Recurrent Neural Networks

LING572 Advanced Statistical Methods for NLP

March 5 2020

Outline

- Word representations and MLPs for NLP tasks
- Recurrent neural networks for sequences
- Fancier RNNs
  - Vanishing/exploding gradients
  - LSTMs (Long Short-Term Memory)
  - Variants
- Seq2seq architecture
  - Attention

MLPs for text classification

Word Representations

- Traditionally: words are *discrete features*
  - e.g. curWord="class"
  - As vectors: *one-hot* encoding
    - Each vector is $|V|$-dimensional, where V is the vocabulary
    - Each dimension corresponds to one word of the vocabulary
    - A 1 for the current word; 0 everywhere else

$$w_1 = [1 \quad 0 \quad 0 \quad \cdots \quad 0]$$
$$w_3 = [0 \quad 0 \quad 1 \quad \cdots \quad 0]$$

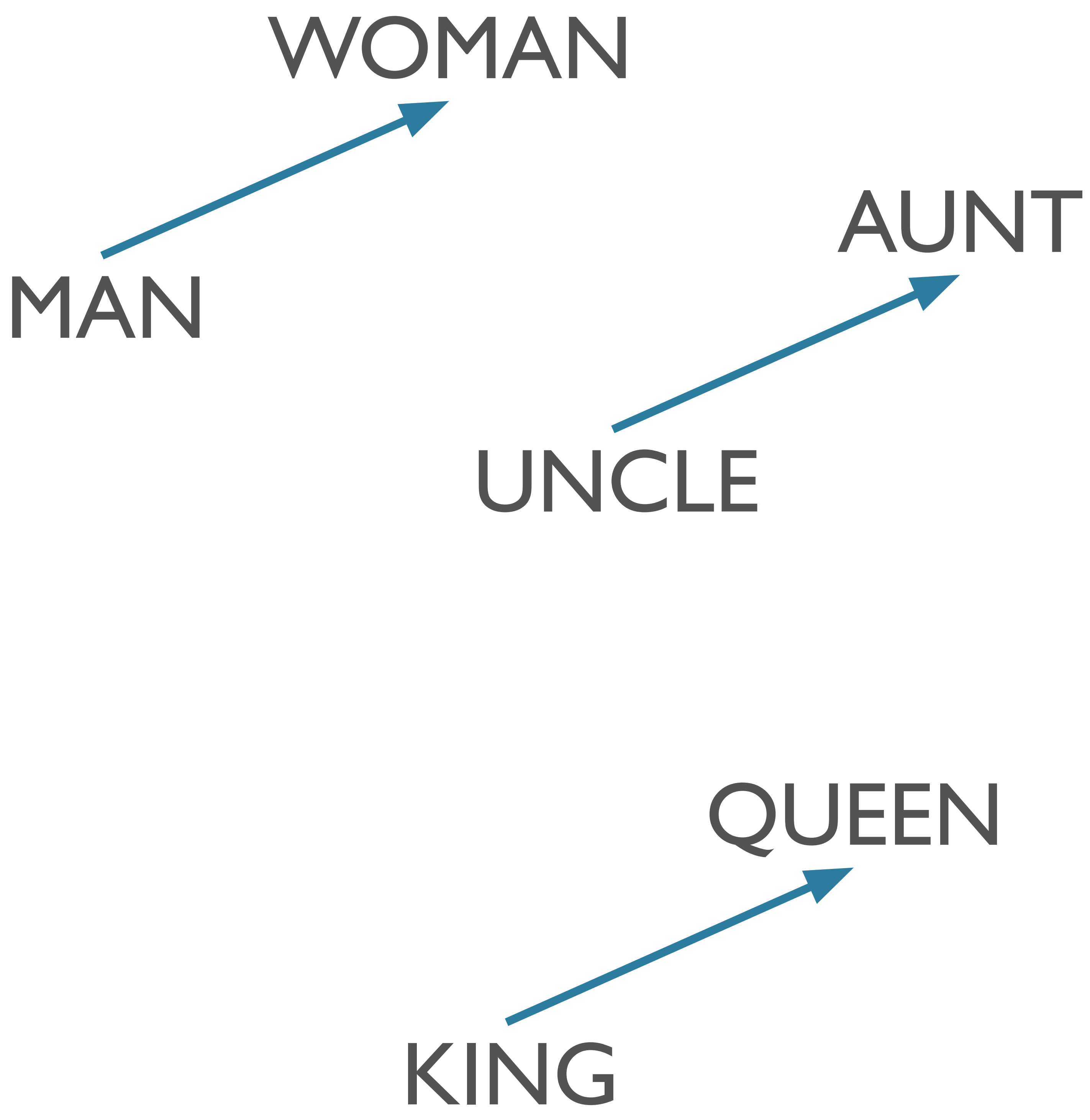
Word Embeddings

- Problem 1: every word is equally different from every other.
  - All words are orthogonal to each other.
- Problem 2: very high dimensionality
- Solution: Move words into *dense*, lower-dimensional space
  - Grouping similar words to each other
  - These denser representations are called *embeddings*

Word Embeddings

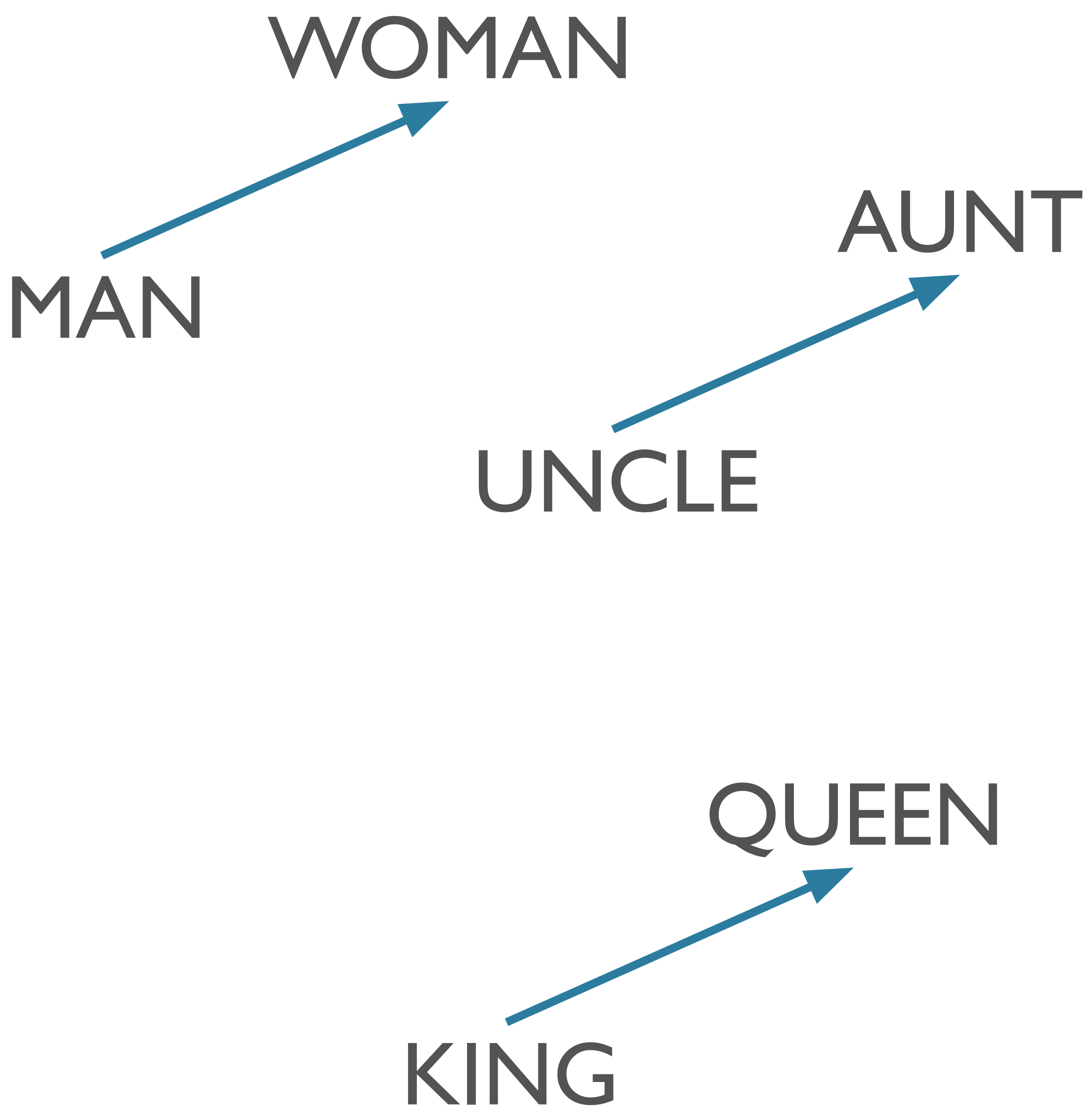
- Formally, a *d*-dimensional embedding is a matrix *E* with shape (|V|, d)
  - Each row is the vector for one word in the vocabulary
  - Matrix multiplying by a one-hot vector returns the corresponding row, i.e. the right word vector
- Trained on prediction tasks (see LING571 slides)
  - Continuous bag of words
  - Skip-gram
  - …
- Can be trained on specific task, or download pre-trained (e.g. GloVe, fastText)
- Fancier versions now to deal with OOV: sub-word (e.g. BPE), character CNN/LSTM

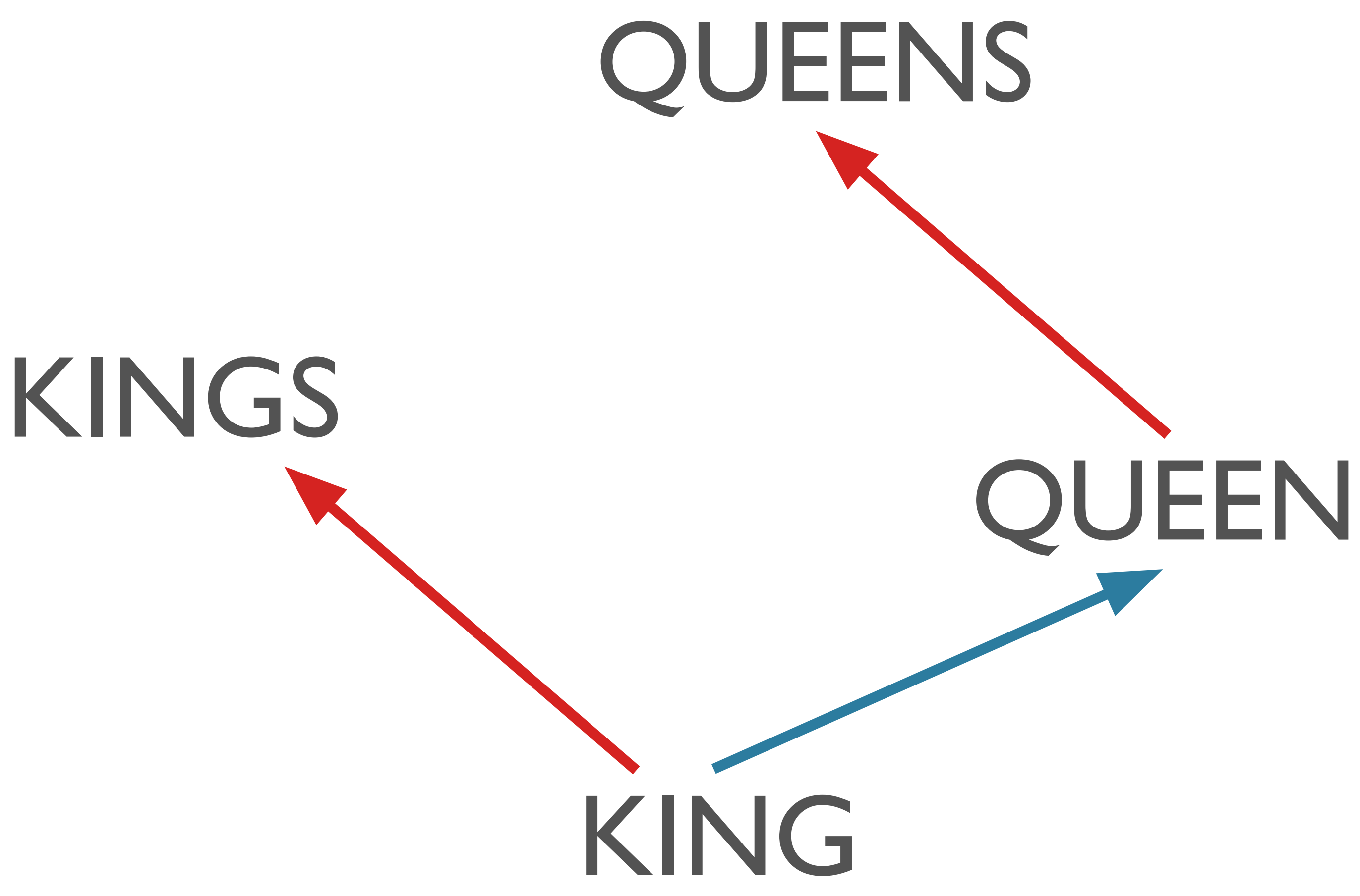
Relationships via Offsets



[Mikolov et al 2013b]

Relationships via Offsets





[Mikolov et al 2013b]

One More Example

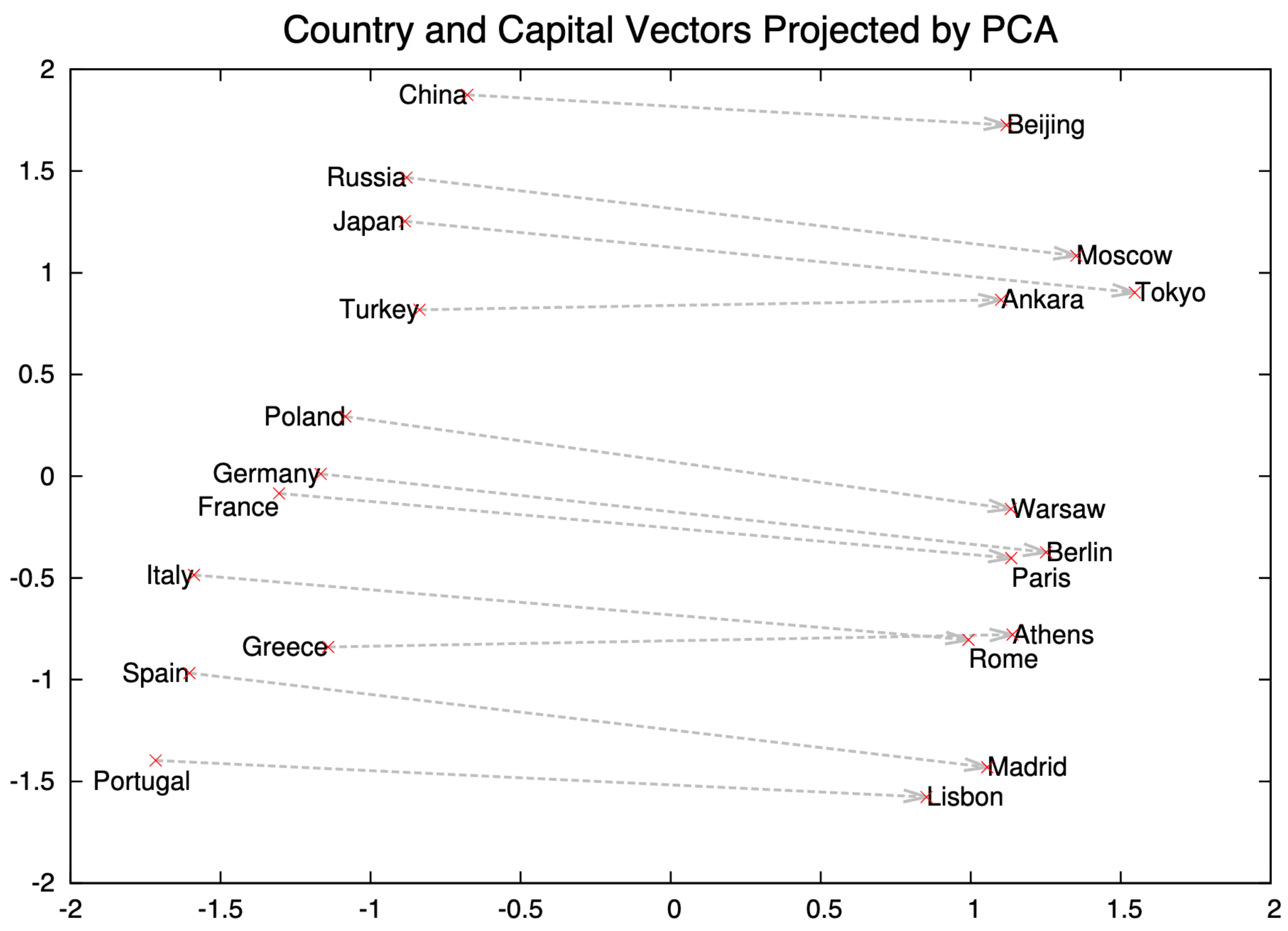
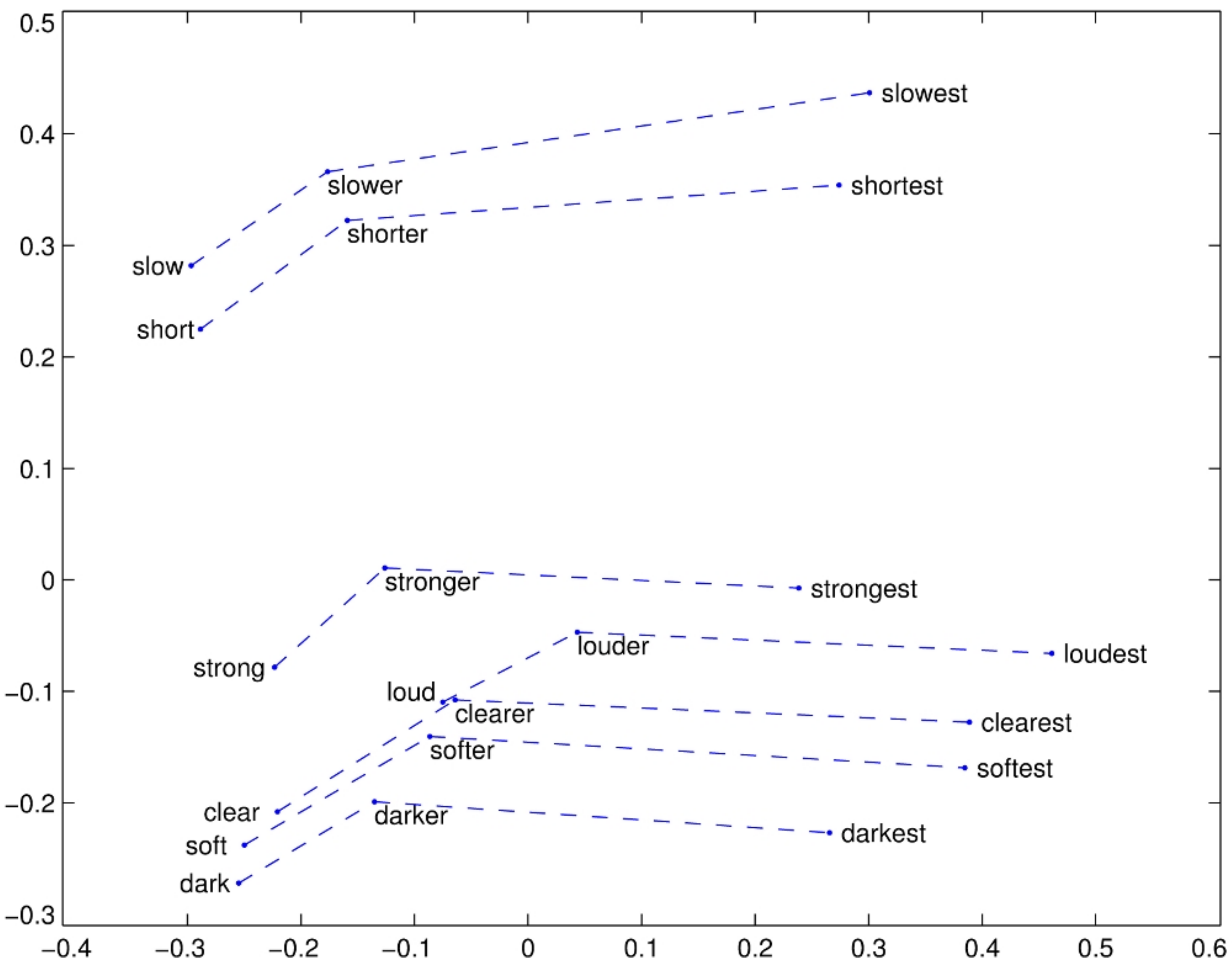


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Mikolov et al 2013c

One More Example



Caveat Emptor

Issues in evaluating semantic spaces using word analogies

Tal Linzen
LSCP & IJN
École Normale Supérieure
PSL Research University
tal.linzen@ens.fr

Abstract

The offset method for solving word analogies has become a standard evaluation tool for vector-space semantic models: it is considered desirable for a space to represent semantic relations as consistent vector offsets. We show that the method's reliance on cosine similarity conflates offset consistency with largely irrelevant neighborhood structure, and propose simple baselines that should be used to improve the utility of the method in vector space evaluation.

