Summary / Review

LING 574 Deep Learning for NLP

Shane Steinert-Threlkeld

Announcements

- Course evaluations open **only until Friday June 6** (more later)
  - https://uw.iasystem.org/survey/307454
- If you haven't used or want to retroactively use your free late assignment, let us know ASAP!
- HW9: prompting and in-context learning. Due **June 9**.
  - Mainly: all the choices / scaffolding for actually using ICL for a task/dataset (SST) + base vs instruction-tuned model
  - OLMo models: stored in our dropbox folder, so that there's one copy instead of each downloading (see `args.hf_home` stuff in `olmo.py`)
  - No unit tests, because there's no "right answer" to the coding portion: room for creativity!
  - Other hyper-parameters we haven't mentioned: various generation ones (sampling, top_k, top_p, etc), how to choose the in-context examples (`get_k_examples`), even more prompting strategies, …. Feel free to play around more on your own!

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

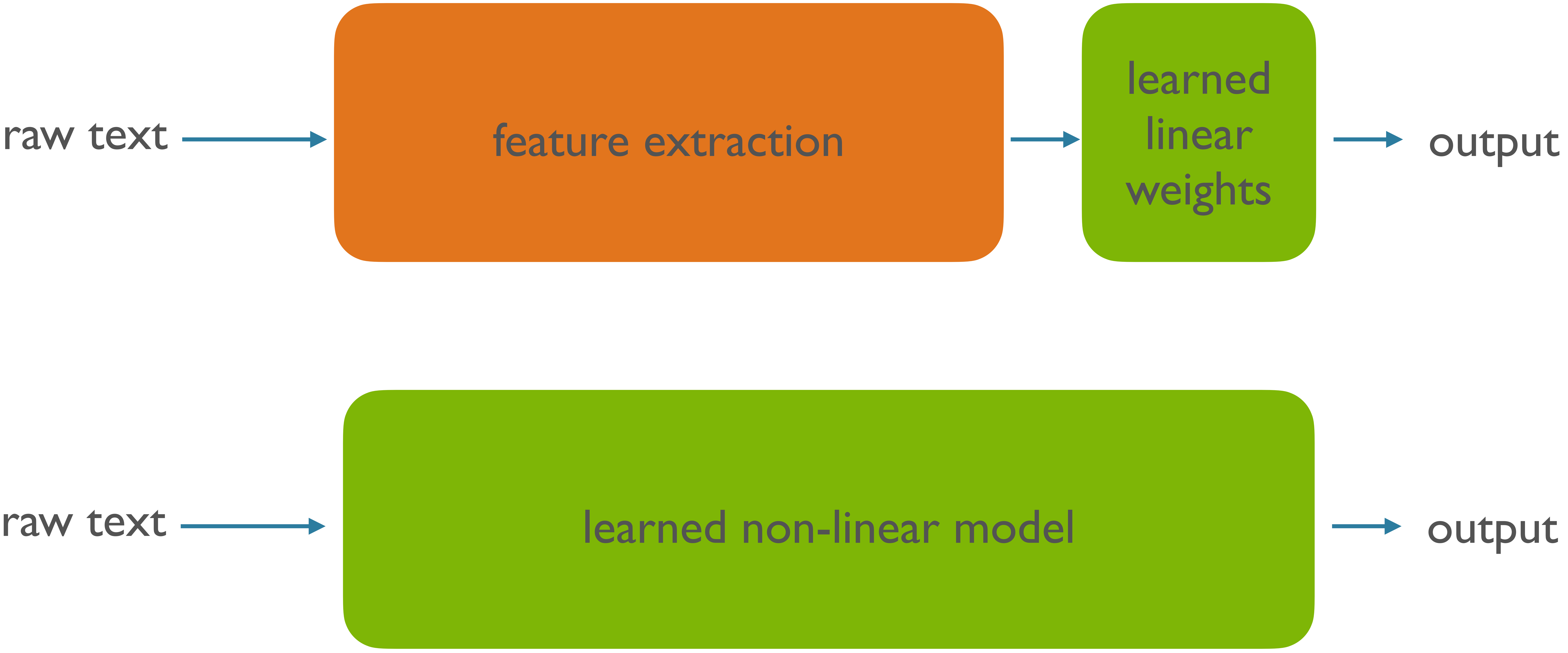
- Provide hands-on experience with building neural networks and using them for NLP tasks
- Theoretical understanding of building blocks
  - Computation graphs + gradient descent
  - Forward/backward API
    - Chain rule for computing gradients [backpropagation]
  - Various network architectures; their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

$$P(1 \mid w, c) = \sigma\left(\underbrace{E_w}_{\text{Target word embedding}} \cdot \underbrace{C_c}_{\text{Context word embedding}}\right)$$

The SGNS Model

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

Target word embedding

Context word embedding

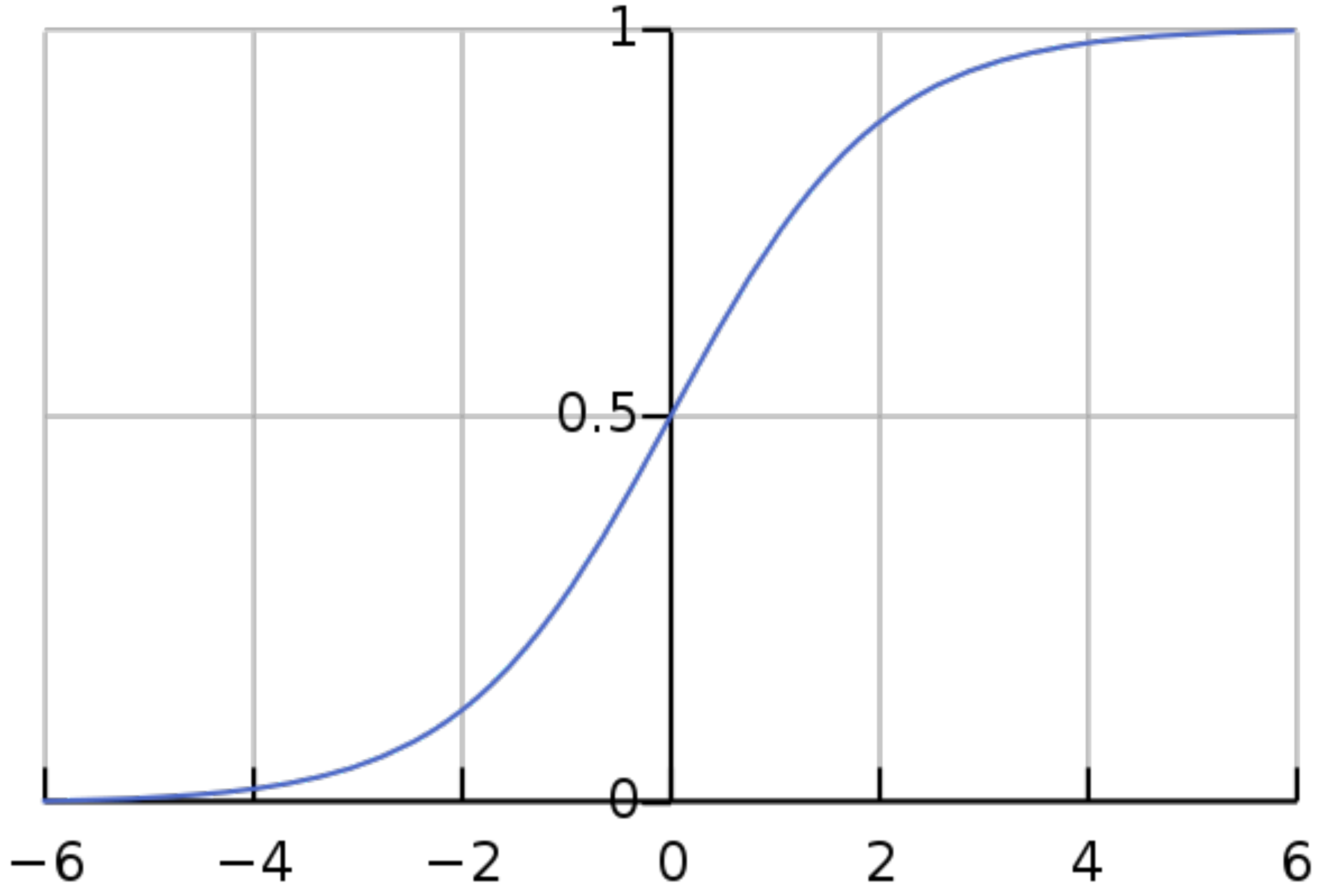
Similarity (dot-product)

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

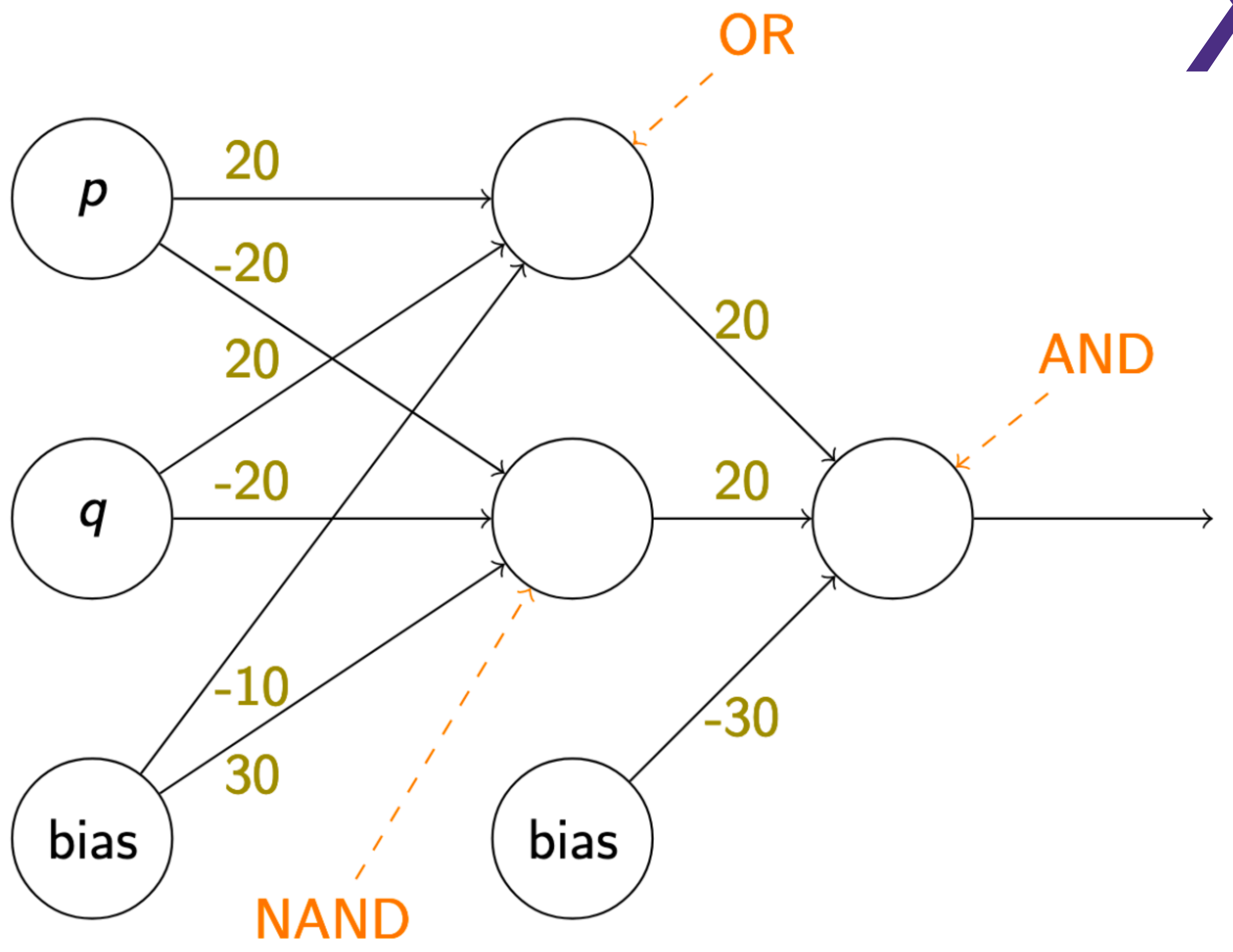
Context word embedding

Similarity (dot-product)

Neural Networks: Foundations

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

XOR Network

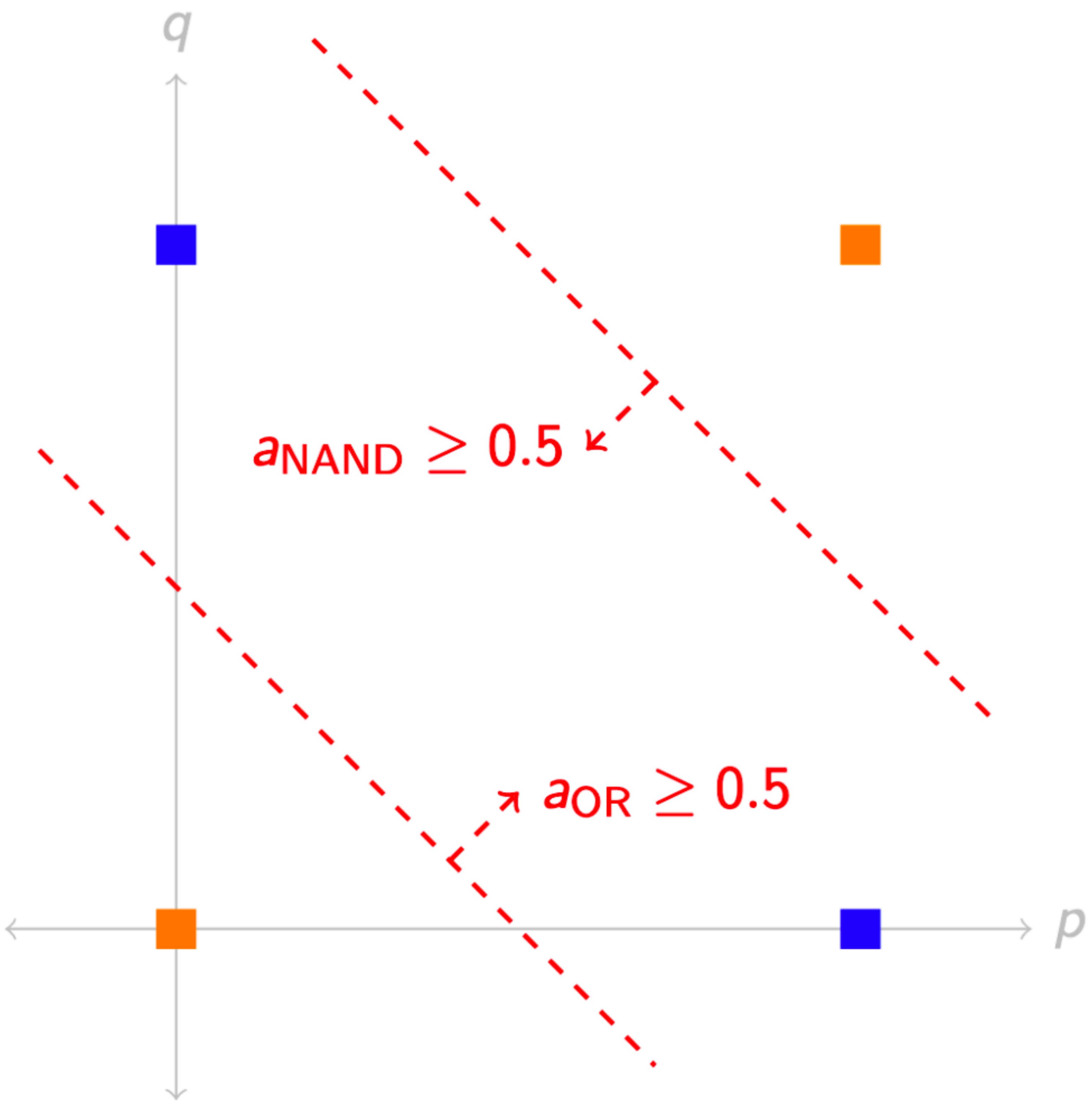


$$a_{\text{and}} = \sigma\left(w^{\text{and}}_{\text{or}} \cdot a_{\text{or}} + w^{\text{and}}_{\text{nand}} \cdot a_{\text{nand}} + b^{\text{and}}\right)$$

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

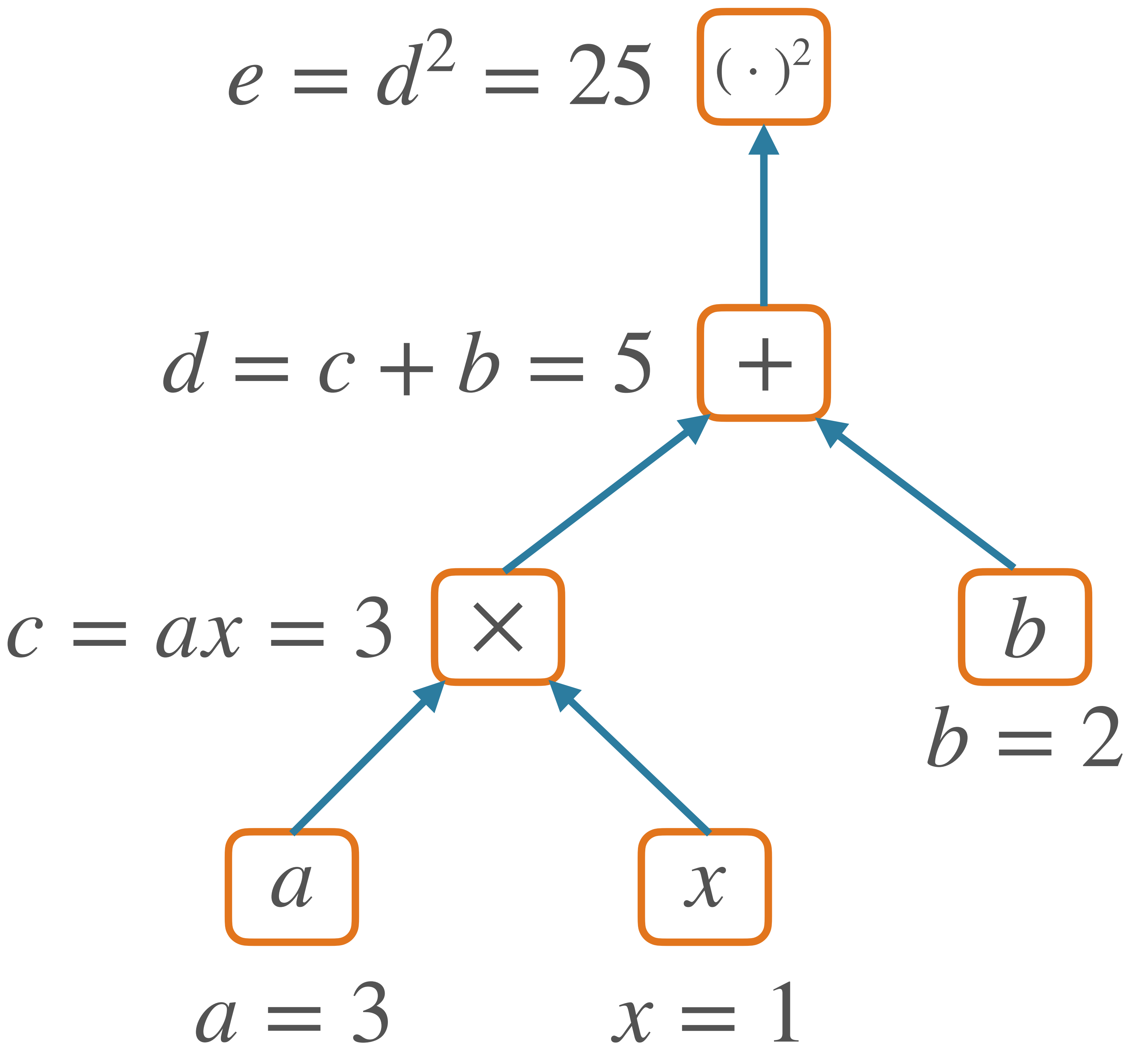
$$a_{\text{and}} = \sigma\left(\sigma\left(\begin{bmatrix} a_p & a_q \end{bmatrix} \begin{bmatrix} w^{\text{or}}_p & w^{\text{nand}}_p \\ w^{\text{or}}_q & w^{\text{nand}}_q \end{bmatrix} + \begin{bmatrix} b^{\text{or}} & b^{\text{nand}} \end{bmatrix}\right) \begin{bmatrix} w^{\text{and}}_{\text{or}}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

Computing XOR (not linearly-separable)



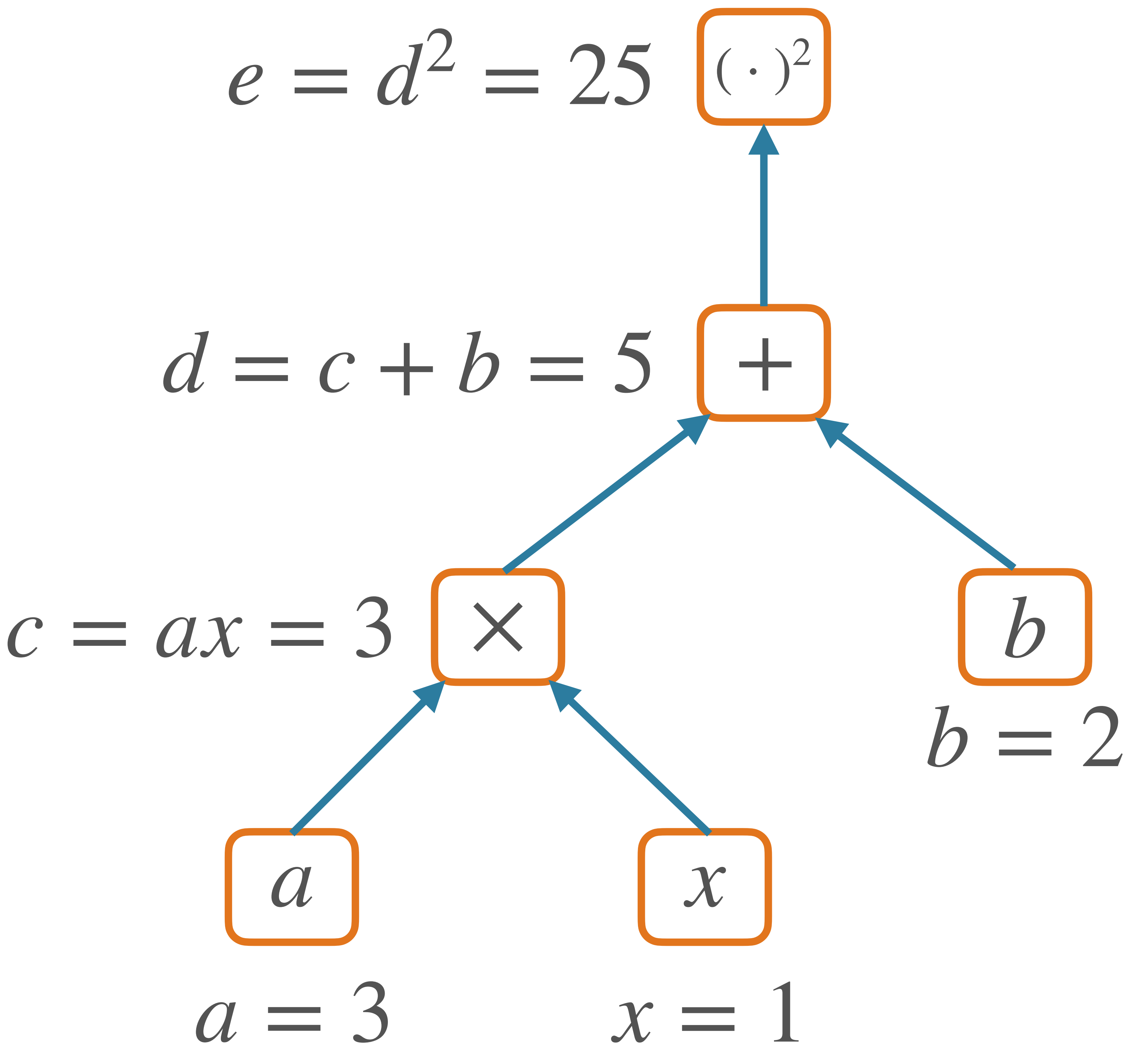
Backpropagation Example

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



Backpropagation Example

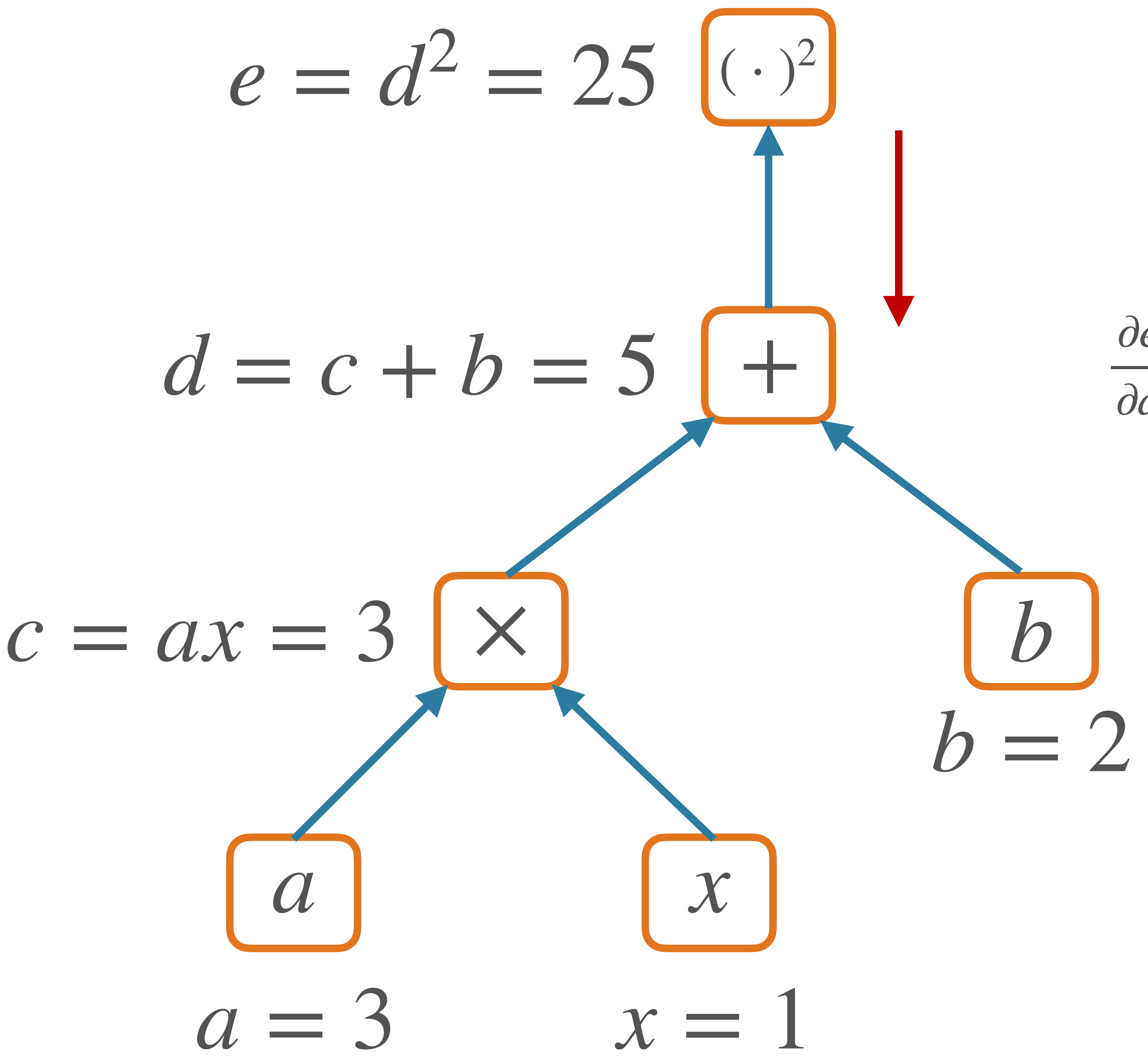
$$f(x; a, b) = (ax + b)^2$$



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

Backpropagation Example

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

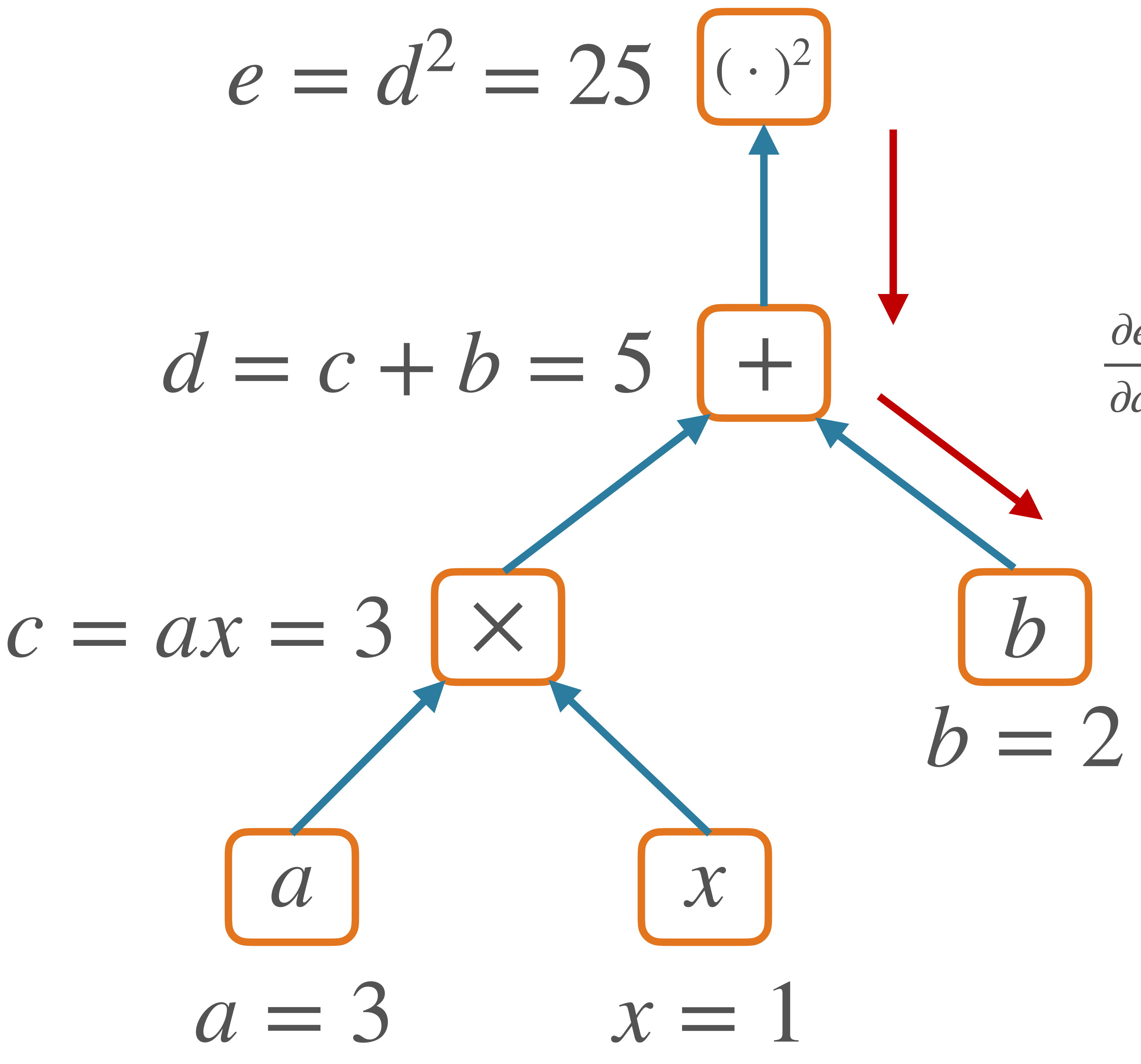


$$\frac{\partial e}{\partial e} = 1$$

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

Backpropagation Example

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



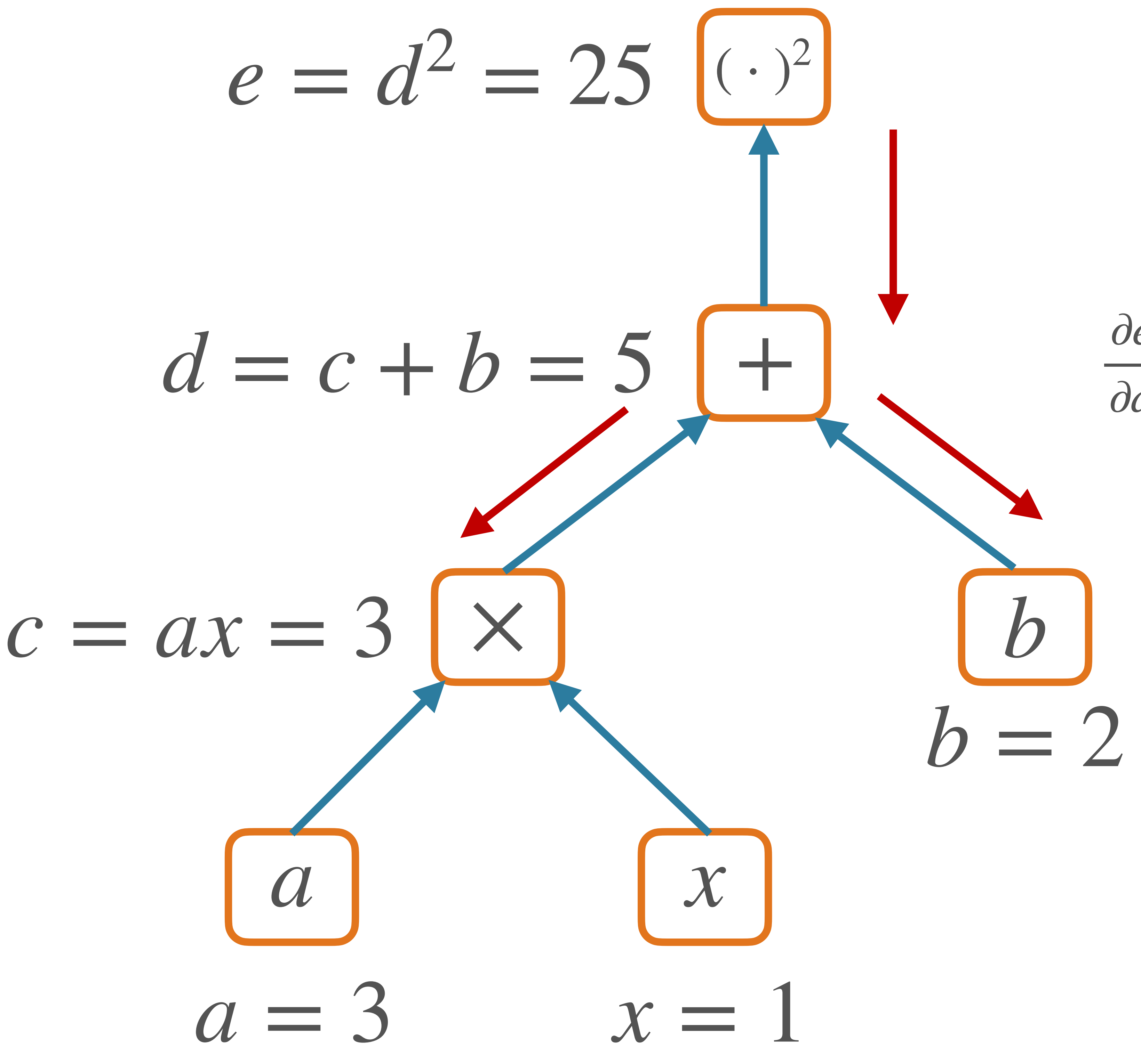
$$\frac{\partial e}{\partial e} = 1$$

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

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

Backpropagation Example

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



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

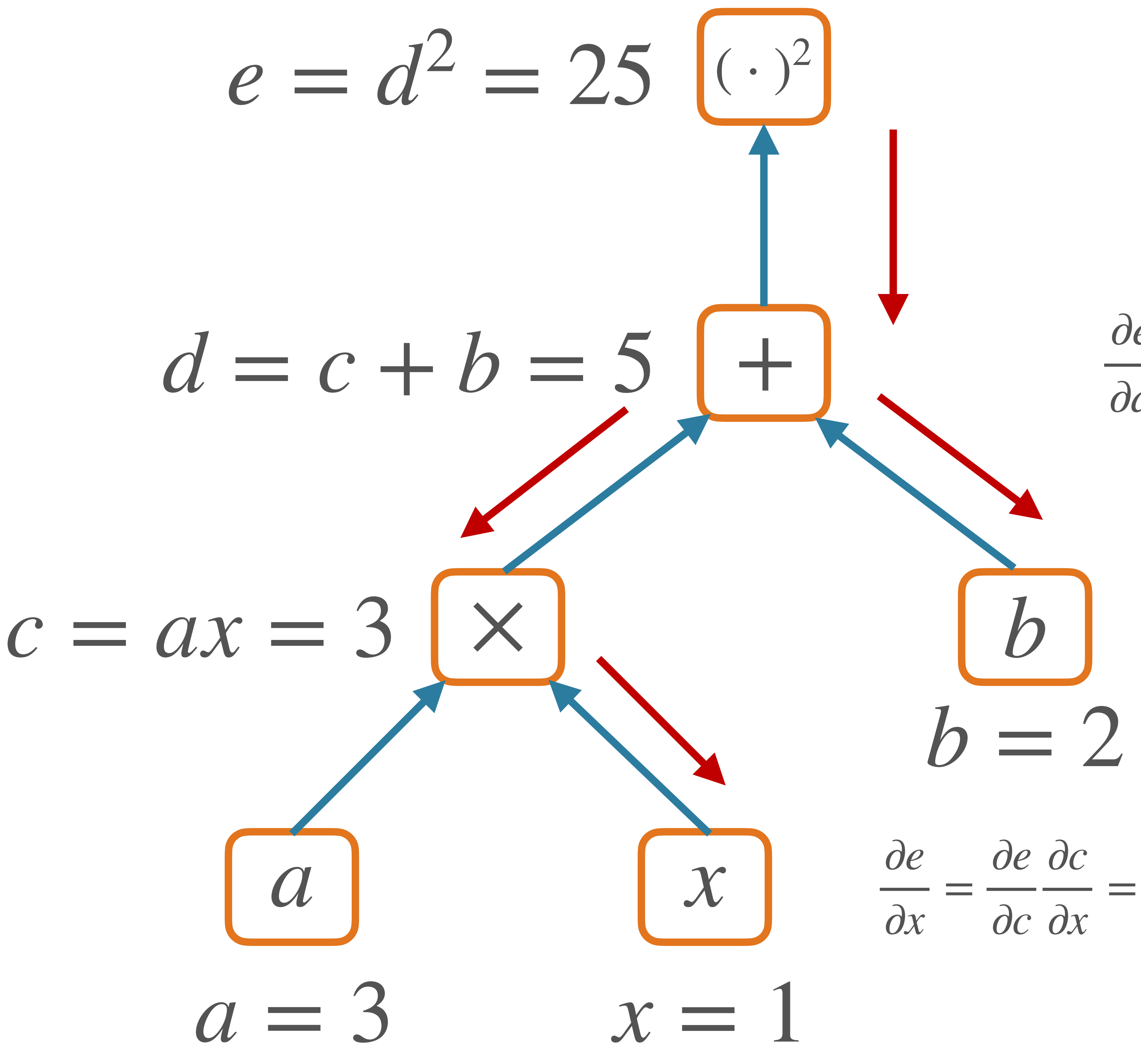
$$\frac{\partial e}{\partial d} = 2d\frac{\partial e}{\partial e} = 10$$

