accuracy at linking the head of a coreferent mention to the head of an antecedent



Results



(a) (b)

(performance improves if fine-tuned on this challenge set)

Guest lectures

- C.M. Downey: Multilingual NLP
- Jack Hessel: Multimodality
- Angelina McMillan-Major: Documenting stochastic parrots

Assignments

- 1: Vocabulary + Data Statement
- 2: Word2Vec (raw numpy)
- 3: Computation graphs (word2vec in edugrad)
- 4: Deep Averaging Network classifier (edugrad)
- 5: Feed-forward language model (edugrad)
- 6: RNN text classifier
- 7: RNN language model
- 8: Seq2Seq + Attention [translation]
- 9: Pre-trained transformer classifier

What’s Next?

Learning Outcomes

- One way of operationalizing the goal: you can hopefully now read many/ most new papers at NLP conferences and understand what they're doing
  - Expressions like "we pre-trained a bi-directional LSTM language model on various tasks and then fine-tuned on a standard suite" are now parseable
  - And with deeper / more hands-on familiarity with the models and their architectures, you are in a position to assess new developments as they come (and contribute to them as well!)

Topics Not Covered

- Full suite of “tips and tricks” for training
  - e.g. learning rate schedules
  - Best methods for hyper parameter tuning
- Other architectures sometimes used: convolutional networks, tree-based RNNs
- Wide variety of NLP tasks: parsing, QA, toxic language detection, etc.
- NB: you are now well-positioned to read and learn about all of these on your own

Where to Learn More

- Where to learn more?
  - Read papers and chase references when confused
  - Cornell's course has lots of online materials: http://www.phontron.com/class/nn4nlp2021/
  - Stanford CS224U (pre-recorded videos) http://web.stanford.edu/class/cs224u/
    - And CS224N (live lectures) http://web.stanford.edu/class/cs224n/
- NLP Newsletter: https://newsletter.ruder.io/
- ACL Anthology: https://www.aclweb.org/anthology/ [more and more videos too]
- Semantic Scholar / arXiv sanity similar paper searches

General Question Time

Wrapping Up

Course Evaluations

- Course evals are open now through June 3
- Please do fill them out as soon as possible!
  - E.g. right now :)
  - Help me:
    - Improve the course for future iterations
    - Get tenure ;)

Thank You!

- I’ve learned a lot from you all this quarter!
- Hopefully you’re in a better place with regard to neural methods in NLP than when the course started.
- And congrats to everyone for handling such a workload amidst all of the challenges of the past year-plus. Very awe-inspiring.
- So: thank you, and have a great summer / future!