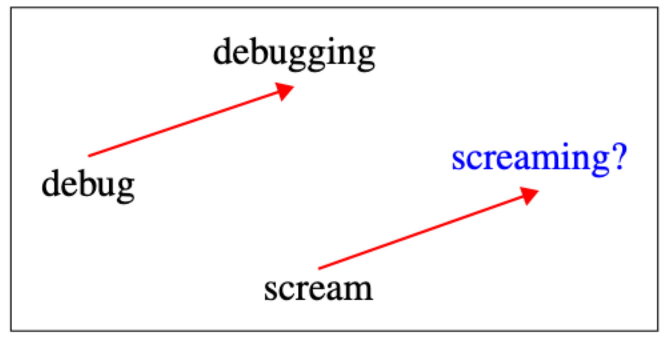


Figure 1: Using the vector offset method to solve the analogy task (Mikolov et al., 2013c).

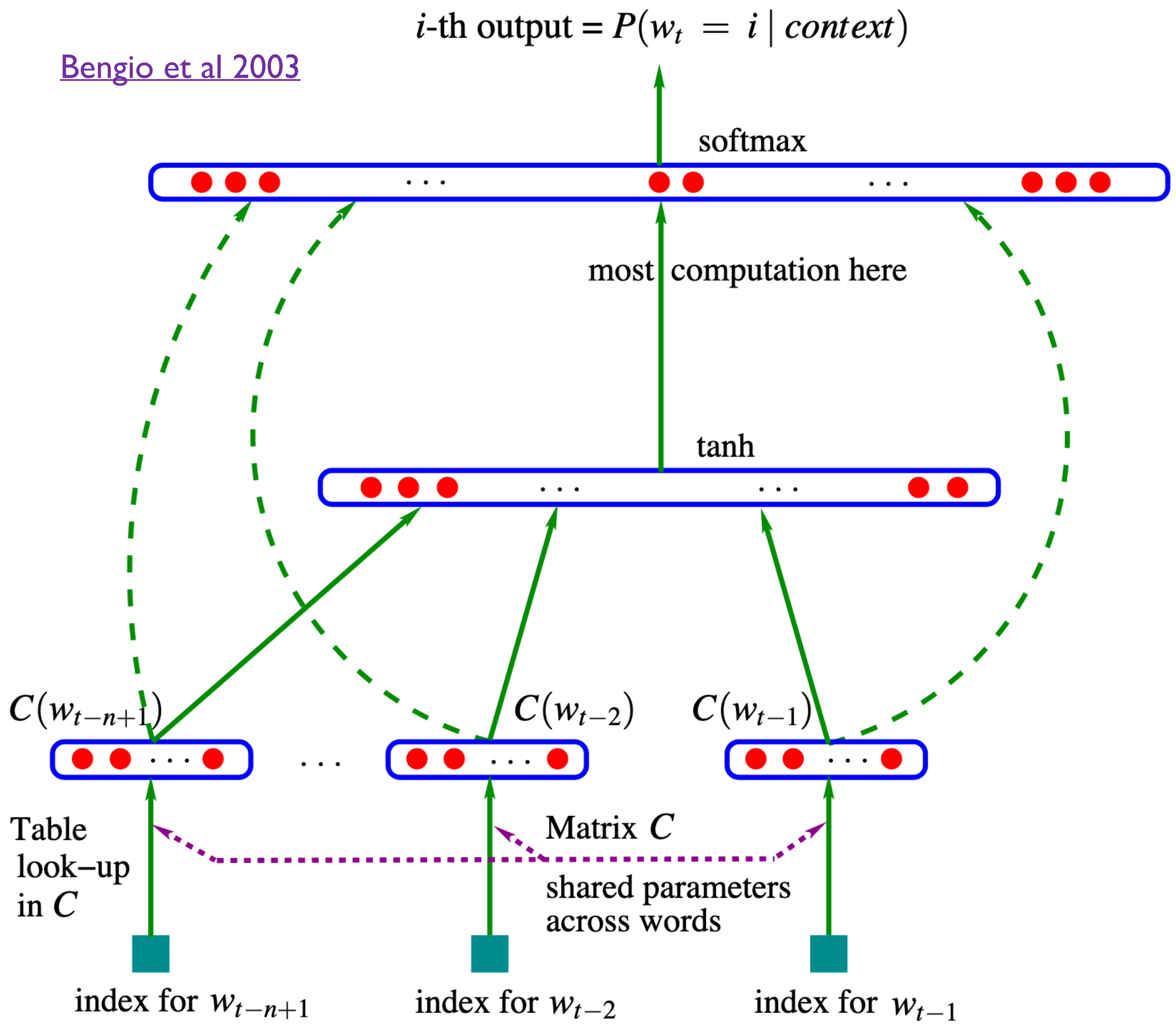
cosine similarity to the landing point. Formally, if the analogy is given by

$$a : a^* :: b : \_\_\_ \qquad (1)$$

Linzen 2016, a.o.

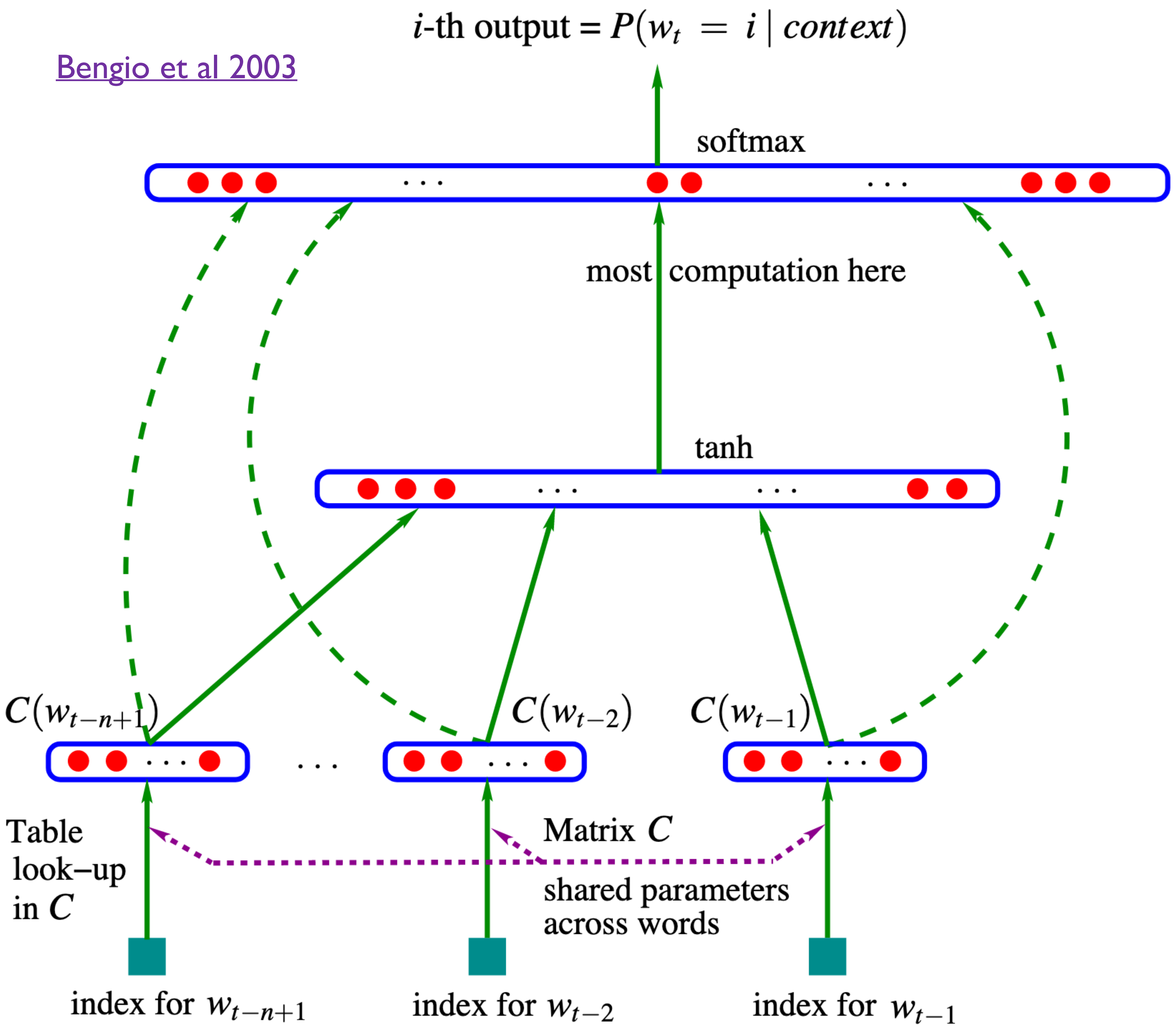
Example MLP for Language Modeling

Bengio et al 2003



Example MLP for Language Modeling

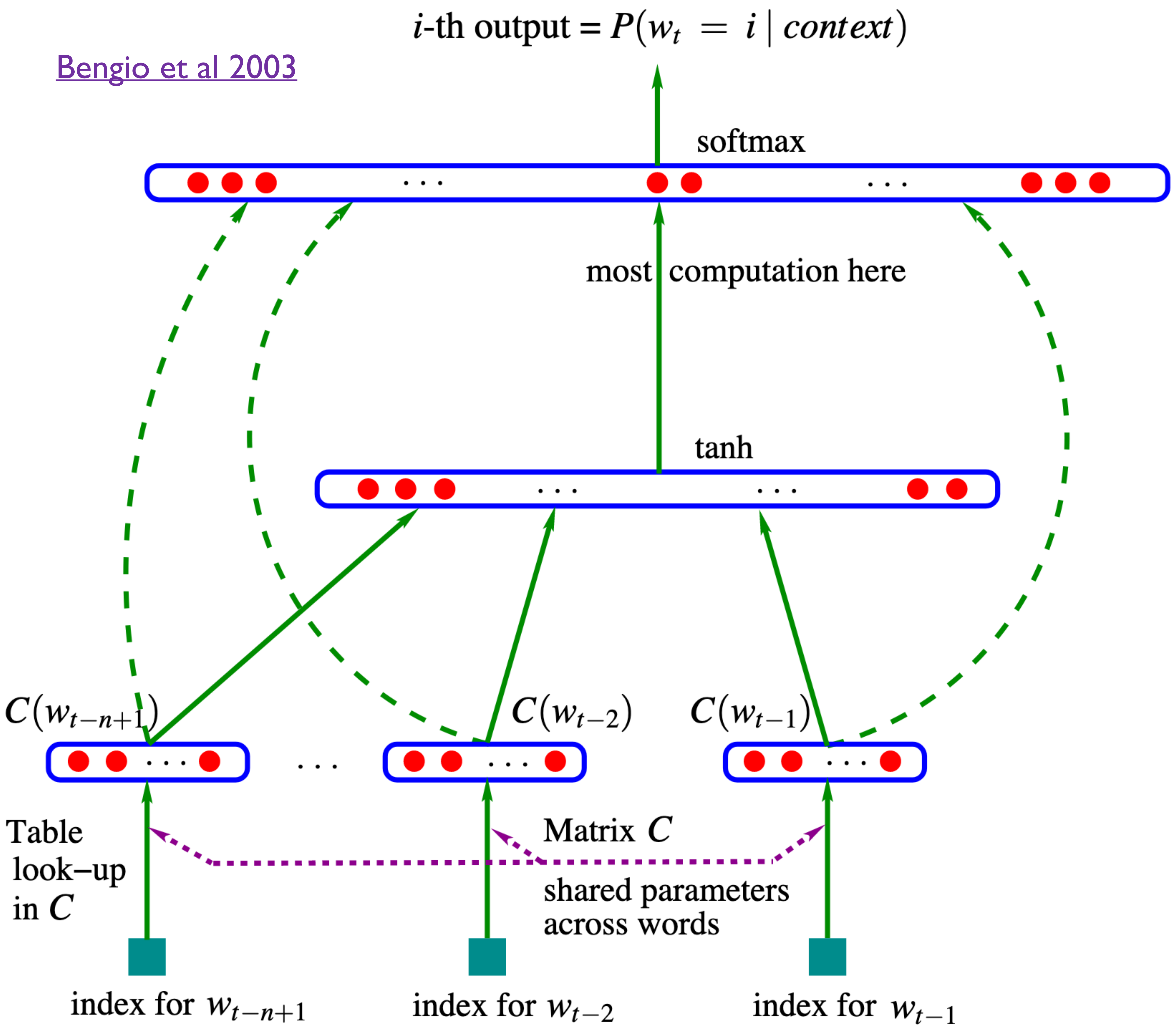
Bengio et al 2003



w_t: one-hot vector

Example MLP for Language Modeling

[Bengio et al 2003]

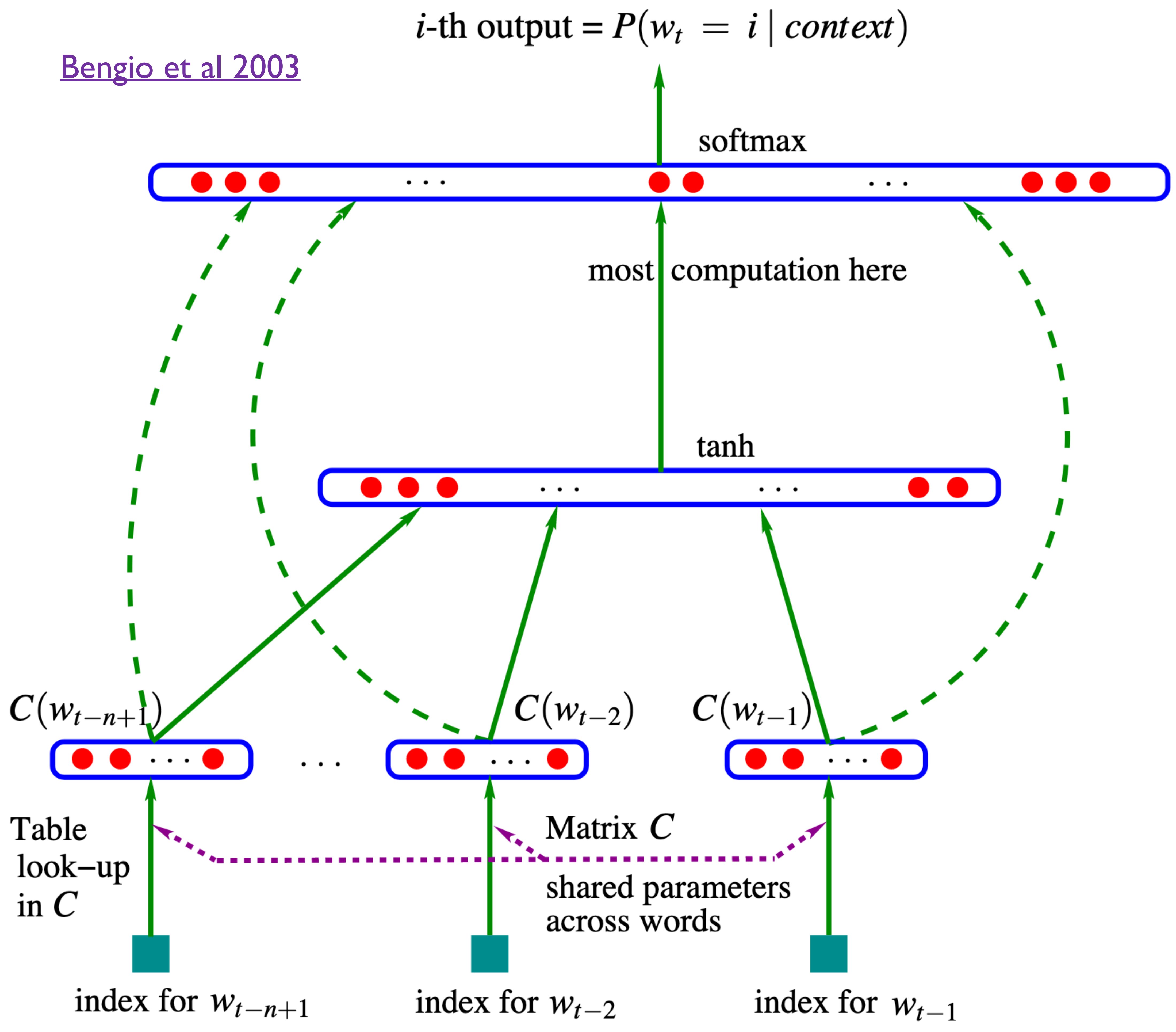


embeddings = concat$(Cw_{t-1}, Cw_{t-2}, \ldots, Cw_{t-(n+1)})$

w_t: one-hot vector

Example MLP for Language Modeling

[Bengio et al 2003]



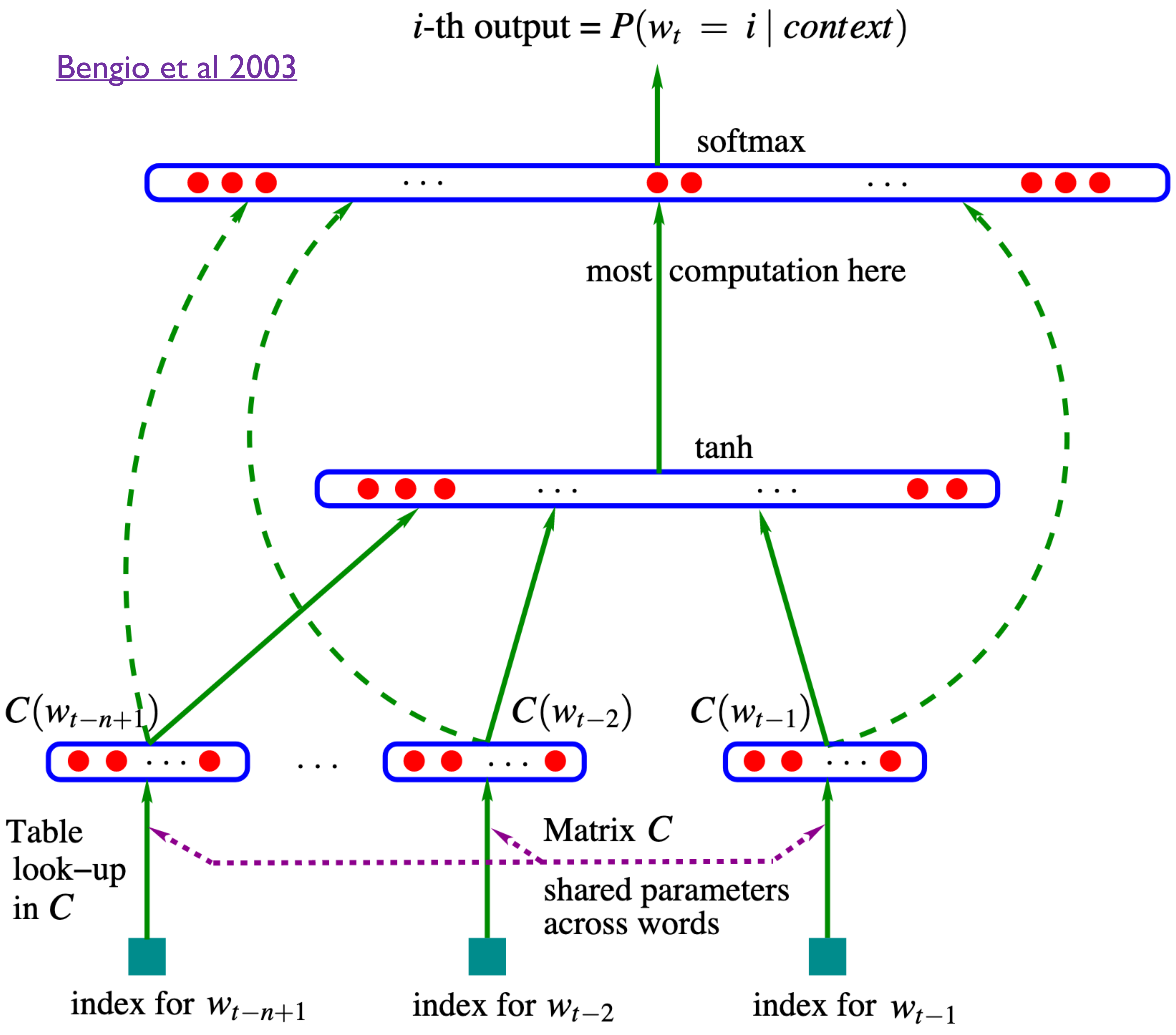
$$\text{hidden} = \tanh(W_1 \text{embeddings} + b_1)$$