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

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

Backpropagation Example

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



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

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

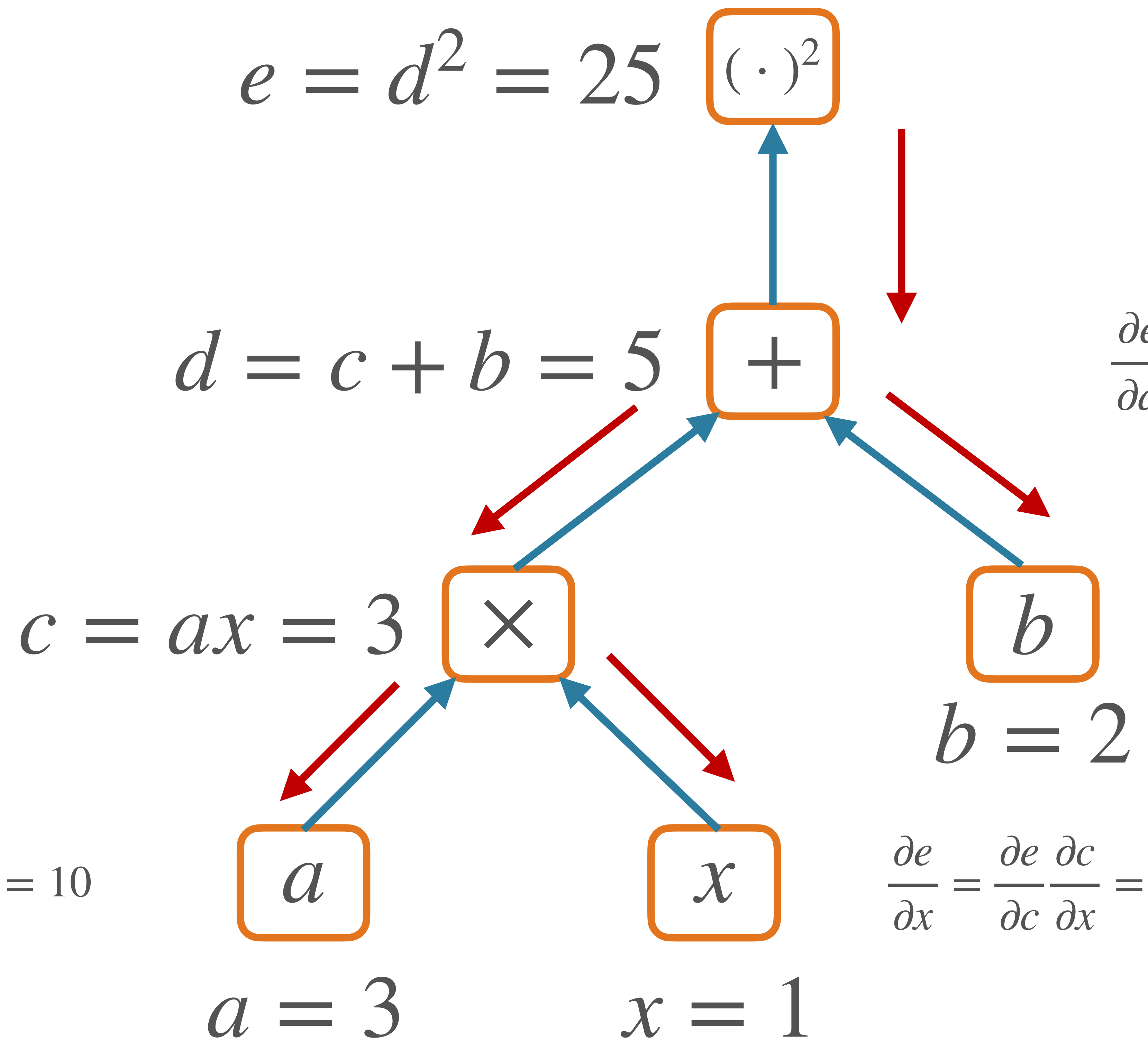
$$\frac{\partial e}{\partial c} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial c} = 10\frac{\partial c + b}{\partial c} = 10$$

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

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

Backpropagation Example

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



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

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

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

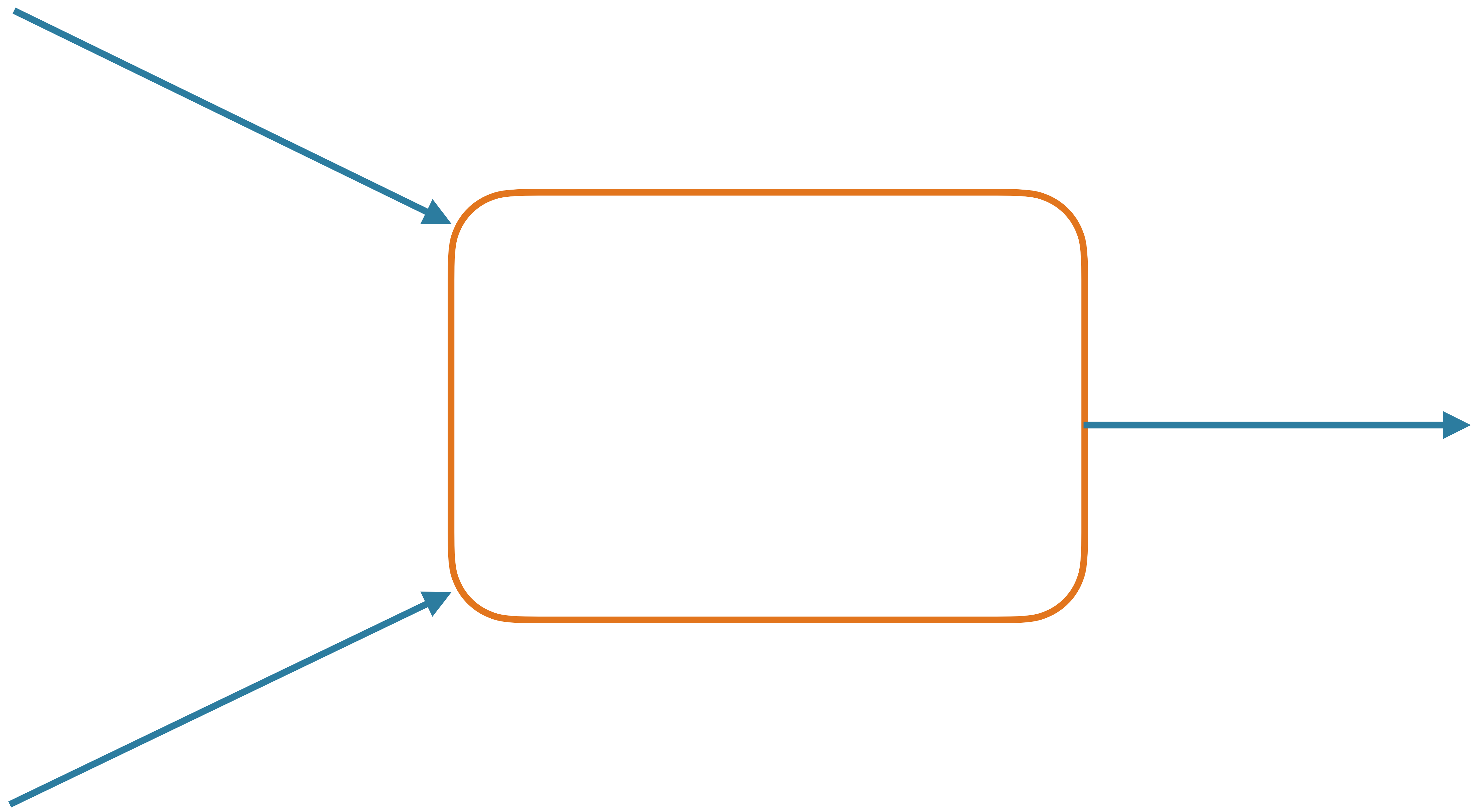
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 10\frac{\partial c + b}{\partial b} = 10$$

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

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

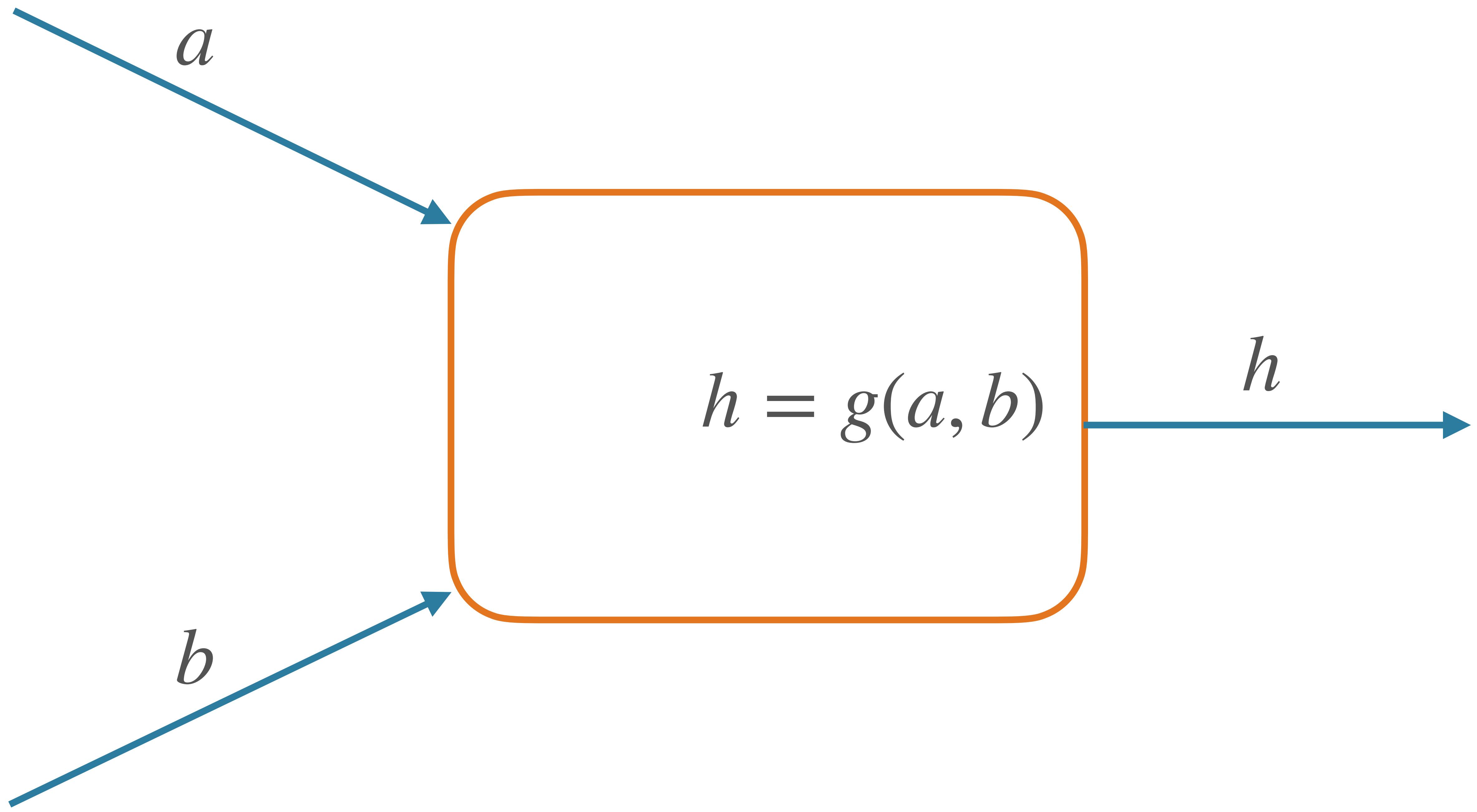
Nodes in Computational Graph

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



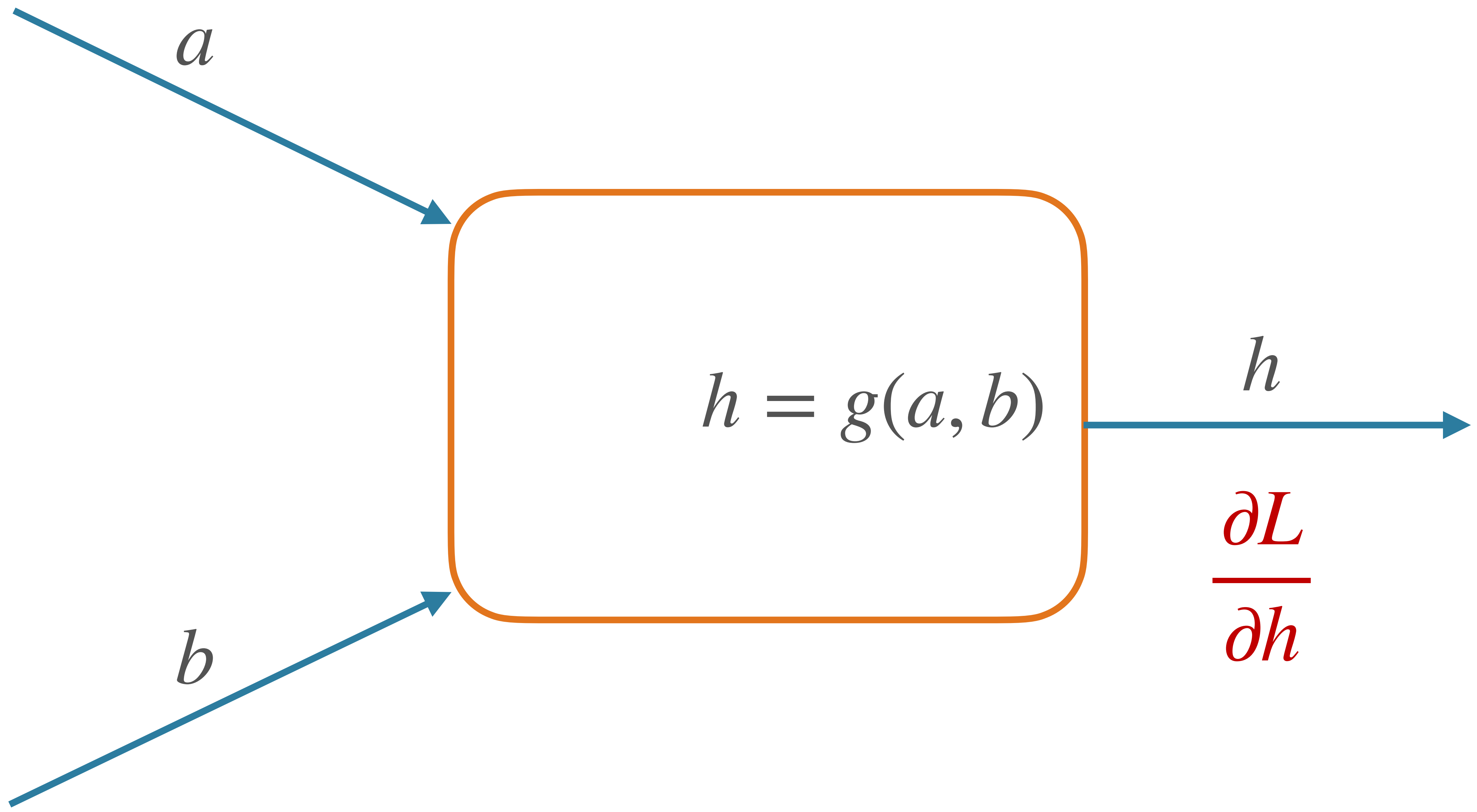
Nodes in Computational Graph

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



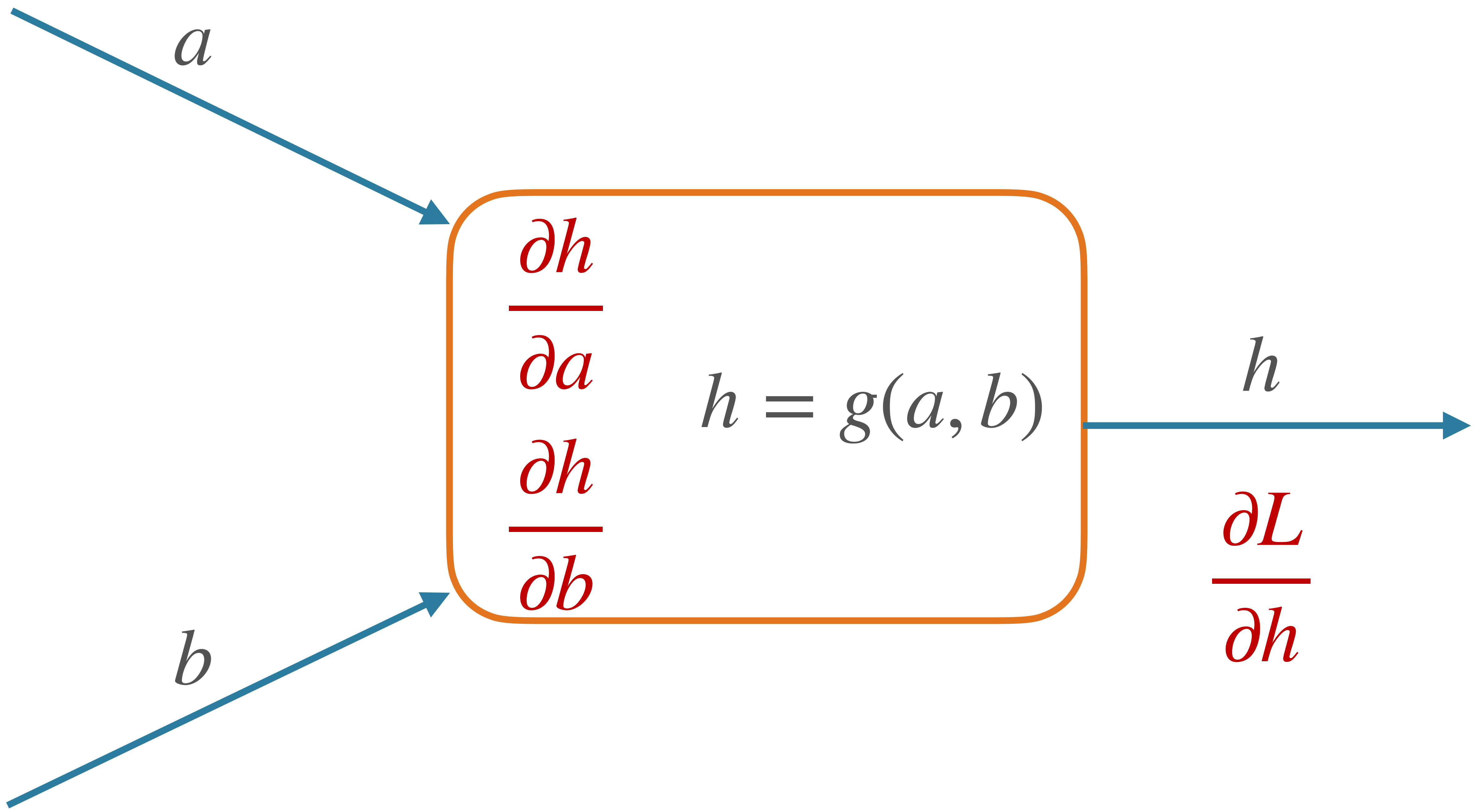
Nodes in Computational Graph

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



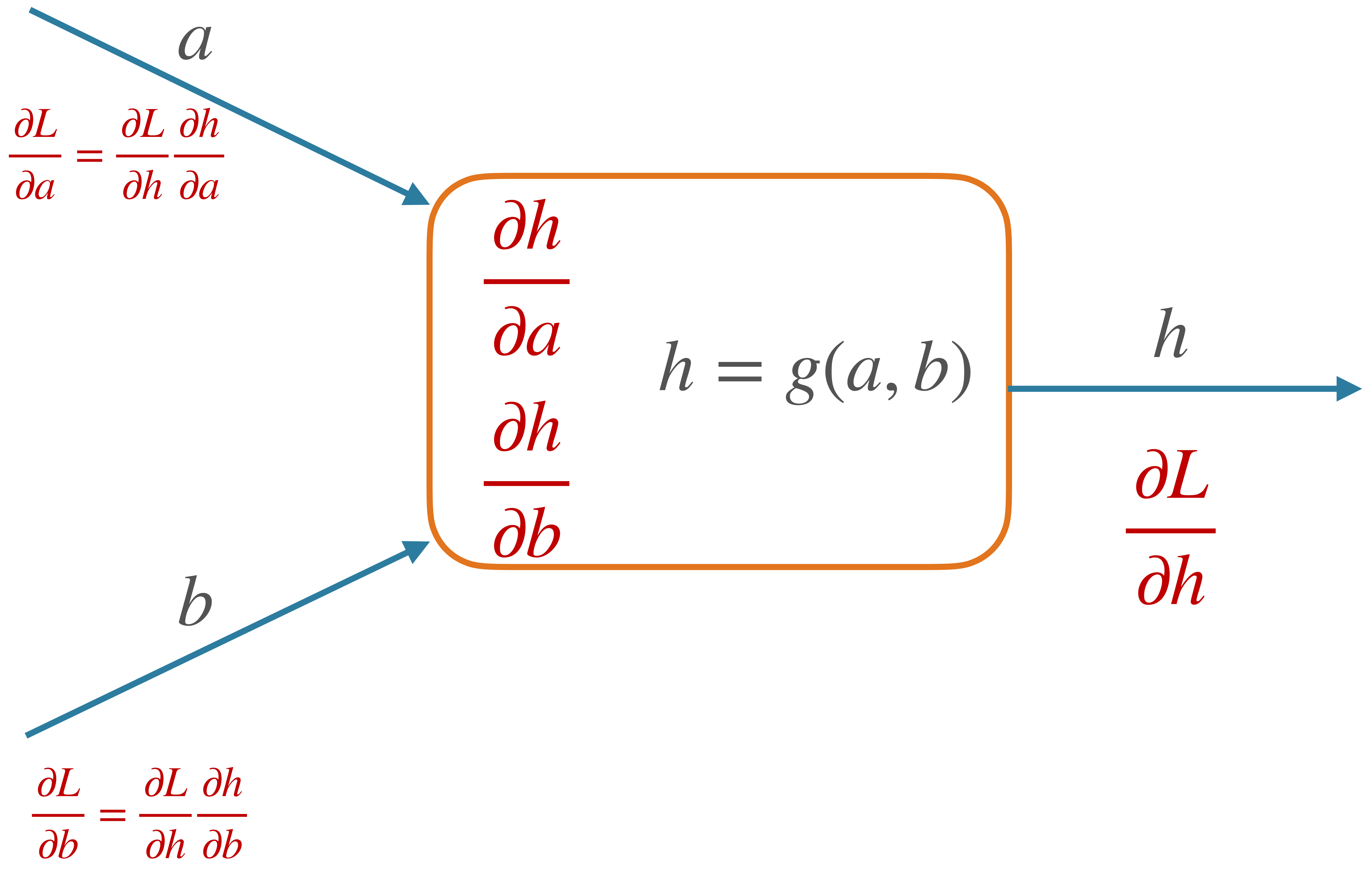
Nodes in Computational Graph

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



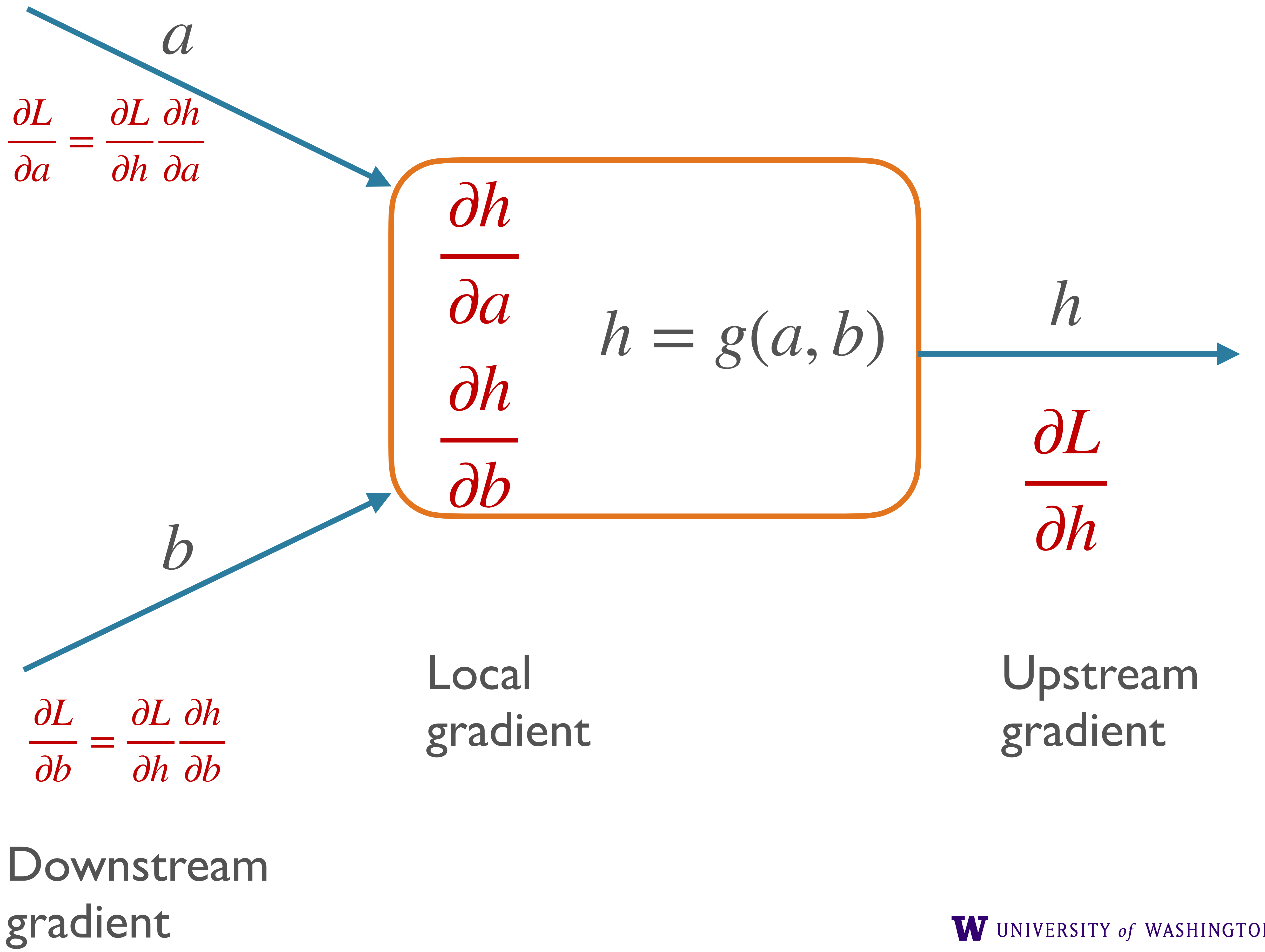
Nodes in Computational Graph

- 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!

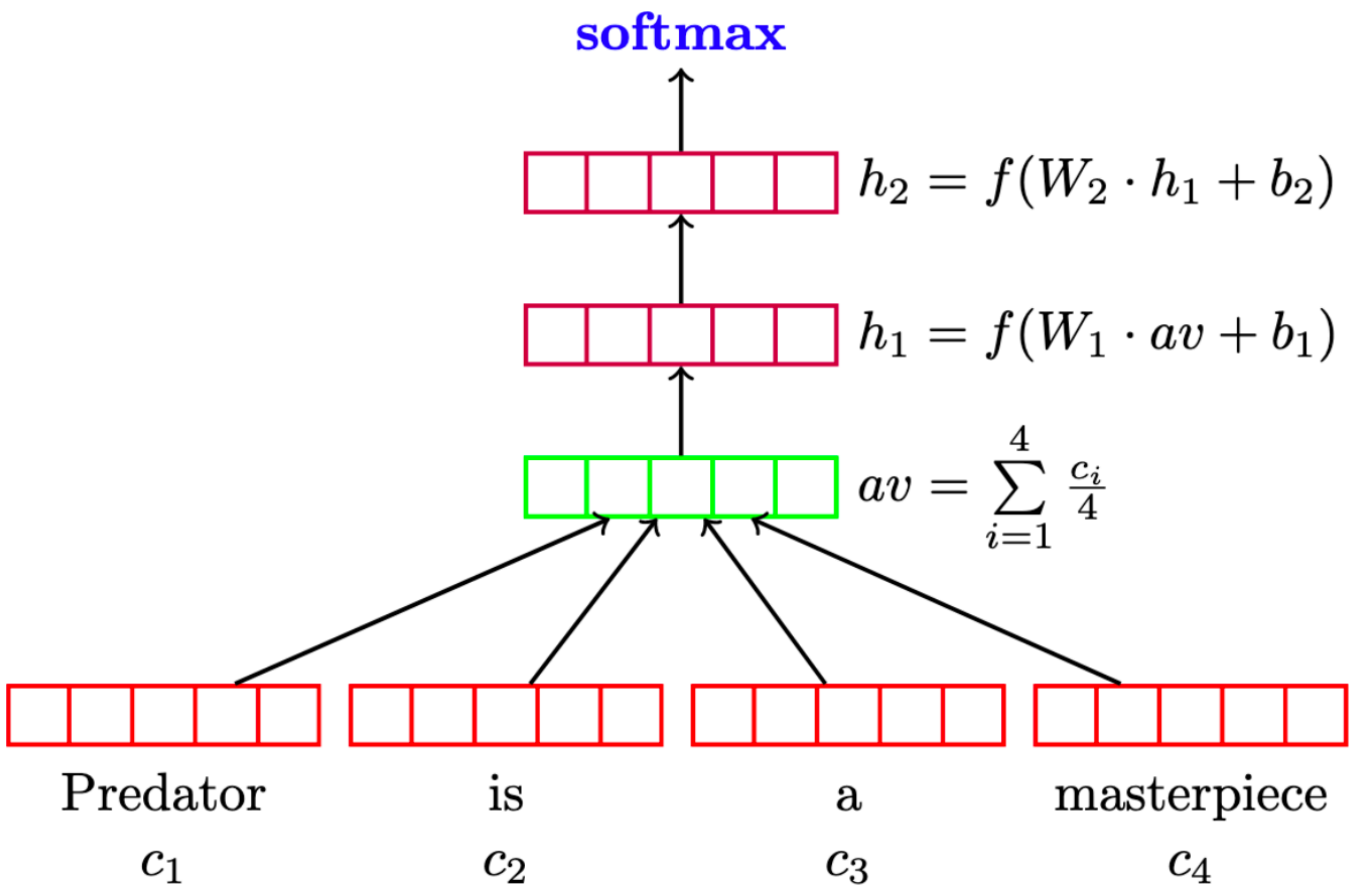
NB: list, one downstream gradient per input (in this case, one)