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

w_t: one-hot vector

Example MLP for Language Modeling

Bengio et al 2003



$$\text{probabilities} = \text{softmax}(W_2 \text{hidden} + b_2)$$

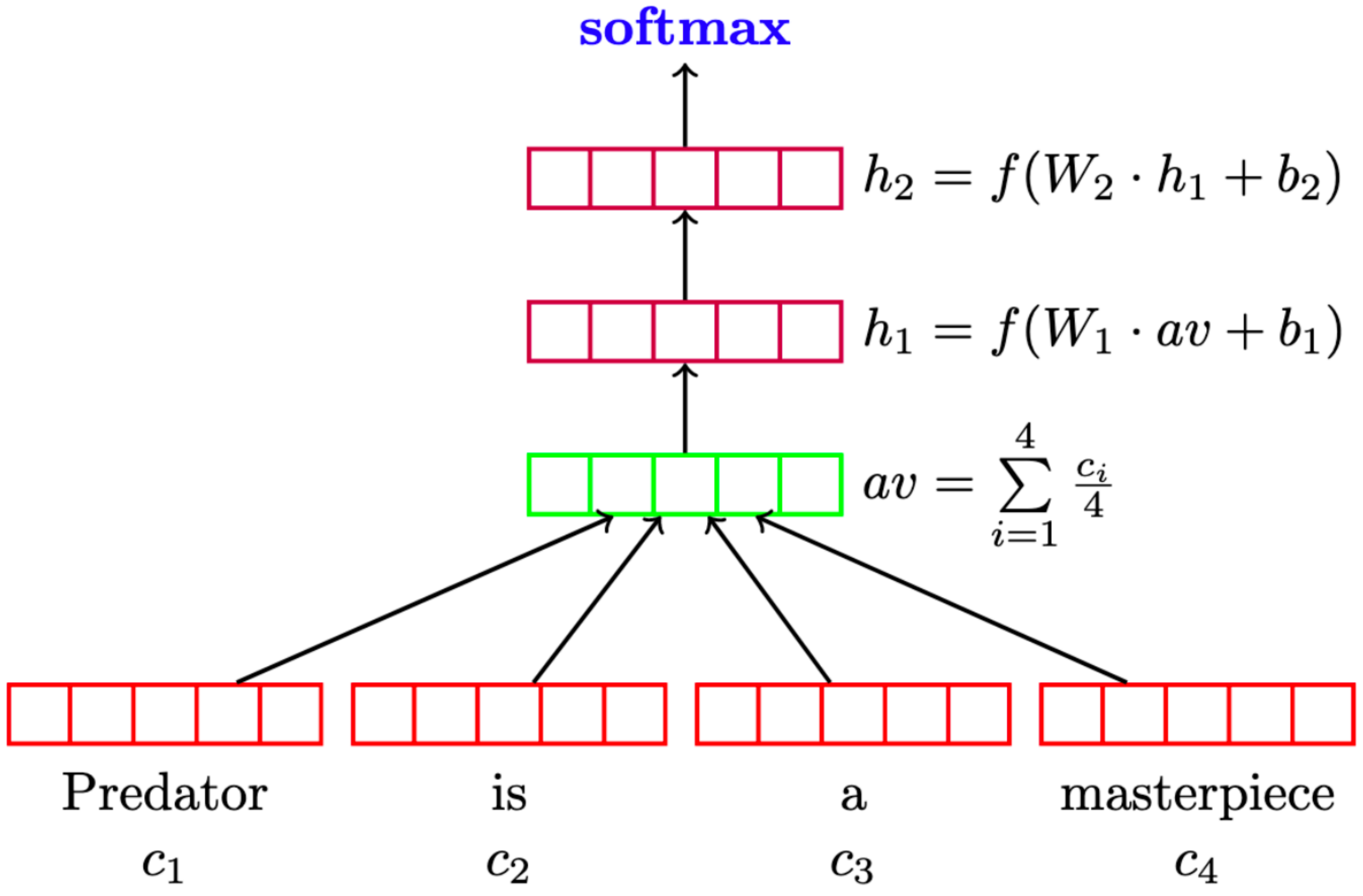
$$\text{hidden} = \tanh(W_1 \text{embeddings} + b_1)$$

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

w_t: one-hot vector

Example MLP for sentiment classification

- Issue: texts of different length.
  - One solution: average (or sum, or…) all the embeddings, which are of same dim



Iyyer et al 2015

Model	IMDB accuracy
Deep averaging network	89.4
NB-SVM (Wang and Manning 2012)	91.2

Recurrent Neural Networks

RNNs: high-level

RNNs: high-level

- Feed-forward networks: fixed-size input, fixed-size output
    - Previous classifier: average embeddings of words
    - Other solutions: *n*-gram assumption (i.e. fixed-size context of word embeddings)

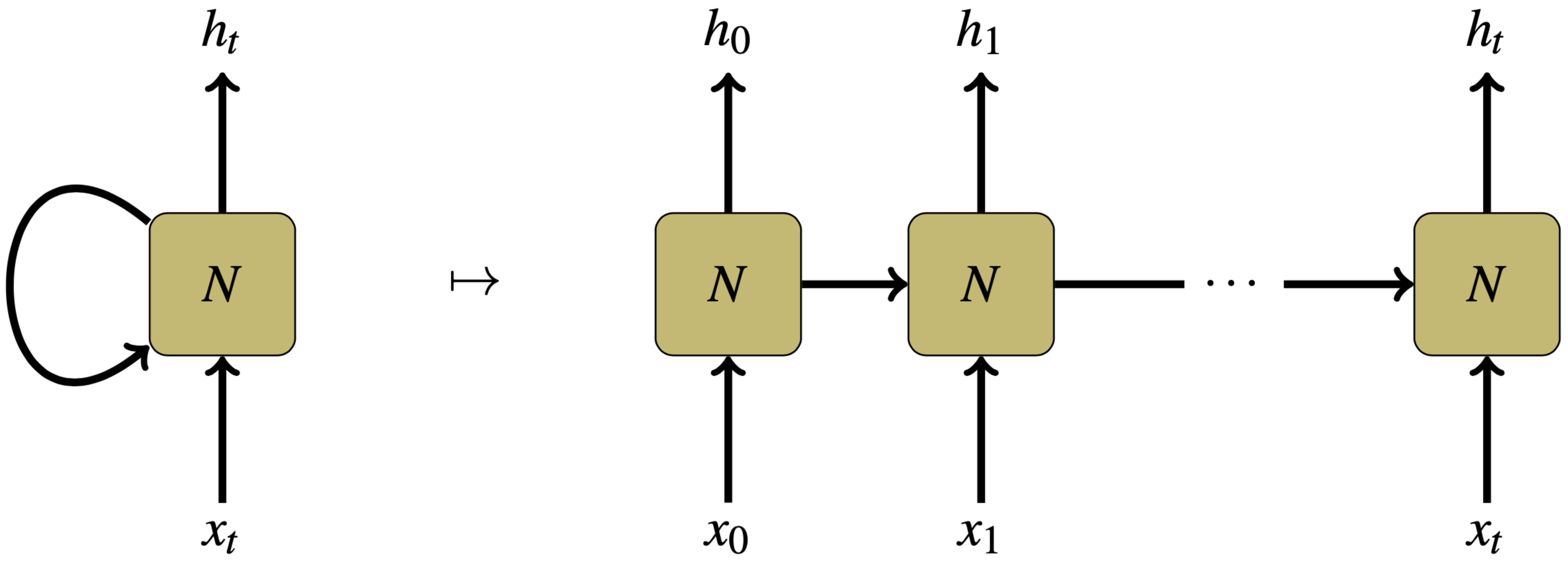
RNNs: high-level

- Feed-forward networks: fixed-size input, fixed-size output
  - Previous classifier: average embeddings of words
  - Other solutions: *n*-gram assumption (i.e. fixed-size context of word embeddings)
- RNNs process *sequences* of vectors
  - Maintaining “hidden” state
  - Applying the same operation at each step

RNNs: high-level

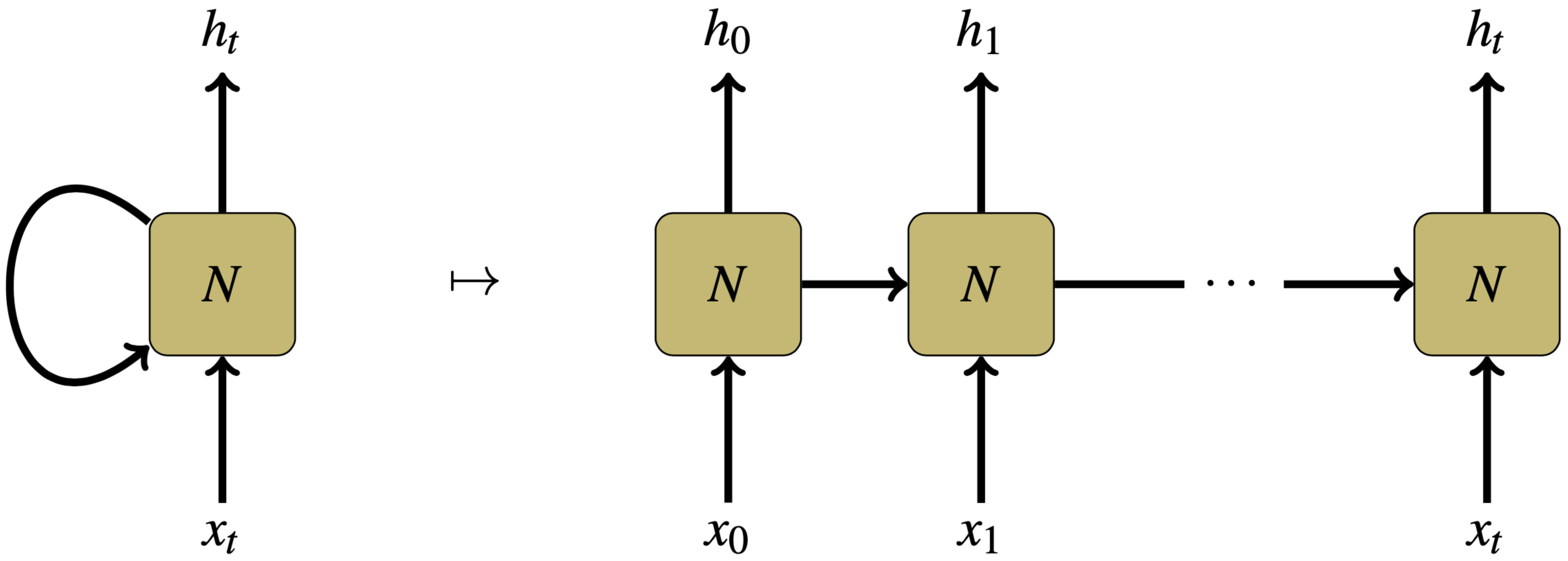
- Feed-forward networks: fixed-size input, fixed-size output
  - Previous classifier: average embeddings of words
  - Other solutions: *n*-gram assumption (i.e. fixed-size context of word embeddings)
- RNNs process *sequences* of vectors
  - Maintaining "hidden" state
  - Applying the same operation at each step
- Different RNNs:
  - Different operations at each step
  - Operation also called "recurrent cell"
  - Other architectural considerations (e.g. depth; bidirectionally)

RNNs



Steinert-Threlkeld and Szymanik 2019; Olah 2015

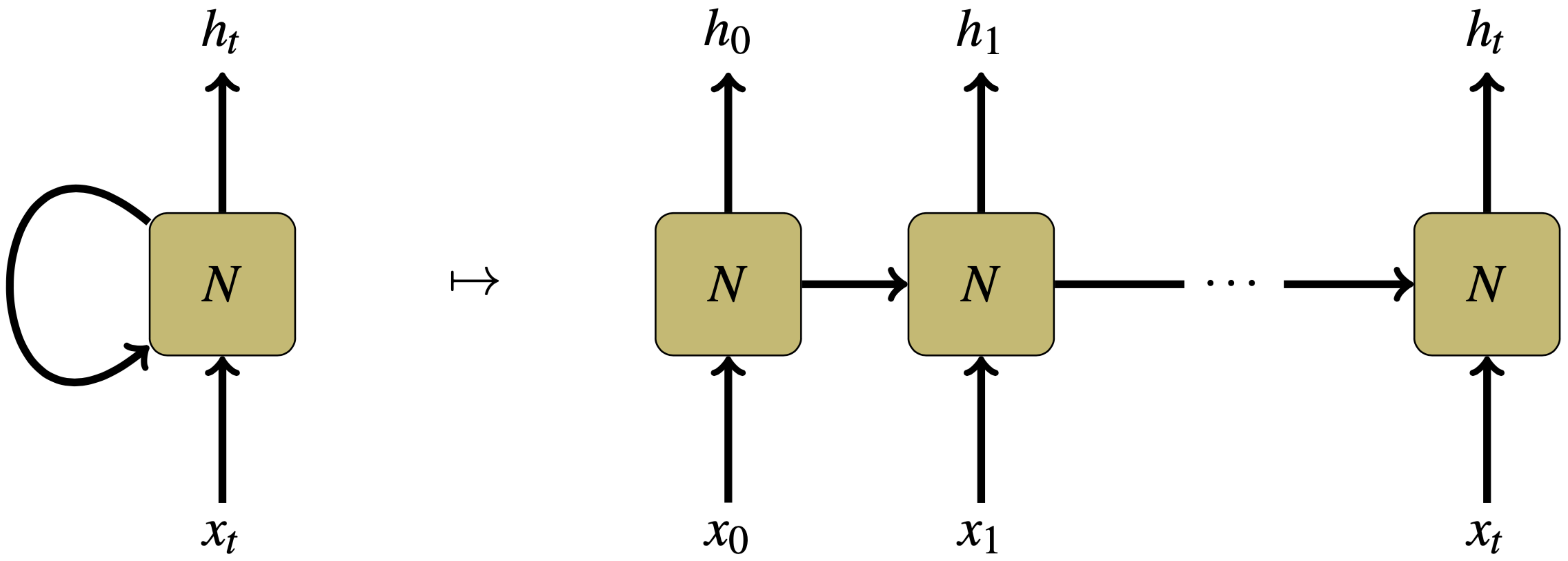
RNNs



$$h_t = f(x_t, h_{t-1})$$

[Steinert-Threlkeld and Szymanik 2019](); [Olah 2015]()

RNNs

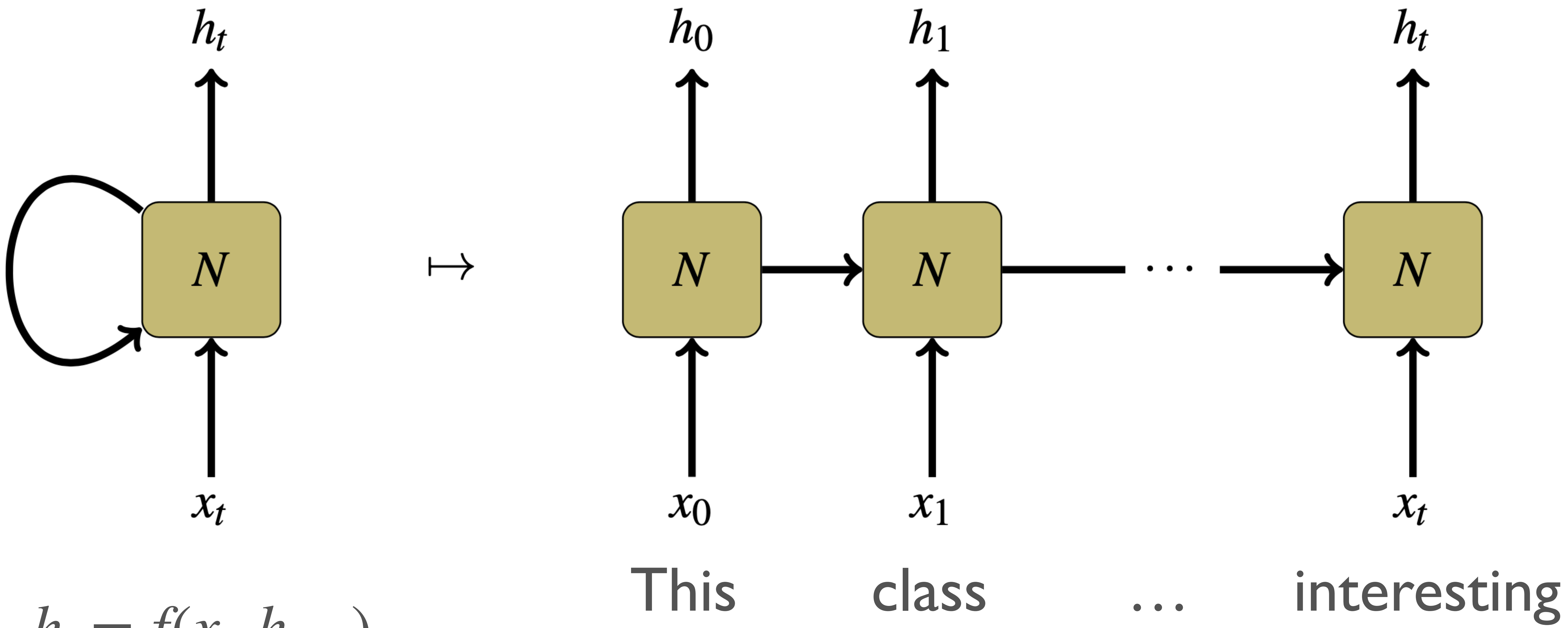


$$h_t = f(x_t, h_{t-1})$$

Simple/"Vanilla" RNN: $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

Steinert-Threlkeld and Szymanik 2019; Olah 2015

RNNs

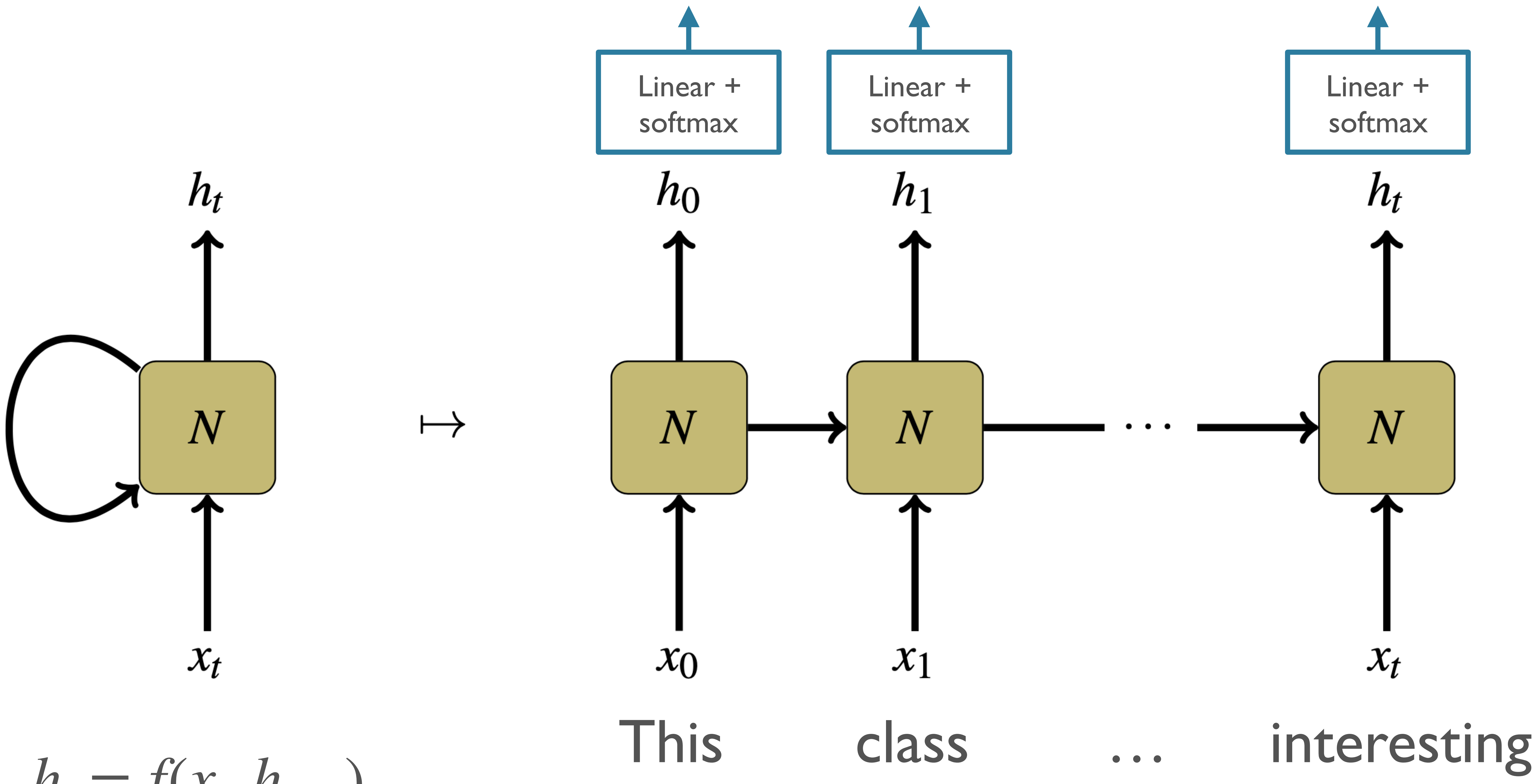


$$h_t = f(x_t, h_{t-1})$$

Simple/"Vanilla" RNN: $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

Steinert-Threlkeld and Szymanik 2019; Olah 2015

RNNs

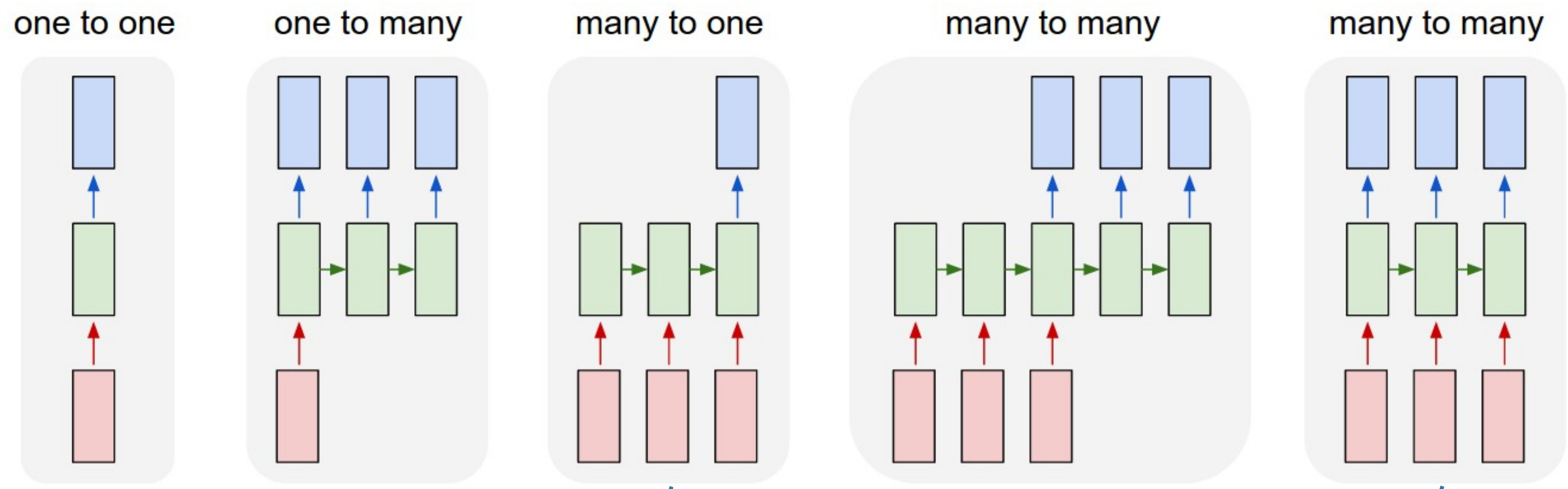


$$h_t = f(x_t, h_{t-1})$$

Simple/"Vanilla" RNN: $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

Steinert-Threlkeld and Szymanik 2019; Olah 2015

Using RNNs



MLP

seq2seq (later)

e.g. text classification

e.g. POS tagging

Training: BPTT

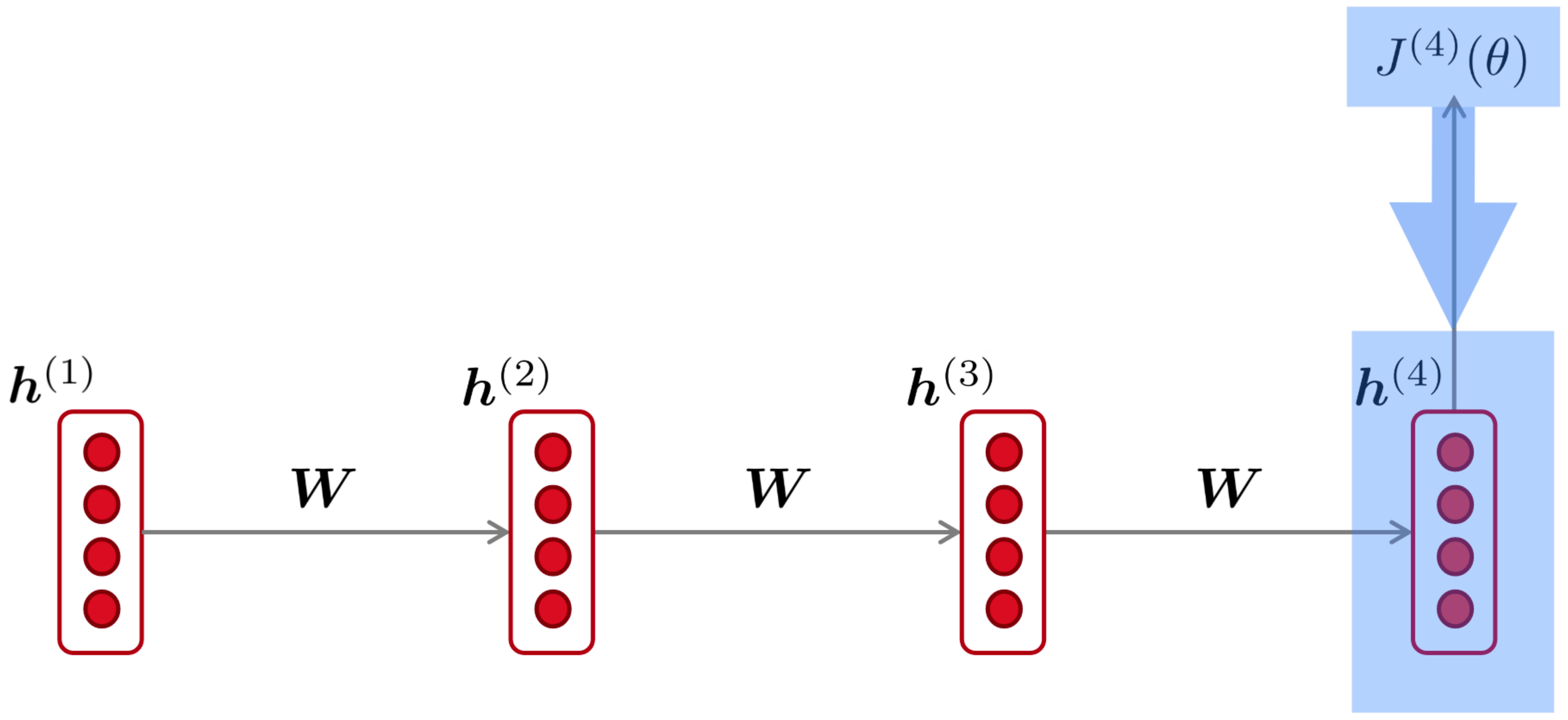
- "Unroll" the network across time-steps
- Apply backprop to the "wide" network
  - Each cell has the *same* parameters
  - When updating parameters using the gradients, take the average across the time steps

Fancier RNNs

Vanishing/Exploding Gradients Problem

- BPTT with vanilla RNNs faces a major problem:
  - The gradients can *vanish* (approach 0) across time
  - This makes it hard/impossible to learn *long distance dependencies*, which are rampant in natural language

Vanishing Gradients

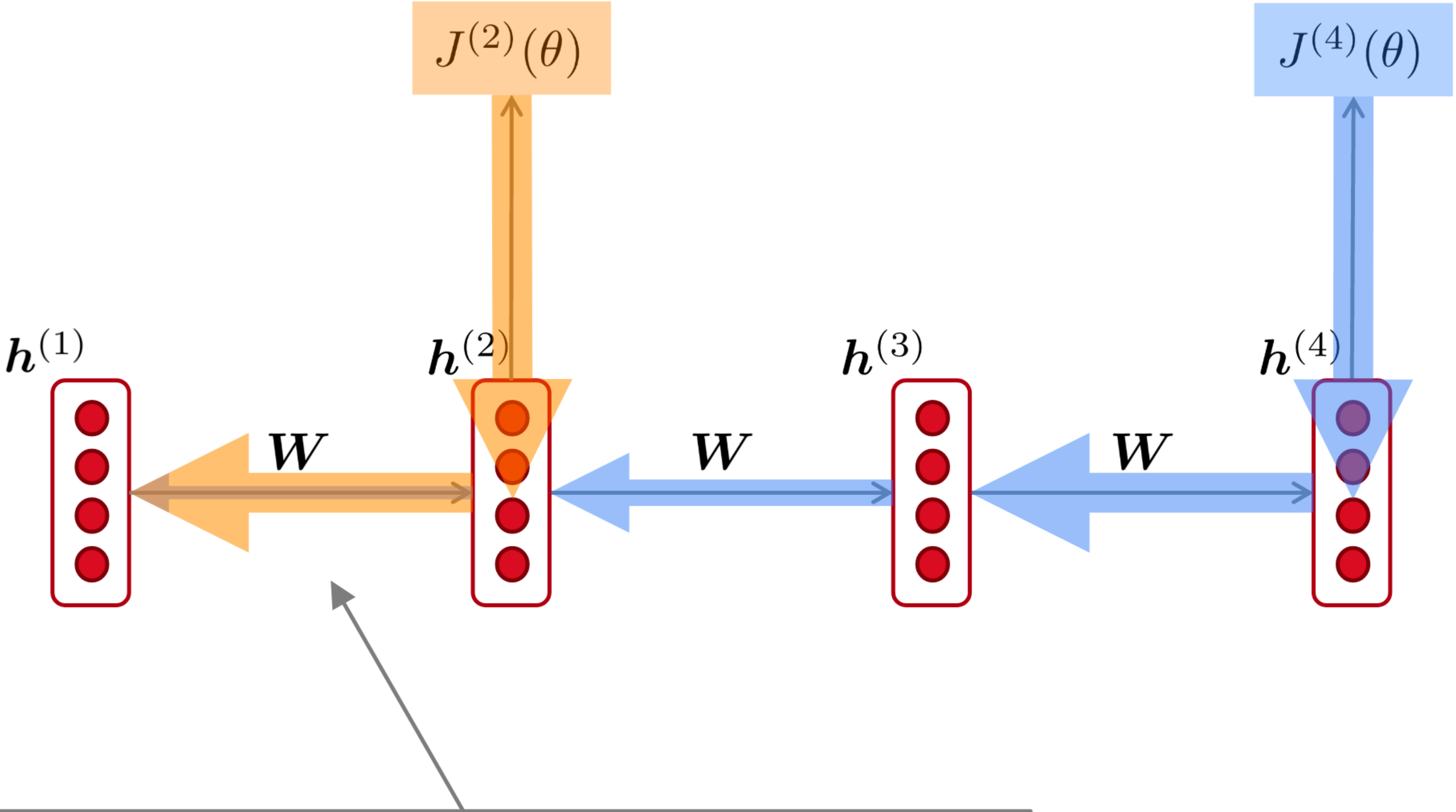


source

$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

If these are small (depends on W), the effect from t=4 on t=1 will be very small

Vanishing Gradient Problem

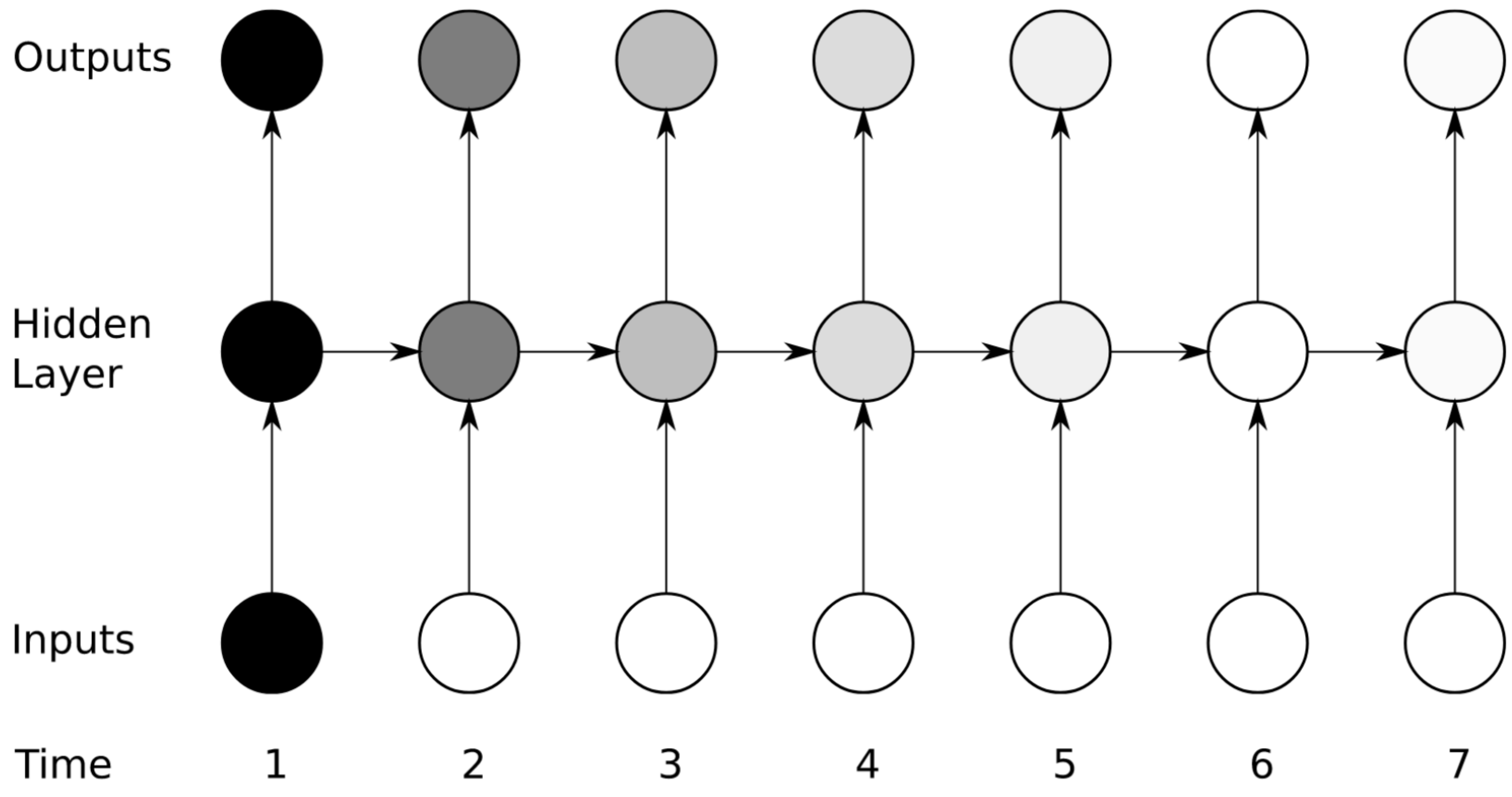


Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are updated only with respect to near effects, not long-term effects.

[source]

Vanishing Gradient Problem



Graves 2012

Vanishing Gradient Problem

- Gradient measures the effect of the past on the future
- If it vanishes between t and t+n, can't tell if:
  - There's no dependency in fact
  - The weights in our network just haven't yet captured the dependency

The need for long-distance dependencies

- Language modeling (fill-in-the-blank)
  - The keys ____
  - The keys on the table ____
  - The keys next to the book on top of the table ____
- To get the number on the verb, need to look at the subject, which can be very far away
  - And number can disagree with linearly-close nouns
- Need models that can capture long-range dependencies like this. Vanishing gradients means vanilla RNNs will have difficulty.

Long Short-Term Memory (LSTM)

LSTMs

- Long Short-Term Memory (Hochreiter and Schmidhuber 1997)
- The gold standard / default RNN
  - If someone says "RNN" now, they almost always mean "LSTM"
- Originally: to solve the vanishing/exploding gradient problem for RNNs
  - Vanilla: re-writes the entire hidden state at every time-step
  - LSTM: separate hidden state and memory
    - Read, write to/from memory; can preserve long-term information

LSTMs

$$f_t = \sigma\left(W^f \cdot h_{t-1}x_t + b^f\right)$$

$$i_t = \sigma\left(W^i \cdot h_{t-1}x_t + b^i\right)$$

$$\hat{c}_t = \tanh\left(W^c \cdot h_{t-1}x_t + b^c\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

$$o_t = \sigma\left(W^o \cdot h_{t-1}x_t + b^o\right)$$

$$h_t = o_t \odot \tanh\left(c_t\right)$$

LSTMs

$$f_t = \sigma\left(W^f \cdot h_{t-1}x_t + b^f\right)$$

$$i_t = \sigma\left(W^i \cdot h_{t-1}x_t + b^i\right)$$

$$\hat{c}_t = \tanh\left(W^c \cdot h_{t-1}x_t + b^c\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

$$o_t = \sigma\left(W^o \cdot h_{t-1}x_t + b^o\right)$$

$$h_t = o_t \odot \tanh\left(c_t\right)$$

🤔🤔🤷‍♀️🤮

LSTMs

- Key innovation:
  - $c_t, h_t = f(x_t, c_{t-1}, h_{t-1})$
  - c_t: a *memory cell*
- Reading/writing (smooth) controlled by *gates*
  - f_t: forget gate
  - i_t: input gate
  - o_t: output gate

$$f_t = \sigma\left(W^f \cdot h_{t-1}x_t + b^f\right)$$

$$i_t = \sigma\left(W^i \cdot h_{t-1}x_t + b^i\right)$$

$$\hat{c}_t = \tanh\left(W^c \cdot h_{t-1}x_t + b^c\right)$$

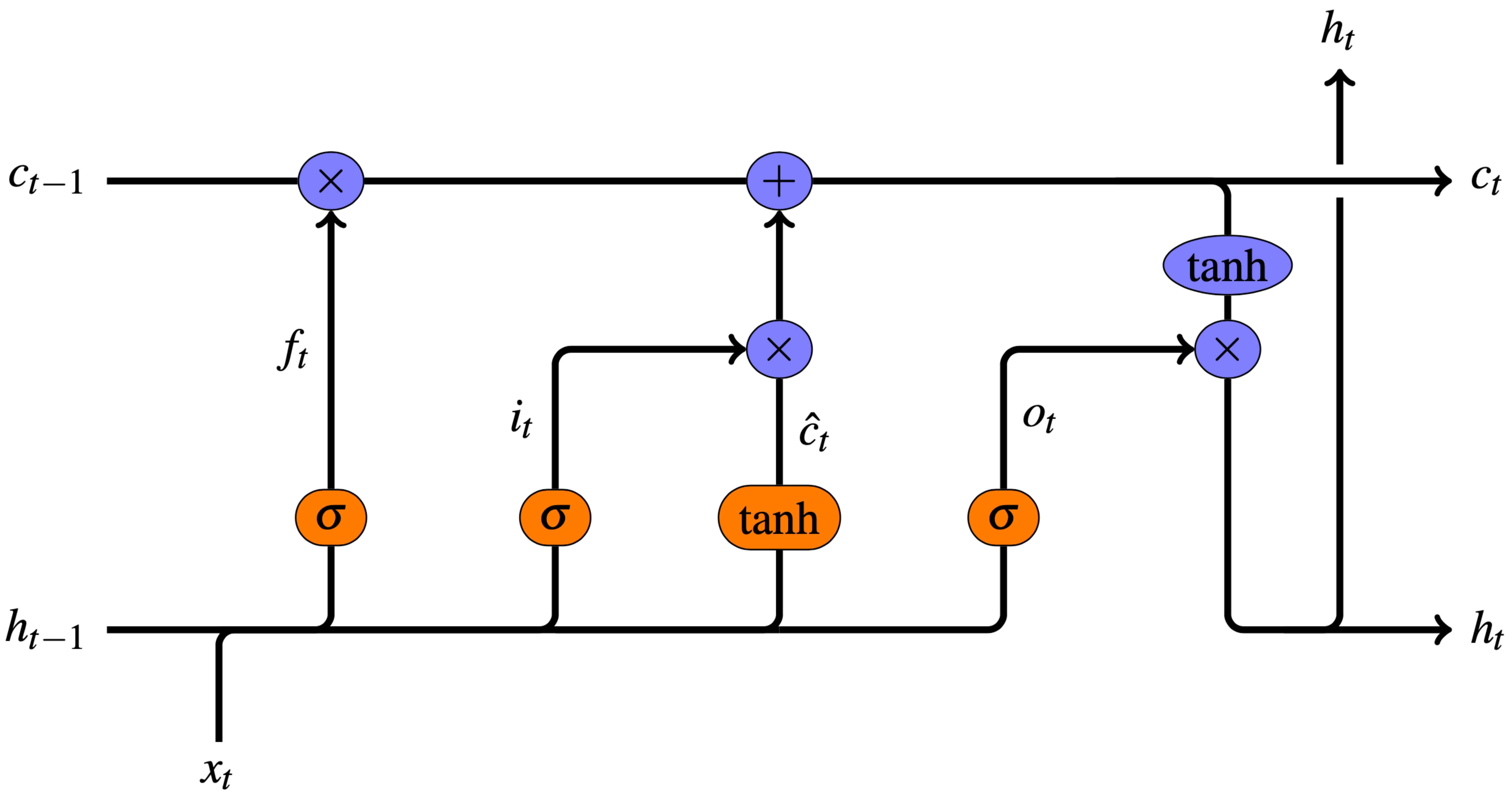
$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