local gradient times upstream gradient

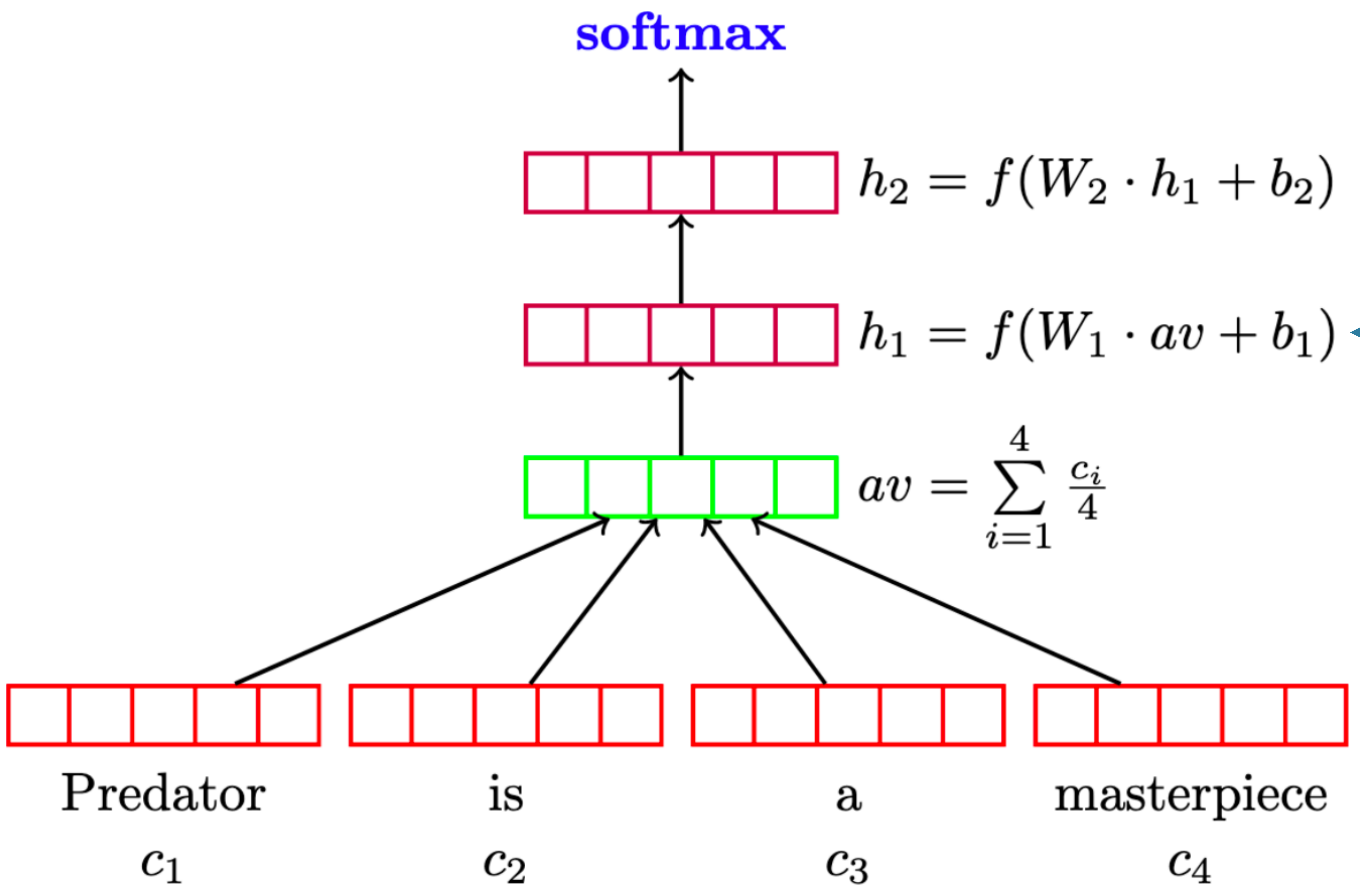
Neural Networks, I

- 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



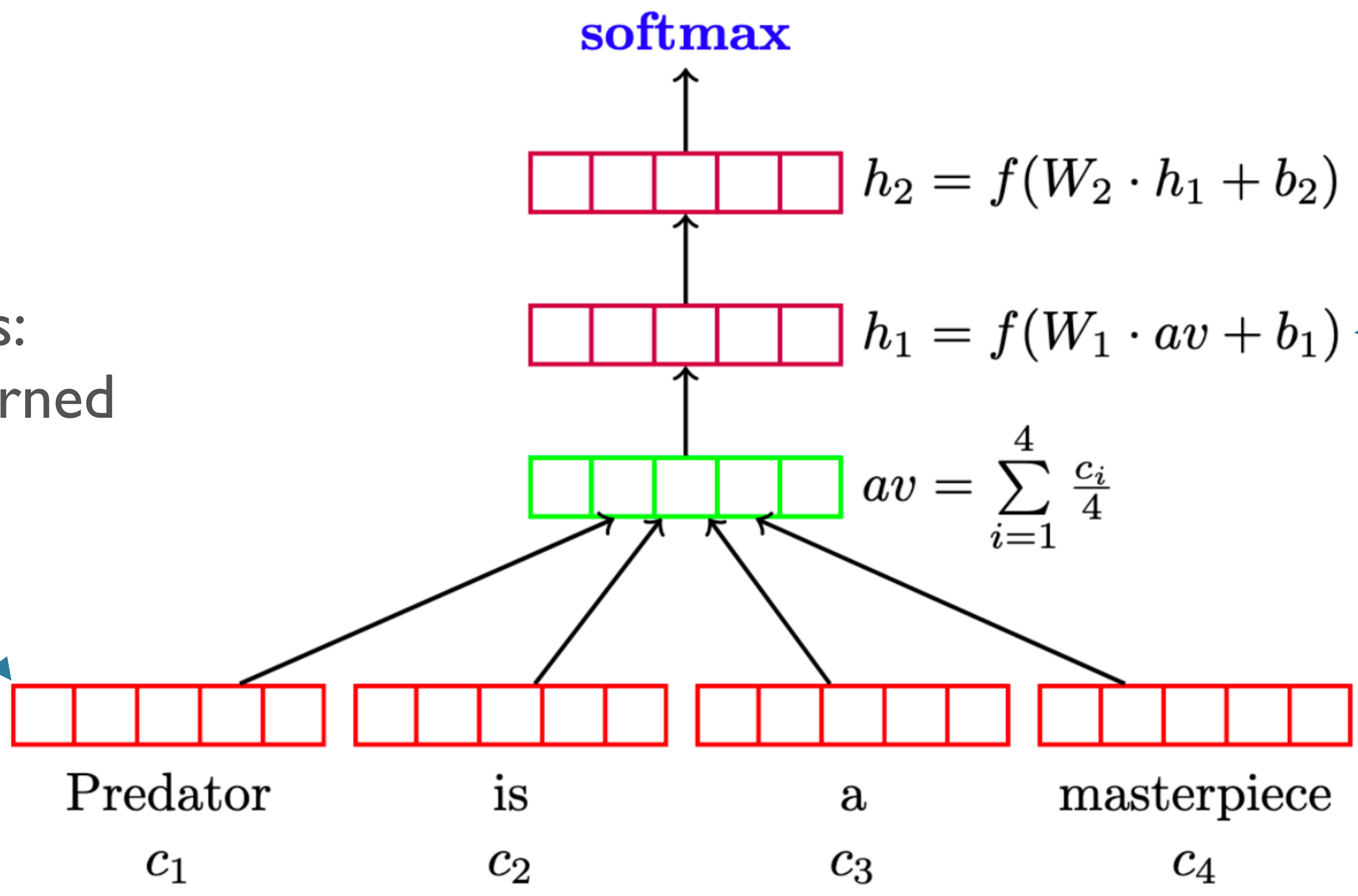
Model Architecture, One Input



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

Model Architecture, One Input

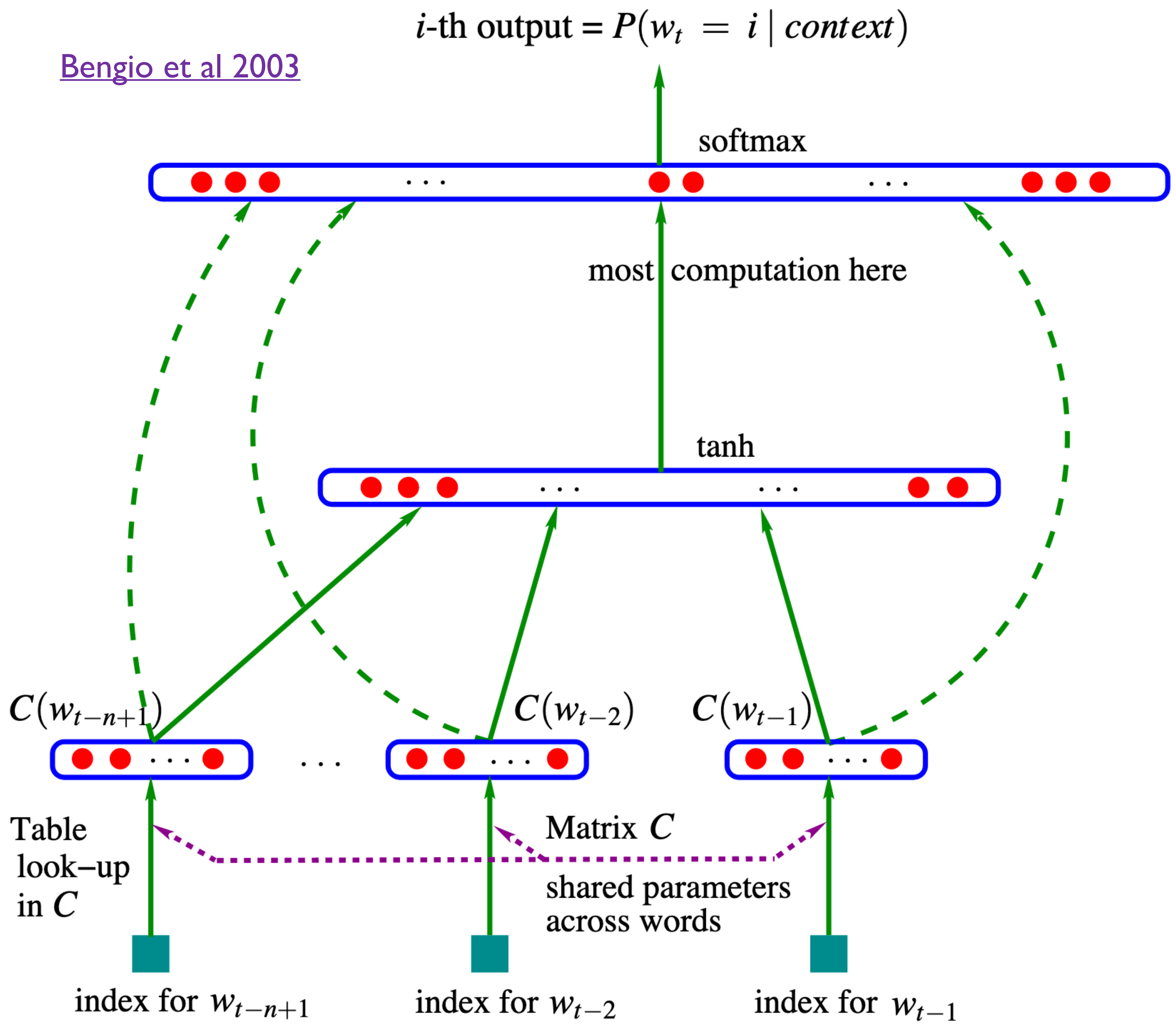
Word embeddings:
Pre-trained or learned



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

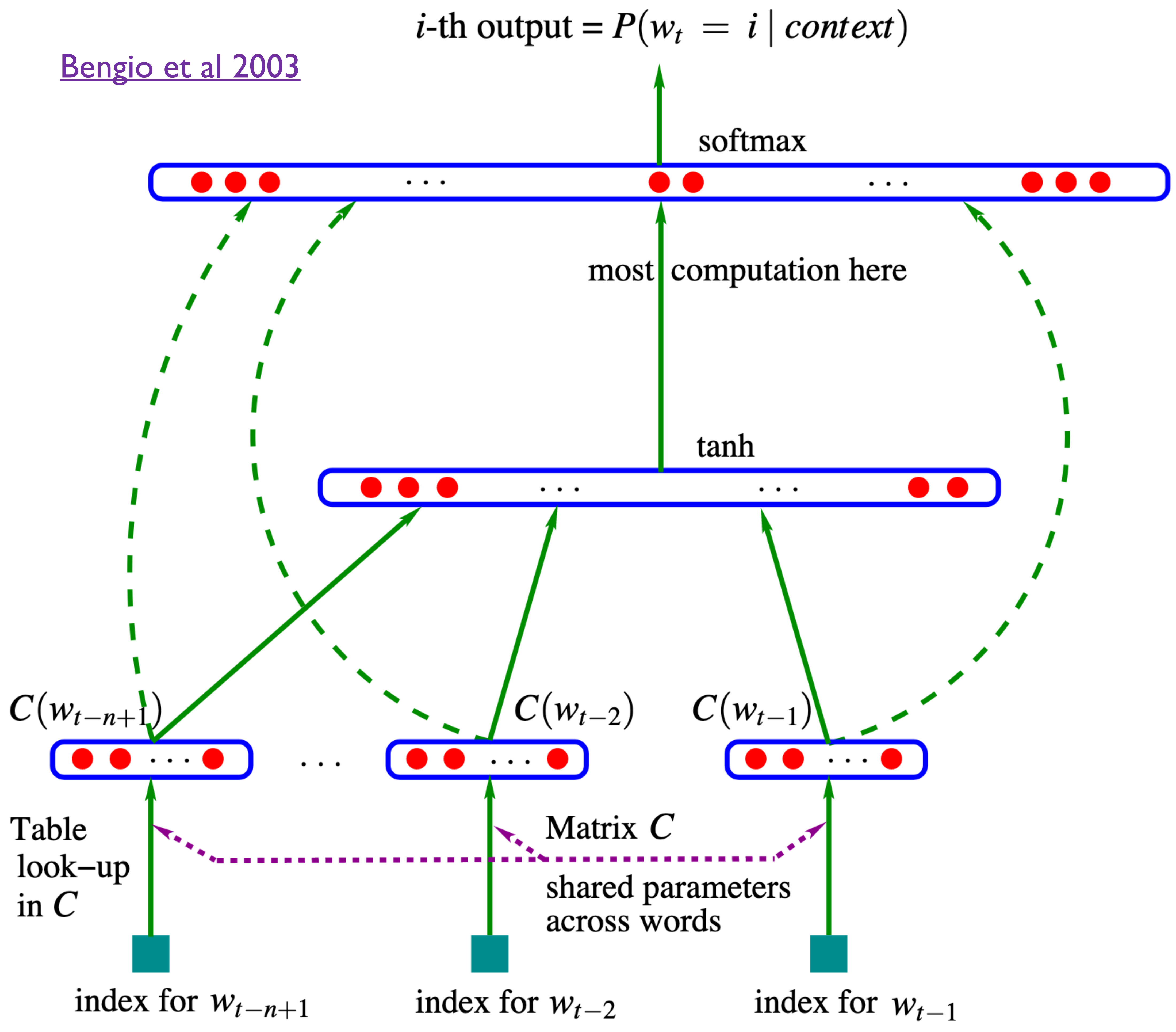
Neural LM Architecture

[Bengio et al 2003]



Neural LM Architecture

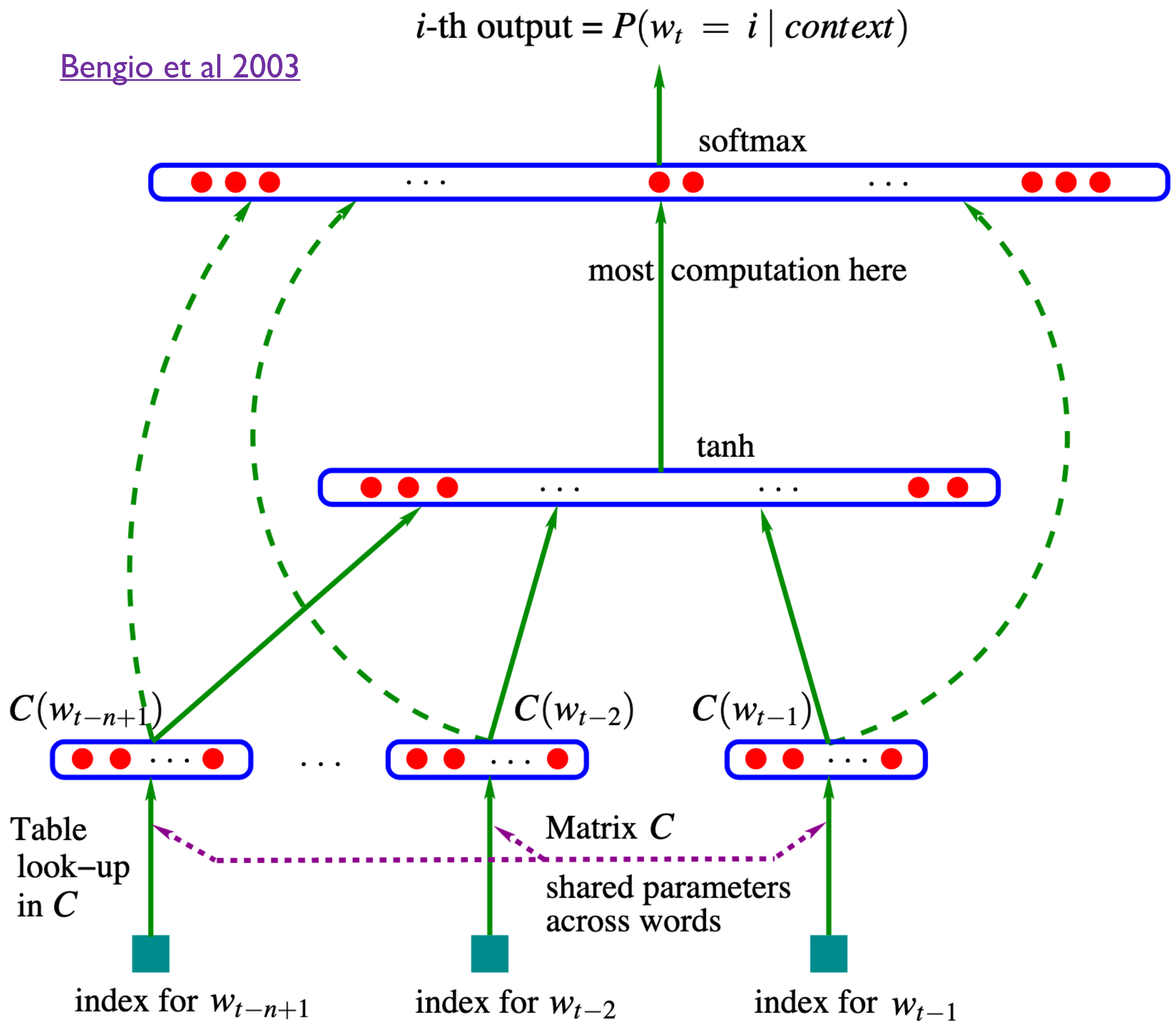
Bengio et al 2003



w_t: one-hot vector

Neural LM Architecture

Bengio et al 2003

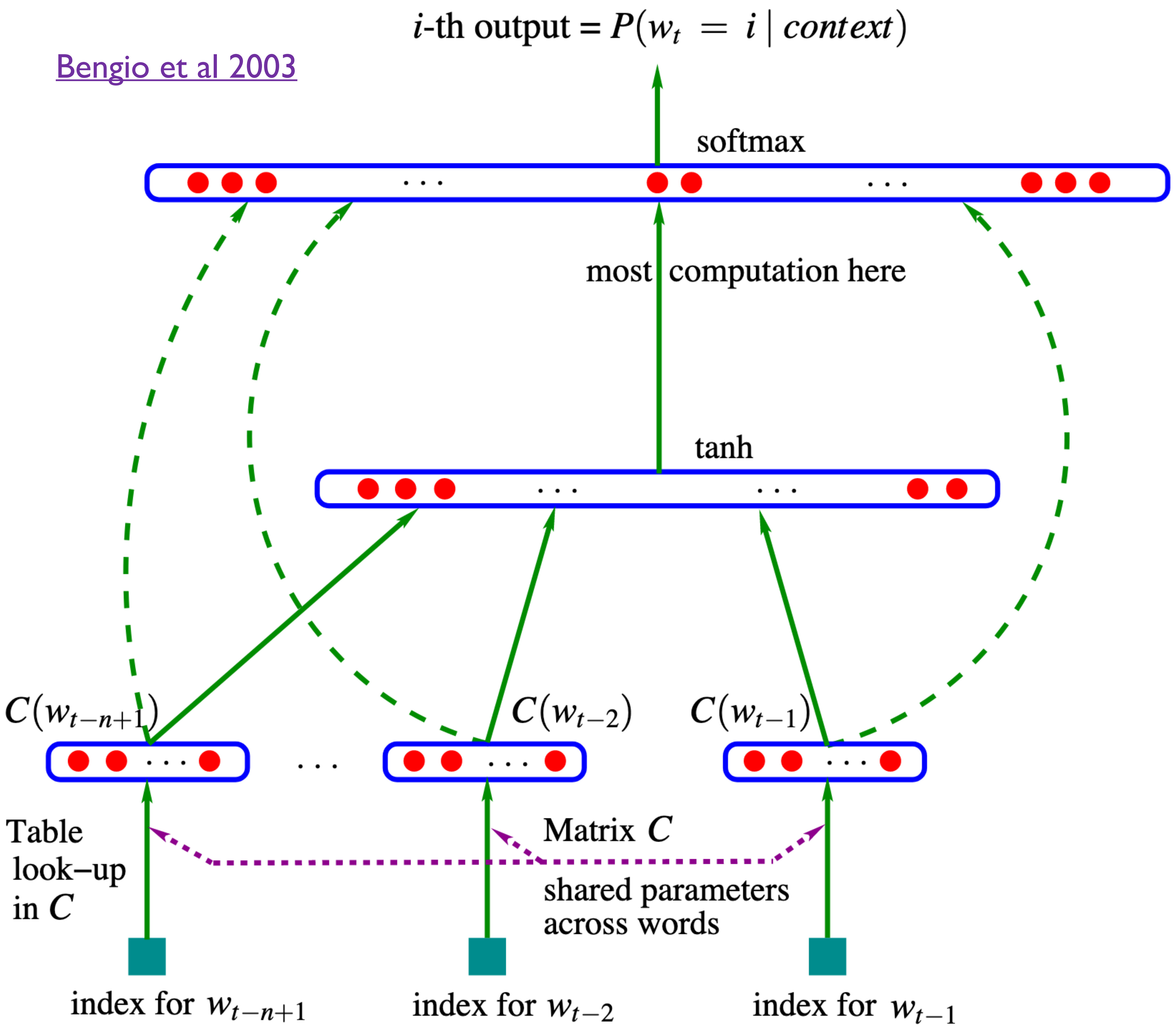


$$\text{embeddings} = \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$$

w_t: one-hot vector

Neural LM Architecture

Bengio et al 2003



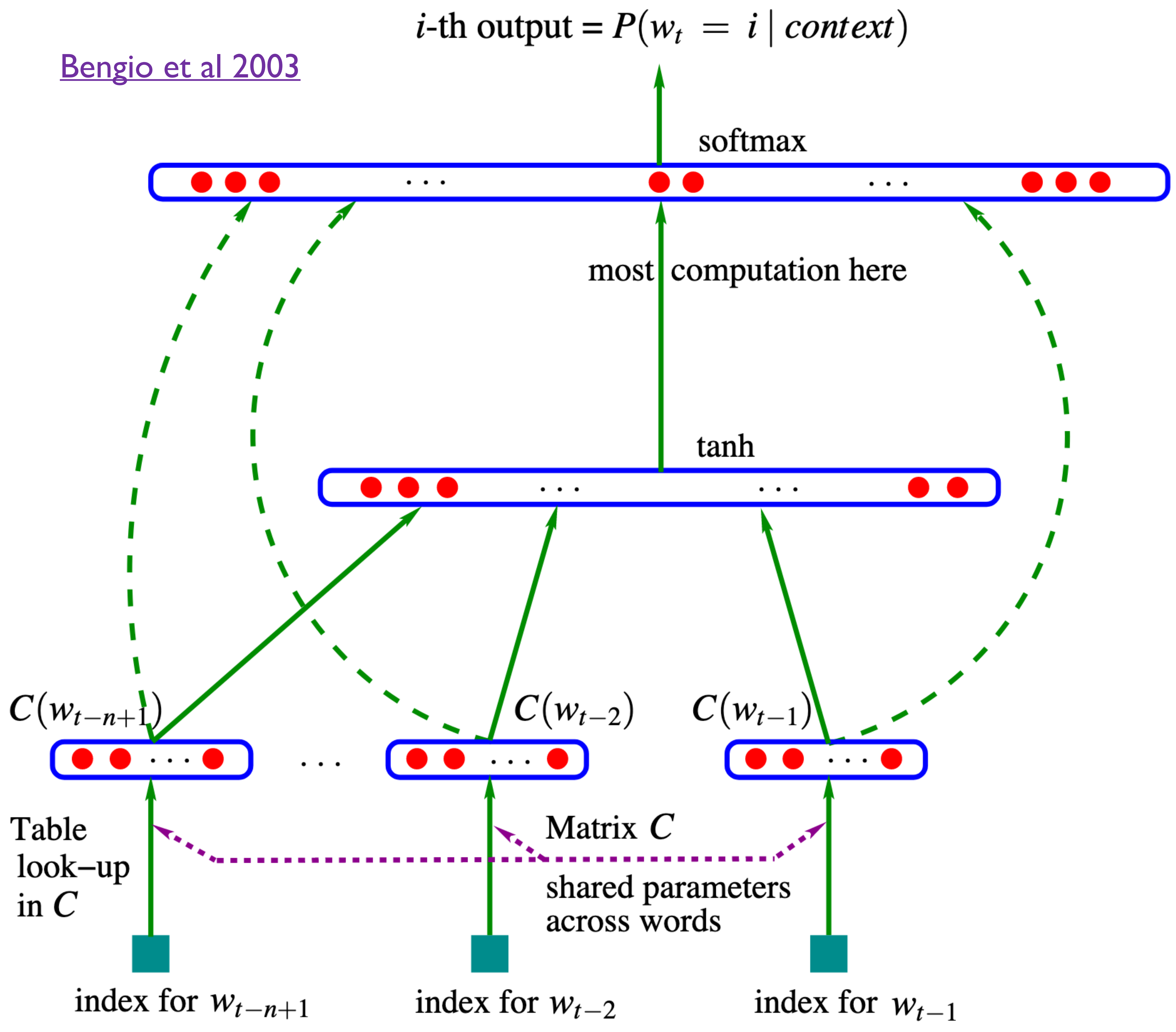
$$\text{hidden} = \tanh(\text{embeddings}W^1 + b^1)$$

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

w_t: one-hot vector

Neural LM Architecture

[Bengio et al 2003]



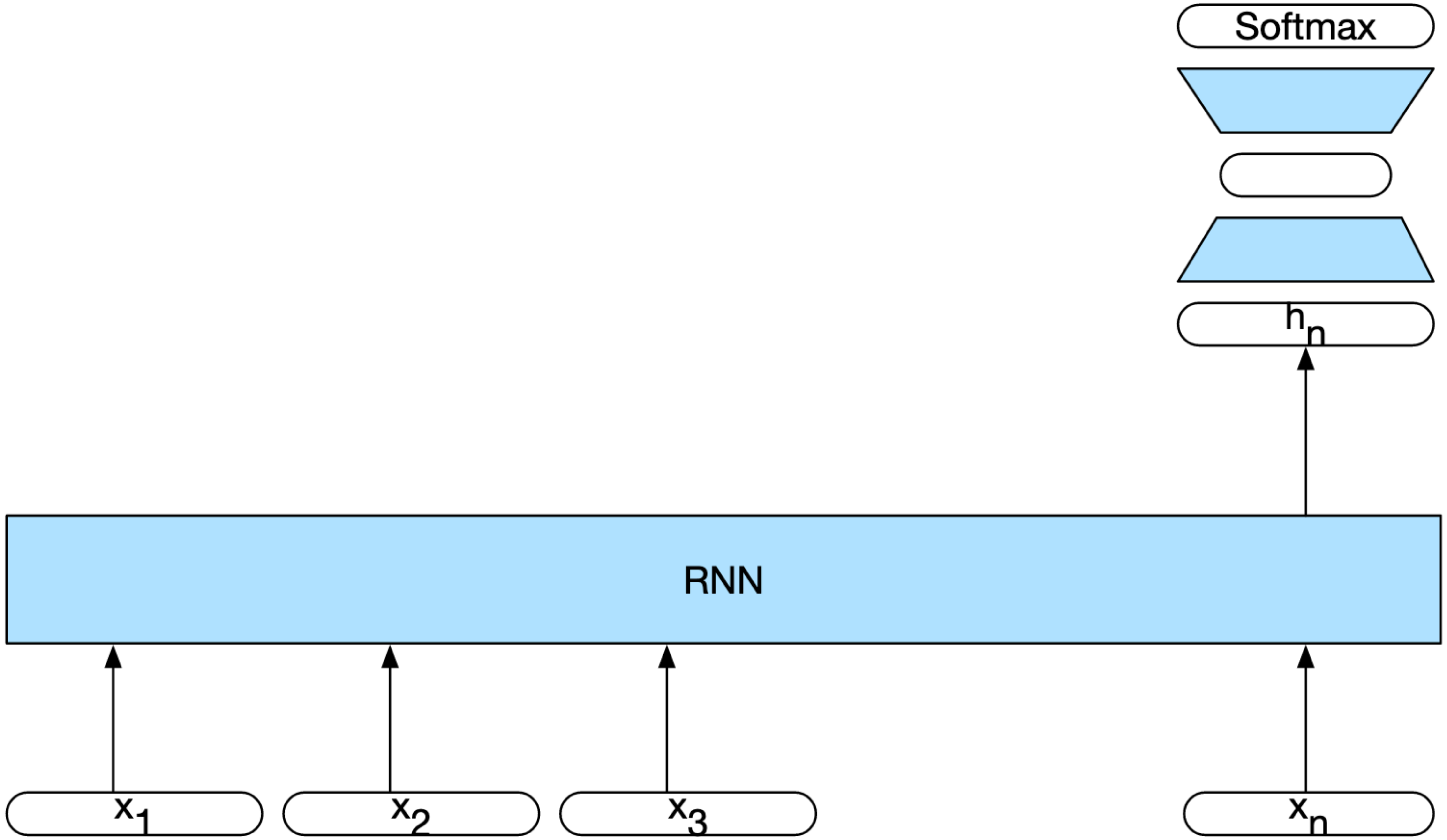
$$\text{probabilities} = \text{softmax}(\text{hidden}W^2 + b^2)$$

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

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

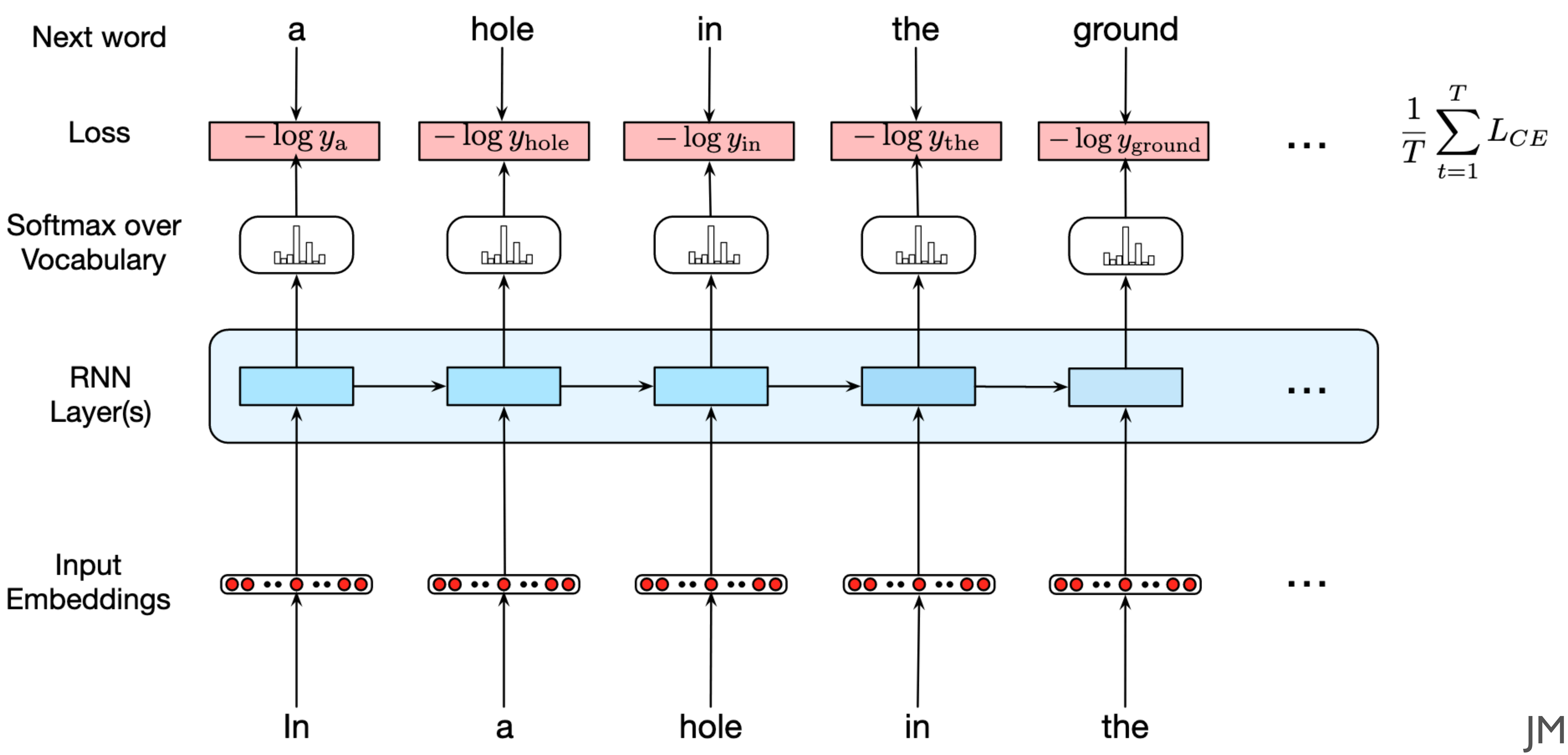
w_t: one-hot vector

RNN for Text Classification



JM sec 9.2.5

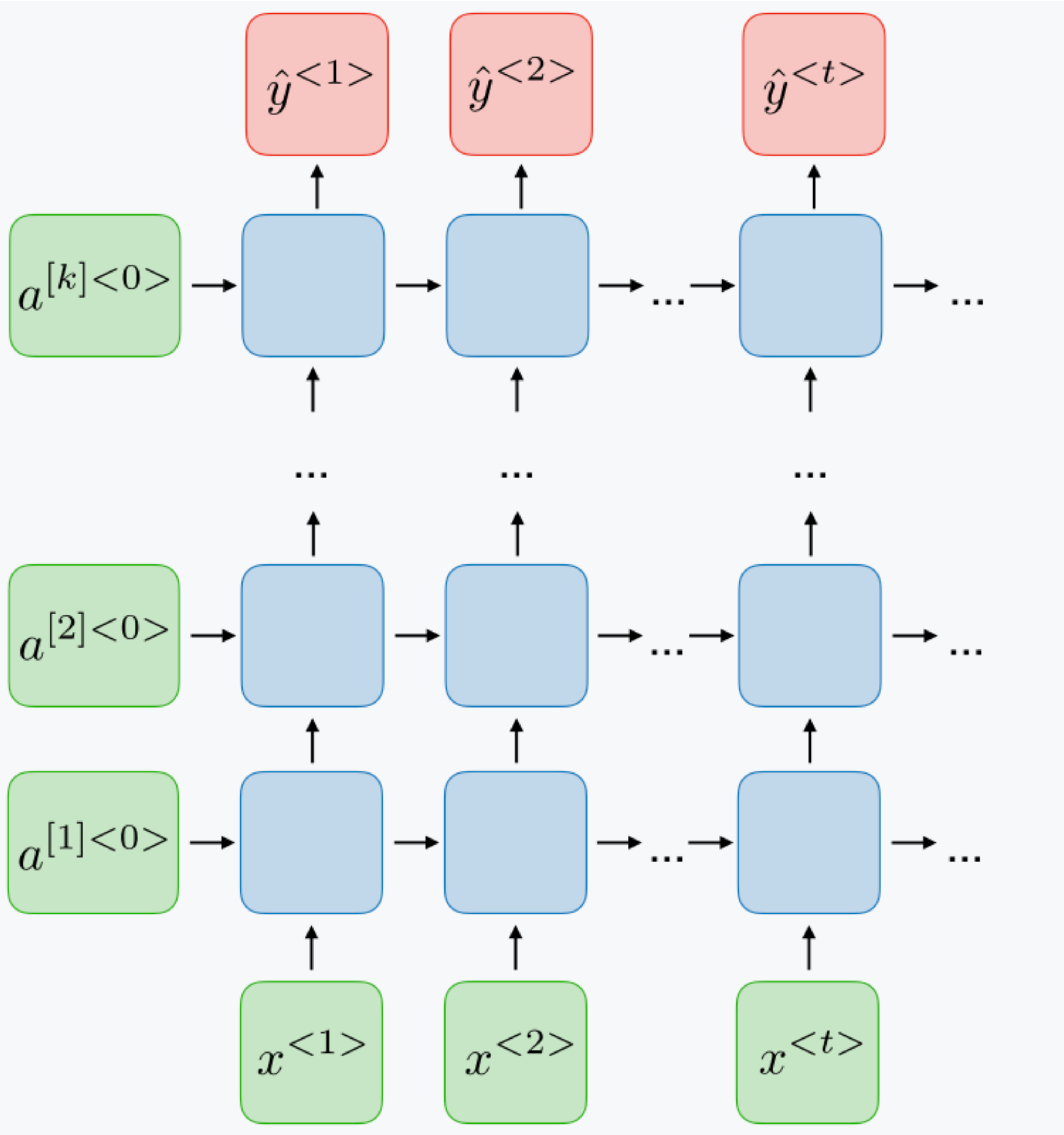
RNNs for Language Modeling



JM sec 9.2.3

Two Extensions

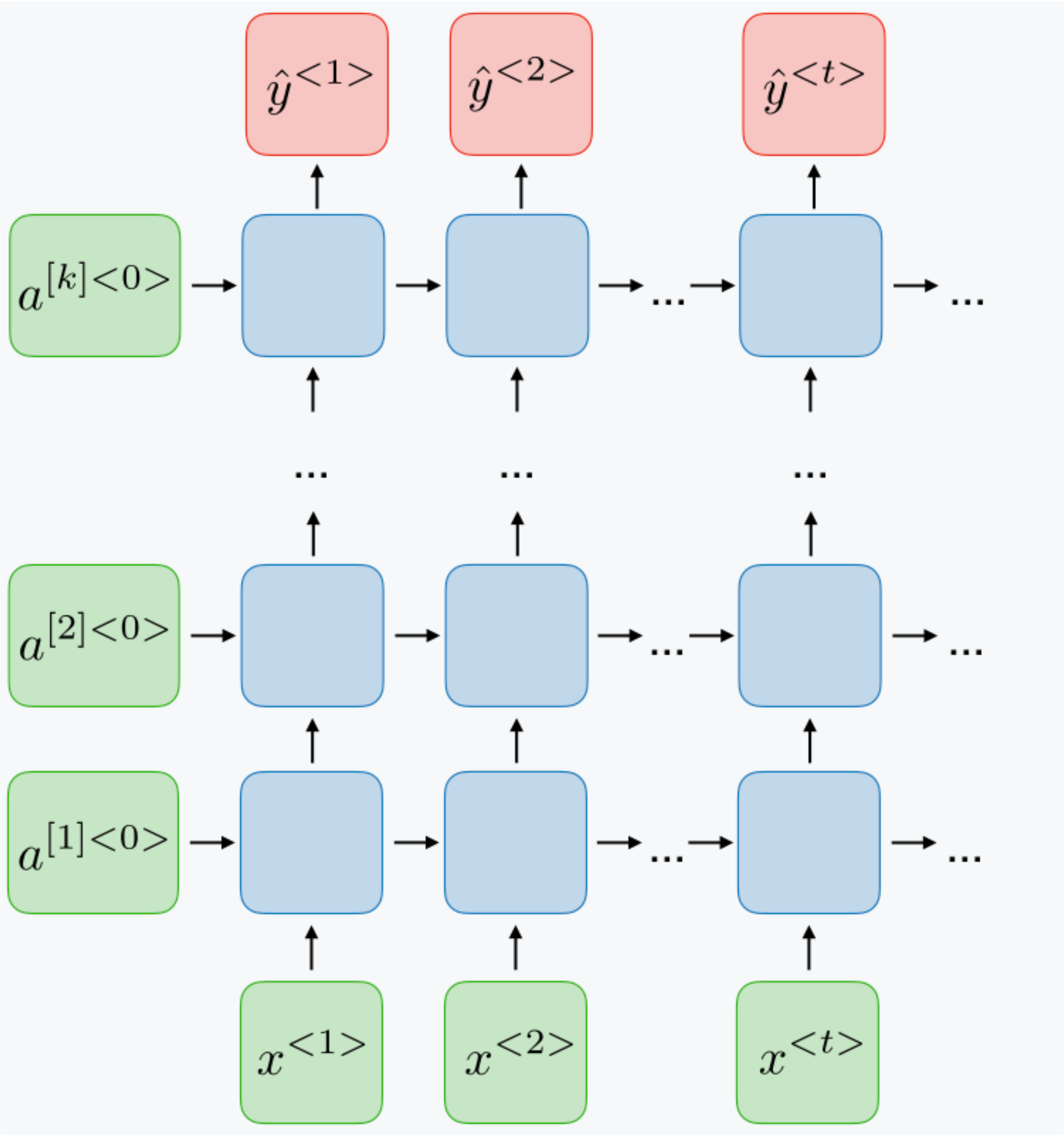
- Deep RNNs:



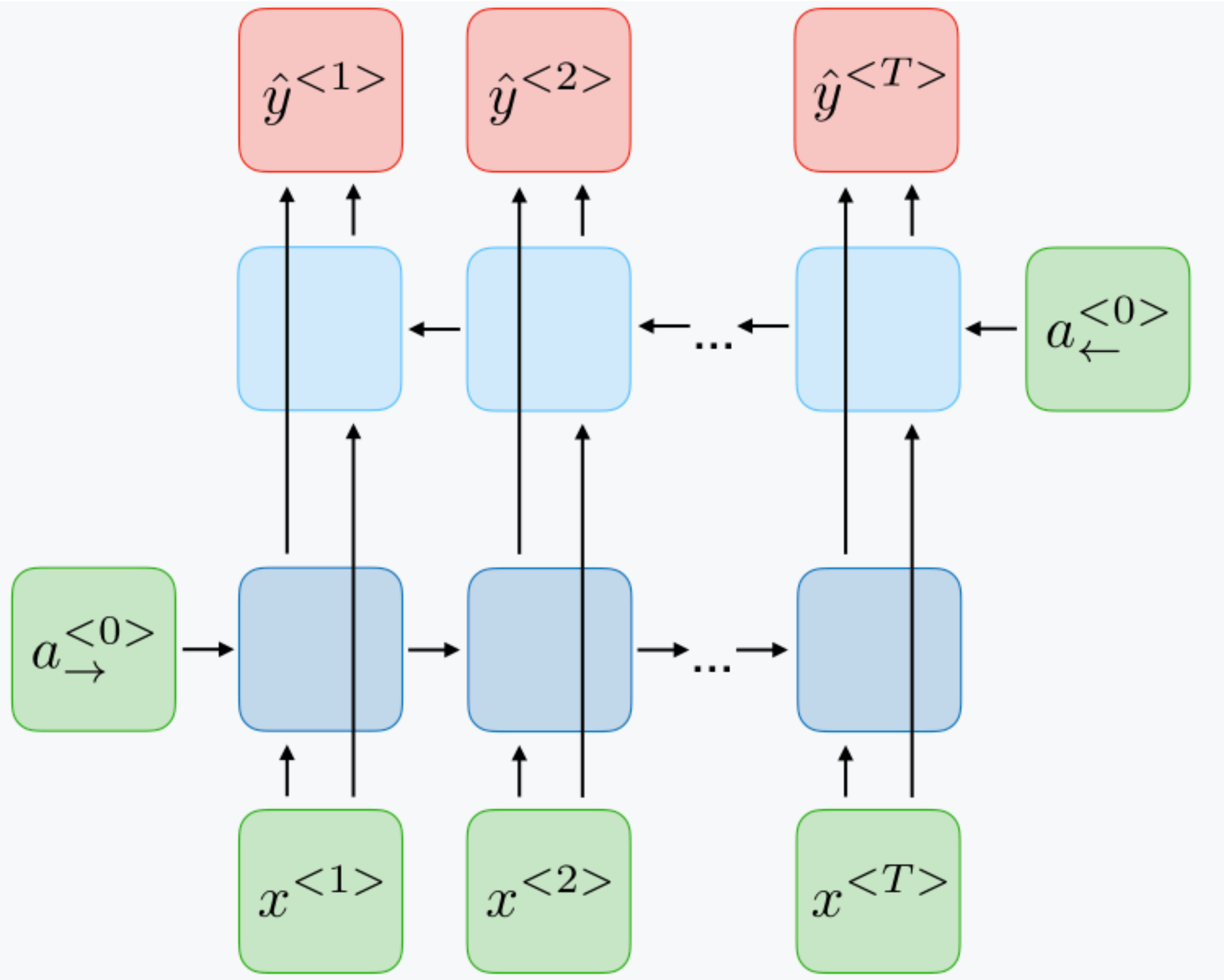
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



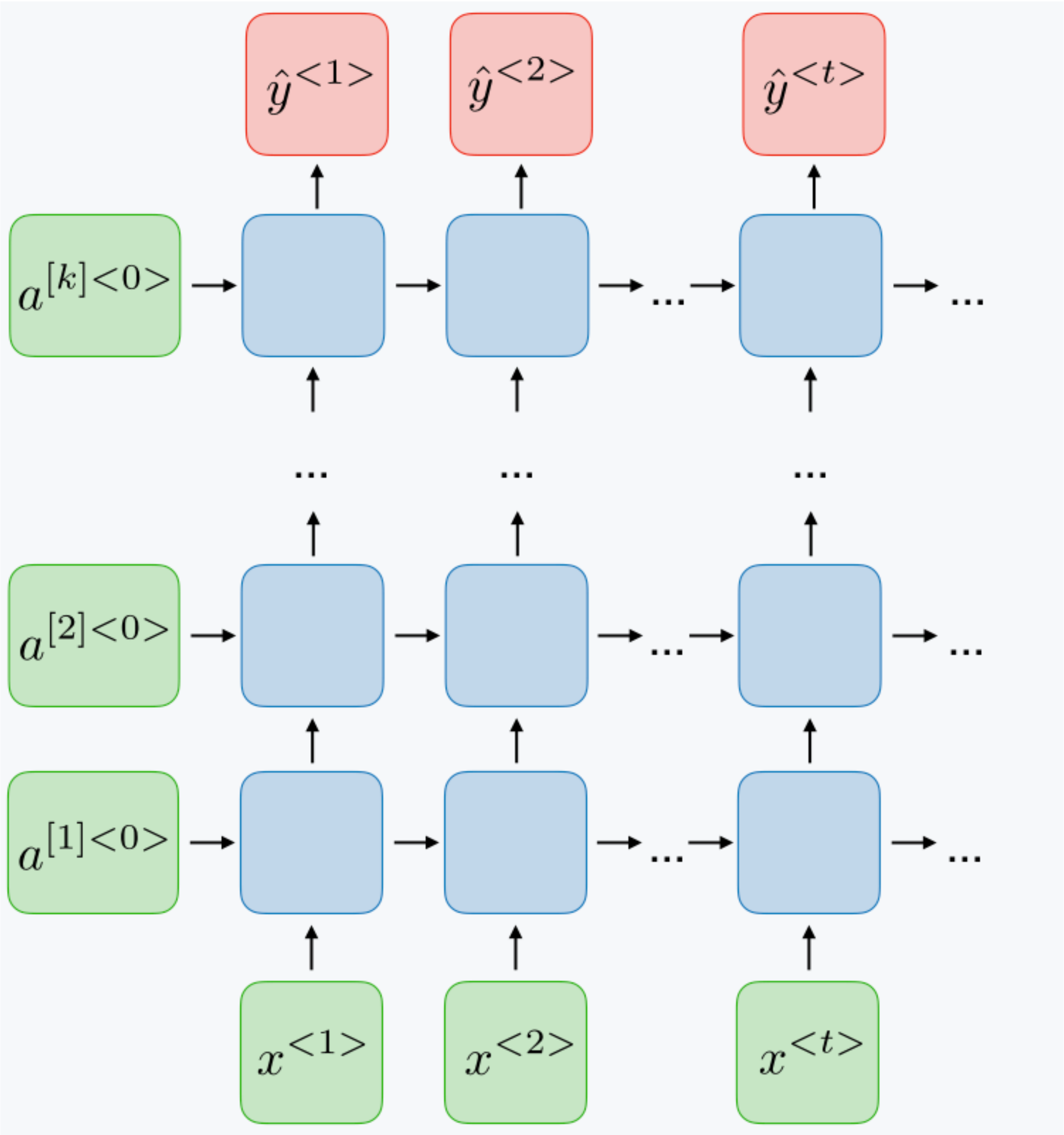
- Bidirectional RNNs:



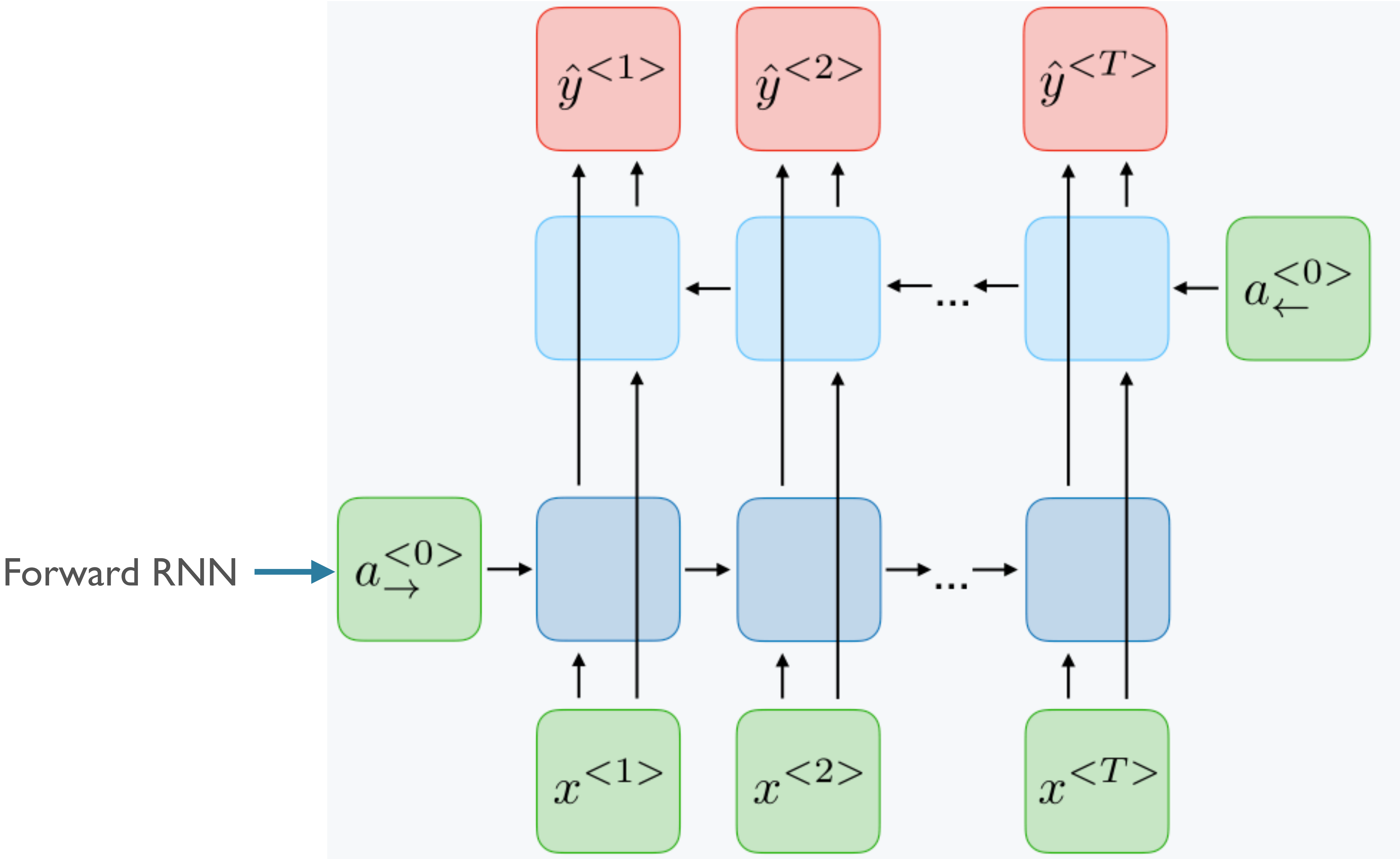
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



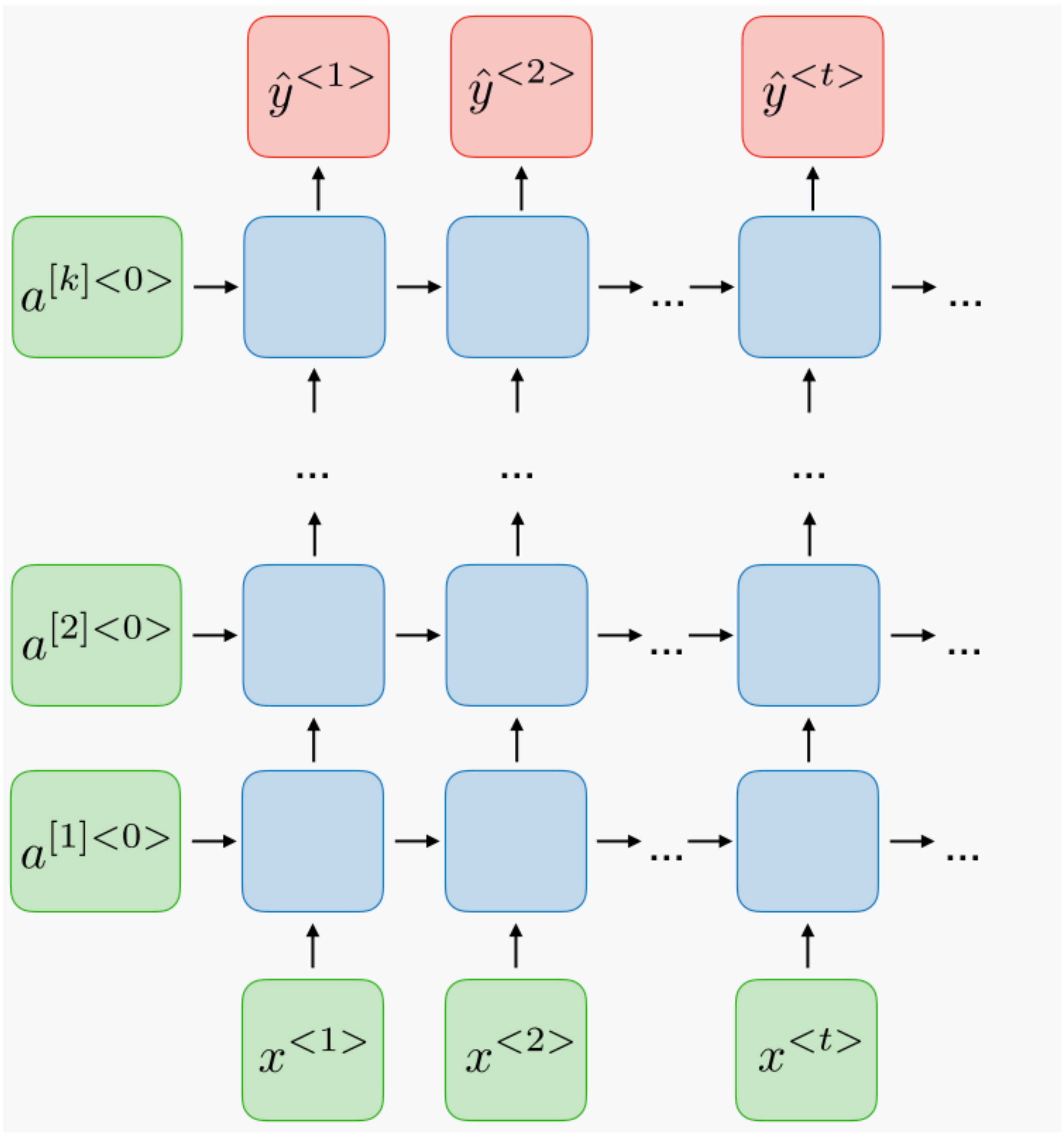
- Bidirectional RNNs:



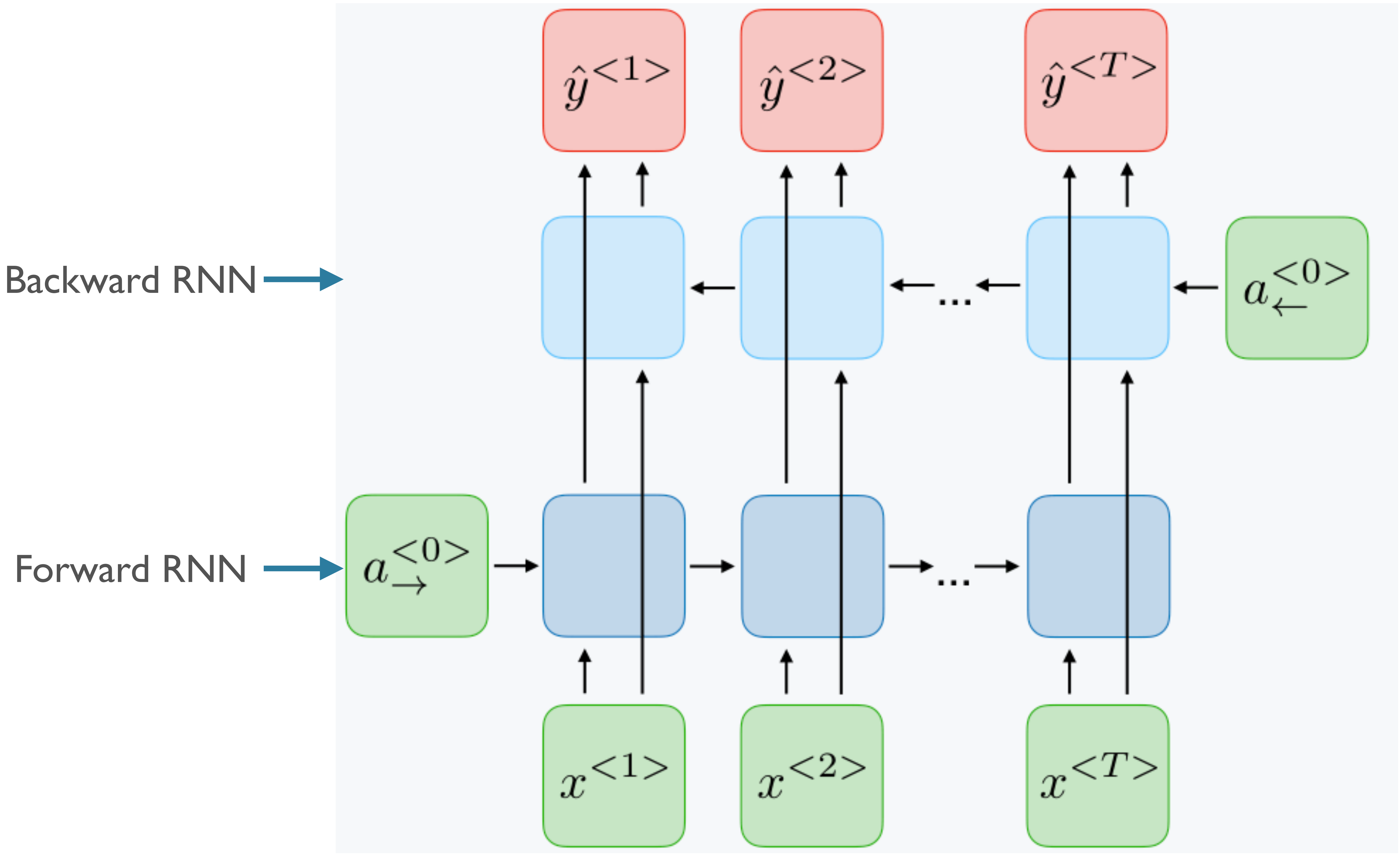
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



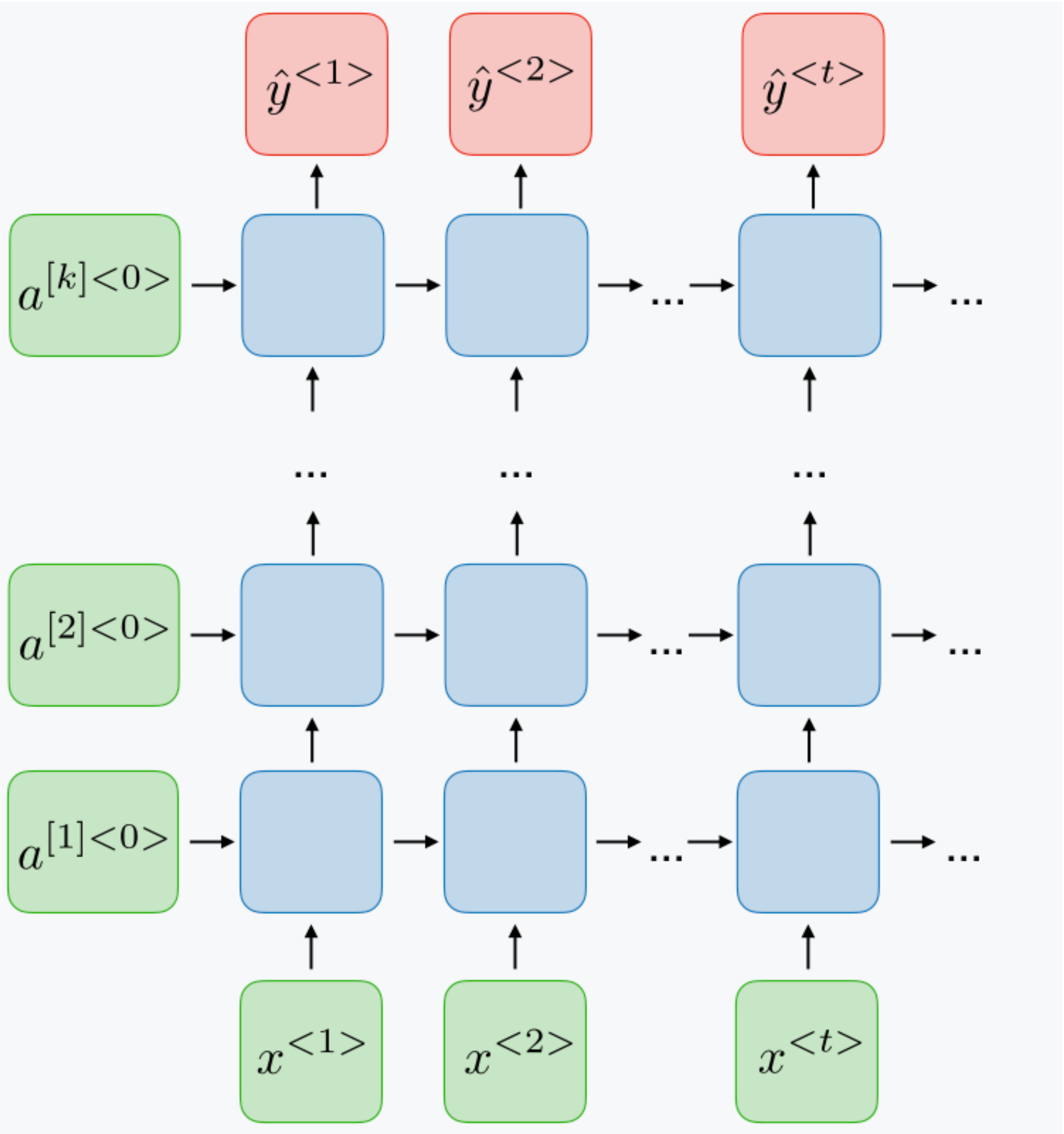
- Bidirectional RNNs:



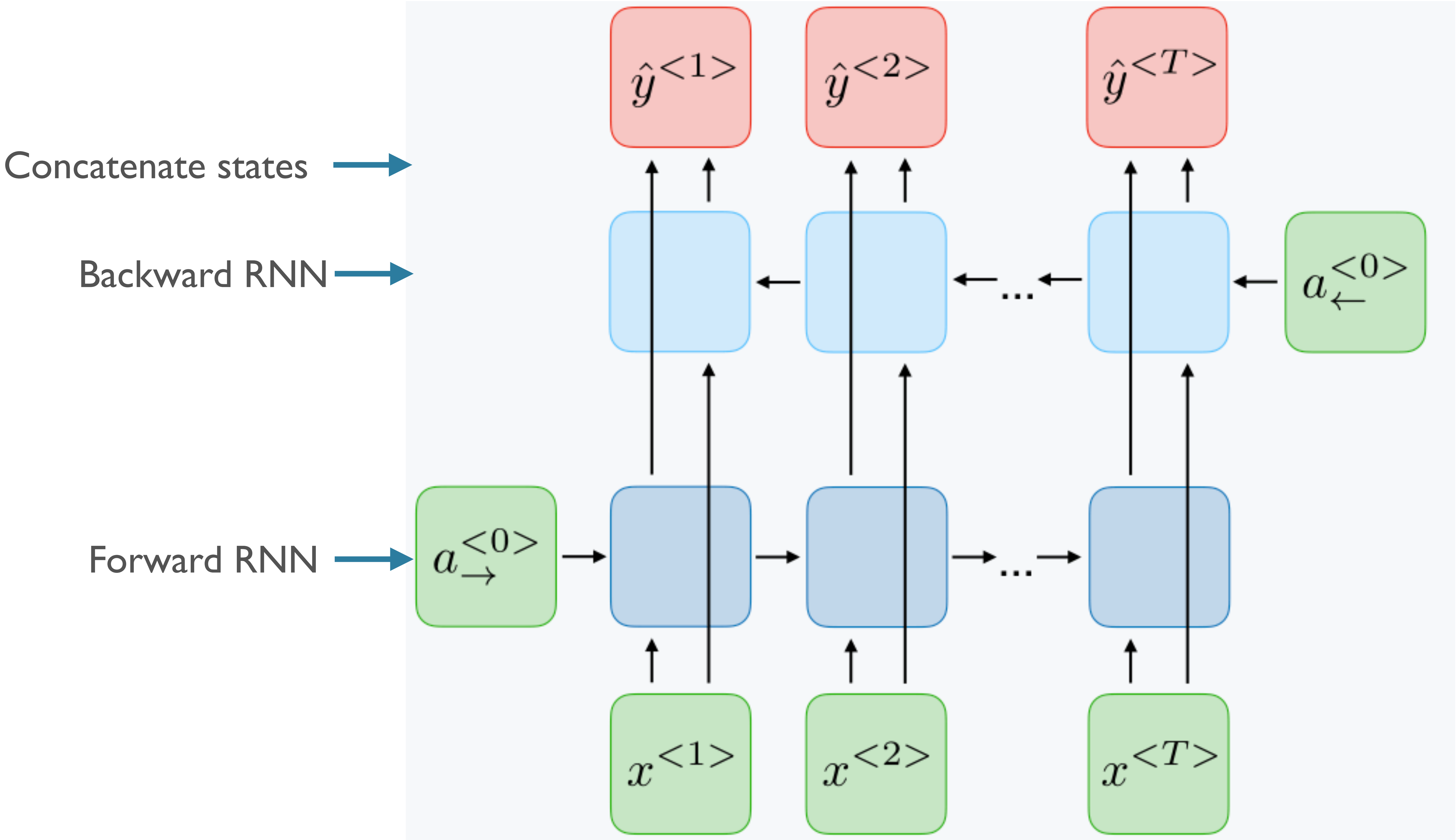
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:

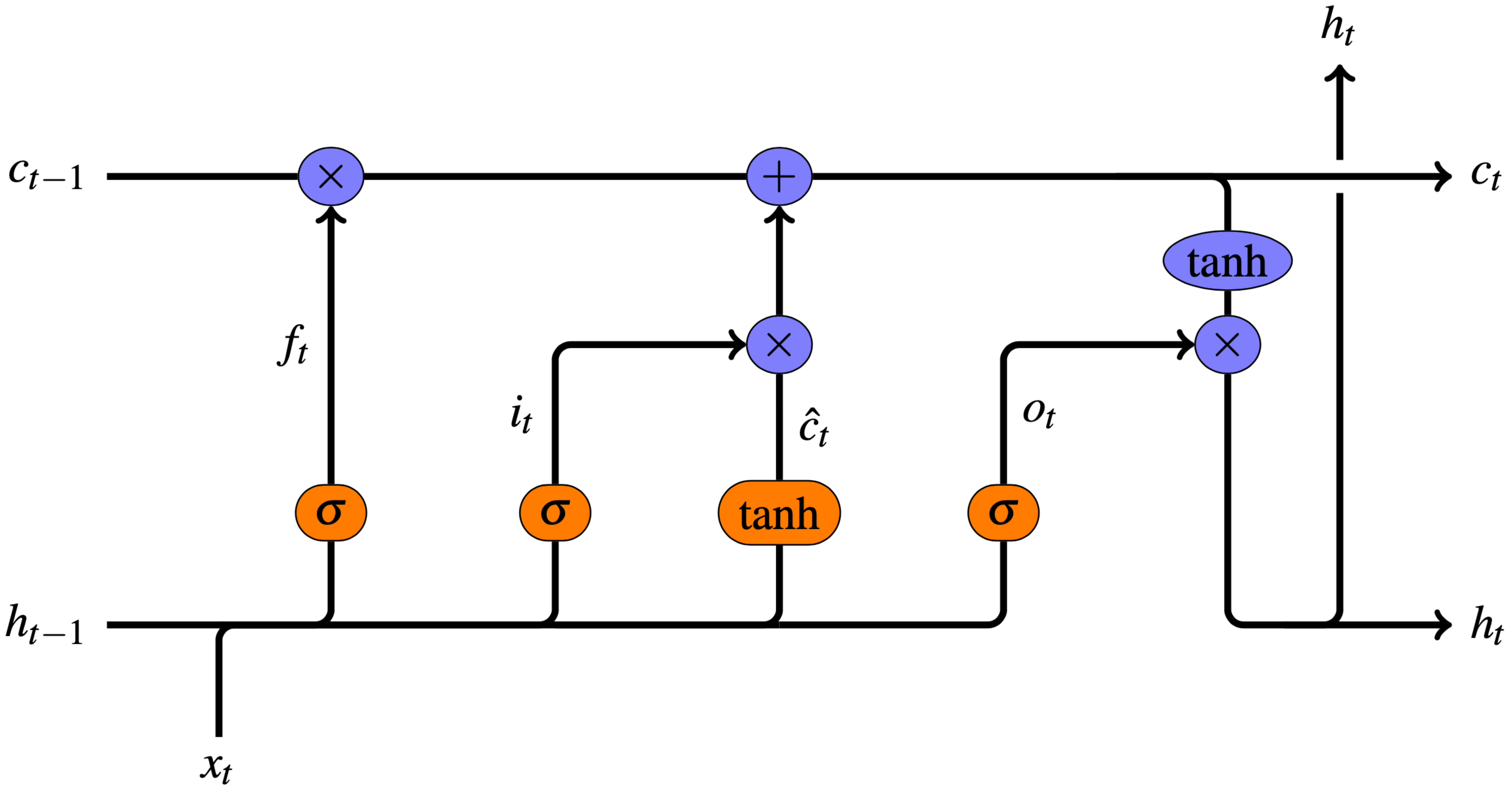


- Bidirectional RNNs:

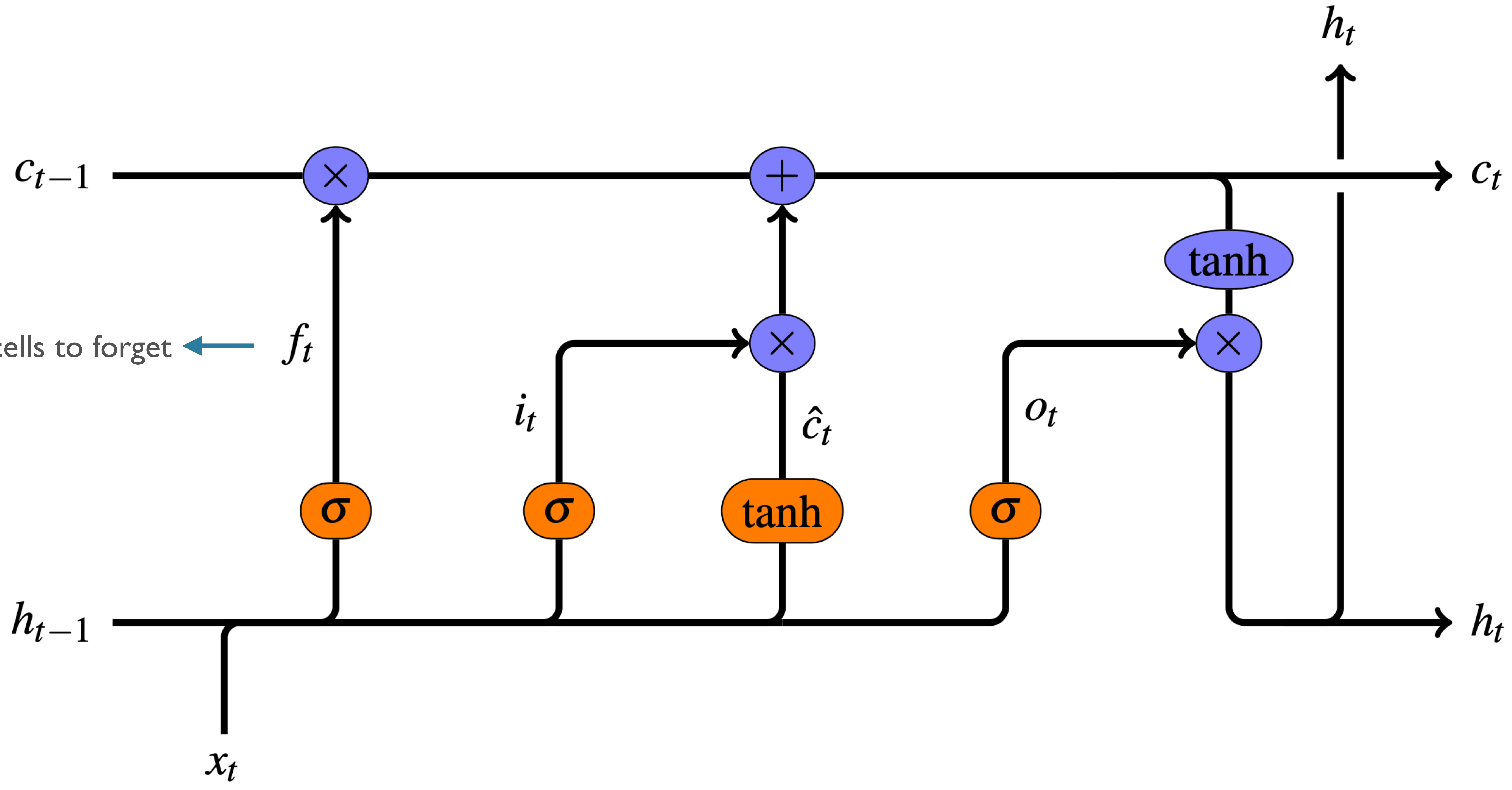


Source: RNN cheat sheet

LSTMs



LSTMs

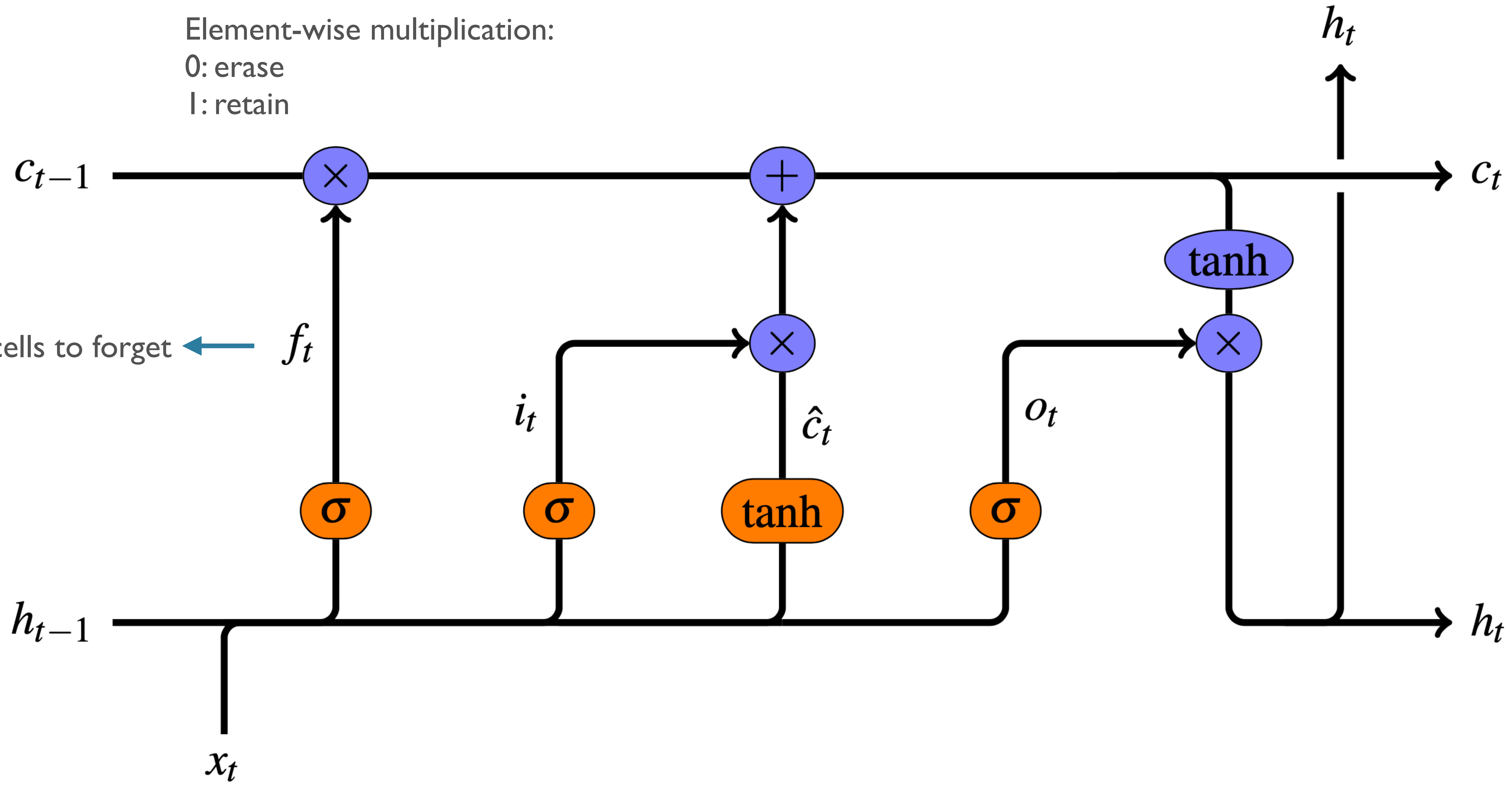


$f_t \in [0,1]^m$: which cells to forget

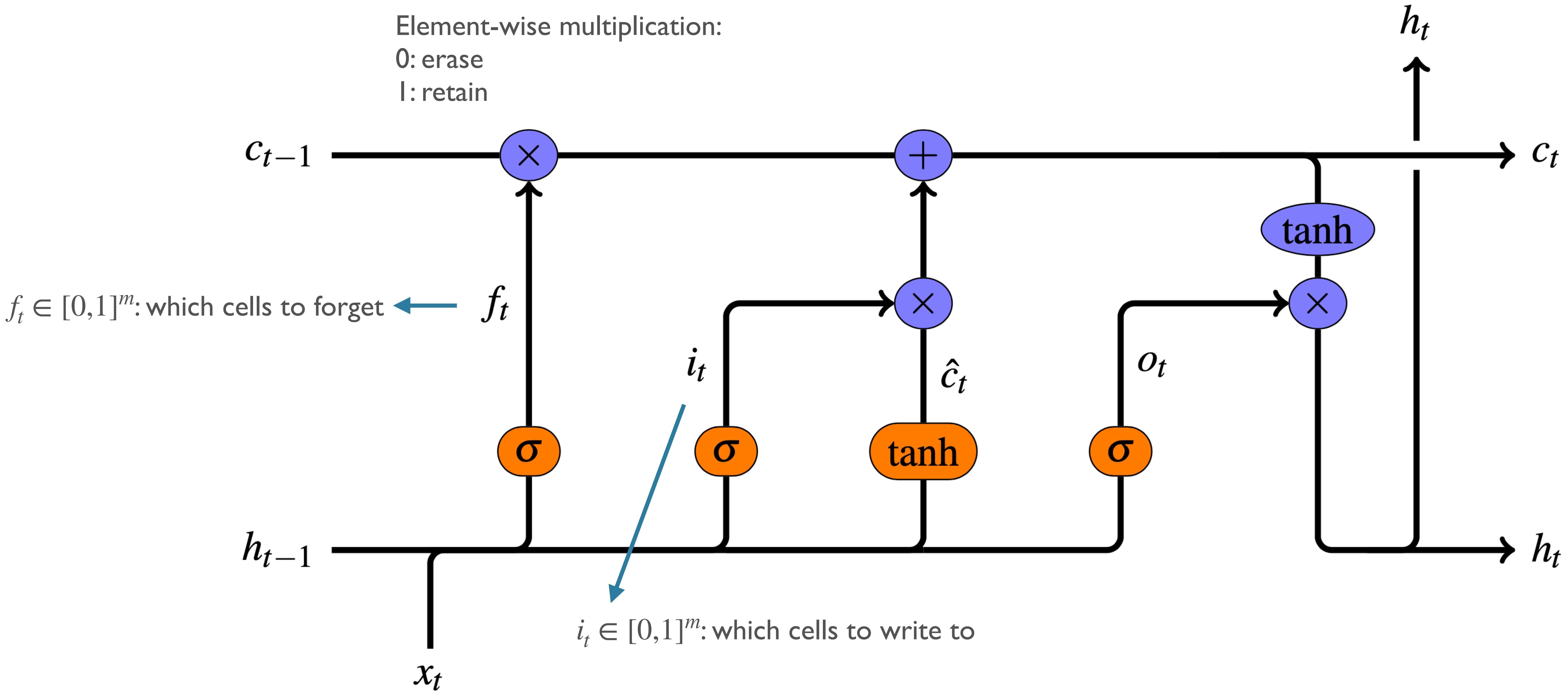
LSTMs

Element-wise multiplication:
0: erase
1: retain

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

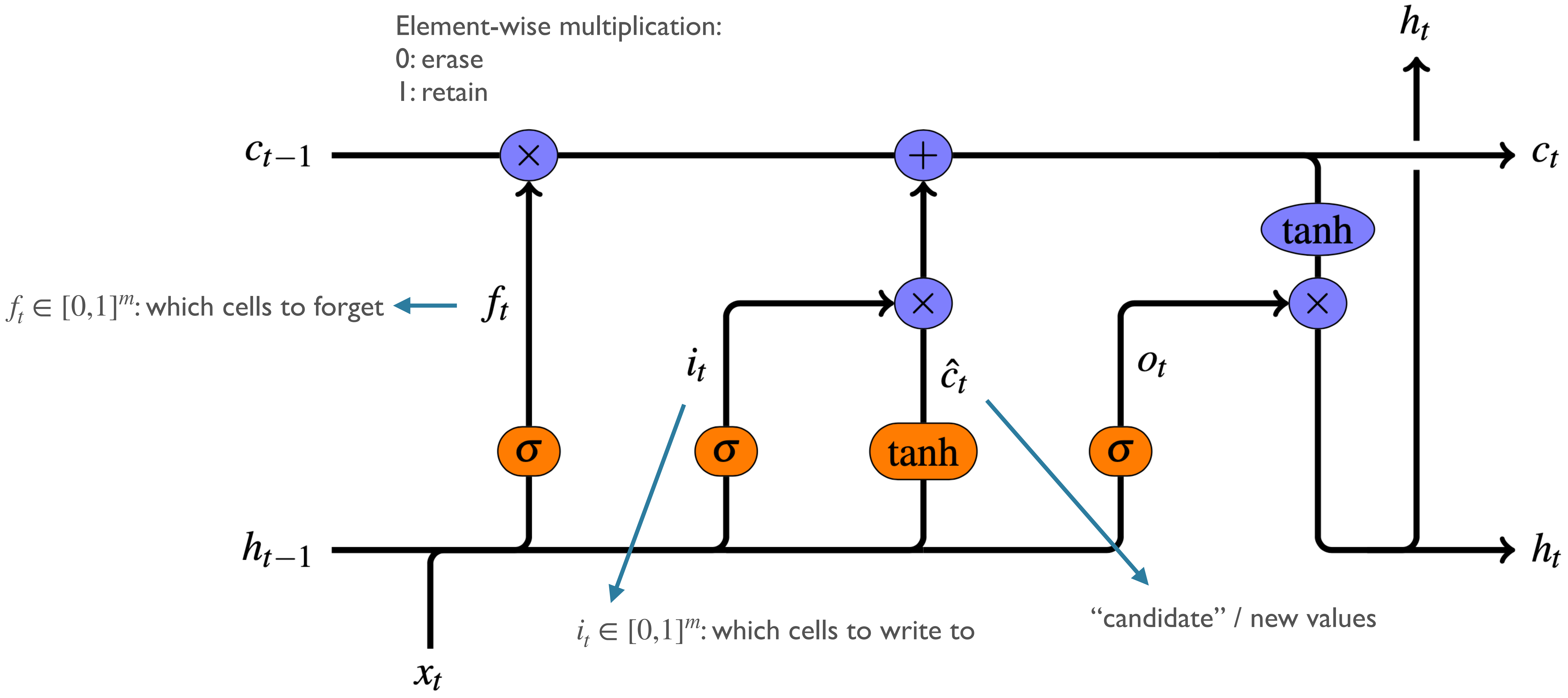


LSTMs



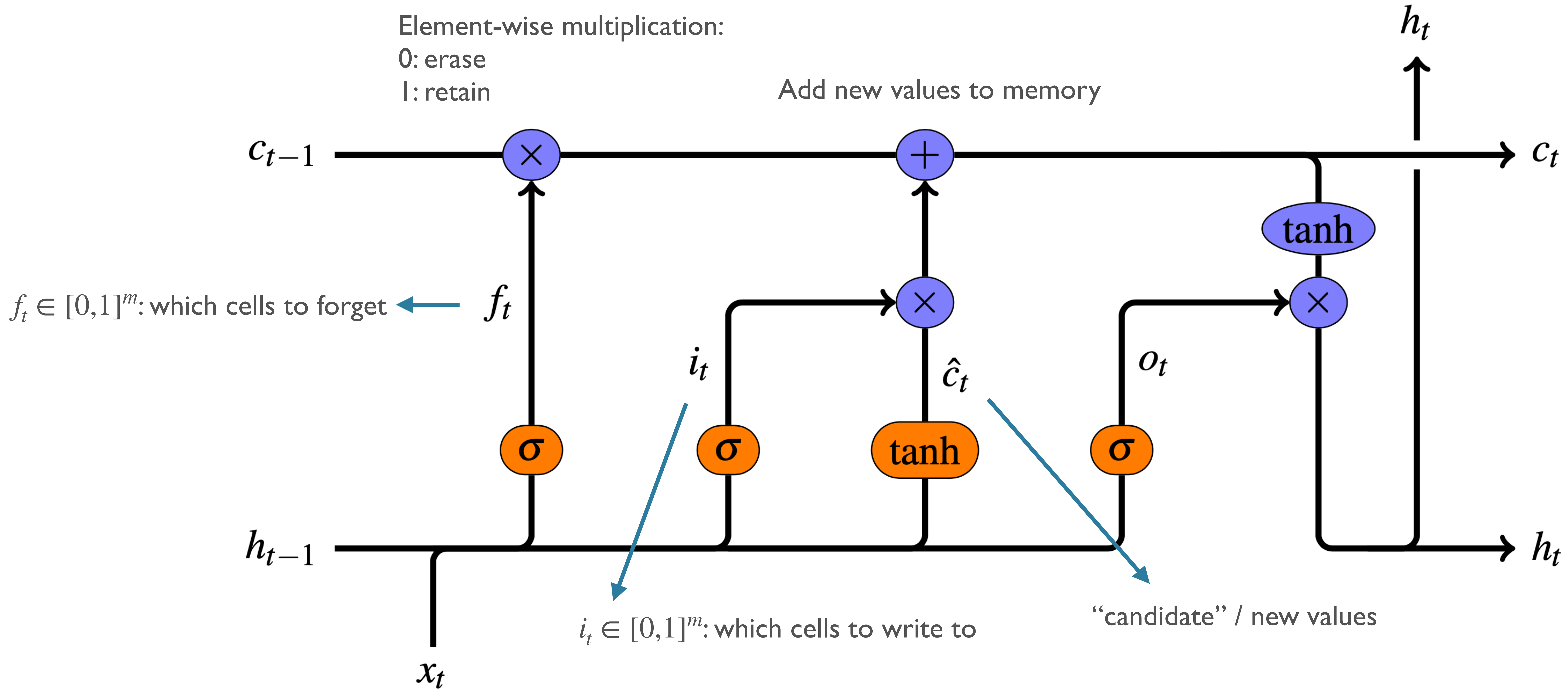
Element-wise multiplication:
0: erase
1: retain
h_t
c_{t-1}
c_t
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
$i_t \in [0,1]^m$: which cells to write to
x_t

LSTMs



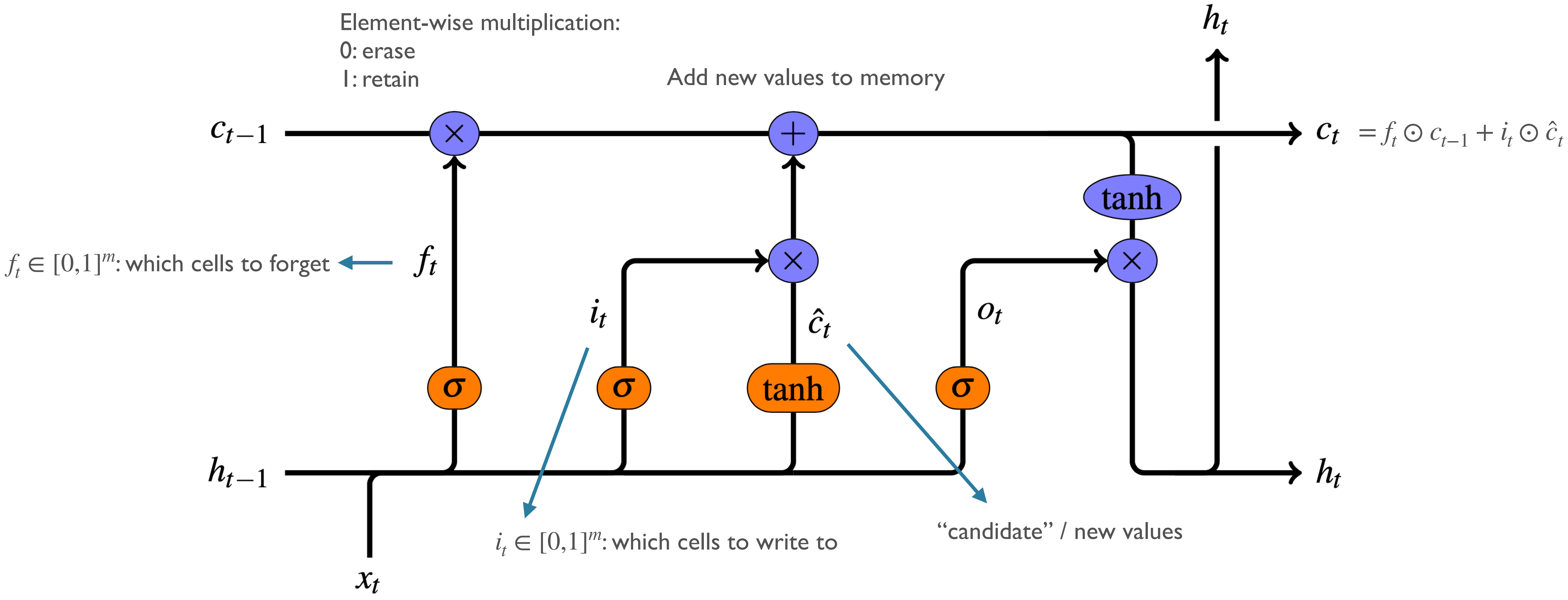
Element-wise multiplication:
0: erase
1: retain
h_t
c_{t-1}
c_t
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values
x_t

LSTMs

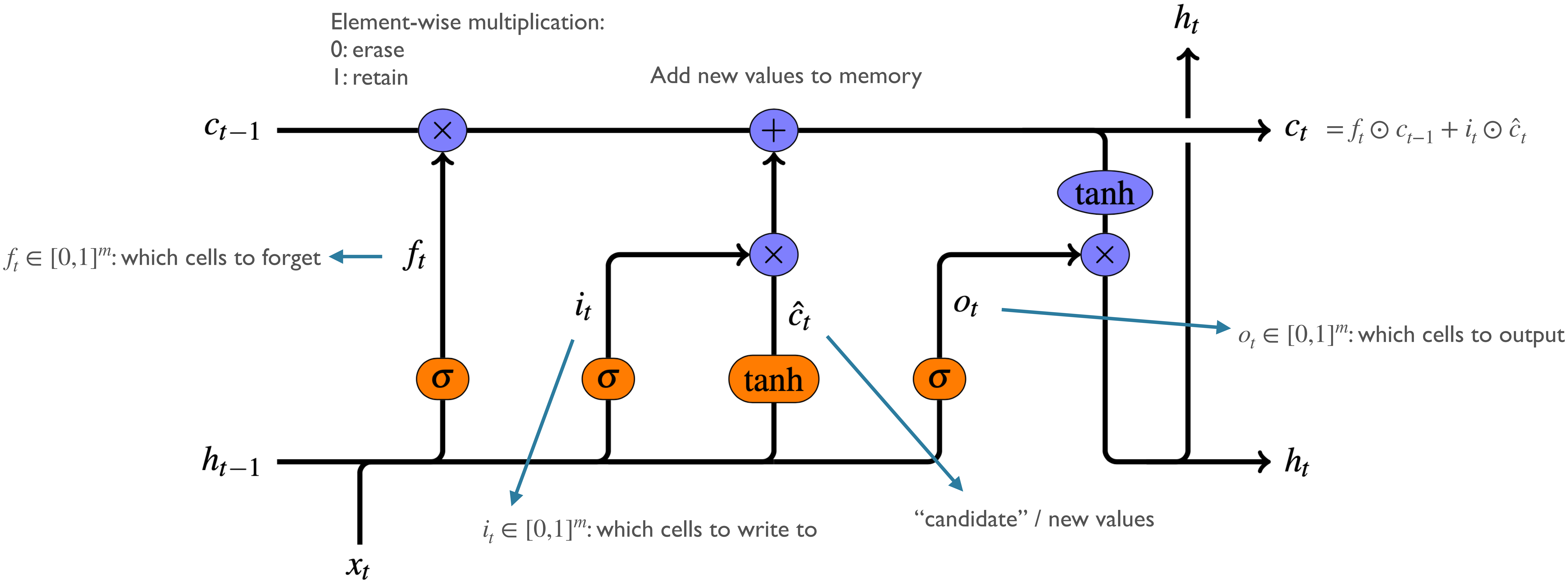


Element-wise multiplication:
0: erase
1: retain
Add new values to memory
h_t
c_{t-1}
c_t
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values
x_t

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
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

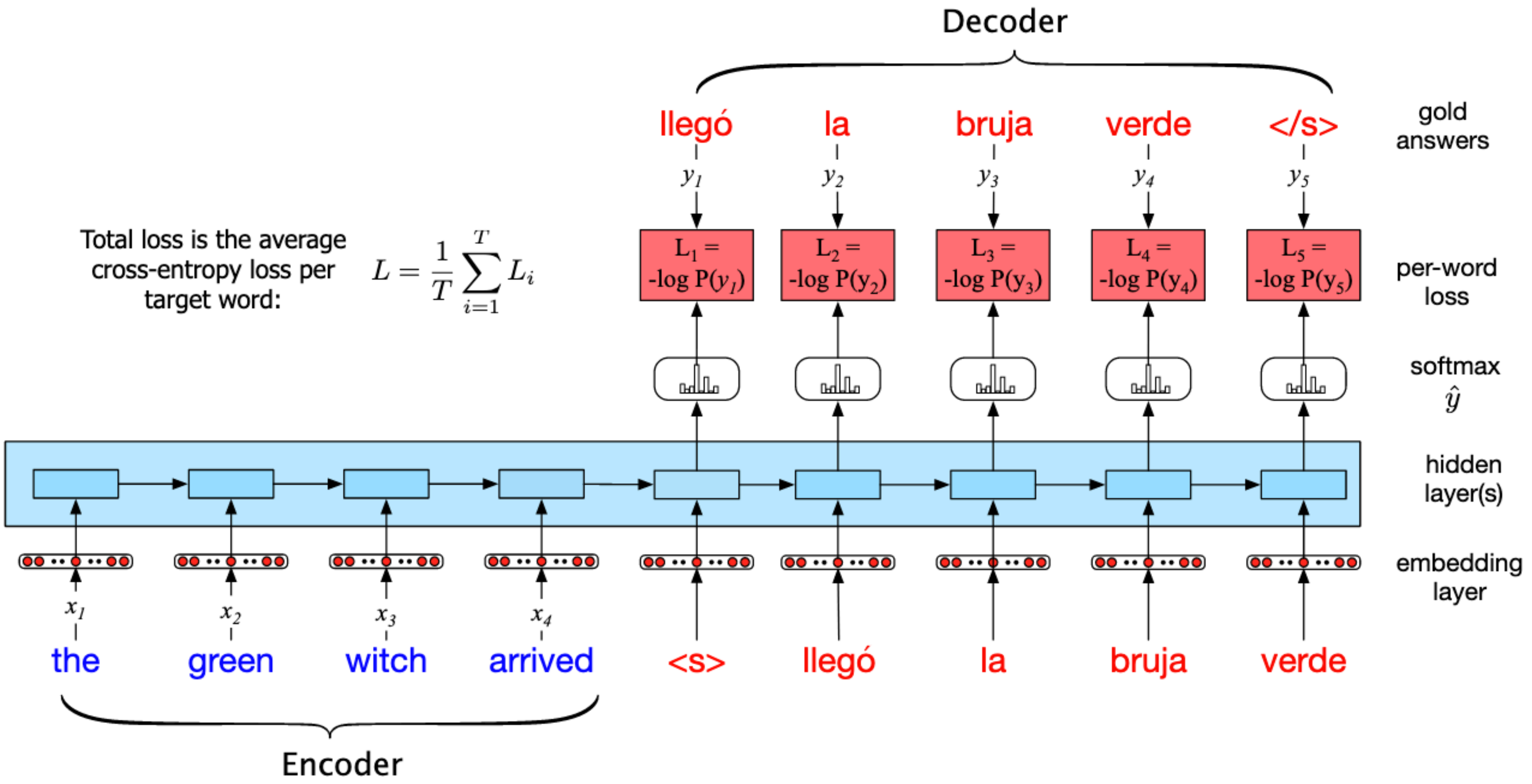
LSTMs



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
$o_t \in [0,1]^m$: 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: $L = \frac{1}{T}\sum_{i=1}^{T} L_i$



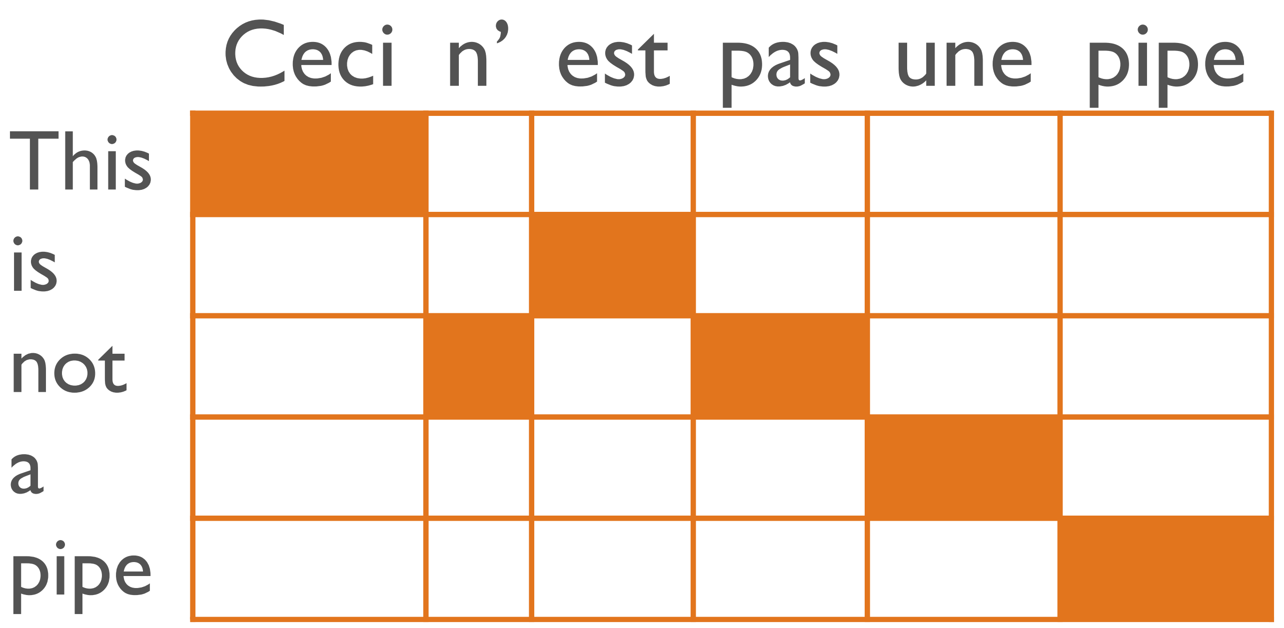
JM 11.3.1

Alignment, example

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

Alignment, example

Ceci n' est pas une pipe



Alignment, example

Ceci n' est pas une pipe

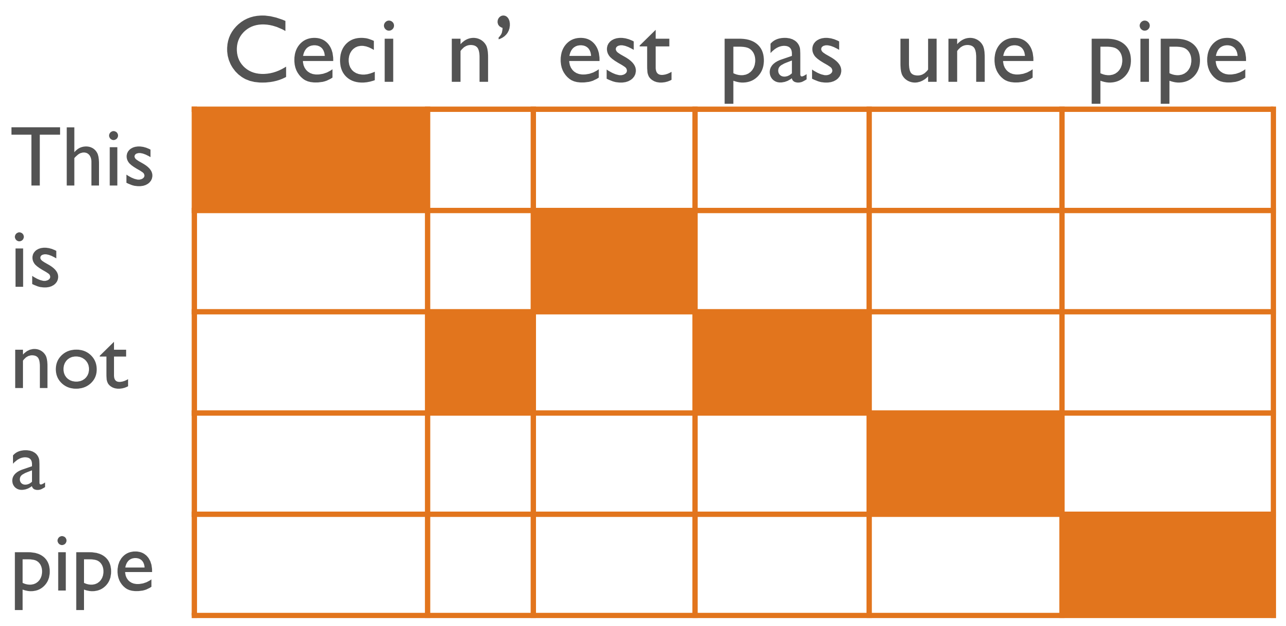
This is not a pipe

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

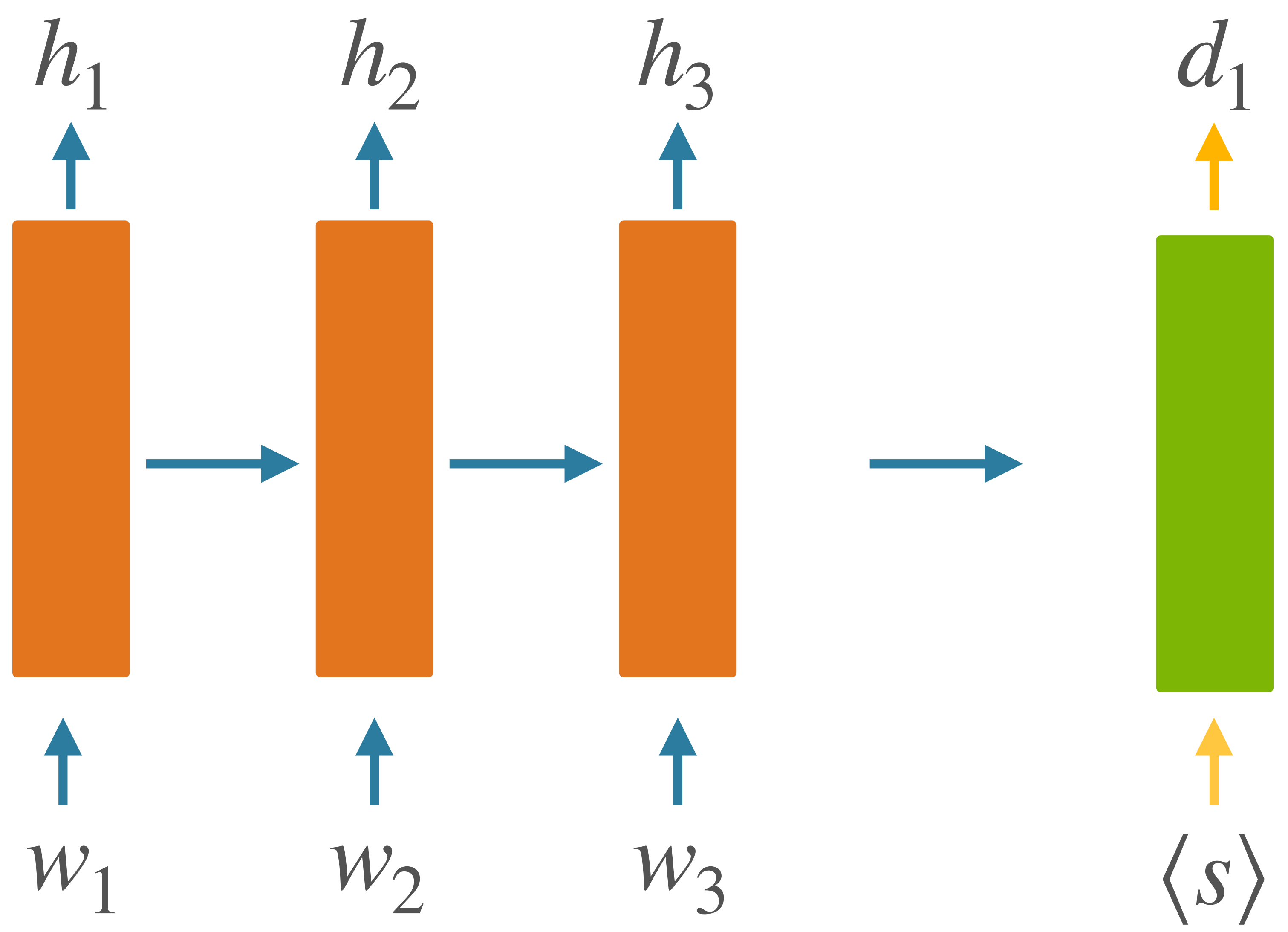
Alignment, example

Ceci n’ est pas une pipe

This is not a pipe

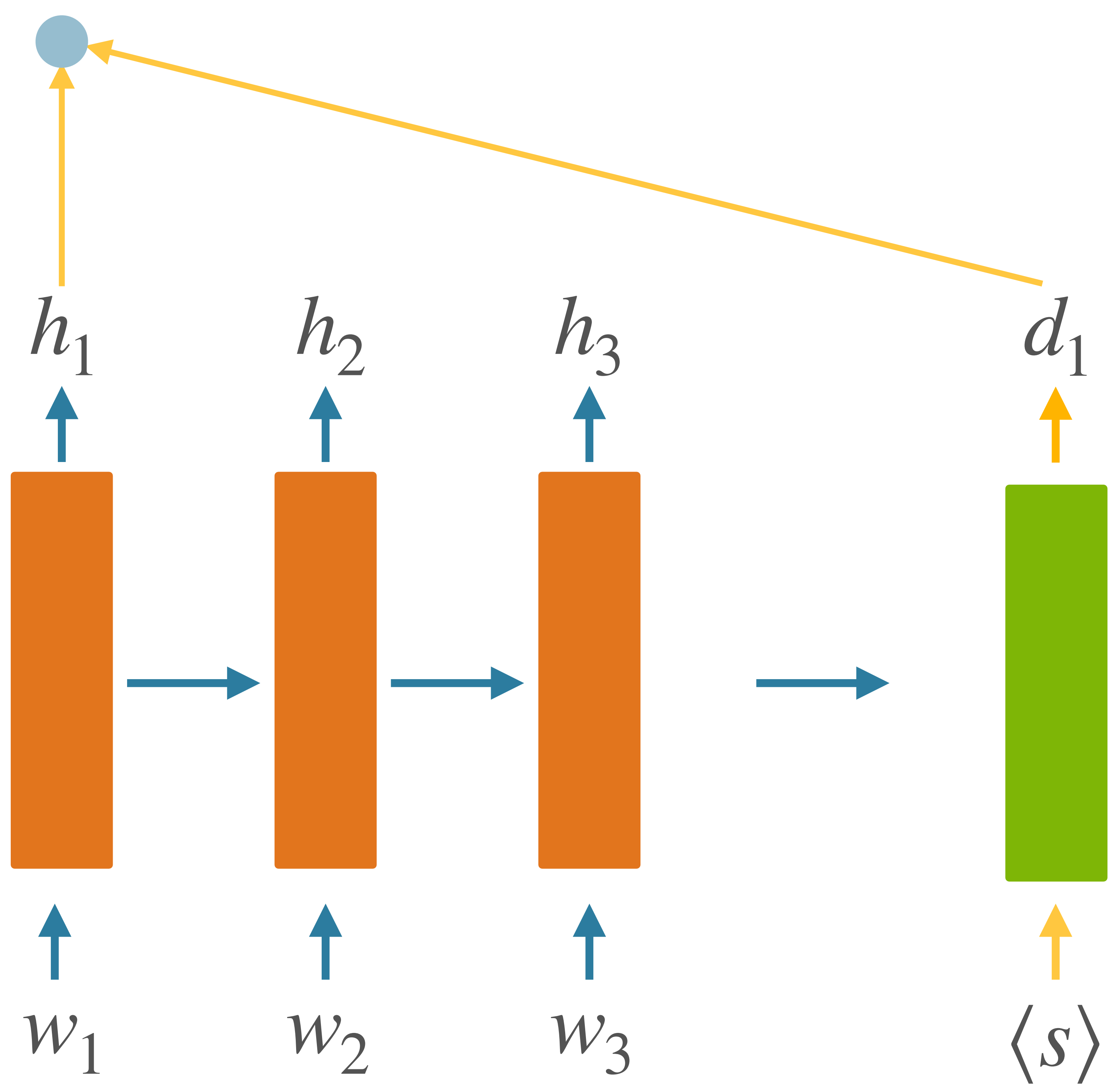


Adding Attention



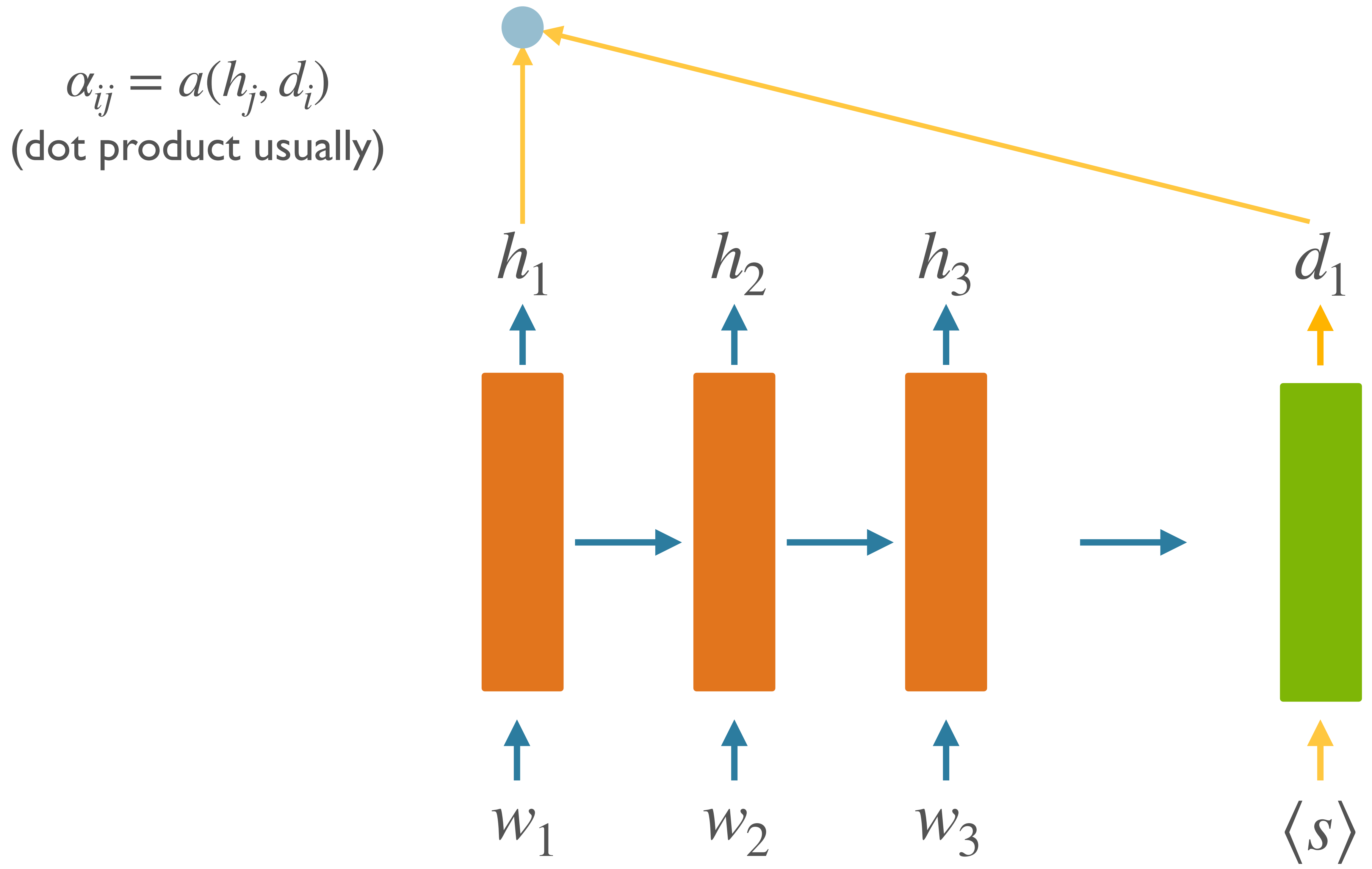
Badhanau et al 2014

Adding Attention



Badhanau et al 2014

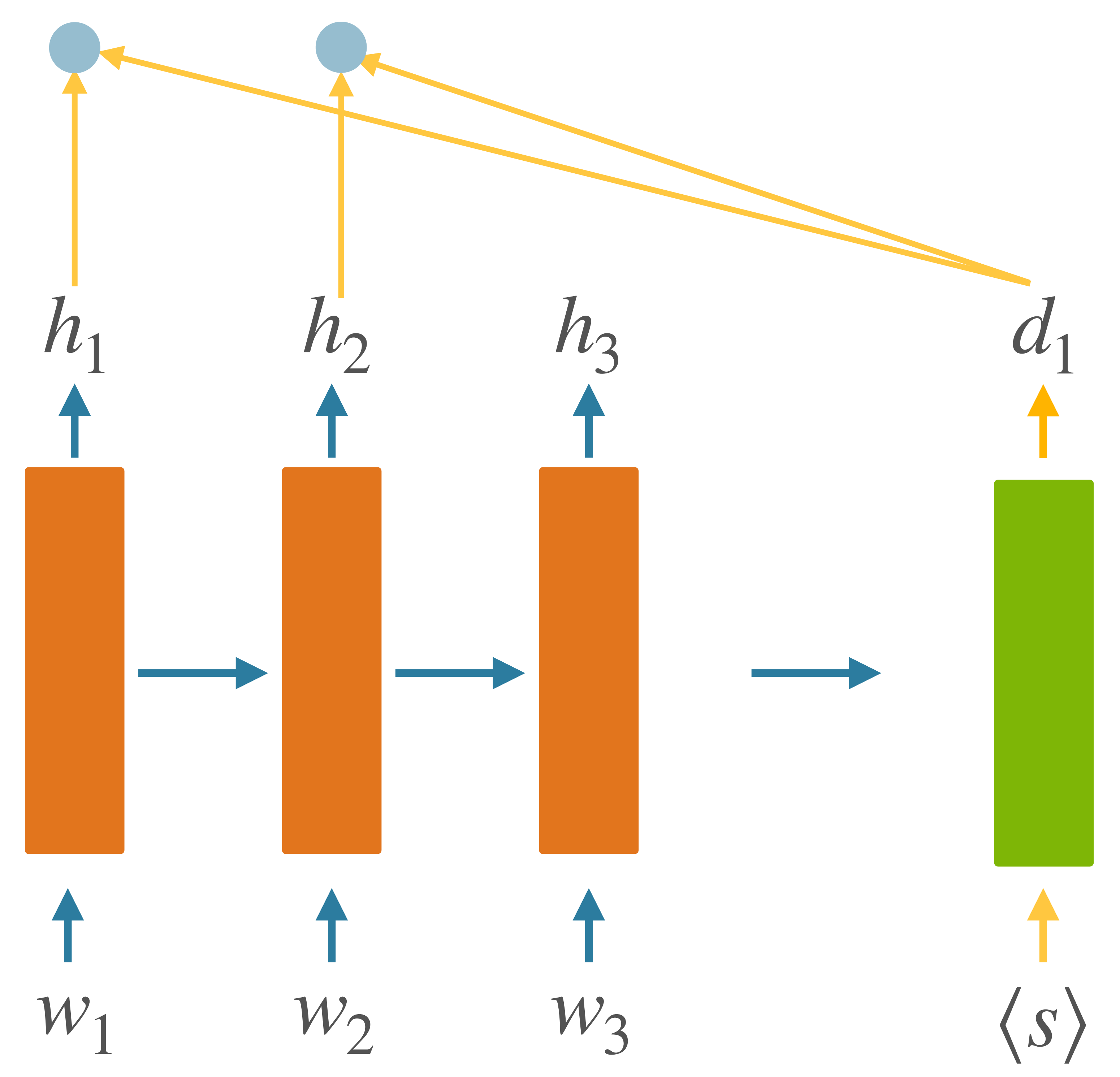
Adding Attention



Badhanau et al 2014

Adding Attention

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

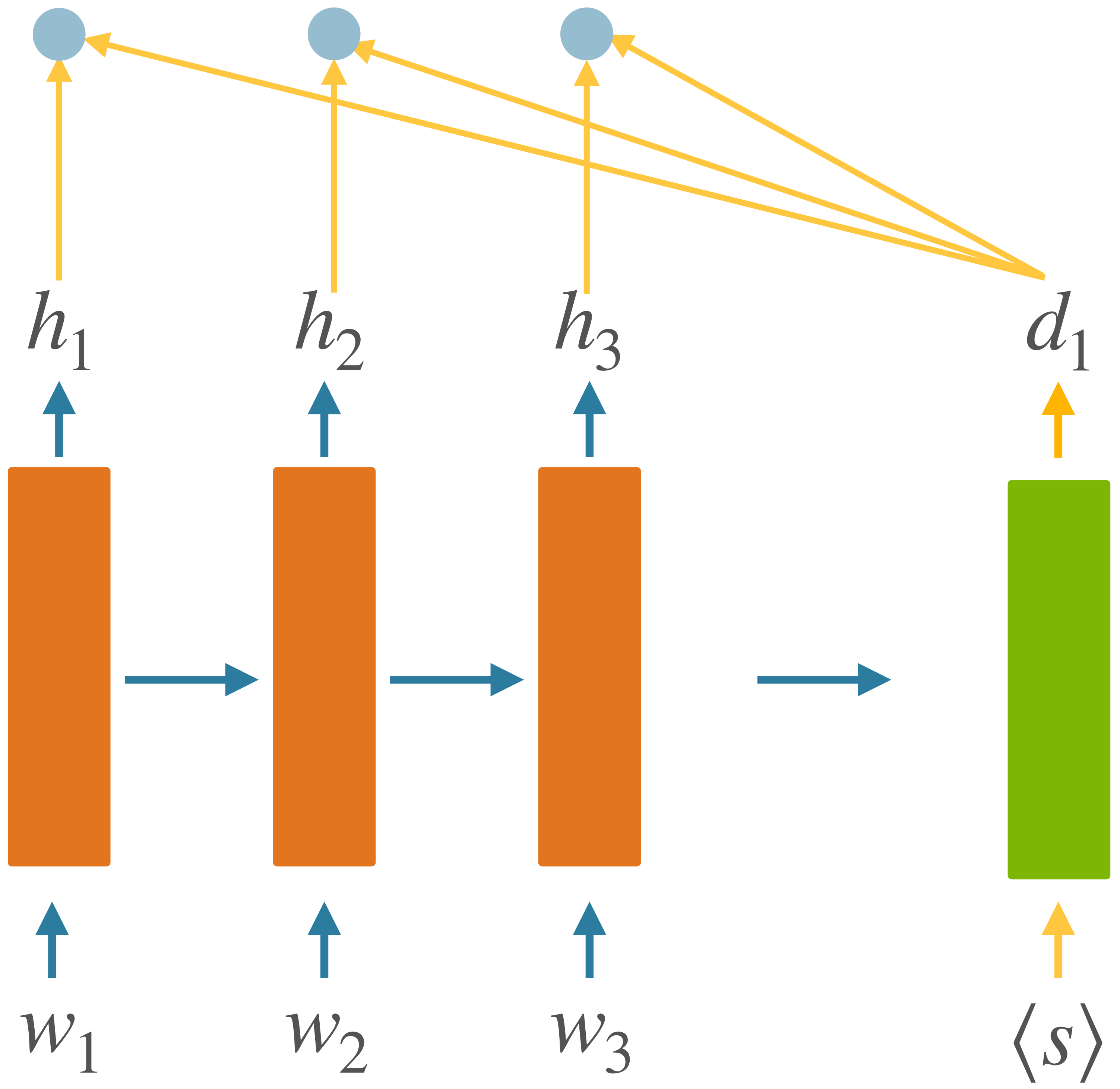


Badhanau et al 2014

Adding Attention

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

(dot product usually)

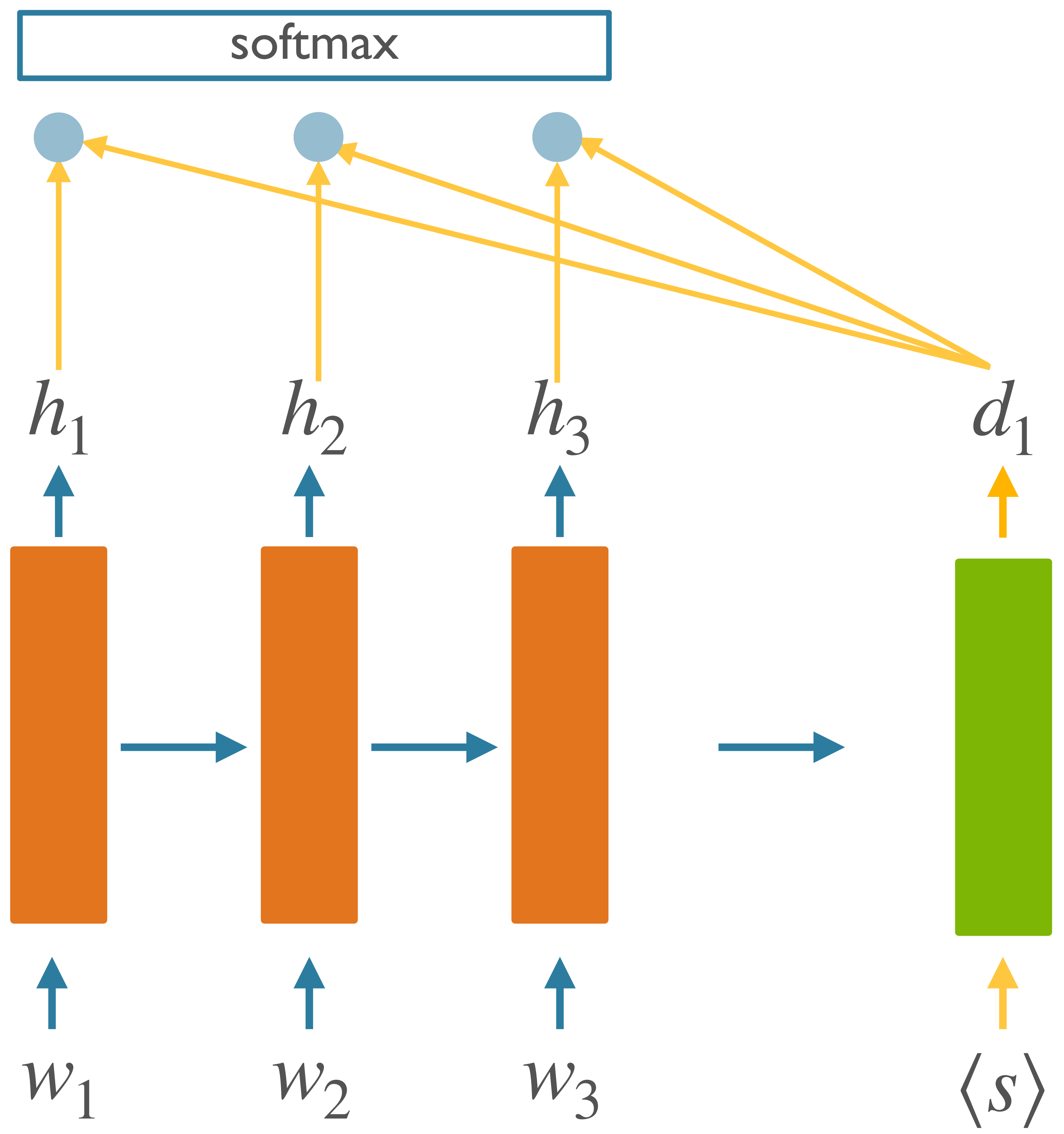


[Badhanau et al 2014]

Adding Attention

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

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



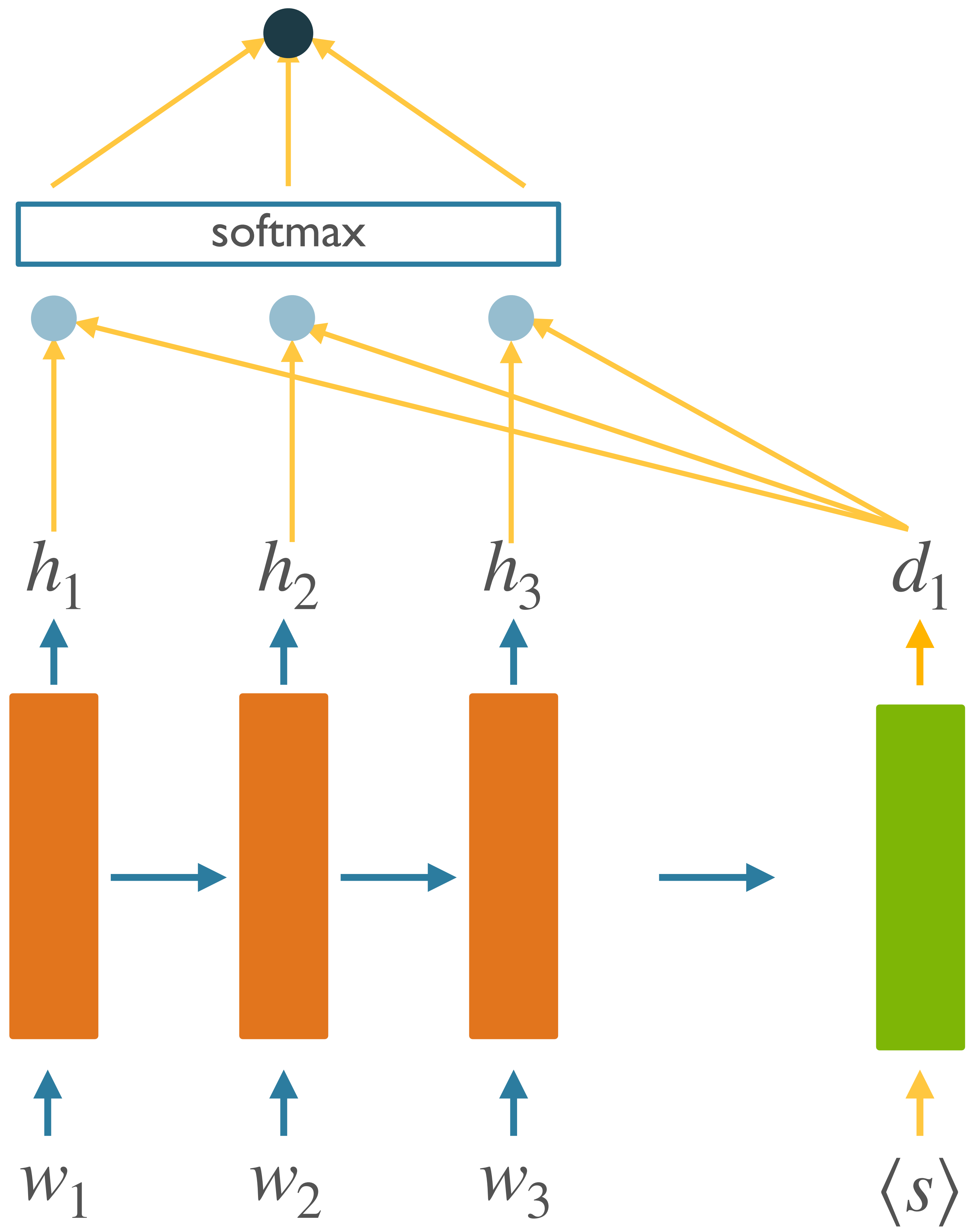
Badhanau et al 2014

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)



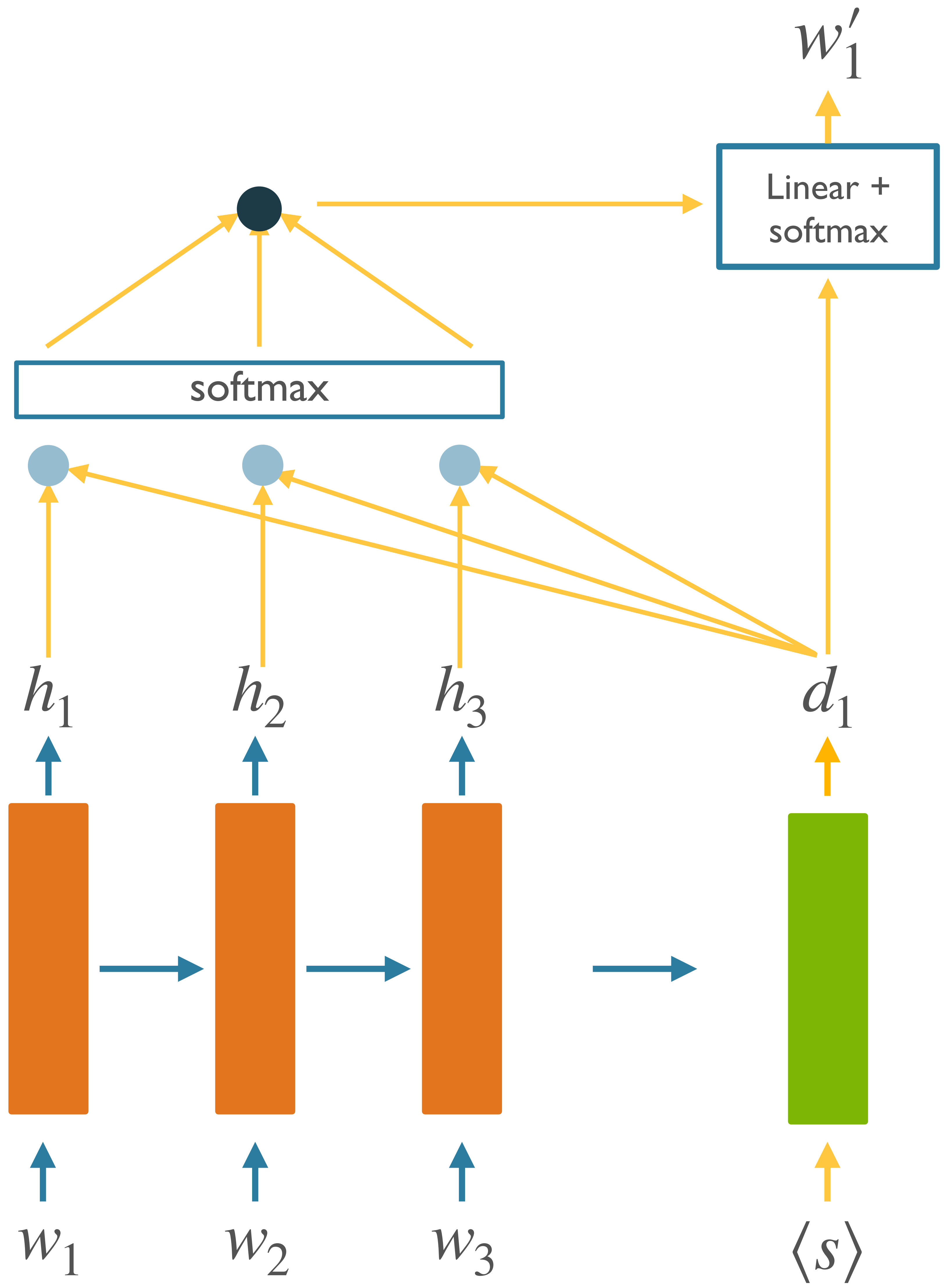
Badhanau et al 2014

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)



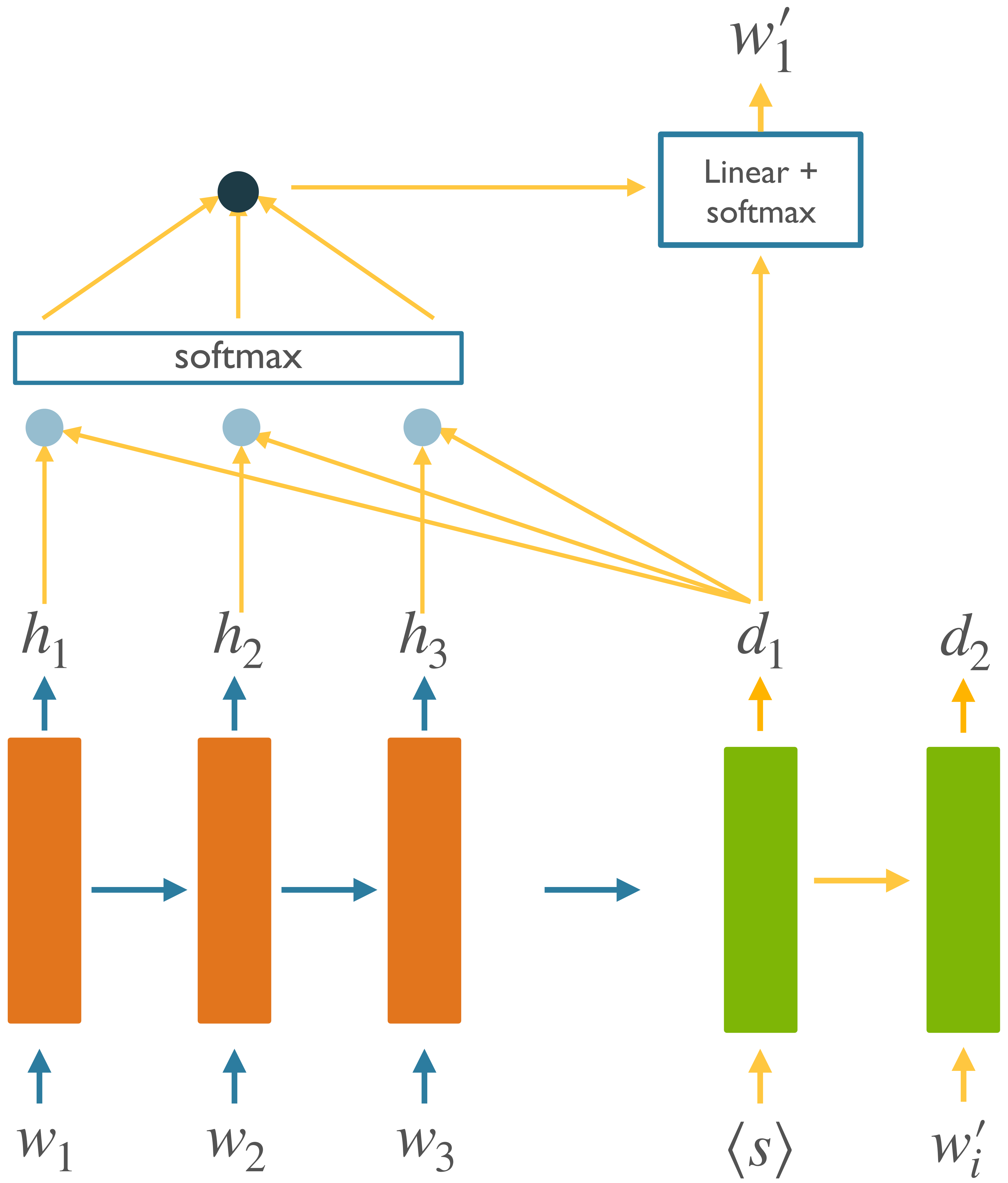
Badhanau et al 2014

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)



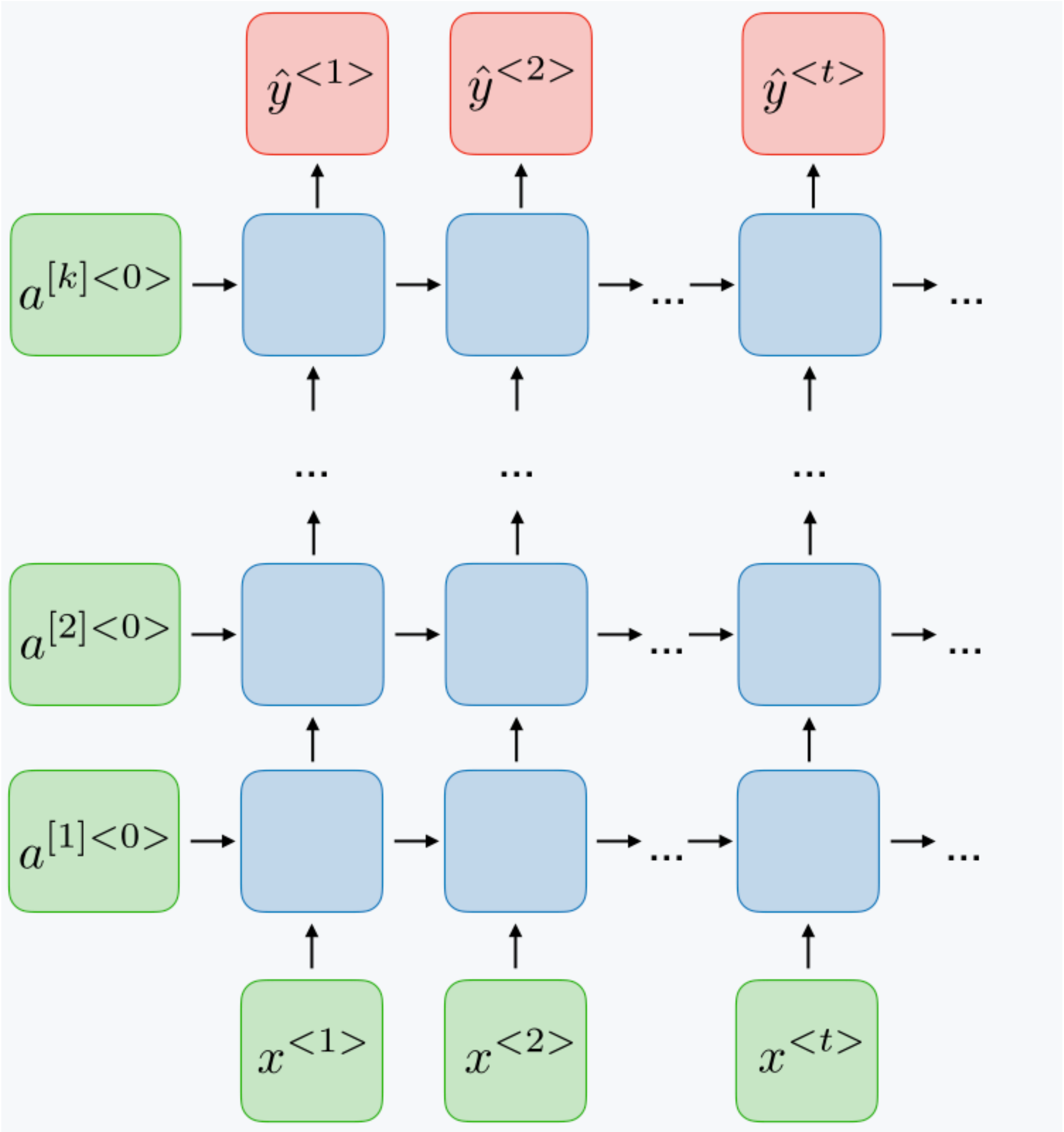
Badhanau et al 2014

Neural Networks, II

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

Lack of Parallelizability

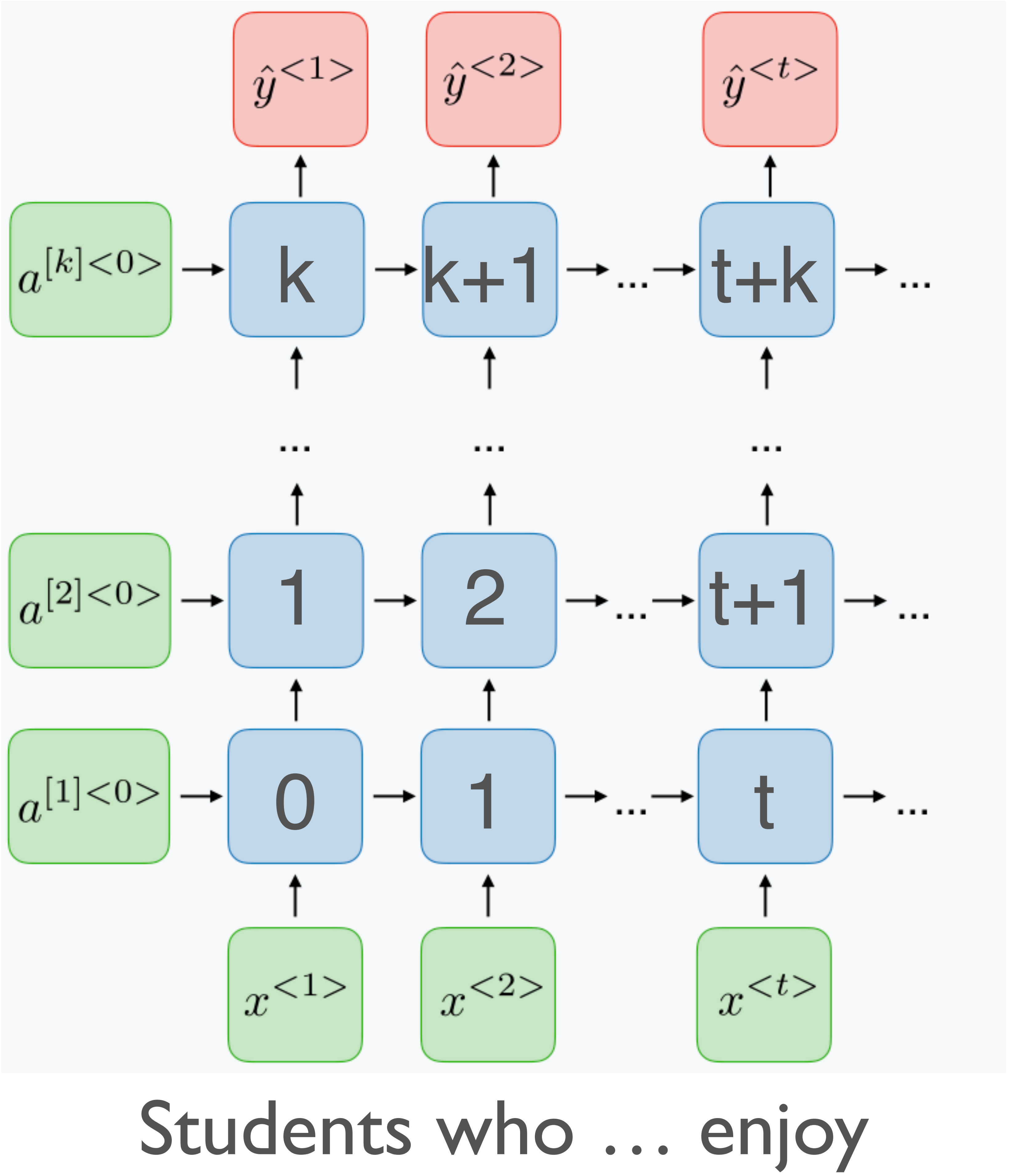
- Modern hardware (e.g. GPUs) are very good at doing *independent* computations in parallel
- RNNs are inherently serial:
  - Cannot compute future time steps without the past
- Bottleneck that makes scaling up difficult