$$o_t = \sigma\left(W^o \cdot h_{t-1}x_t + b^o\right)$$

$$h_t = o_t \odot \tanh\left(c_t\right)$$

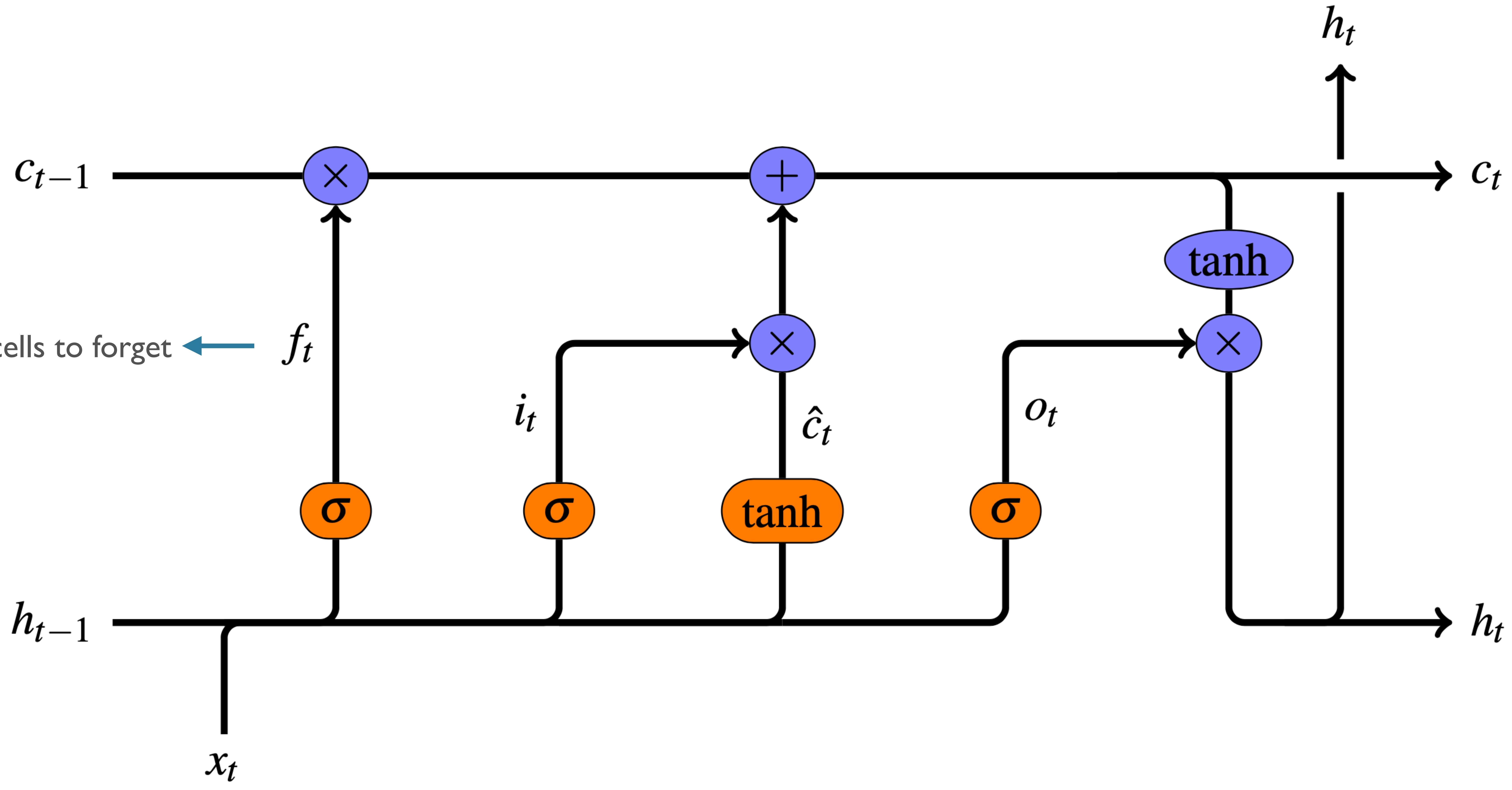
🤔🤔🤷‍♀️🤮

LSTMs

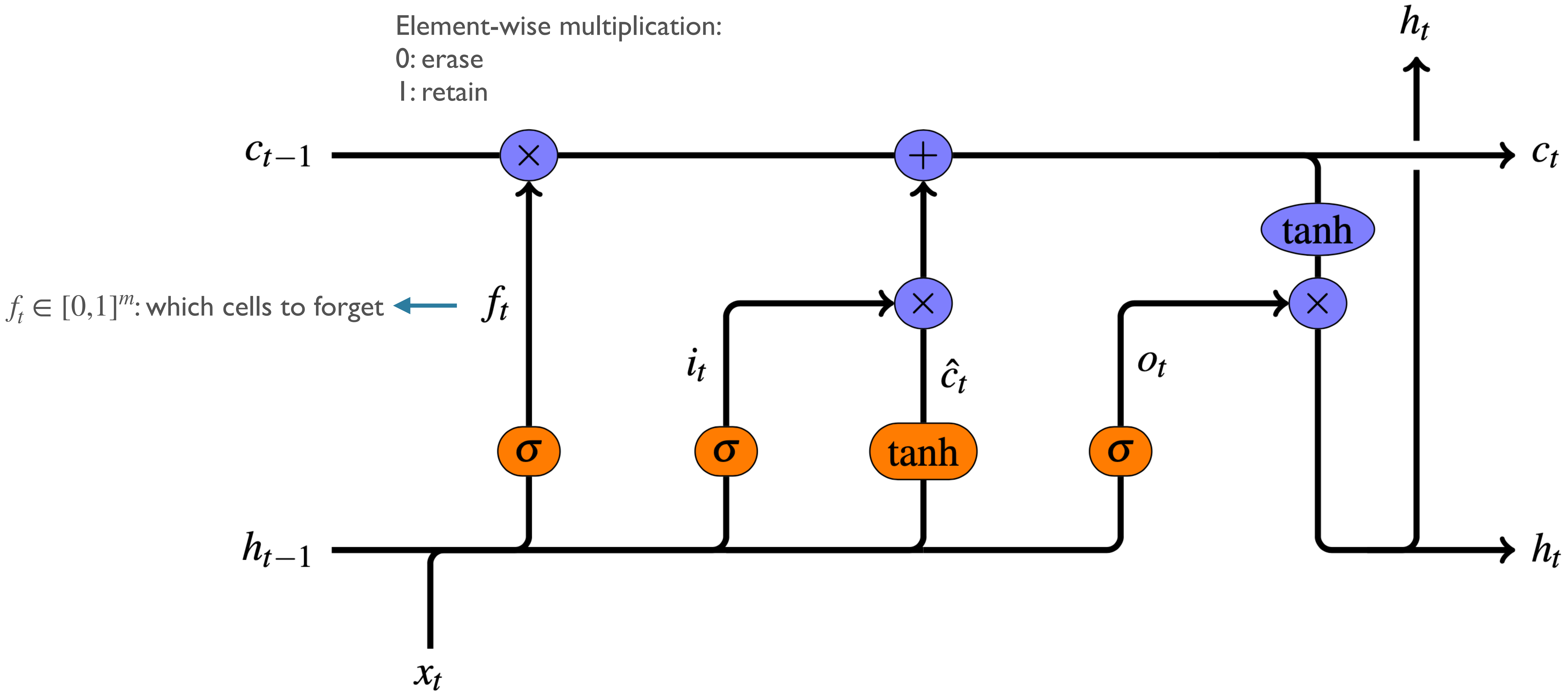


LSTMs

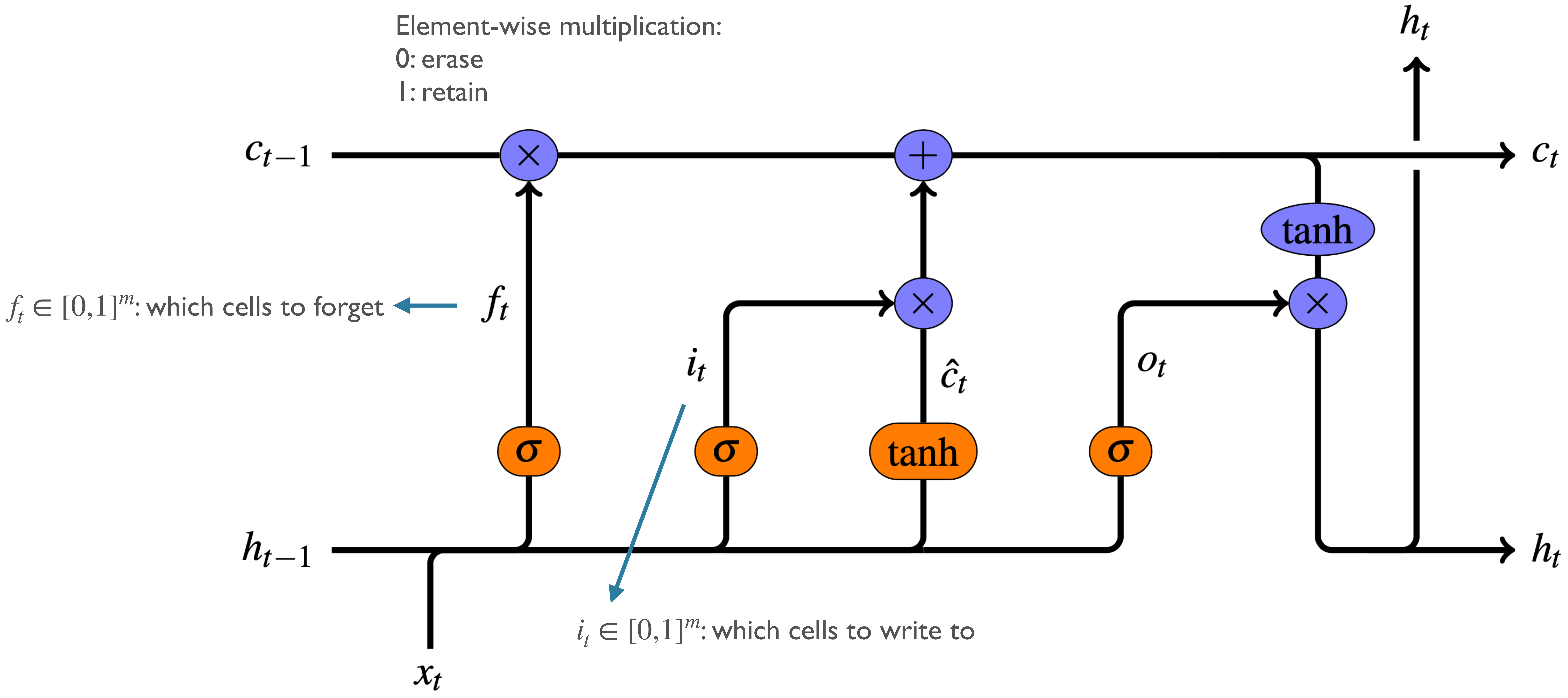
$f_t \in [0,1]^m$: which cells to forget



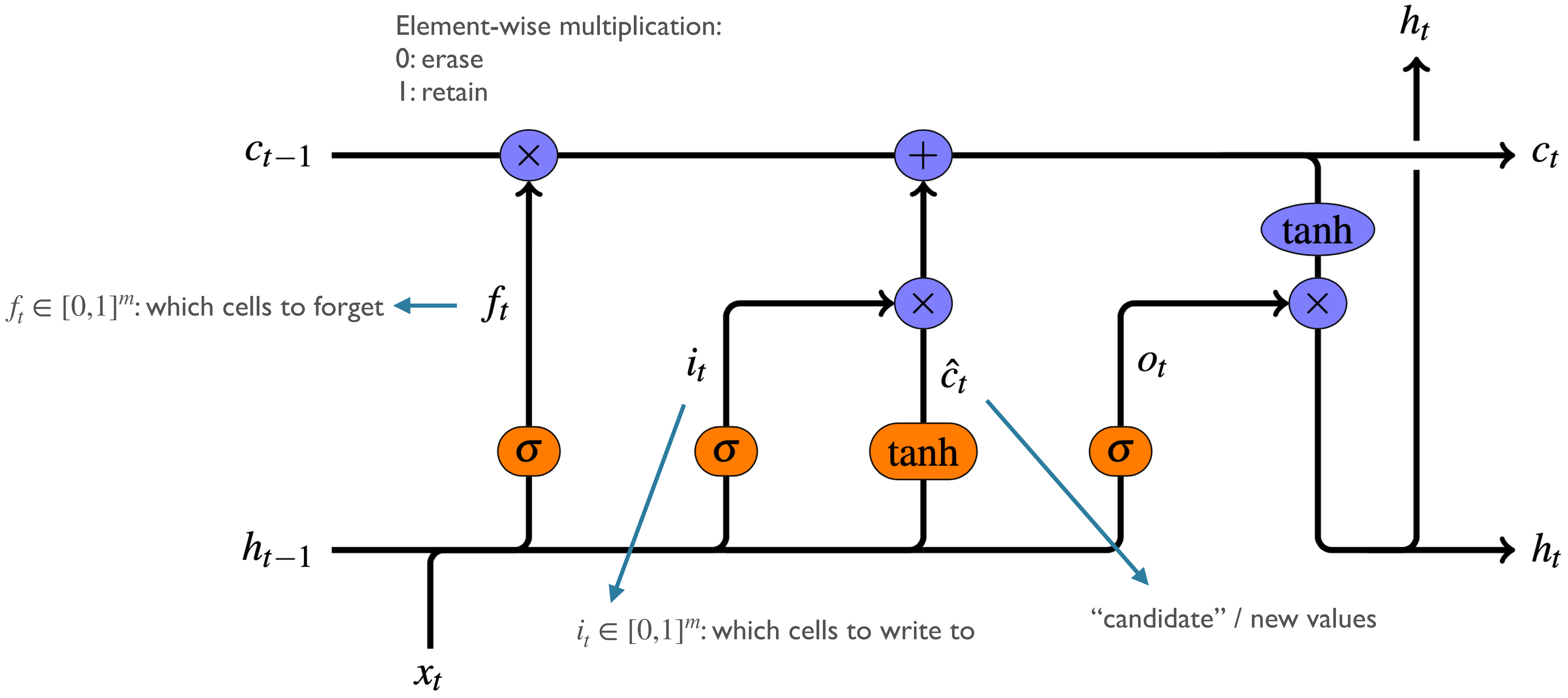
LSTMs


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

LSTMs

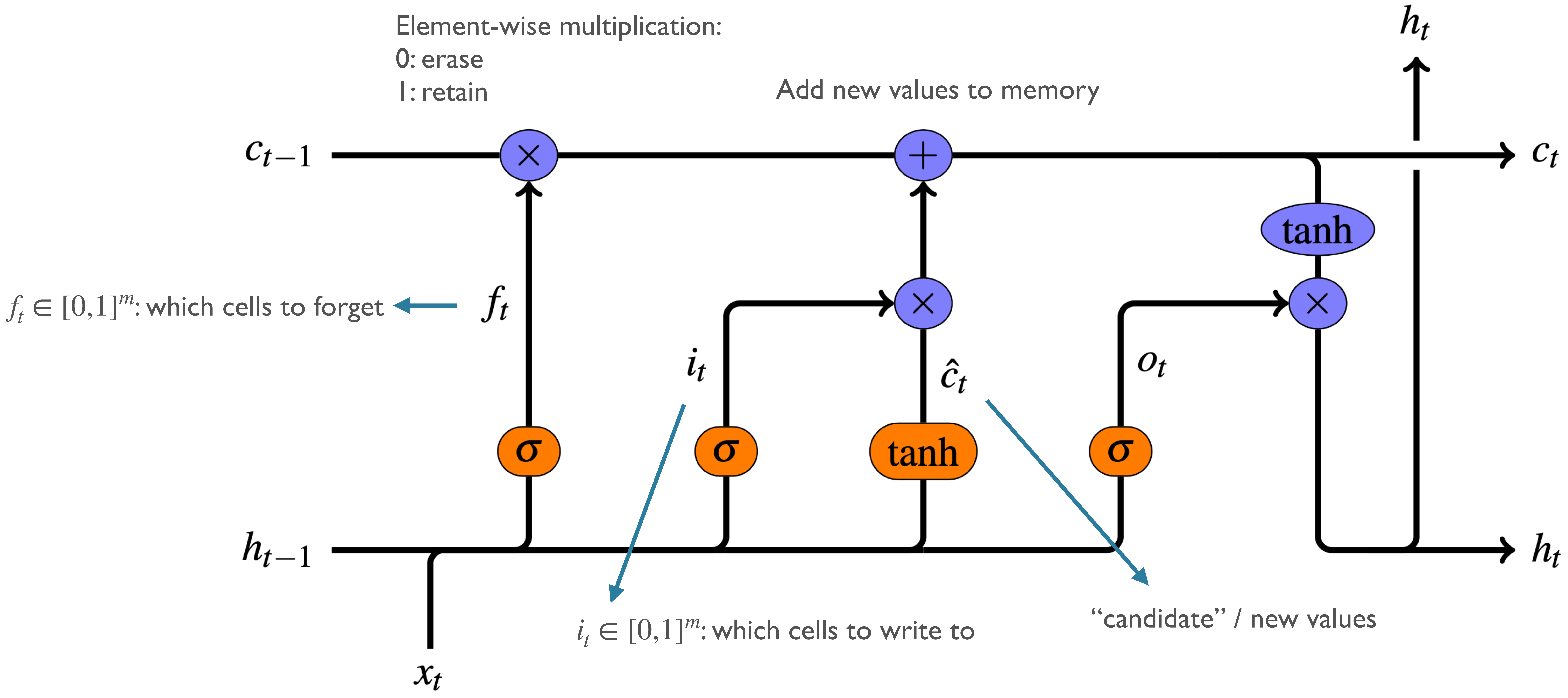

Element-wise multiplication:
0: erase
1: retain
c_{t-1}
c_t
h_t
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
$i_t \in [0,1]^m$: which cells to write to

LSTMs



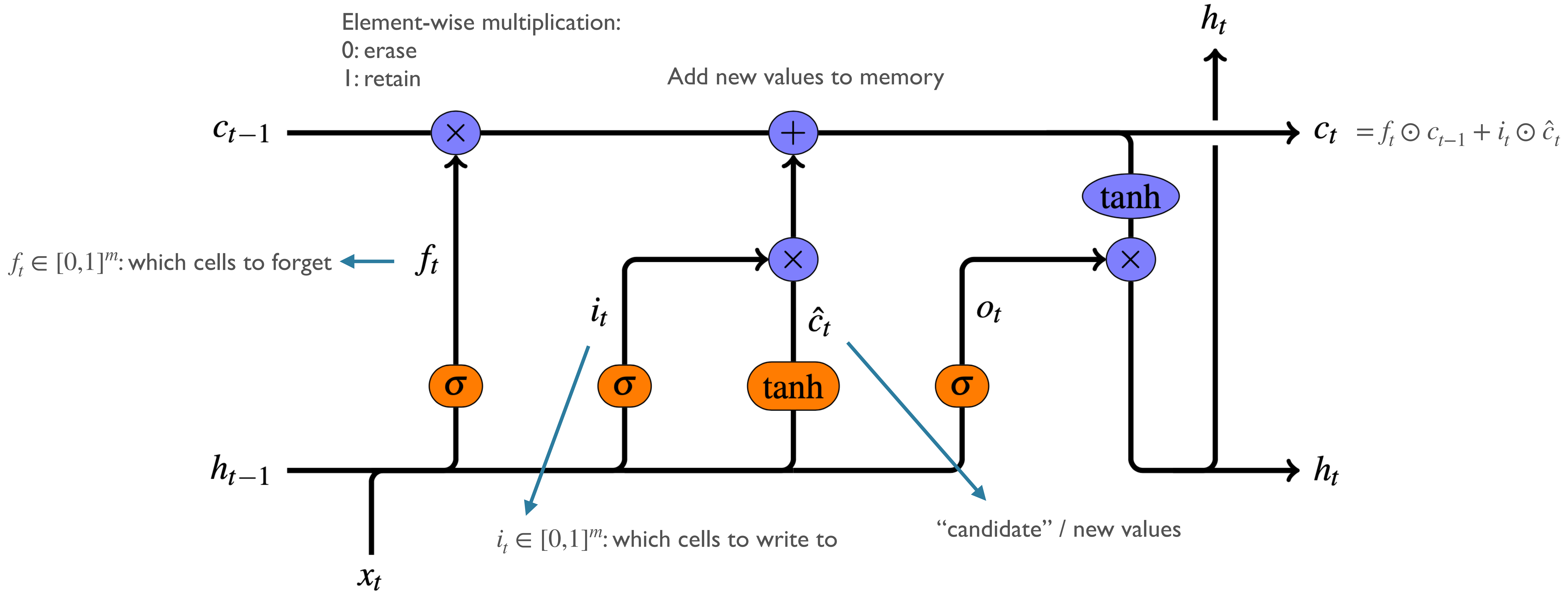
Steinert-Threlkeld and Szymanik 2019; Olah 2015

LSTMs



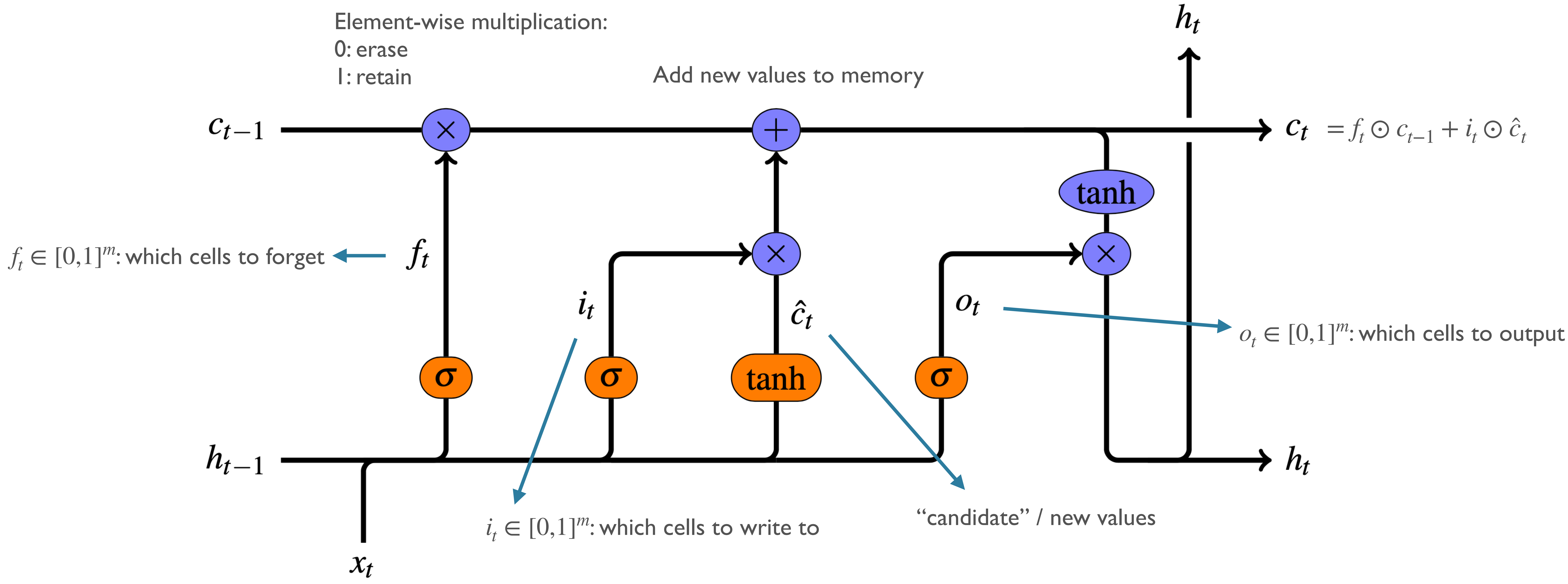
Element-wise multiplication:
0: erase
1: retain
Add new values to memory
c_{t-1}
c_t
h_t
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values

LSTMs

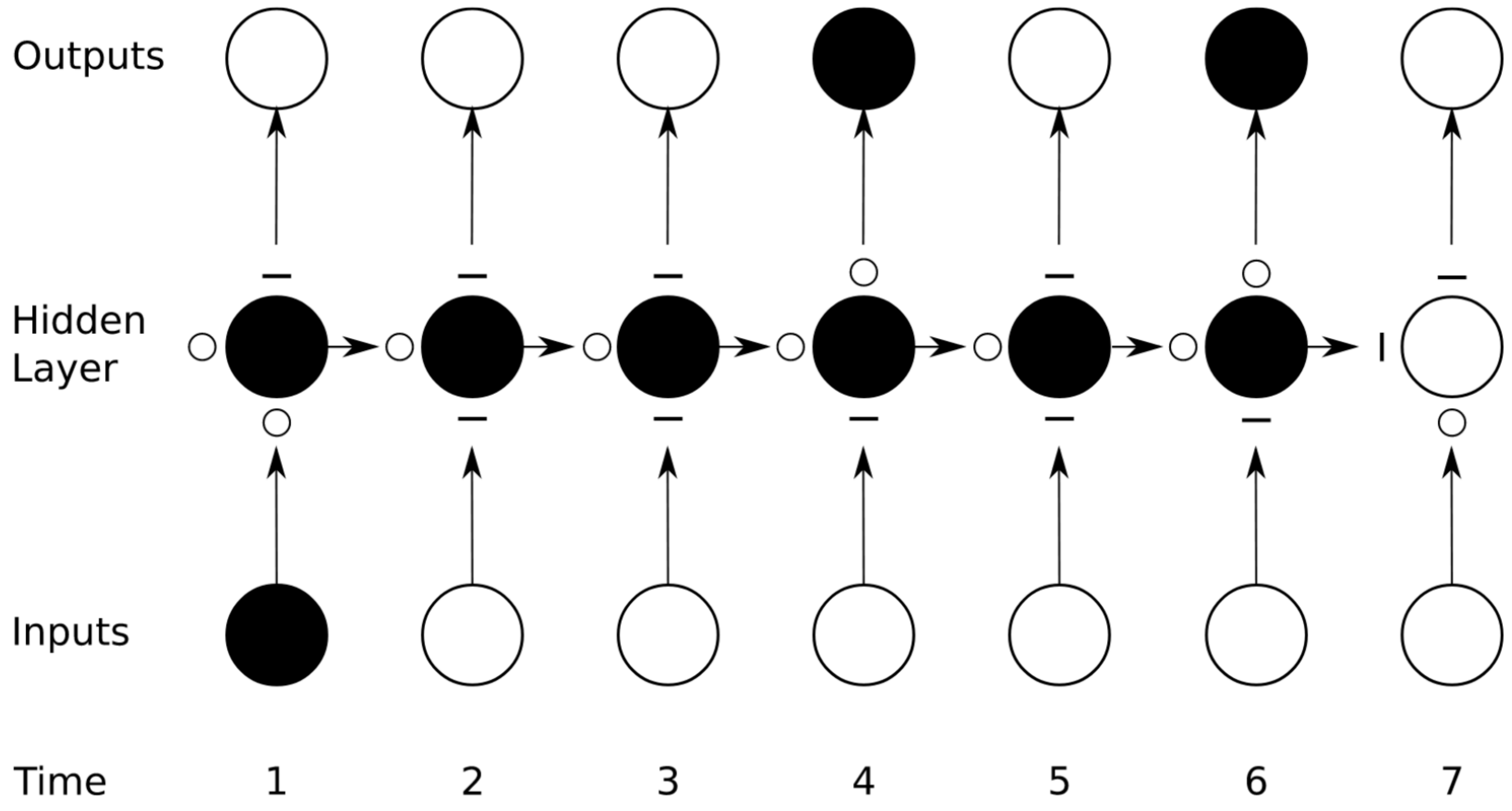


Element-wise multiplication:
0: erase
1: retain
Add new values to memory
h_t
c_{t-1}
c_t $= f_t \odot c_{t-1} + i_t \odot \hat{c}_t$
tanh
$f_t \in [0,1]^m$: which cells to forget
f_t
i_t
$\hat{c}_t$
o_t
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
$i_t \in [0,1]^m$: which cells to write to
"candidate" / new values

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

LSTMs solve vanishing gradients



[Graves 2012]

Gated Recurrent Unit (GRU)

- Cho et al 2014: gated like LSTM, but no separate memory cell
  - "Collapses" execution/control and memory
- Fewer gates = fewer parameters, higher speed
  - Update gate
  - Reset gate

$$u_t = \sigma(W_u h_{t-1} + U_u x_t + b_u)$$

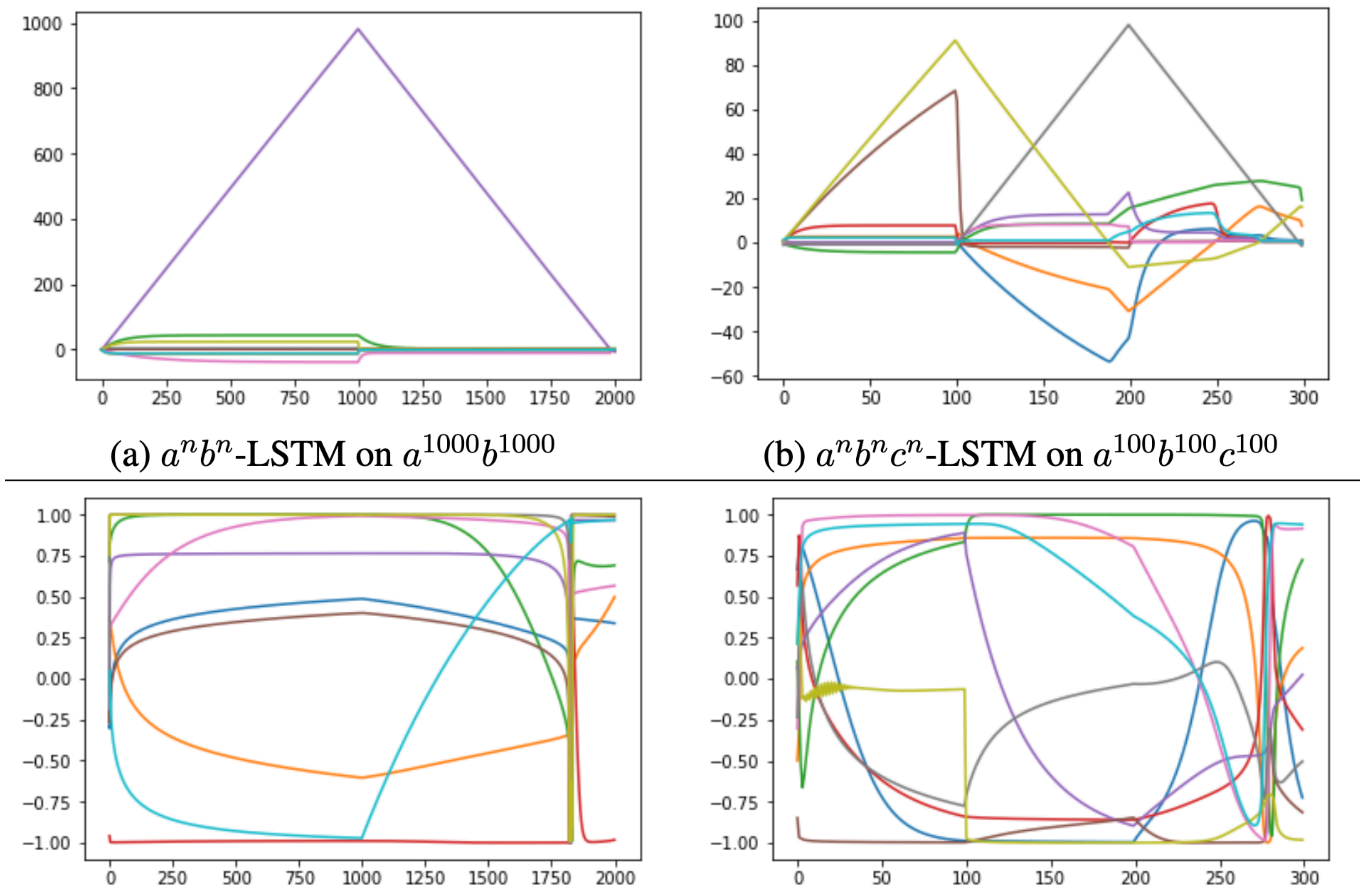
$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r)$$

$$\tilde{h}_t = \tanh(W_h(r_t \odot h_t) + U_h x_t + b_h)$$

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

LSTM vs GRU

- Generally: LSTM a good default choice
  - GRU can be used if speed and fewer parameters are important
- Full differences between them not fully understood
- Performance often comparable, but: LSTMs can store unboundedly large values in memory, and seem to e.g. count better



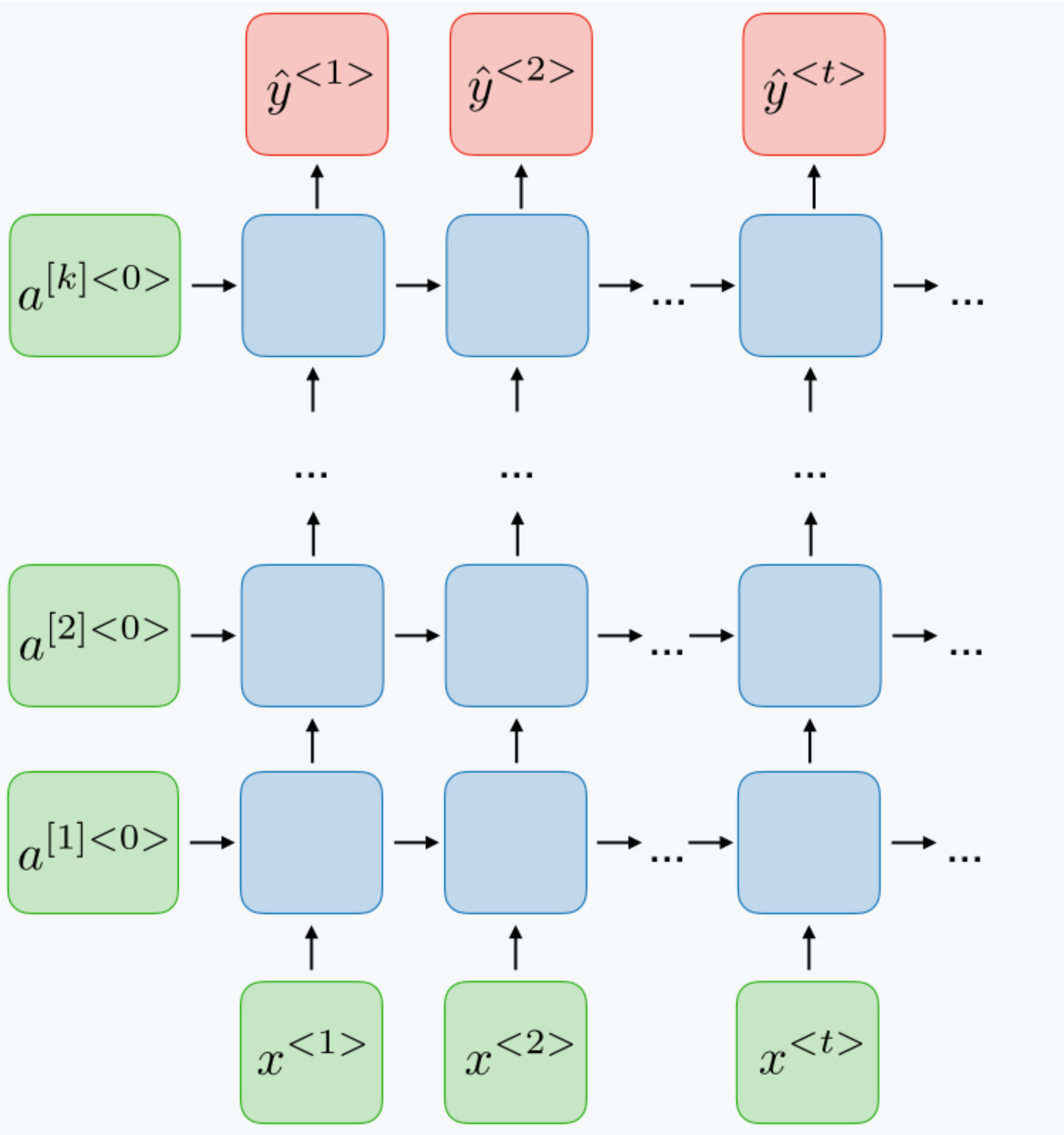
(a) a^nb^n-LSTM on $a^{1000}b^{1000}$ (b) $a^nb^nc^n$-LSTM on $a^{100}b^{100}c^{100}$

(c) a^nb^n-GRU on $a^{1000}b^{1000}$ (d) $a^nb^nc^n$-GRU on $a^{100}b^{100}c^{100}$

[source]

Two Extensions

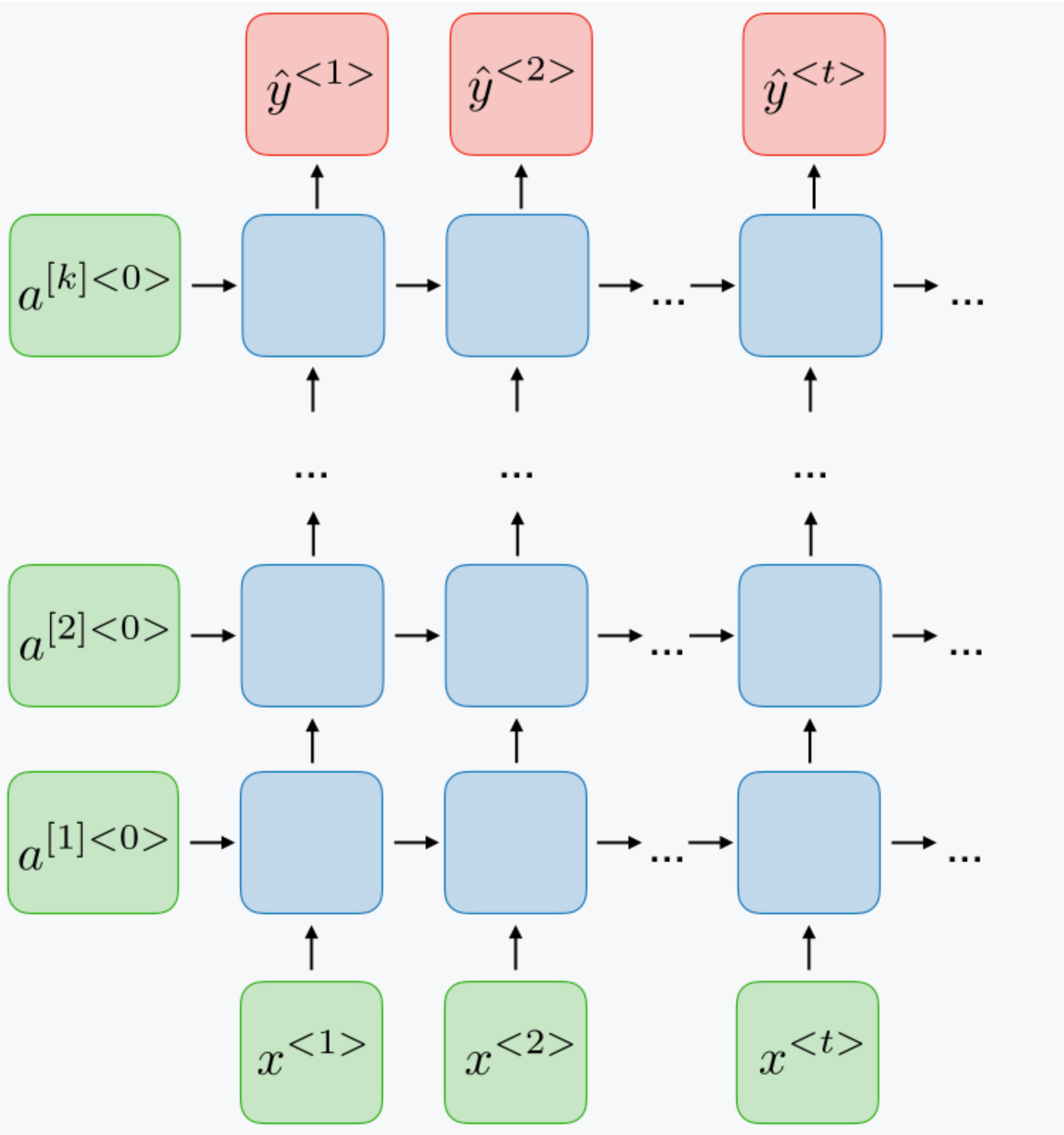
- Deep RNNs:



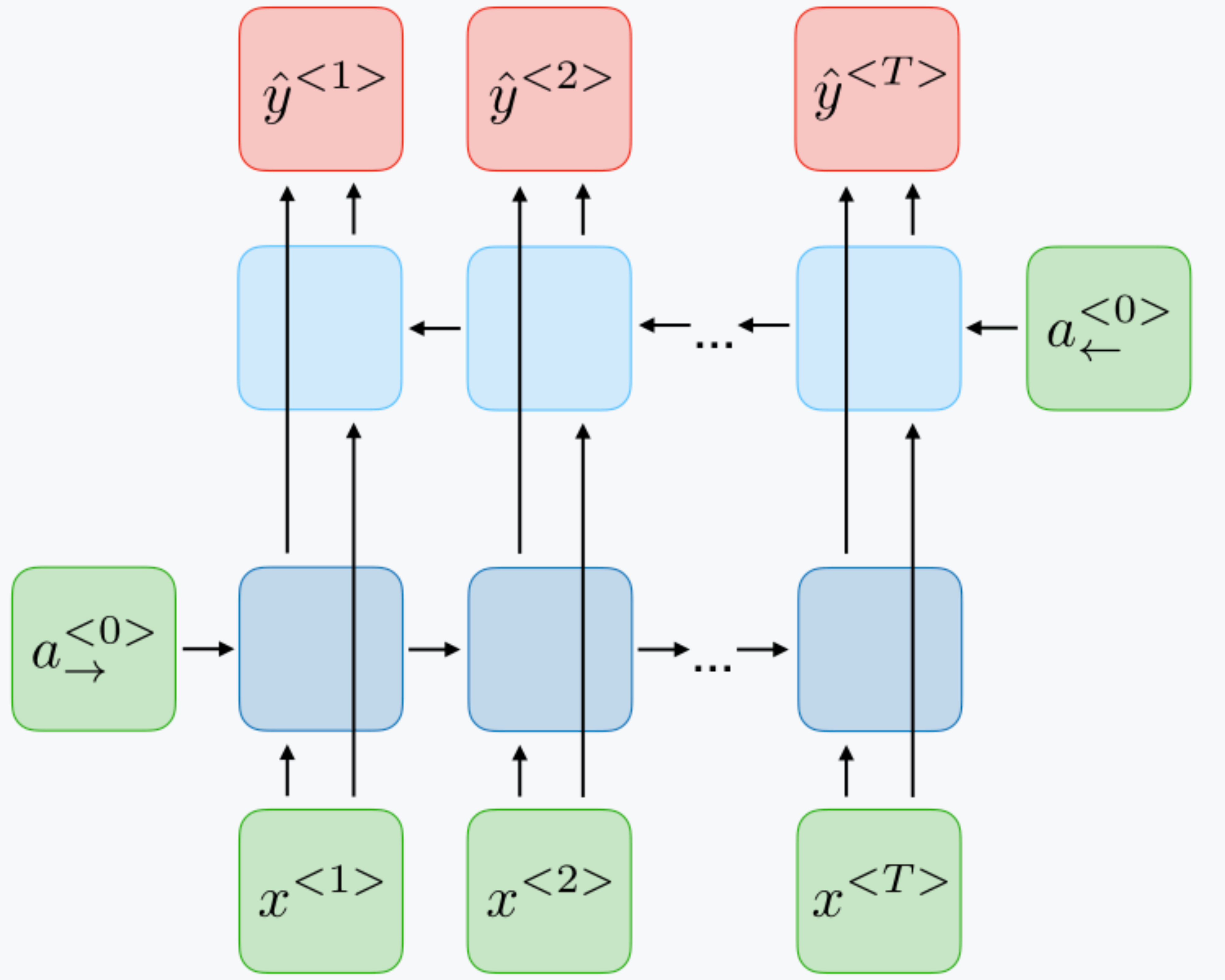
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