Students who … enjoy

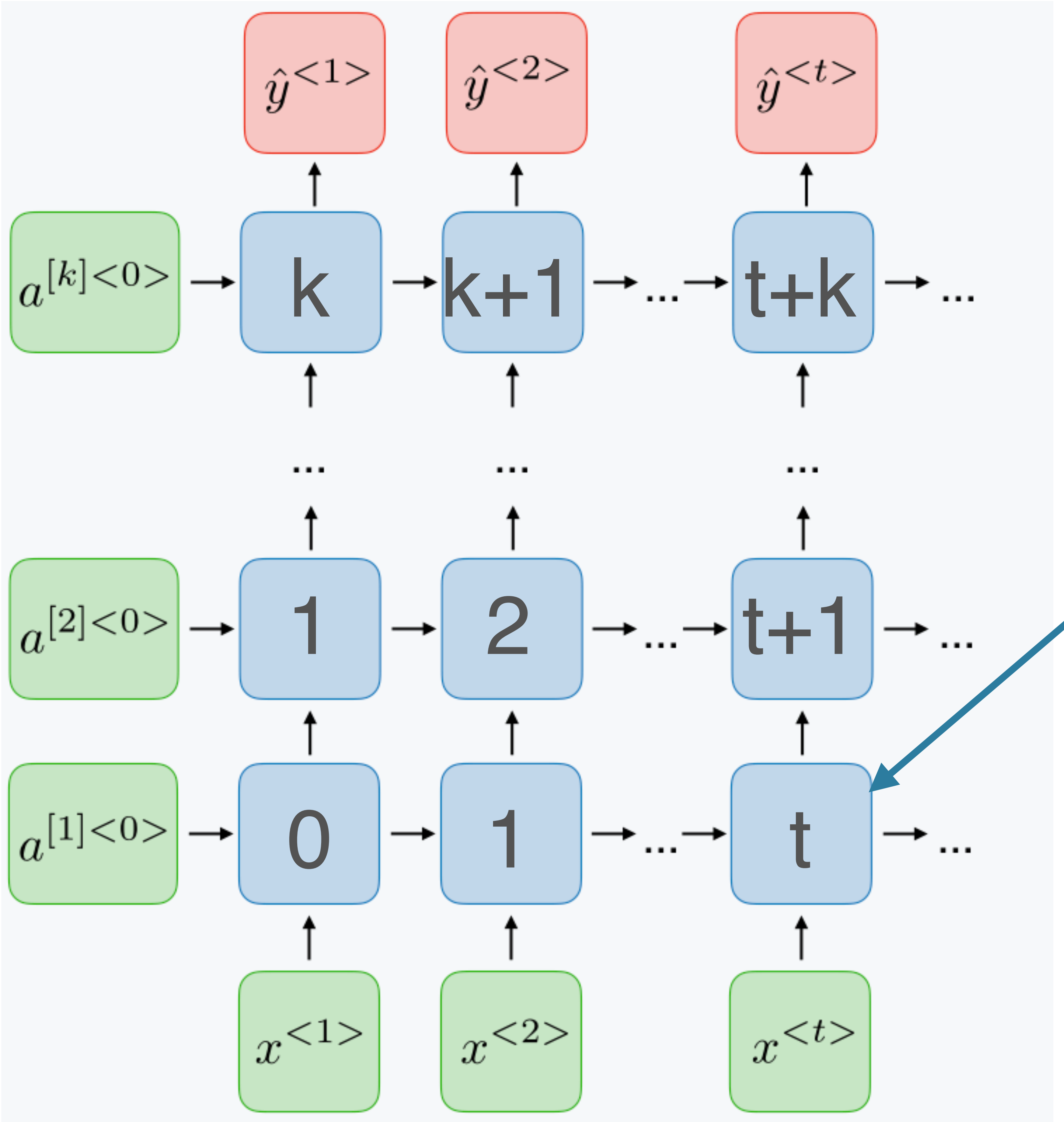
Lack of Parallelizability

- 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



Lack of Parallelizability

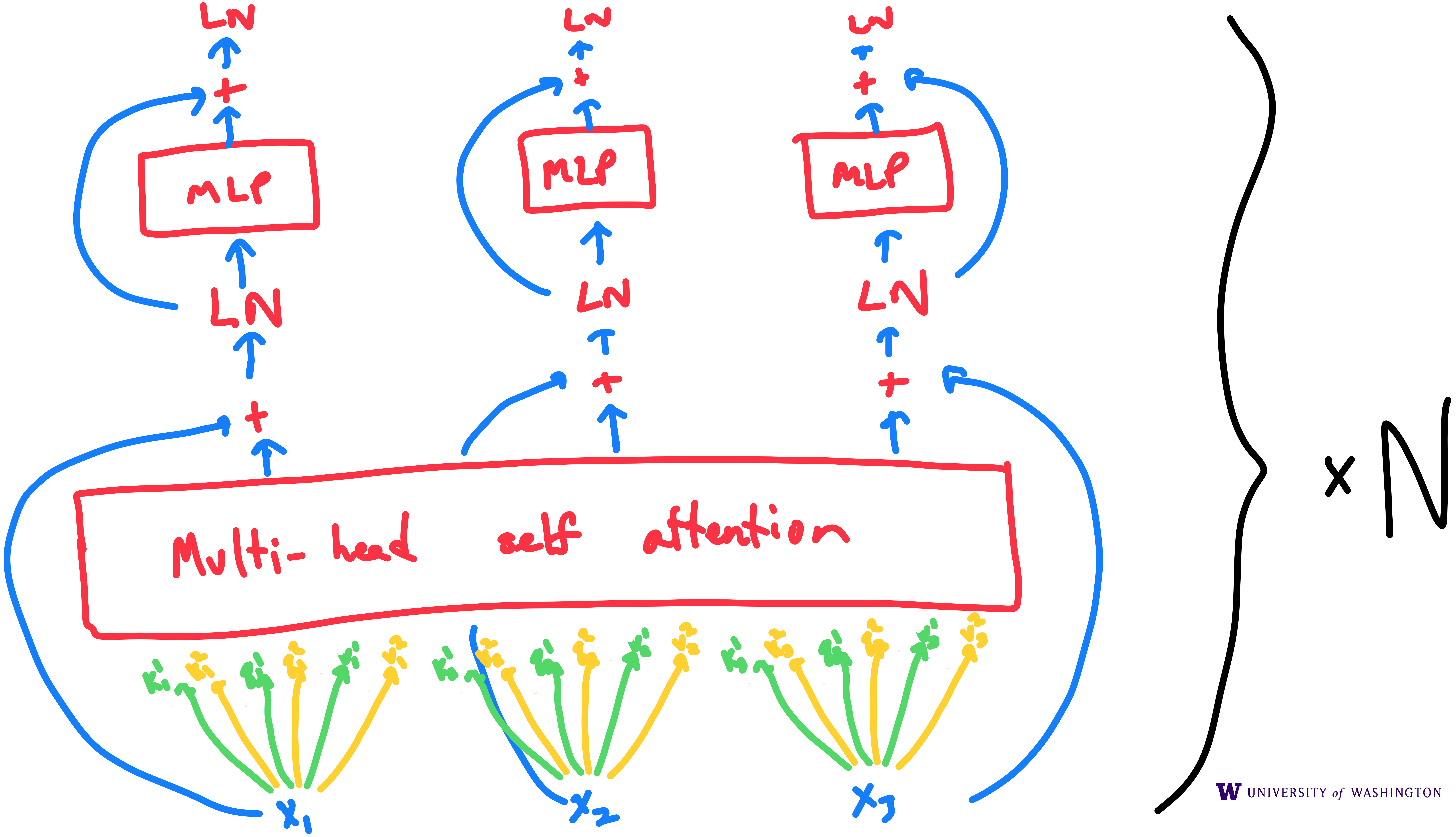
- Modern hardware (e.g. GPUs) are very good at doing *independent* computations in parallel
- RNNs are inherently serial:
  - Cannot compute future time steps without the past
- Bottleneck that makes scaling up difficult



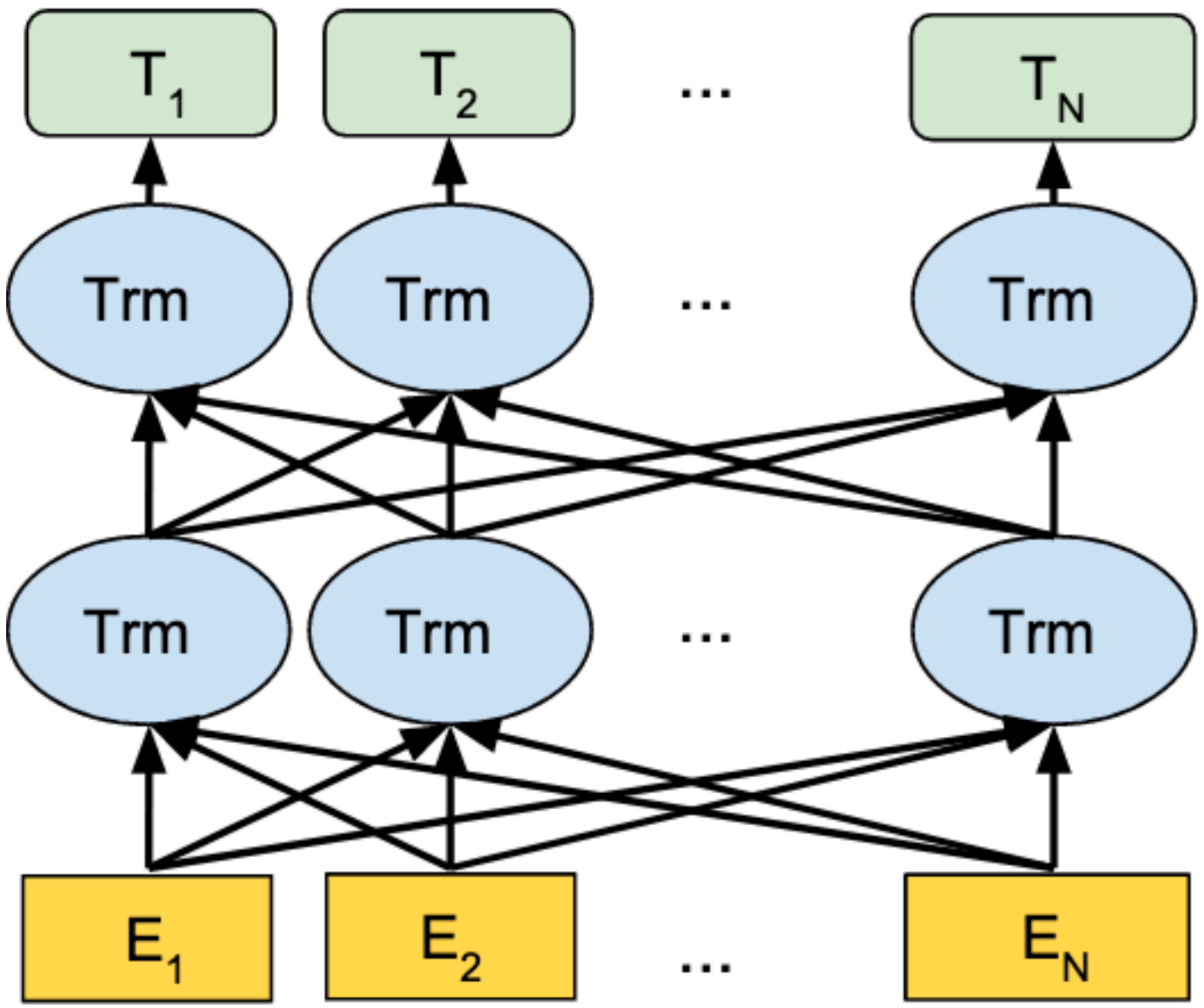
Students who … enjoy

Number of computation steps required: linear in sequence length

Full Transformer Encoder Block

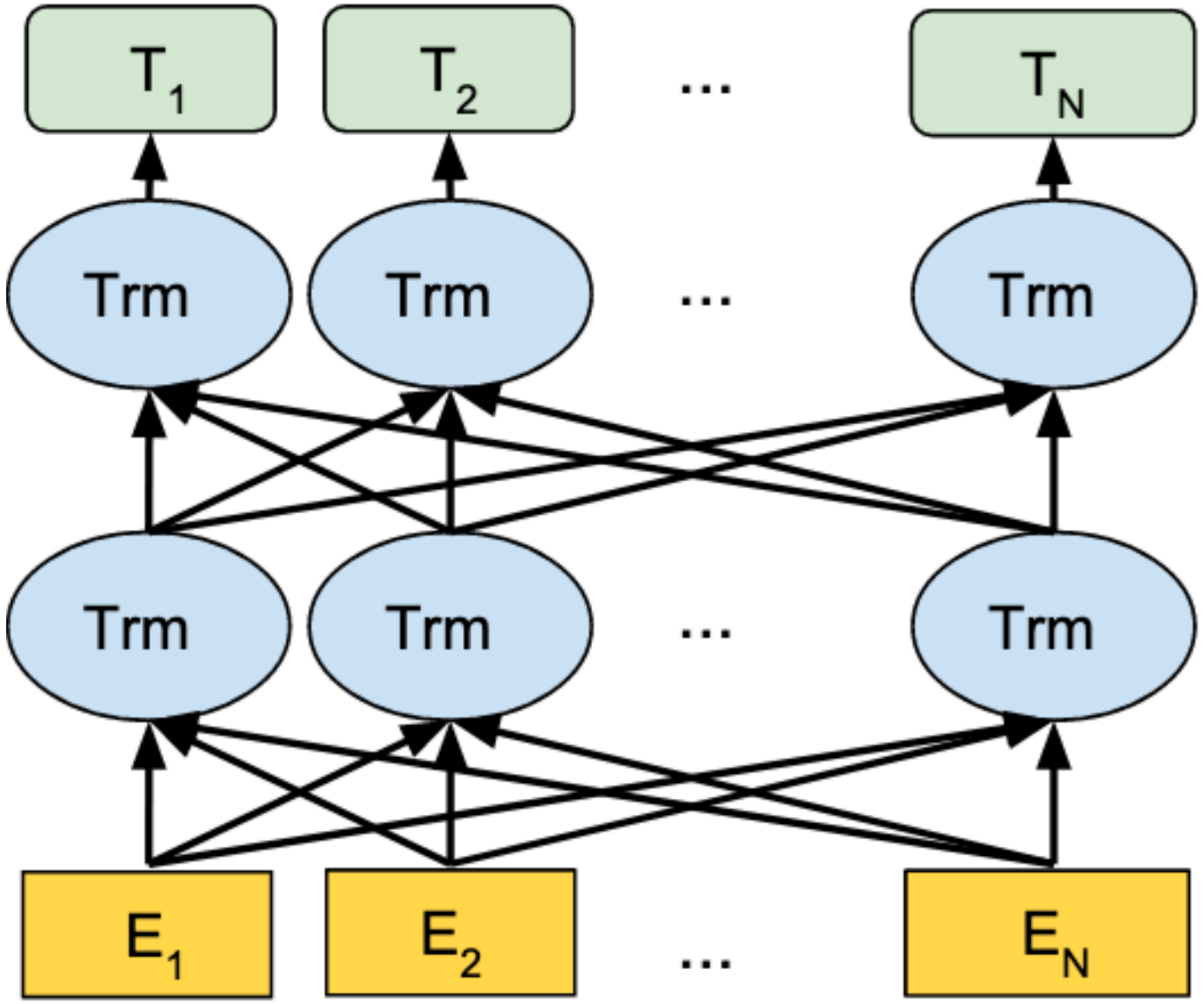


Transformer: Path Lengths + Parallelism



source (BERT paper)

Transformer: Path Lengths + Parallelism



source (BERT paper)

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

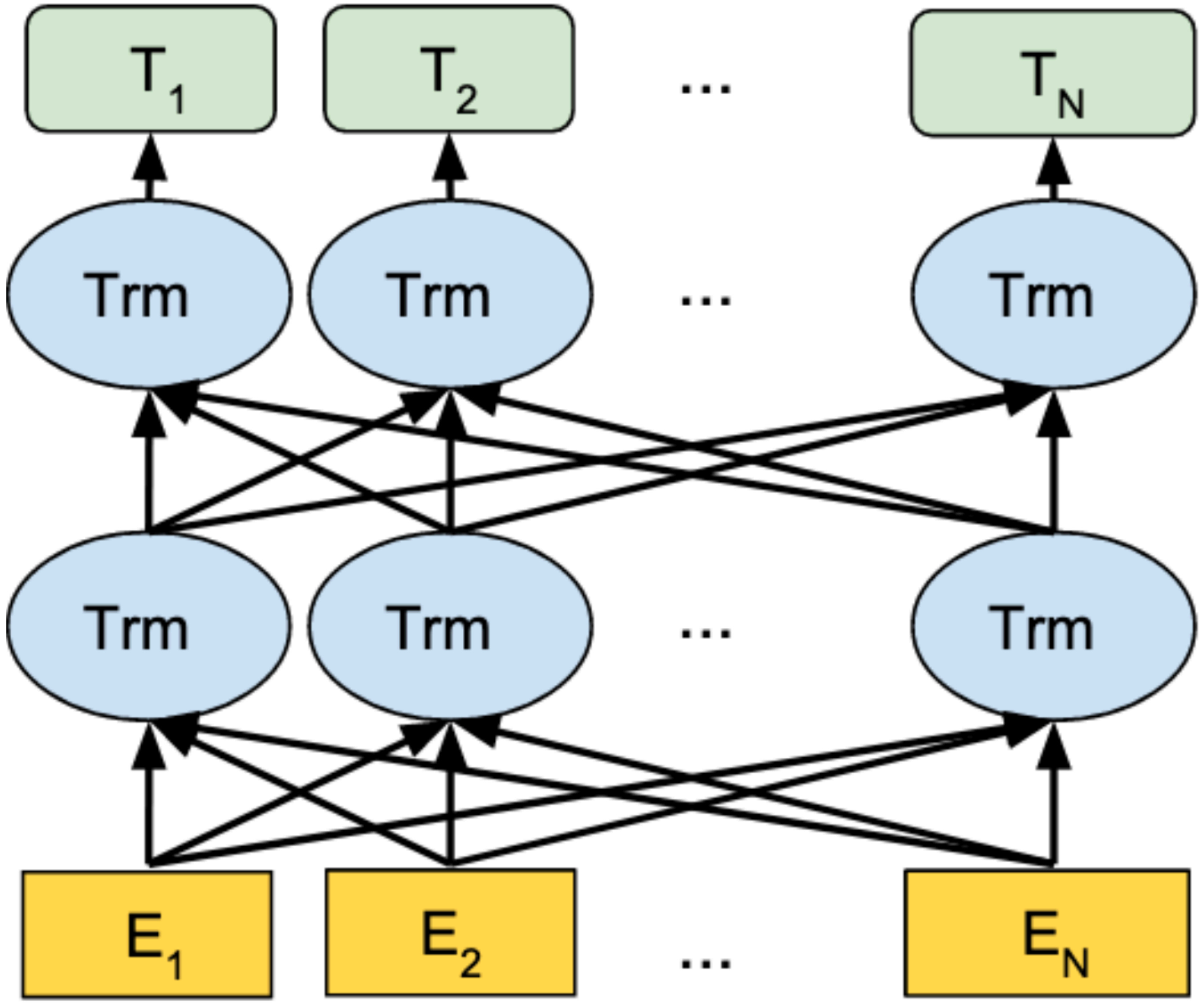
Transformer: Path Lengths + Parallelism

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

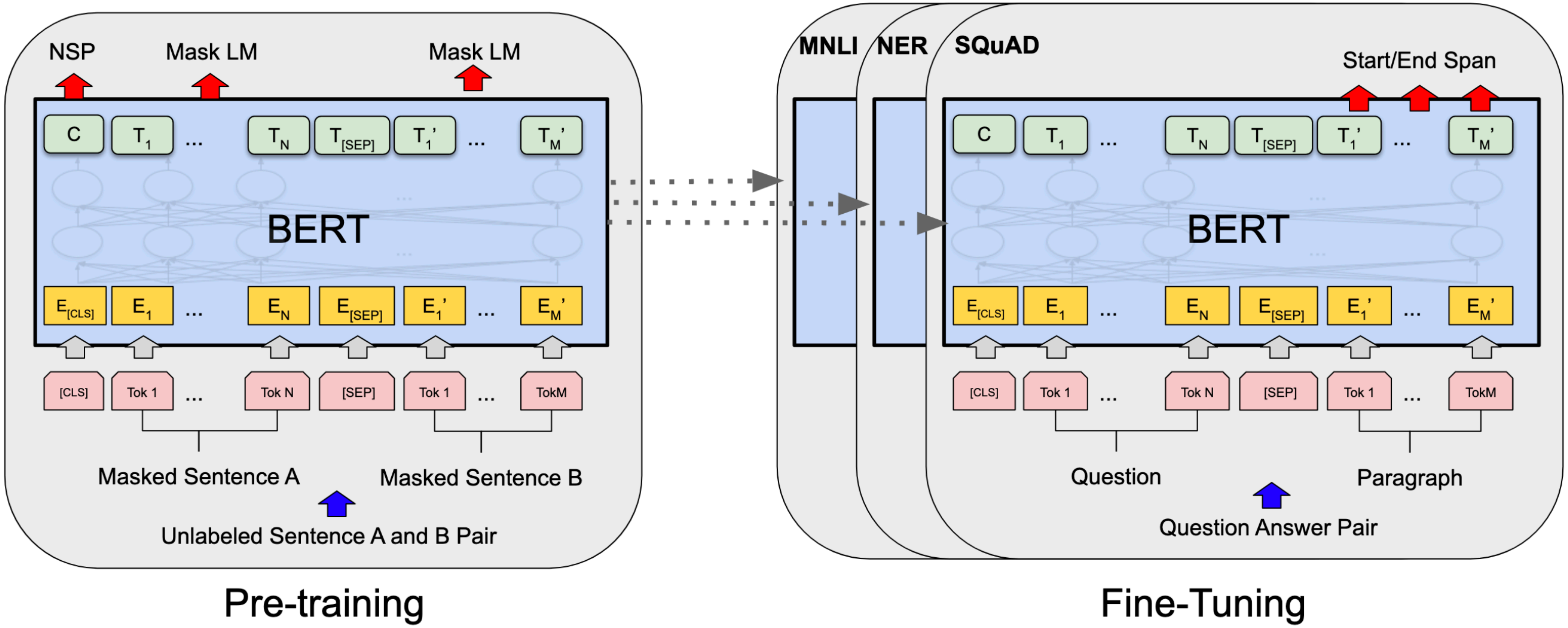
QK^T: total attention scores

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

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

	<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

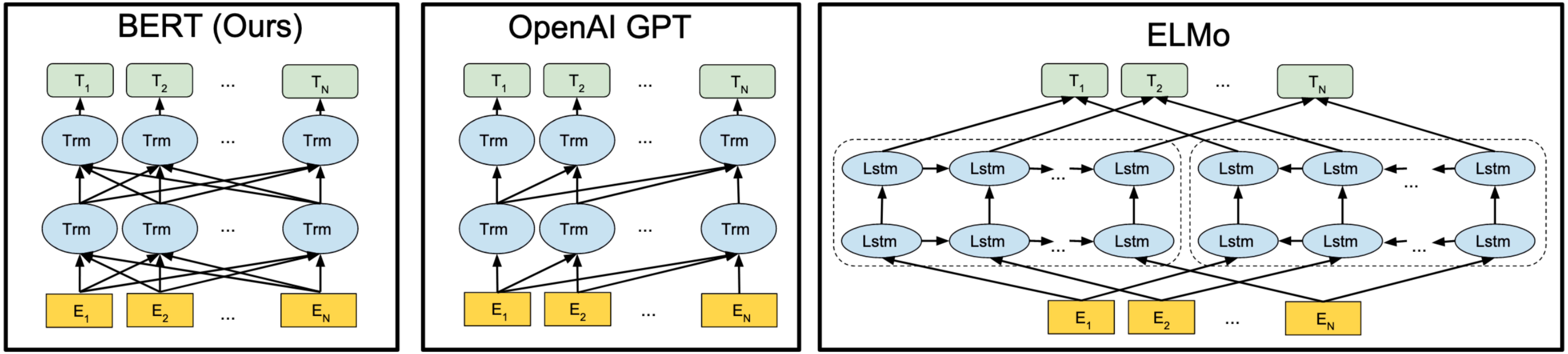
Schematically



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



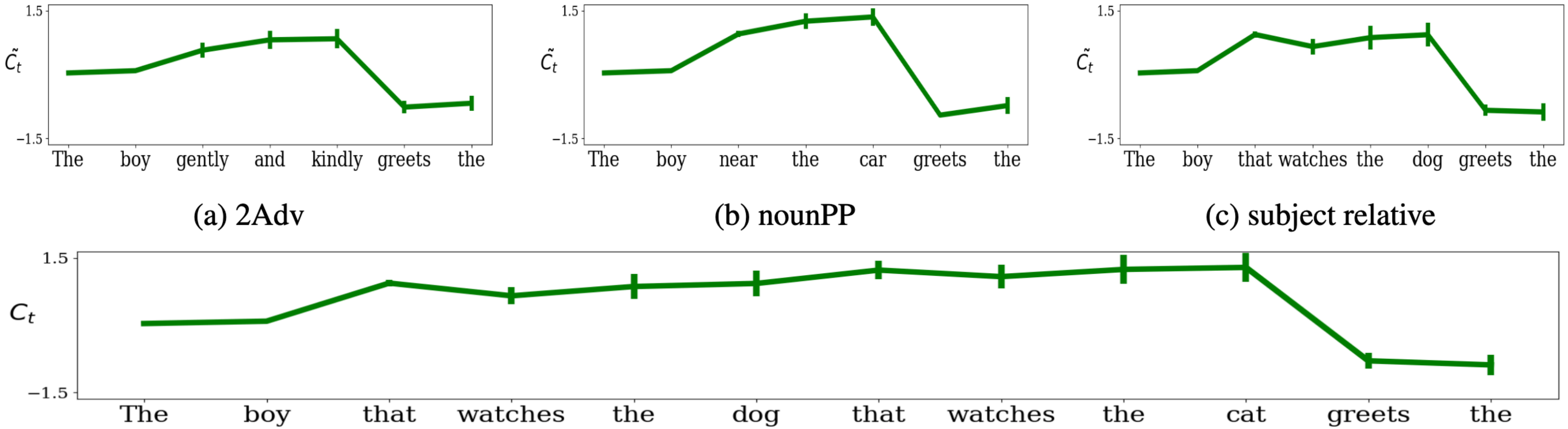
Source: BERT paper

Multilingual Pre-training

- One other main dimension: *mono-* vs *multi*-lingual pre-training
  - Roughly: concatenate (in fancy way) corpora from many languages, then do the same kind of pre-training
  - *[More in C.M. Downey guest lecture]*

	Encoder-only	Decoder-only	Encoder-decoder
English-only *	BERT, RoBERTa, XLNet, ALBERT, …	GPT-n	BART
Multilingual	mBERT, XLM(-R), …	BLOOM (HF BigScience), XGLM	mBART, MASS, mT5