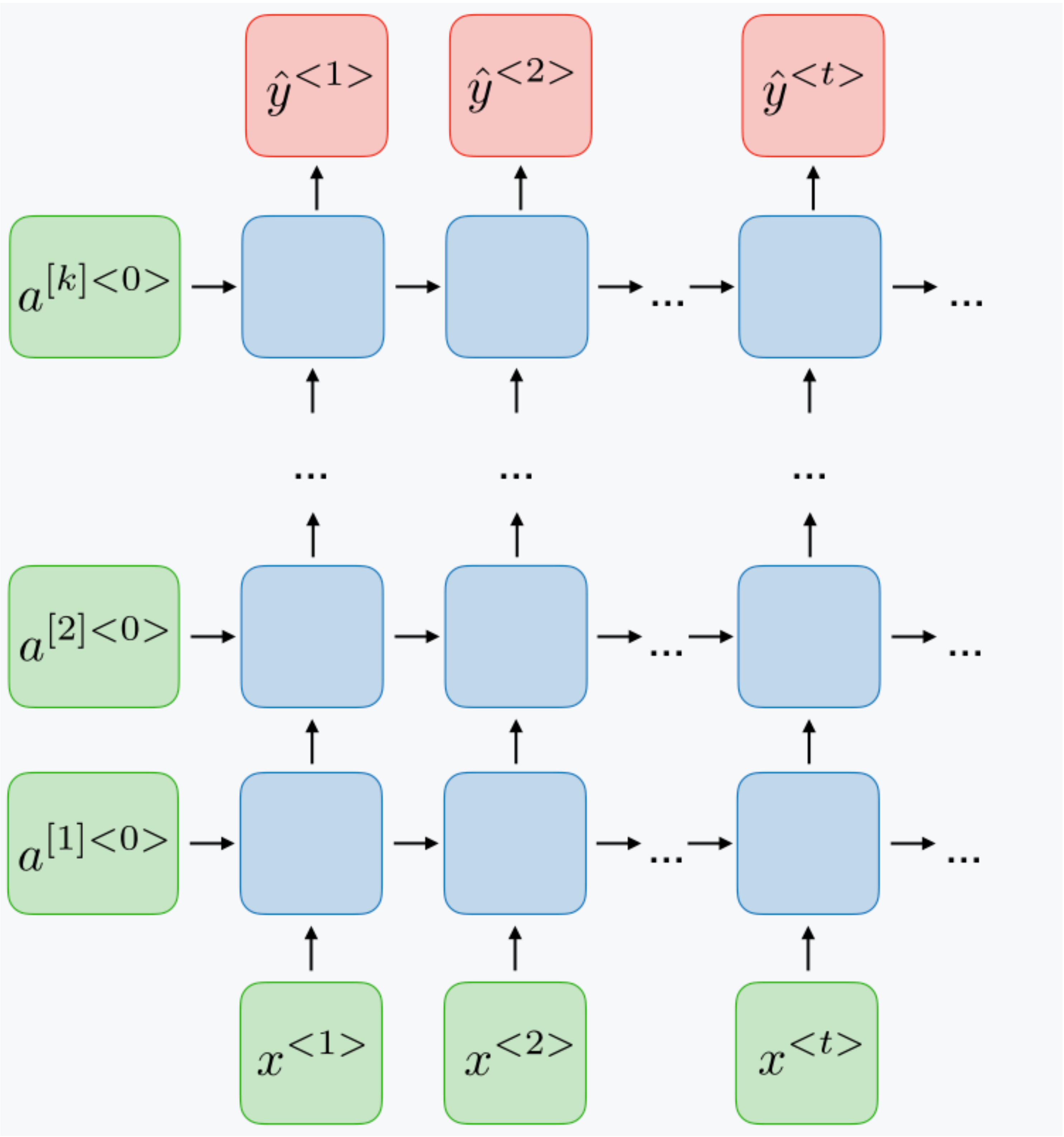
- Bidirectional RNNs:



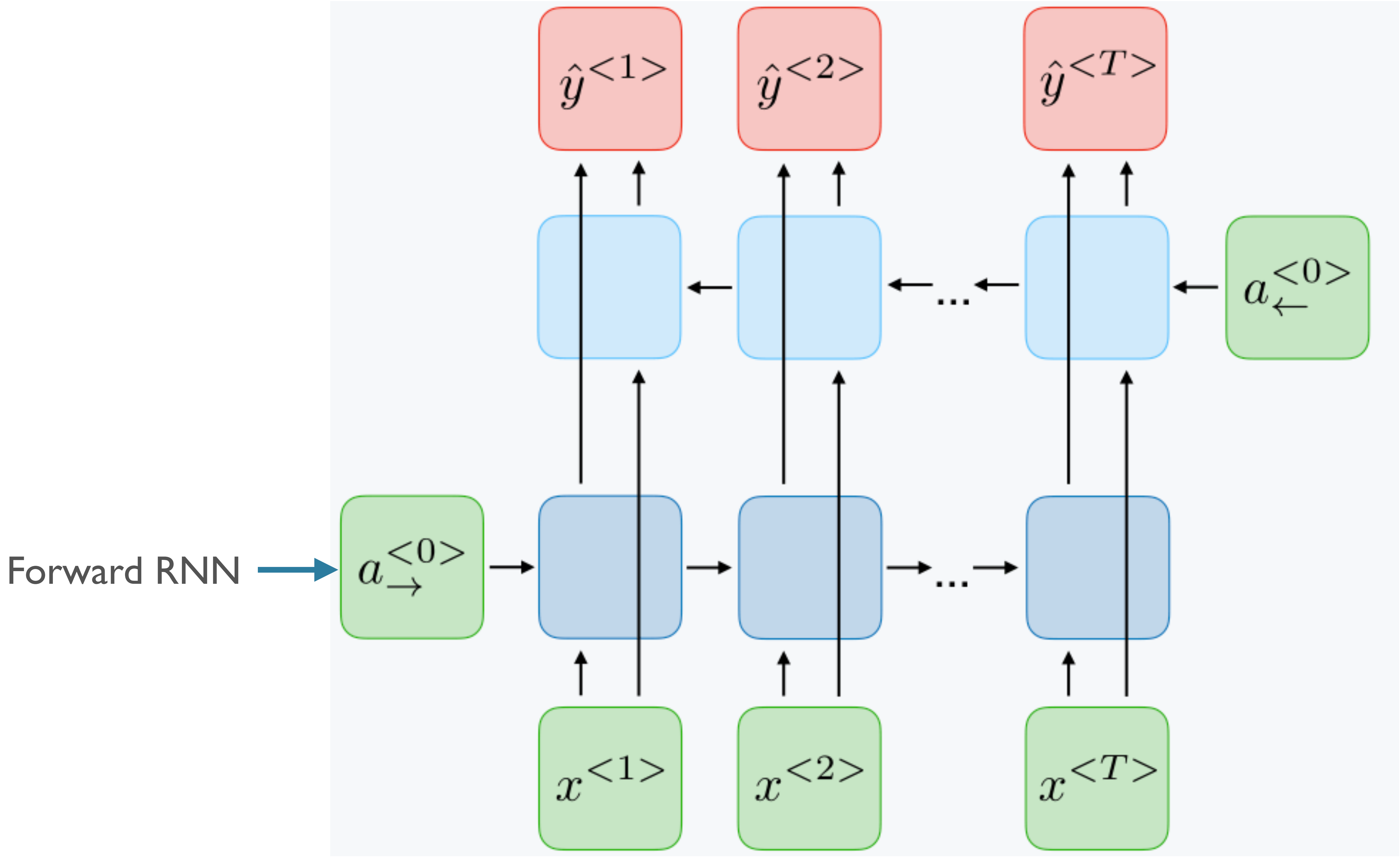
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



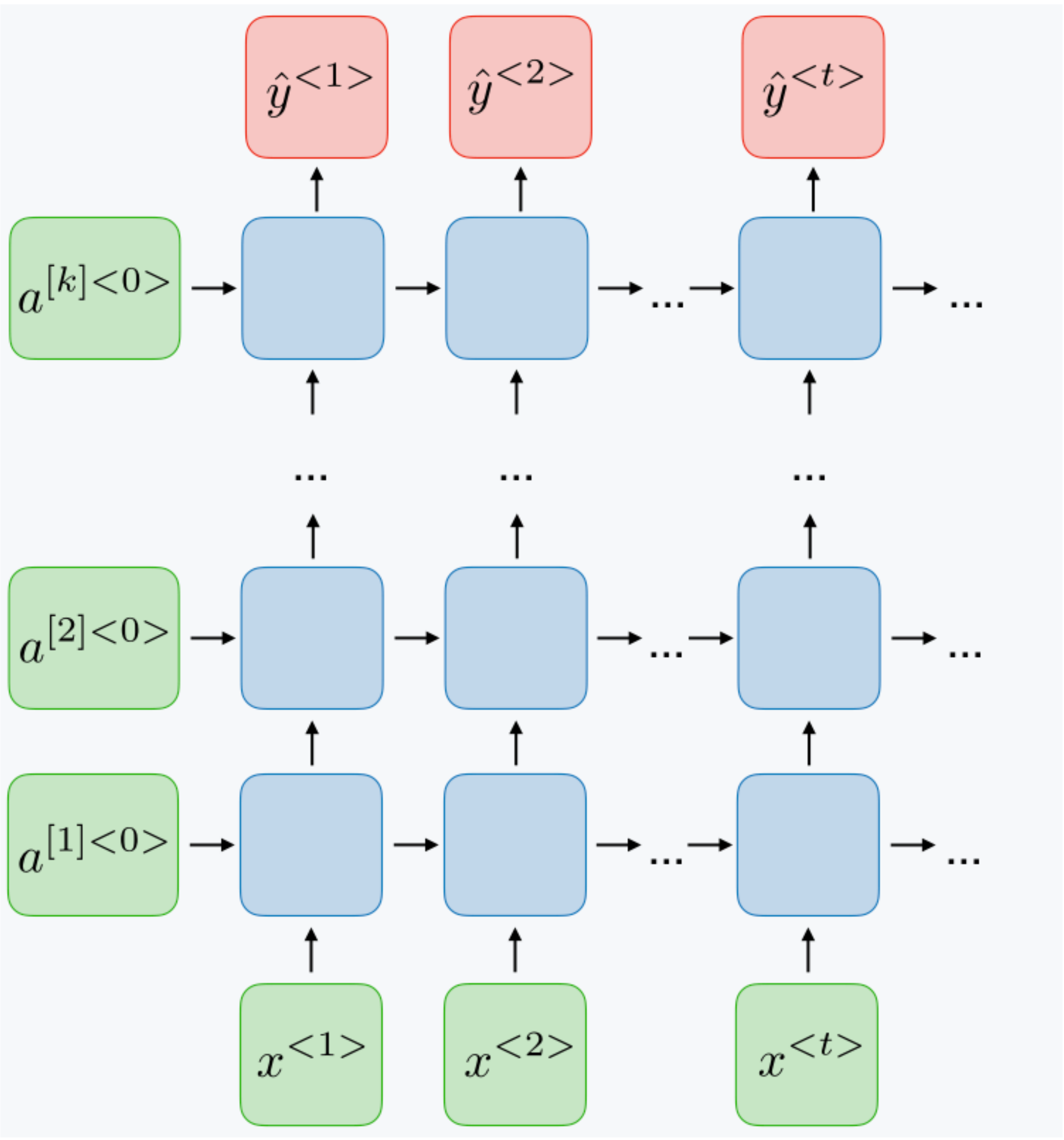
- Bidirectional RNNs:



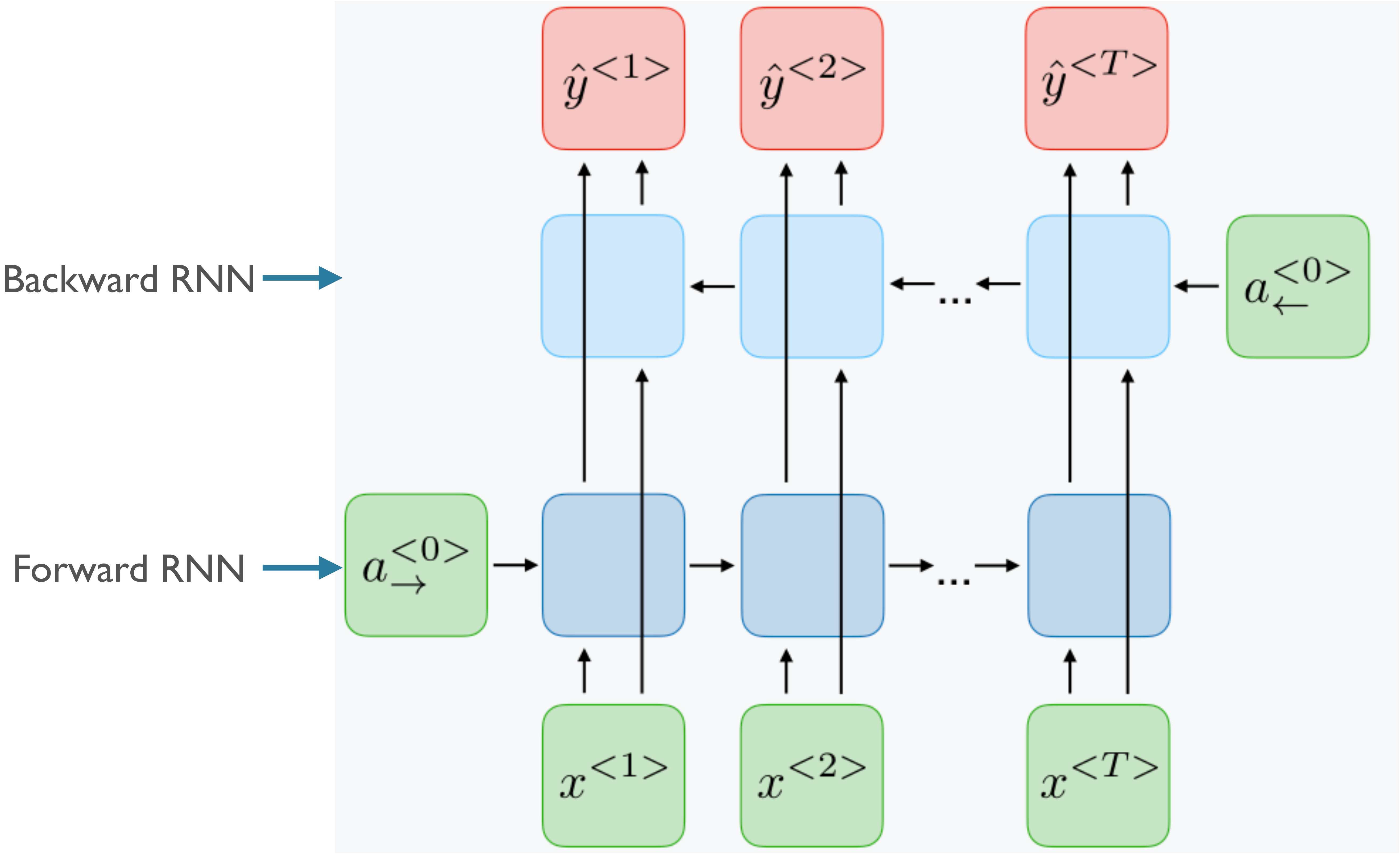
Source: RNN cheat sheet

Two Extensions

- Deep RNNs:



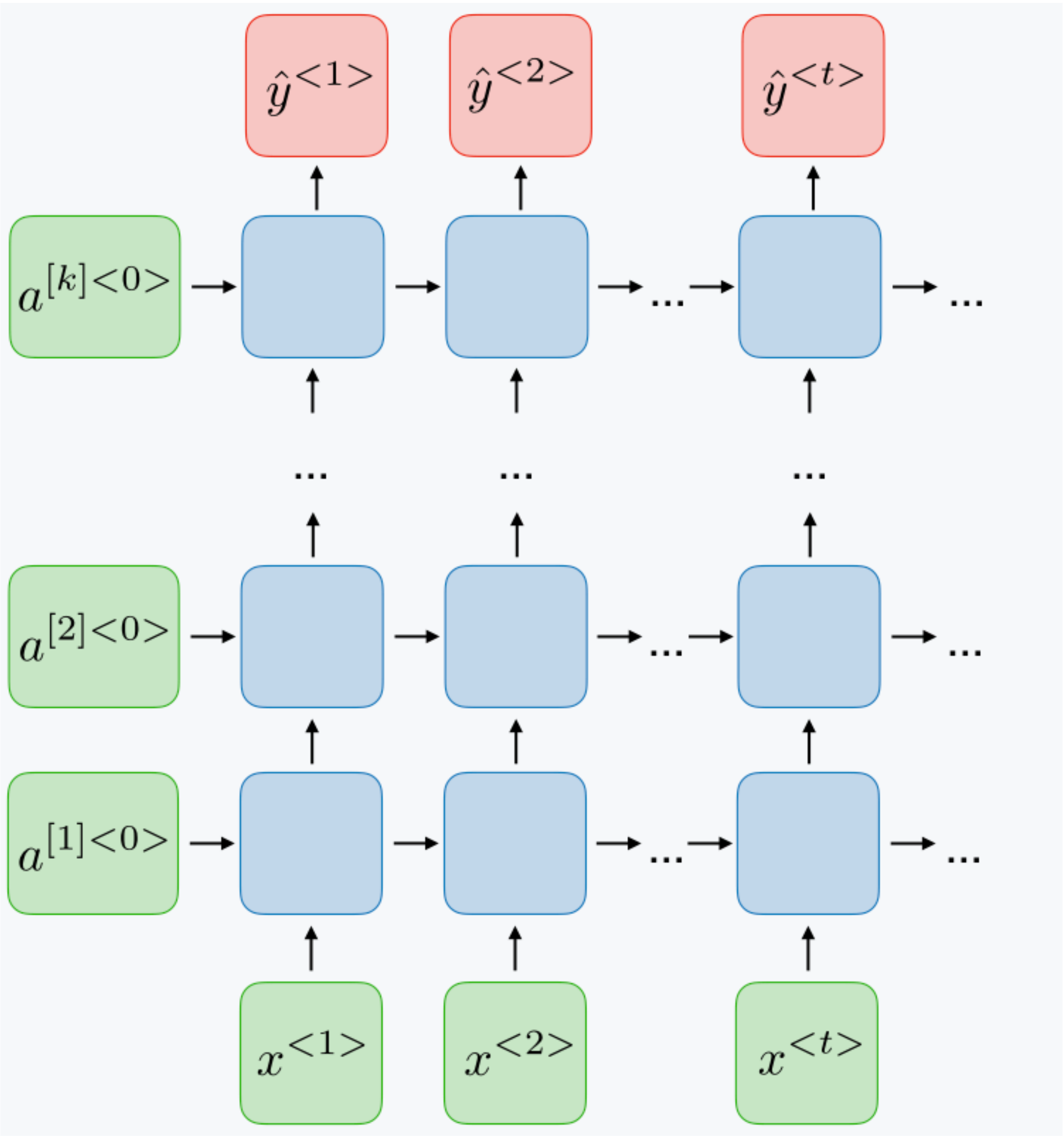
- Bidirectional RNNs:



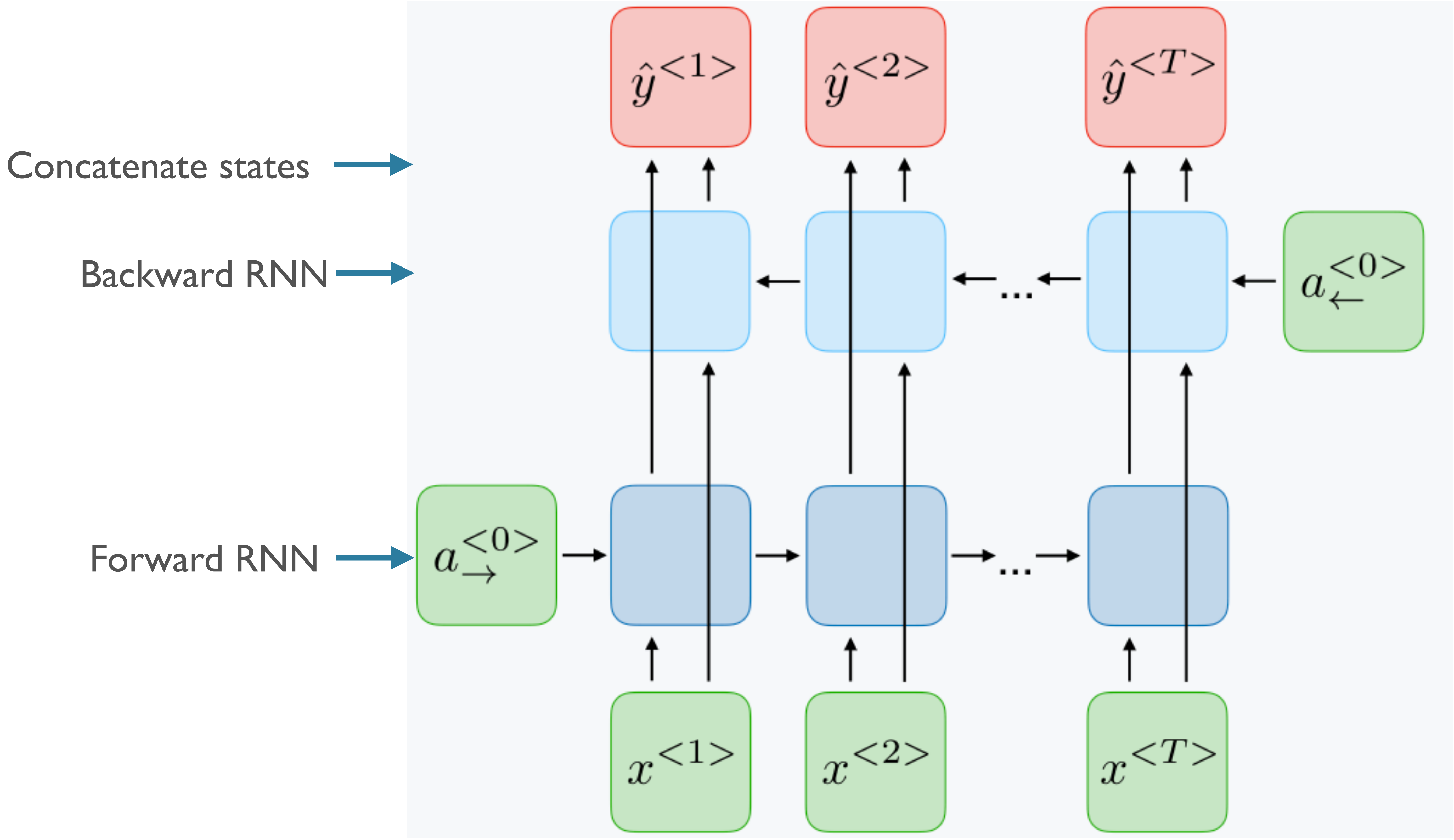
[Source: RNN cheat sheet]