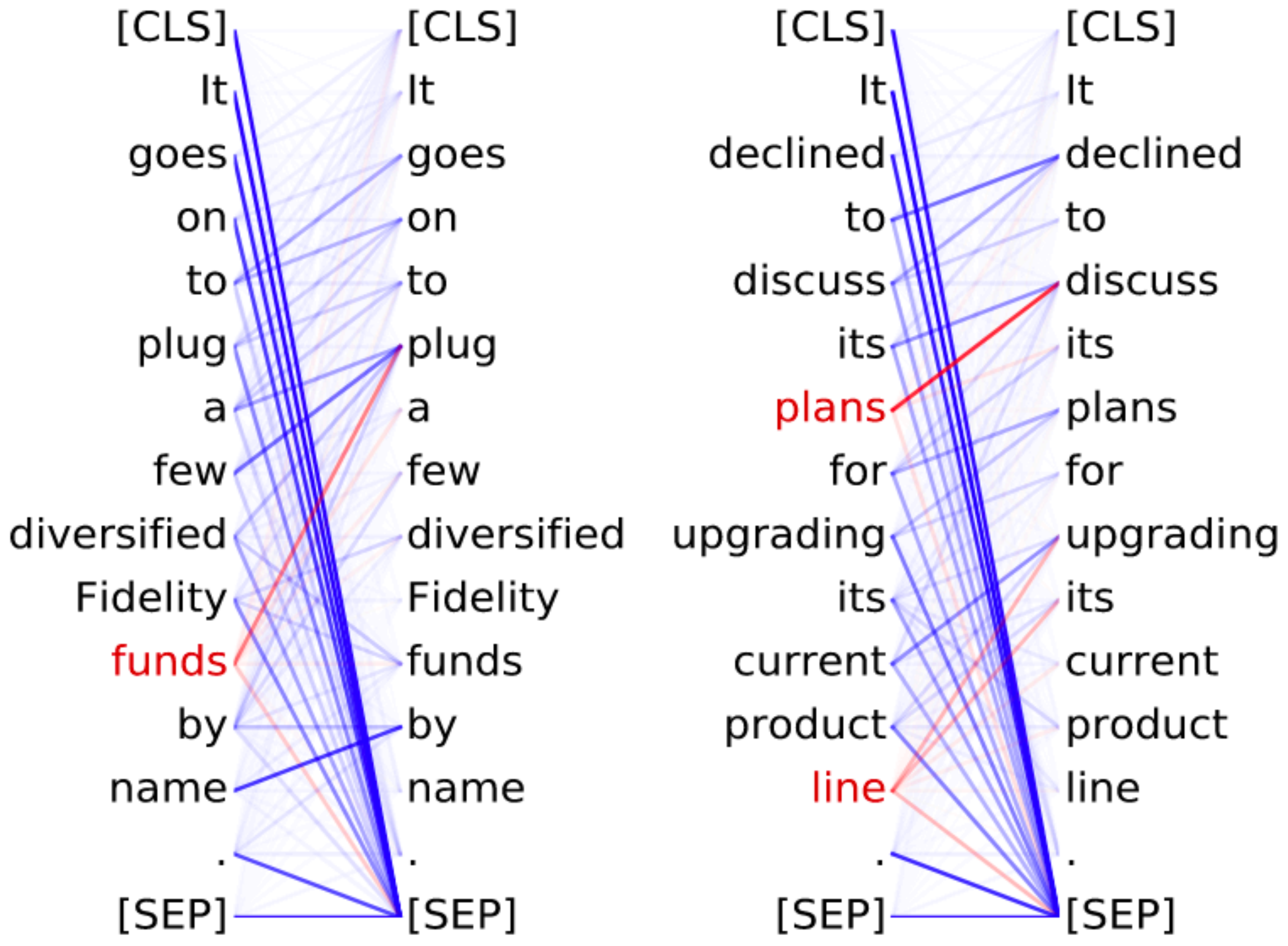
Cell dynamics for a syntax unit



Examples

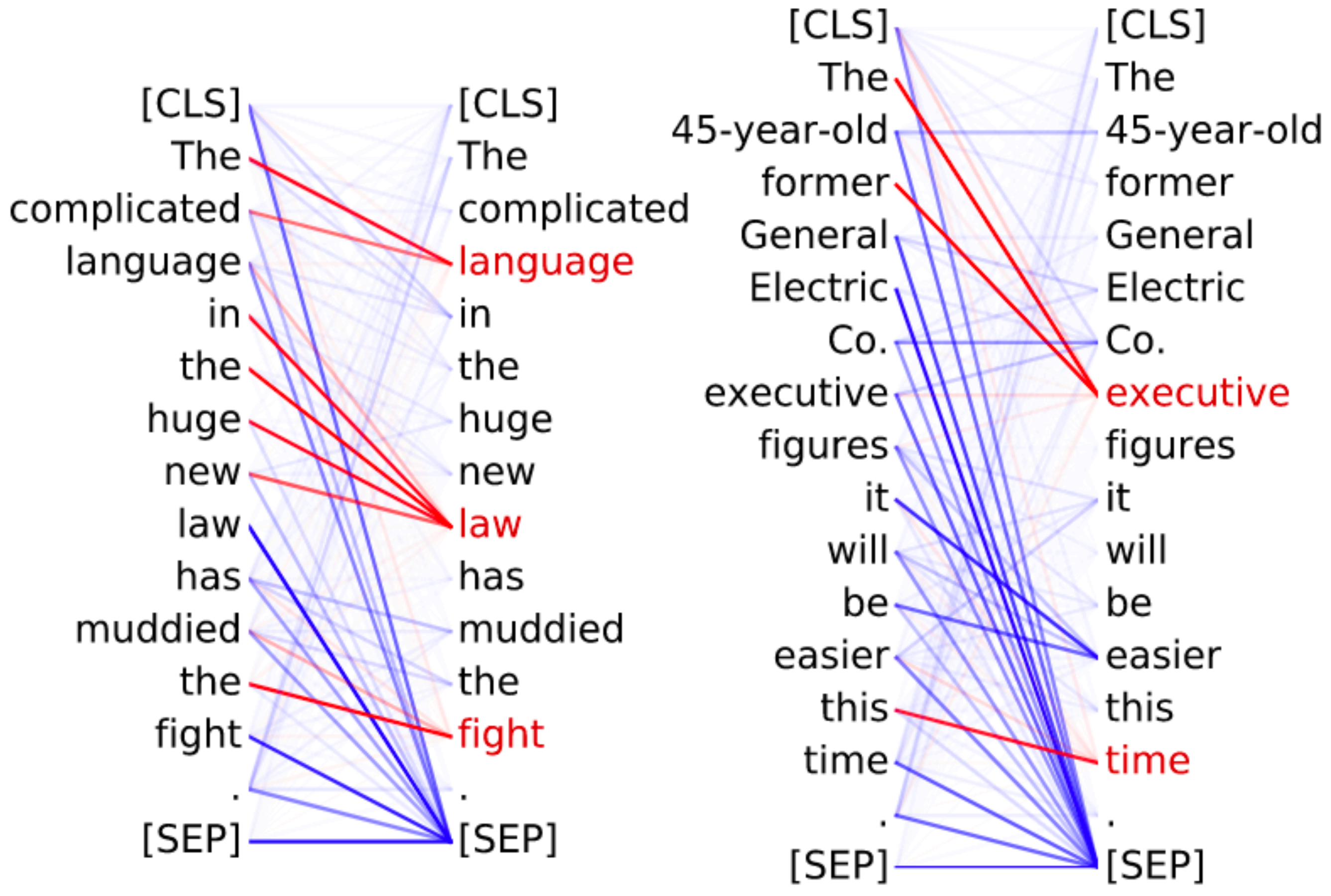
Head 8-10

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



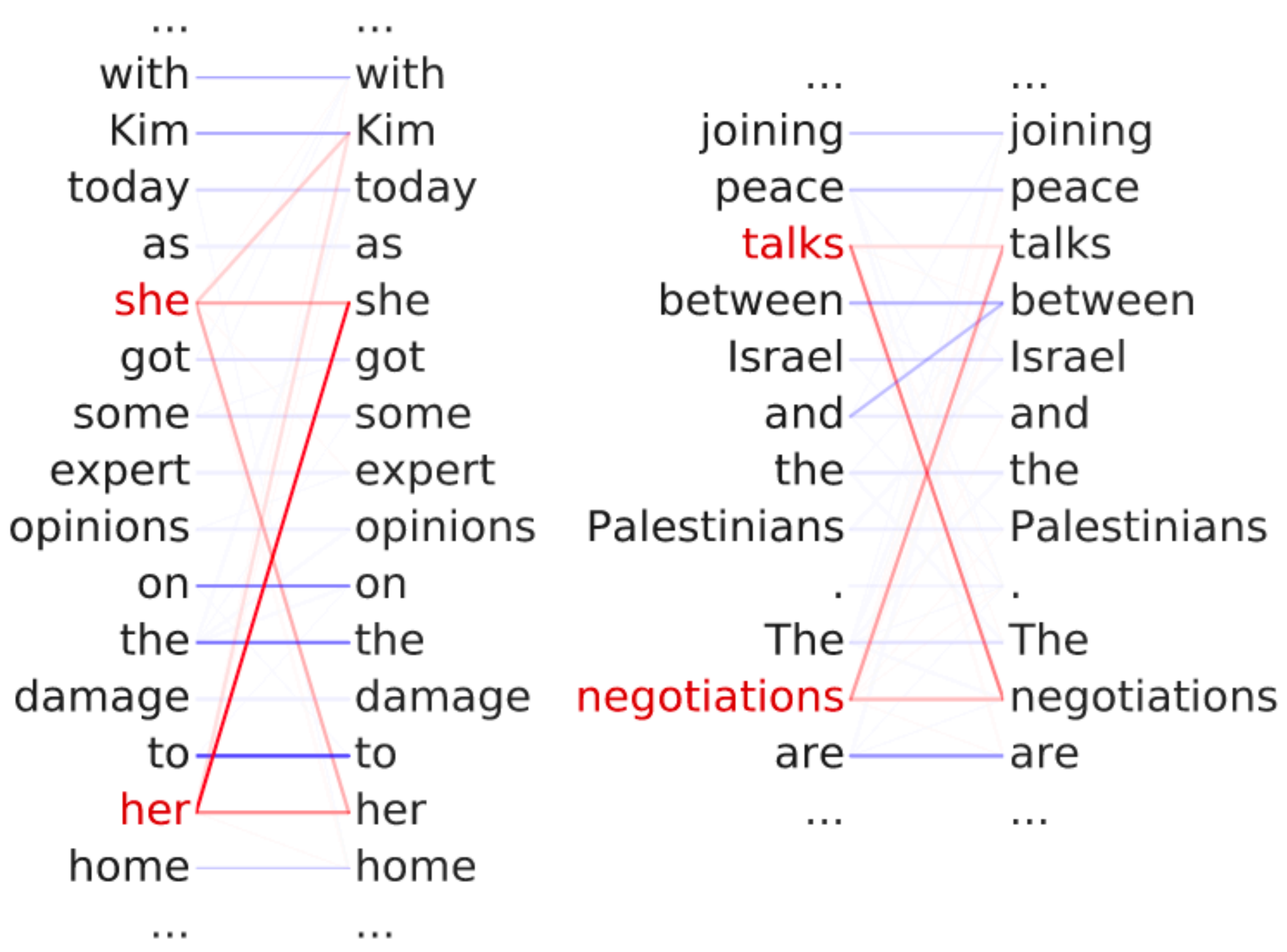
Head 8-11

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

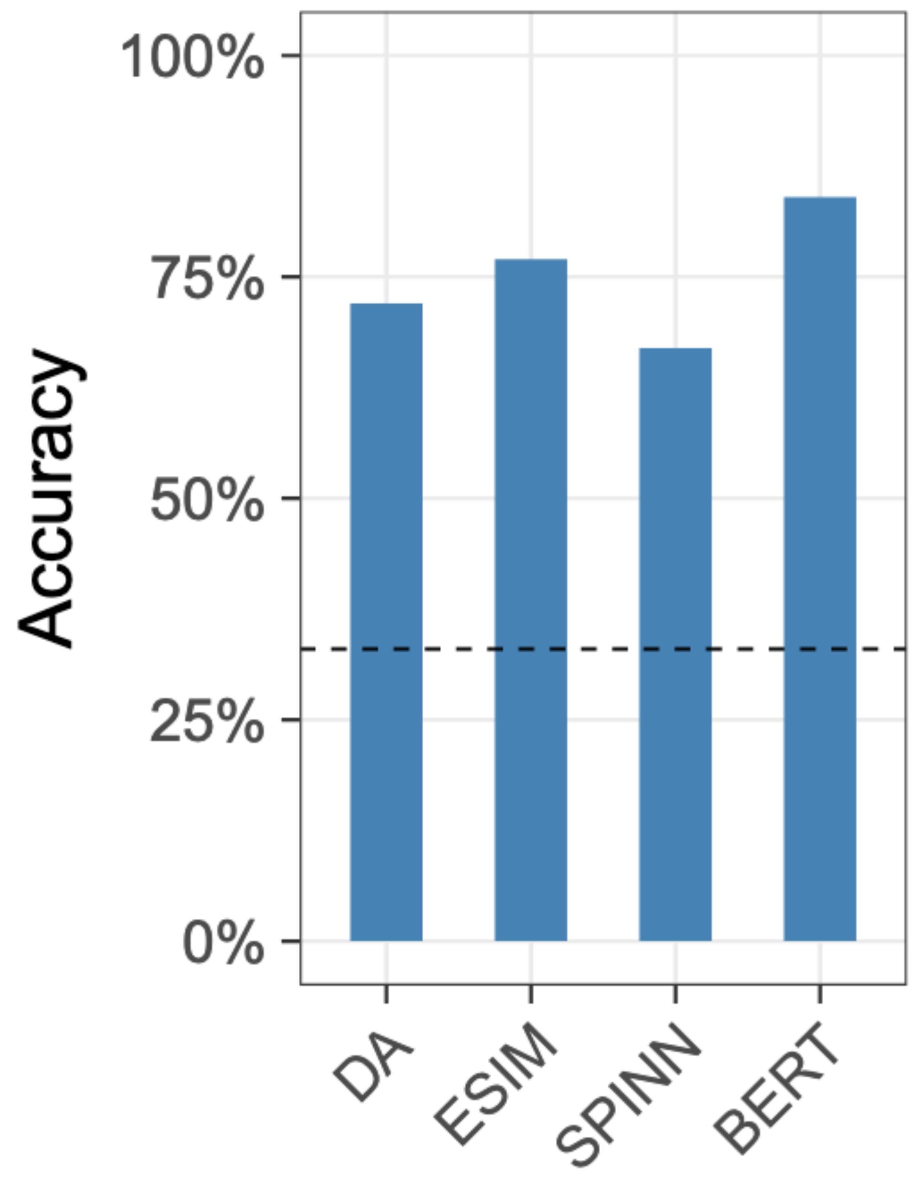


Head 5-4

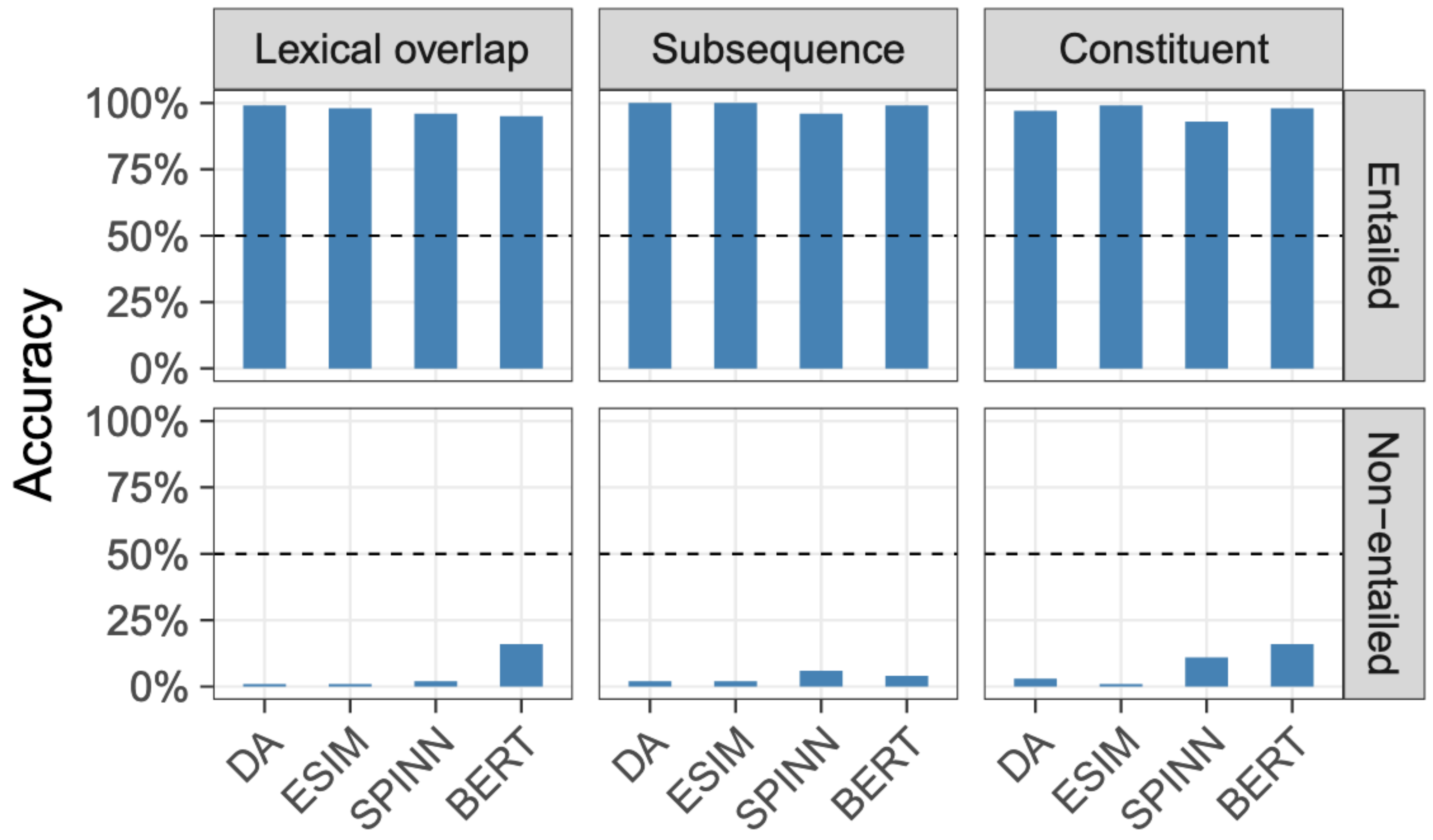
- **Coreferent** mentions attend to their antecedents
- 65.1% accuracy at linking the head of a coreferent mention to the head of an antecedent



Results



(a)

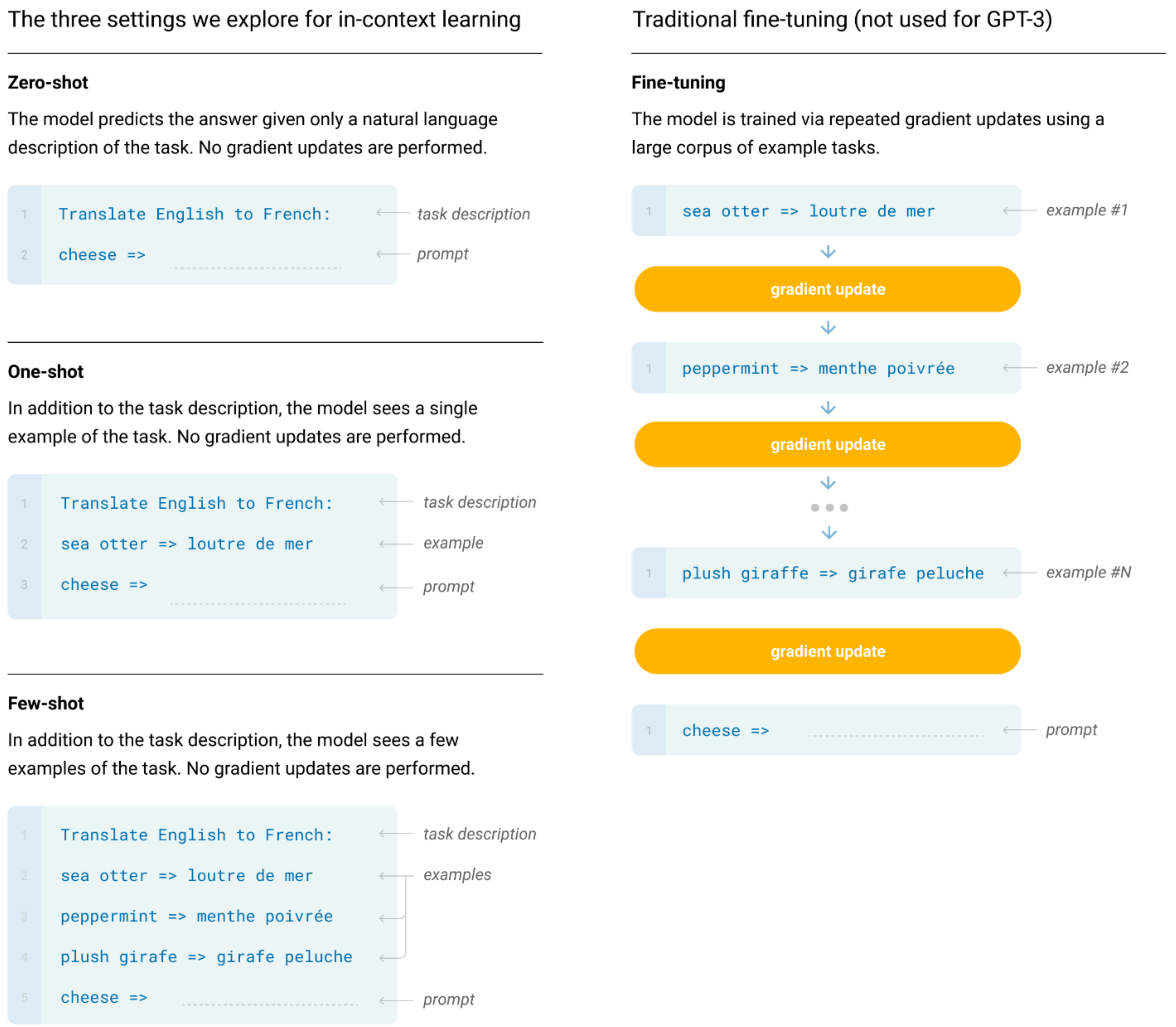


(b)

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

GPT3

- Same approach: pure Transformer decoder trained on LM
  - Scale: 175B params
  - Data size: ~500billion tokens, majority from filtered Common Crawl
- Few-shot "fine-tuning" paradigm:
  - Prompt with a few examples, ask to continue
  - *No parameter updates*



Some Mysteries

Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?

Sewon Min[1,2] **Xinxi Lyu**[1] **Ari Holtzman**[1] **Mikel Artetxe**[2]
Mike Lewis[2] **Hannaneh Hajishirzi**[1,3] **Luke Zettlemoyer**[1,2]
[1]University of Washington [2]Meta AI [3]Allen Institute for AI
{sewon,alrope,ahai,hannaneh,lsz}@cs.washington.edu
{artetxe,mikelewis}@meta.com

Abstract

Large language models (LMs) are able to in-context learn—perform a new task via inference alone by conditioning on a few input-label pairs (demonstrations) and making predictions for new inputs. However, there has been little understanding of *how* the model learns and *which* aspects of the demonstrations contribute to end task performance. In this paper, we show that ground truth demonstrations are in fact not required—randomly replacing labels in the demonstrations barely hurts performance on a range of classification and multi-choce tasks, consistently over 12 different models including GPT-3. Instead, we find that other aspects of the demonstrations are the key drivers of end task performance, including the fact that they provide a few examples of (1) the label space, (2) the distribution of the input text, and (3) the overall format of the sequence. Together, our analysis provides a new way of understanding how and why in-context learning works, while opening up new questions about how much can be learned from large language models through inference alone.

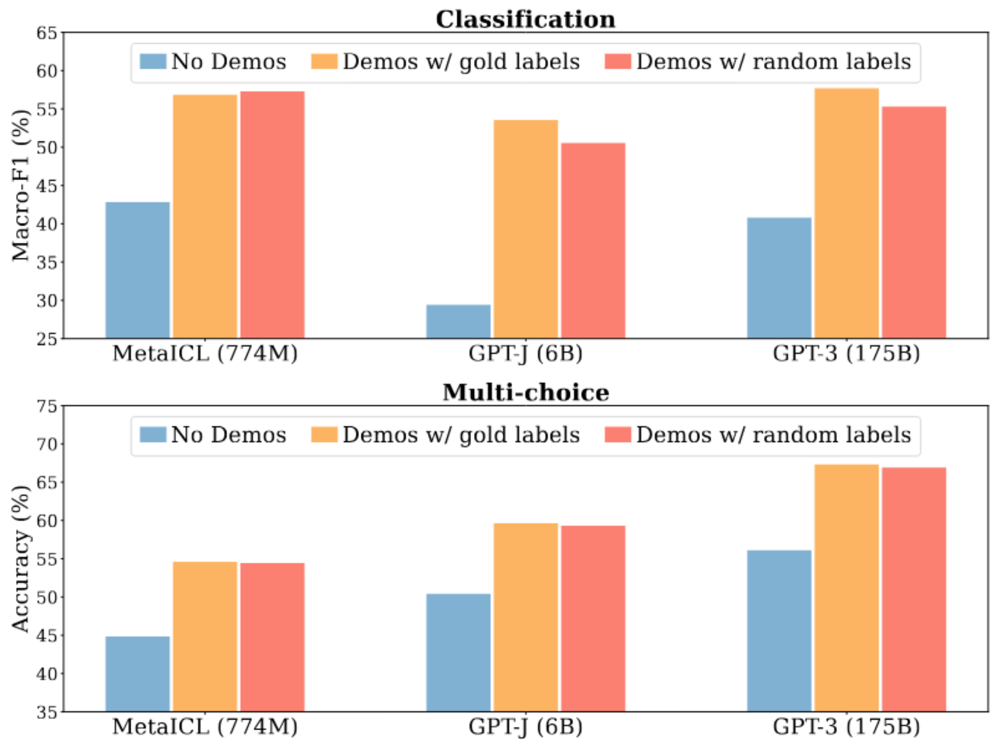
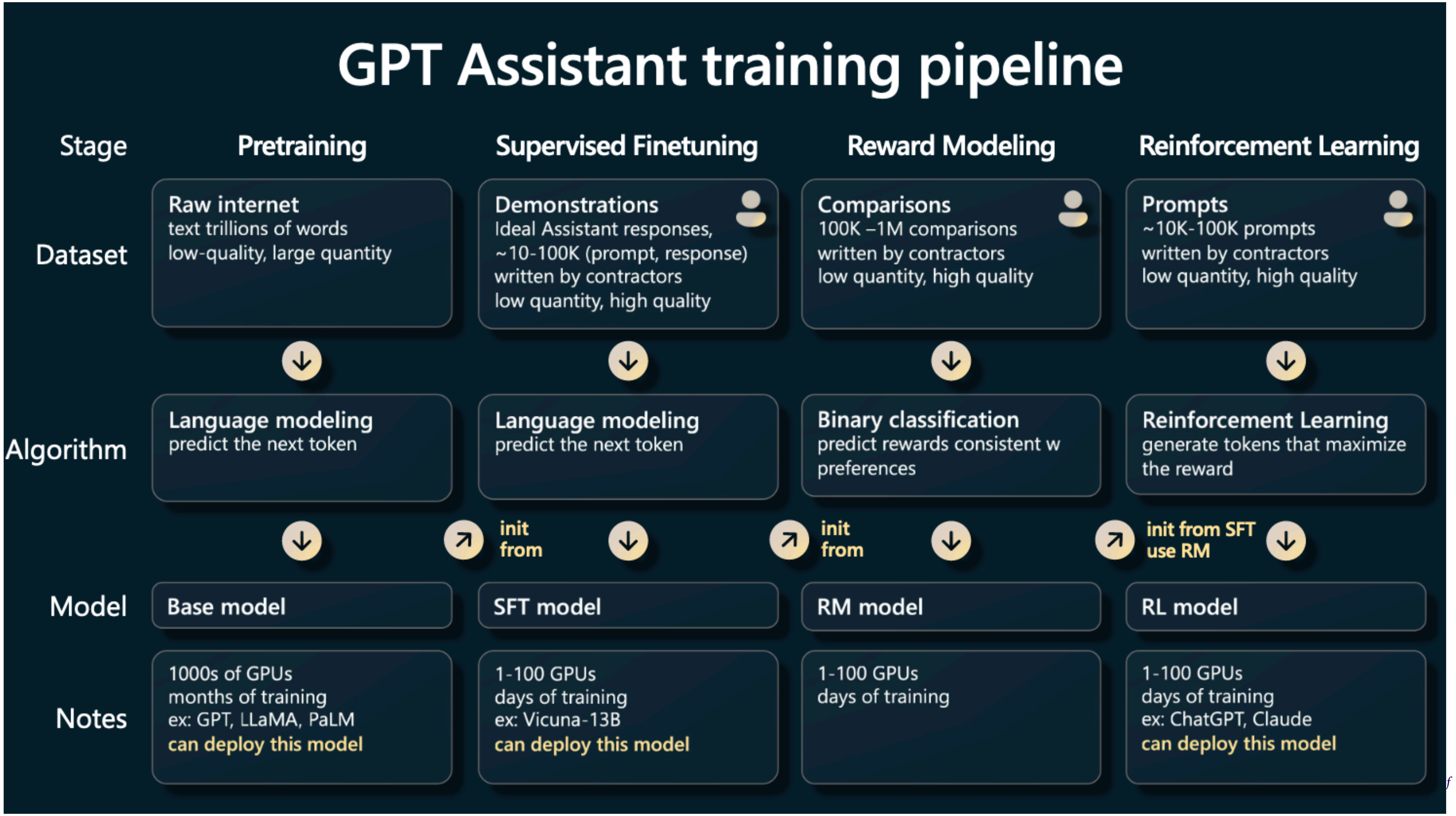


Figure 1: Results in classification (top) and multi-choice tasks (bottom), using three LMs with varying size. Reported on six datasets on which GPT-3 is evaluated; the channel method is used. See Section 4 for the full results. In-context learning performance drops only marginally when labels in the demonstrations are replaced by random labels.

is consistent over 12 different models including the GPT-3 family (Radford et al., 2019; Min et al., 2021b; Wang and Komatsuzaki, 2021; Artetxe

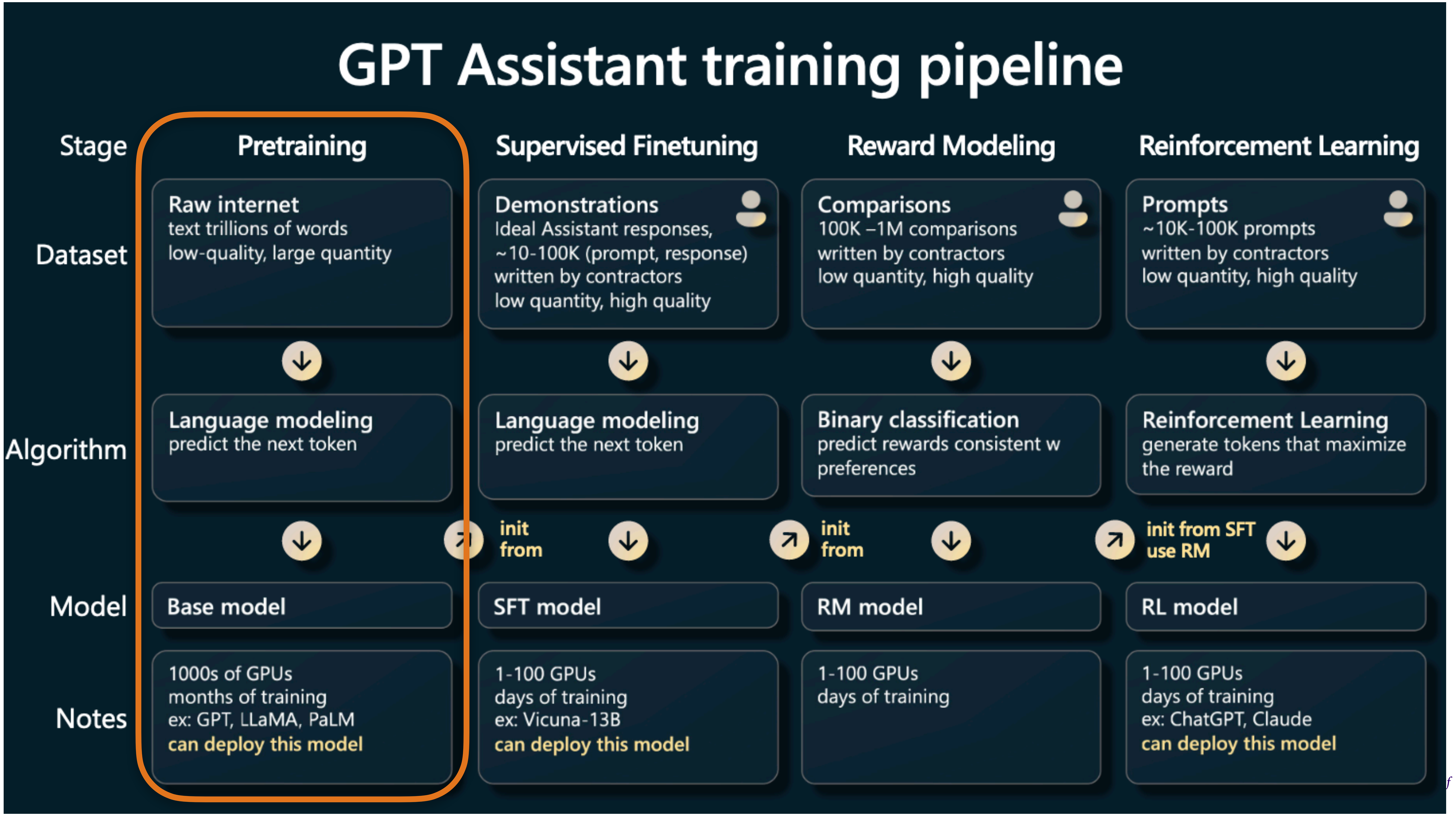
source

From GPT to ChatGPT



source

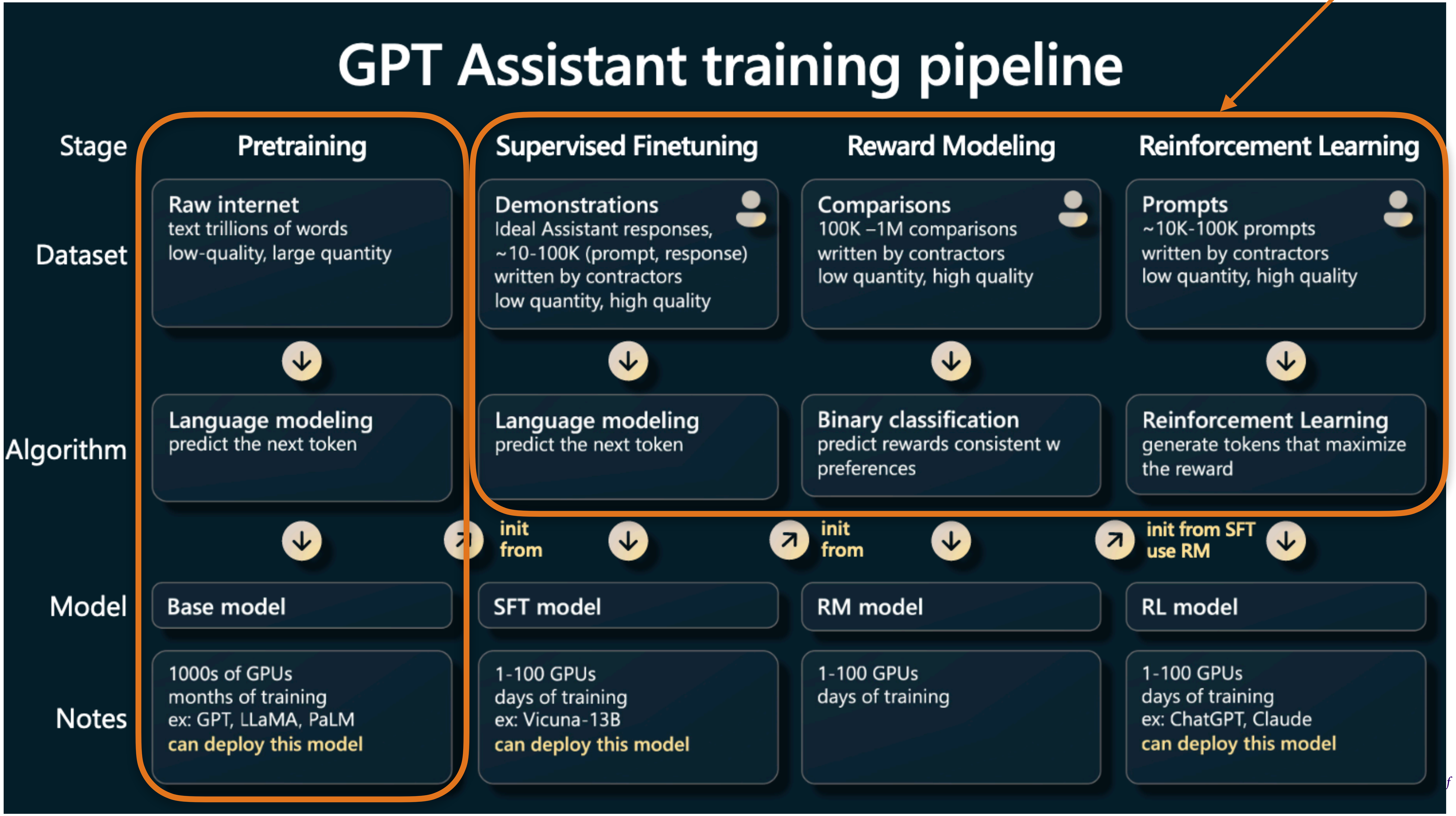
From GPT to ChatGPT



source

From GPT to ChatGPT

“Post-training”

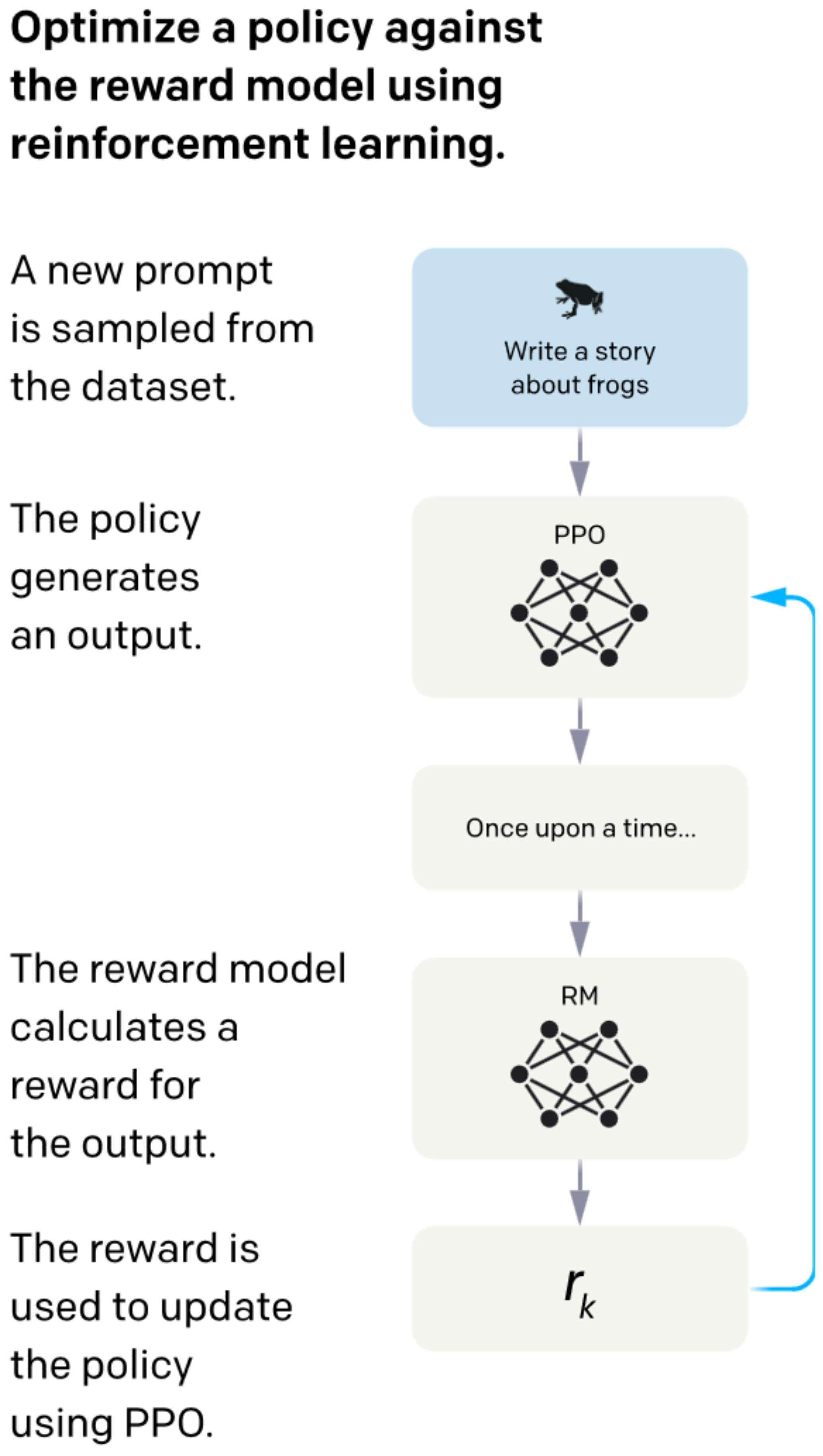


source

RLHF: Reinforcement Learning

- Take a pretrained LM
  - Prompt it, generate response
  - Feed (prompt, response) to reward model RM
  - Use that reward to update LM
- This is reinforcement learning with the RM playing the role of external environment (provider of rewards)

$$\mathcal{L}(\theta_{\mathrm{LM}}) = -\mathbb{E}_{x,\hat{y}\sim P_{\mathrm{LM}}(\cdot\,|\,x;\theta_{\mathrm{LM}})}\left(\mathrm{RM}(x,\hat{y}) - \beta \log\left(\frac{P_{\mathrm{LM}}(\hat{y}\,|\,x;\theta_{\mathrm{LM}})}{P_{\mathrm{LM}}(\hat{y}\,|\,x;\theta_{\mathrm{pretrained}})}\right)\right)$$



Special Topics

- Angie McMillan-Major: societal impacts
- C.M. Downey: Multilingual NLP

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 + language model
- 7: Seq2Seq + Attention [translation]
- 8: Pre-trained transformer classifier
- 9: Prompting and in-context learning

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, state-space models
- Wide variety of NLP tasks: parsing, QA, toxic language detection, etc.
- Generation: wide range of decoding strategies, evaluation
- N.B.: 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
  - CMU's course has lots of online materials: http://www.phontron.com/class/nn4nlp2021/
    - Advanced NLP: https://www.phontron.com/class/anlp-fall2024/
  - Stanford CS224U (pre-recorded videos) http://web.stanford.edu/class/cs224u/
    - And CS224N (live lectures) http://web.stanford.edu/class/cs224n/
- 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 6**
  - https://uw.iasystem.org/survey/307454
- Please do fill them out as soon as possible!
  - E.g. right now :)
  - Help improve the course for future iterations!

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 chaos in the wider world. Very awe-inspiring.
- So: thank you, and have a great summer / future!