Two Extensions

- Deep RNNs:



- Bidirectional RNNs:



Source: RNN cheat sheet

"The BiLSTM Hegemony"

- Chris Manning, in 2017:

To a first approximation,
the de facto consensus in NLP in 2017 is
that no matter what the task,
you throw a BiLSTM at it, with
attention if you need information flow

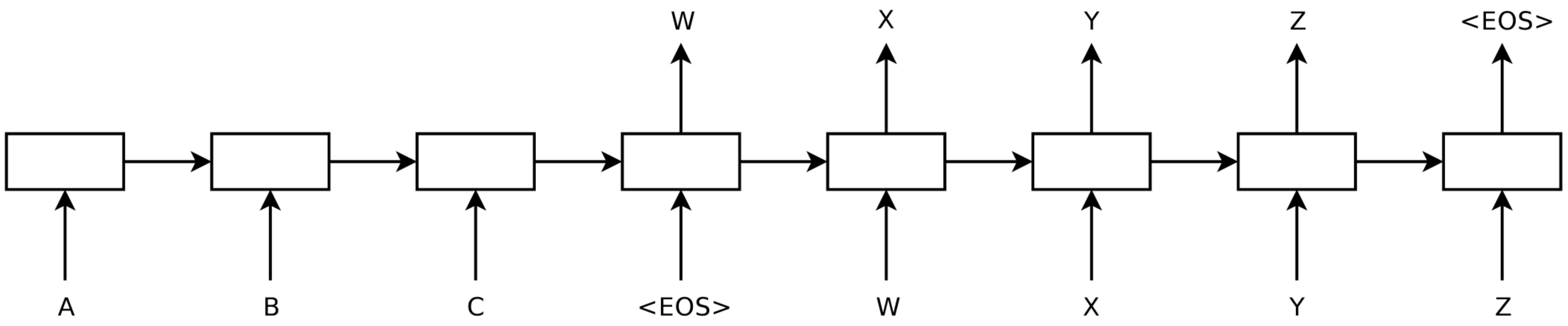
source

Seq2Seq + attention

Sequence to sequence problems

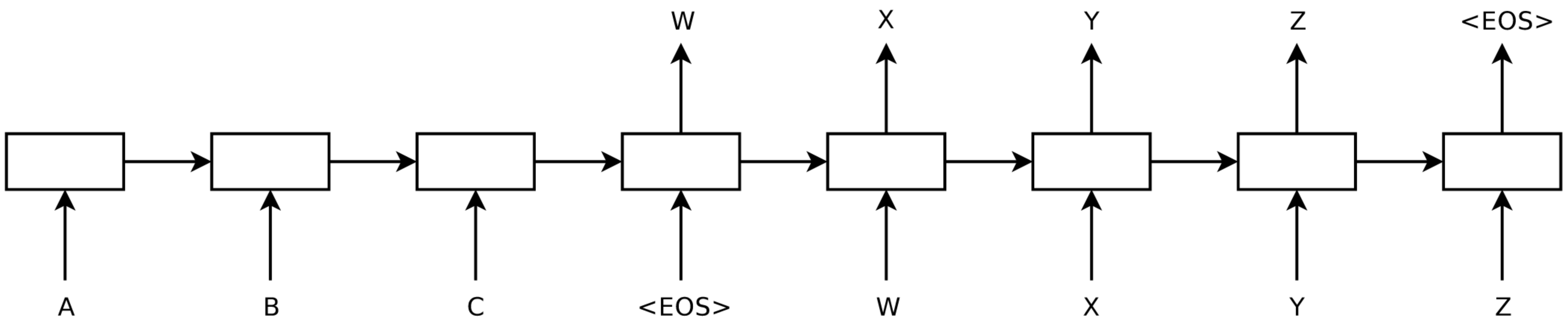
- Many NLP tasks can be construed as *sequence-to-sequence* problems
  - Machine translations: sequence of source lang tokens to sequence of target lang tokens
  - Parsing: “Shane talks.” —> “(S (NP (N Shane)) (VP V talks))”
    - Incl semantic parsing
  - Summarization
  - …
- NB: not the same as *tagging*, which assigns a label to each position in a given sequence

seq2seq architecture [e.g. NMT]



Sutskever et al 2013

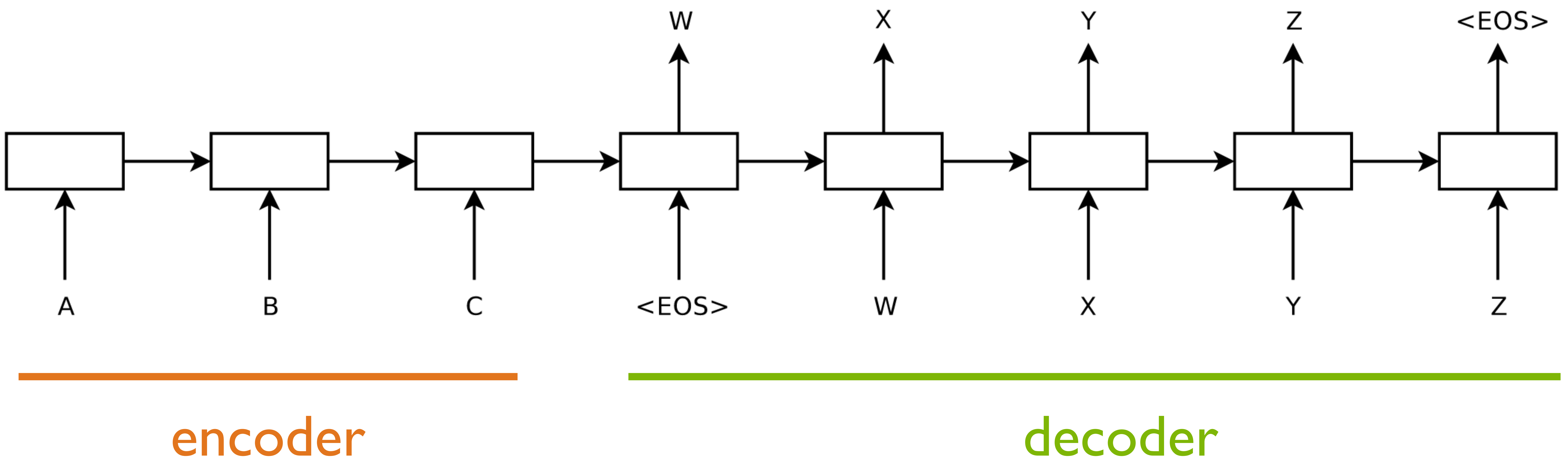
seq2seq architecture [e.g. NMT]



encoder

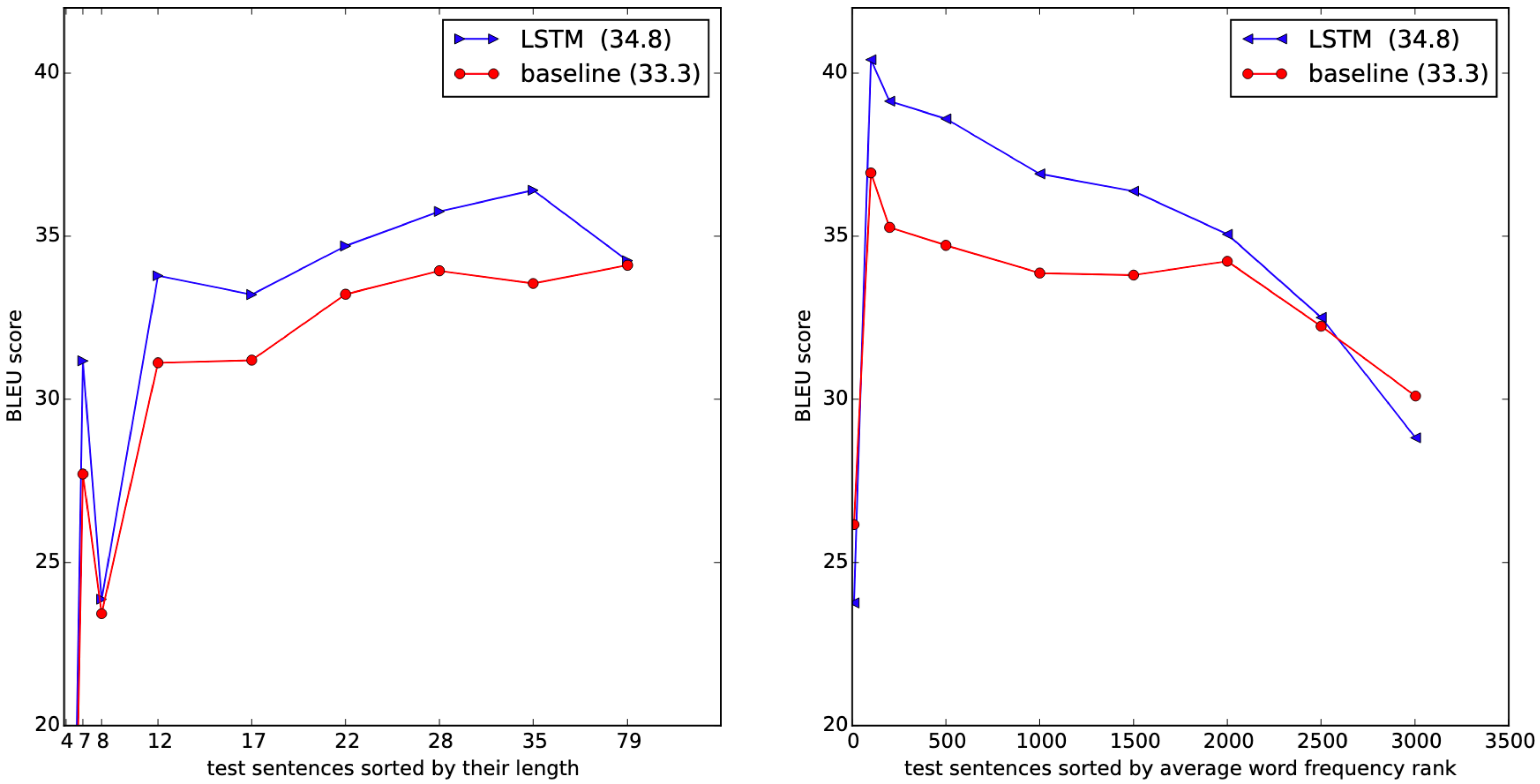
Sutskever et al 2013

seq2seq architecture [e.g. NMT]

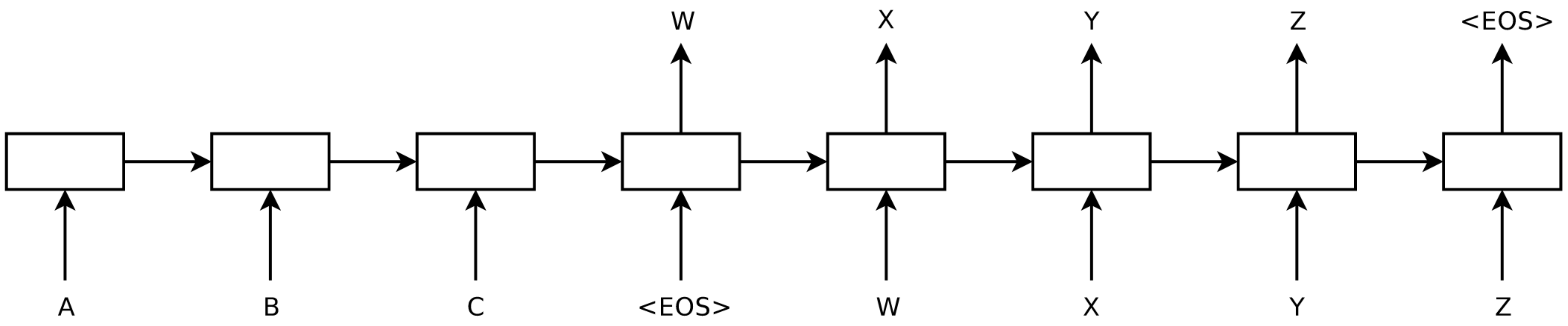


Sutskever et al 2013

seq2seq results

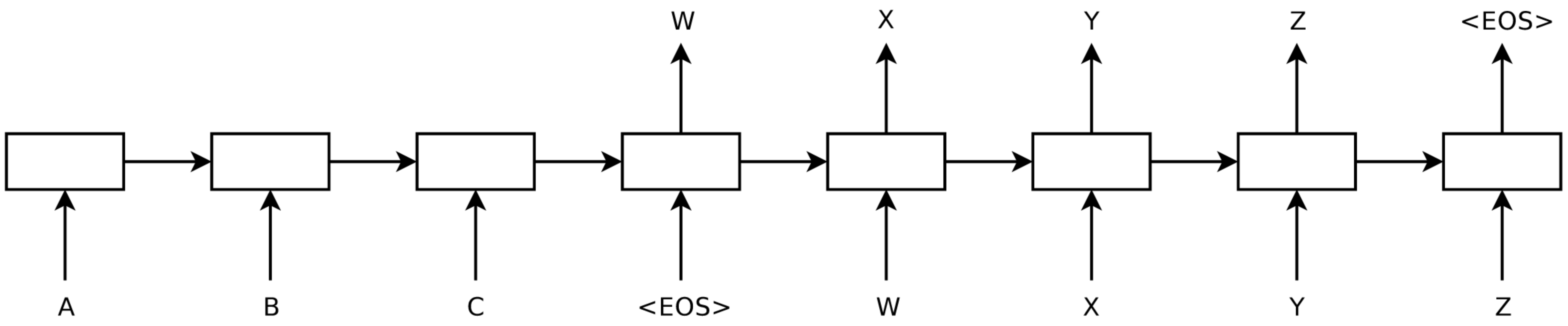


seq2seq architecture: problem



Sutskever et al 2013

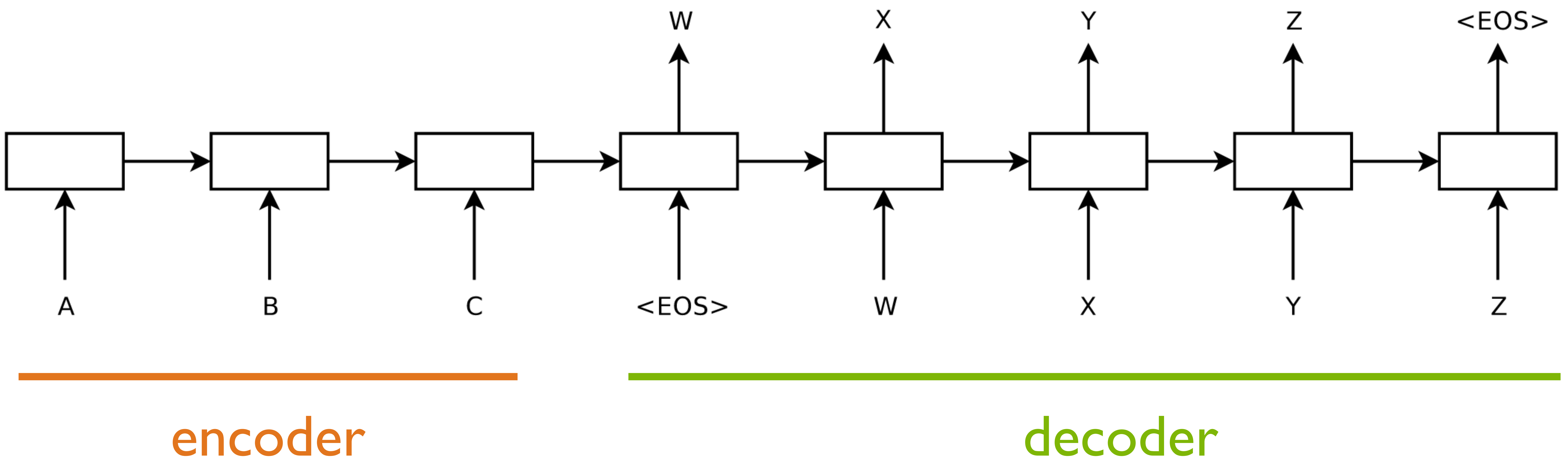
seq2seq architecture: problem



encoder

Sutskever et al 2013

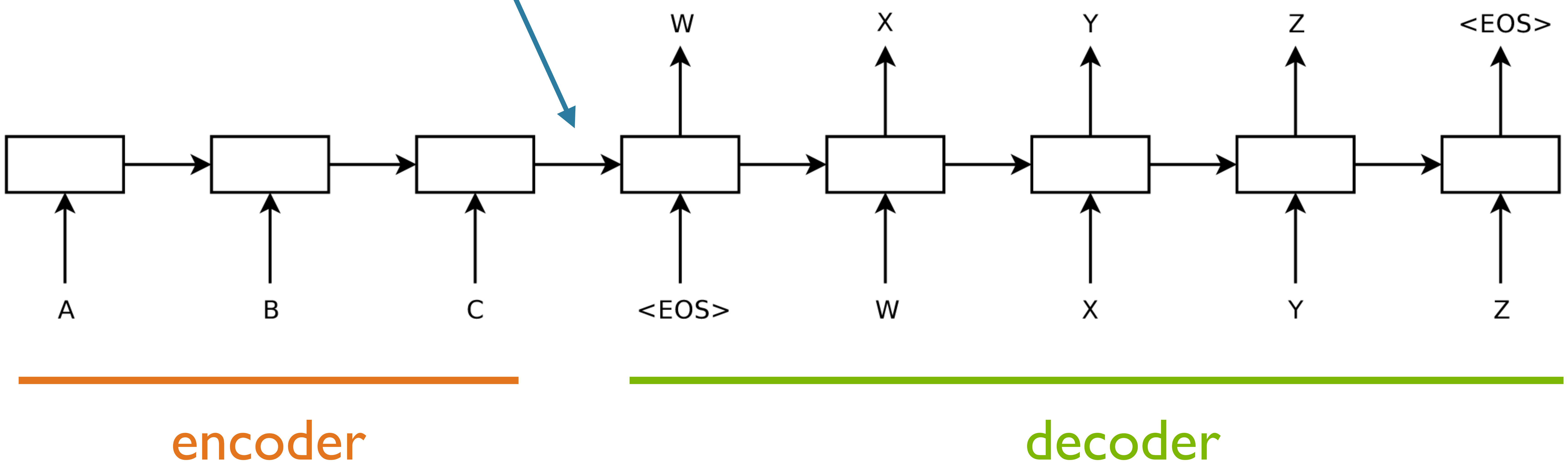
seq2seq architecture: problem



Sutskever et al 2013

seq2seq architecture: problem

Decoder can only see info in this one vector
all info about source must be "crammed" into here

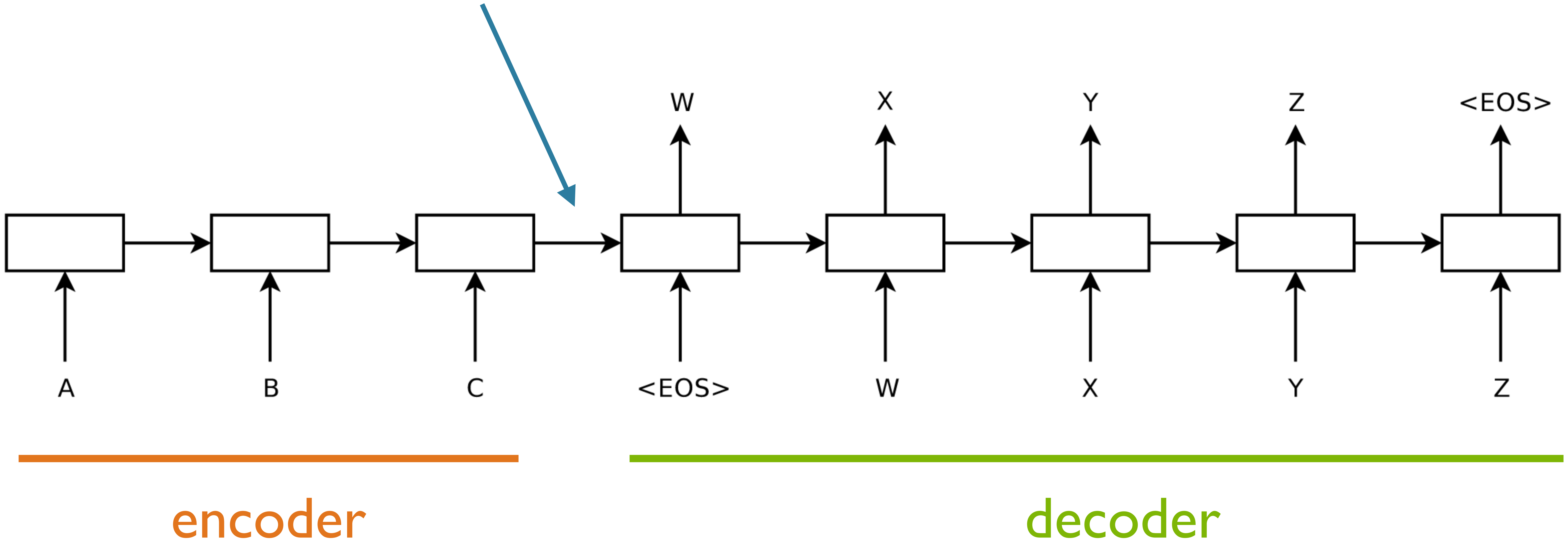


Sutskever et al 2013

seq2seq architecture: problem

Decoder can only see info in this one vector
all info about source must be "crammed" into here

Mooney 2014: "You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!"



Sutskever et al 2013

Neural Machine Translation by Jointly Learning to Align and Translate

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

Abstract

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

source

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

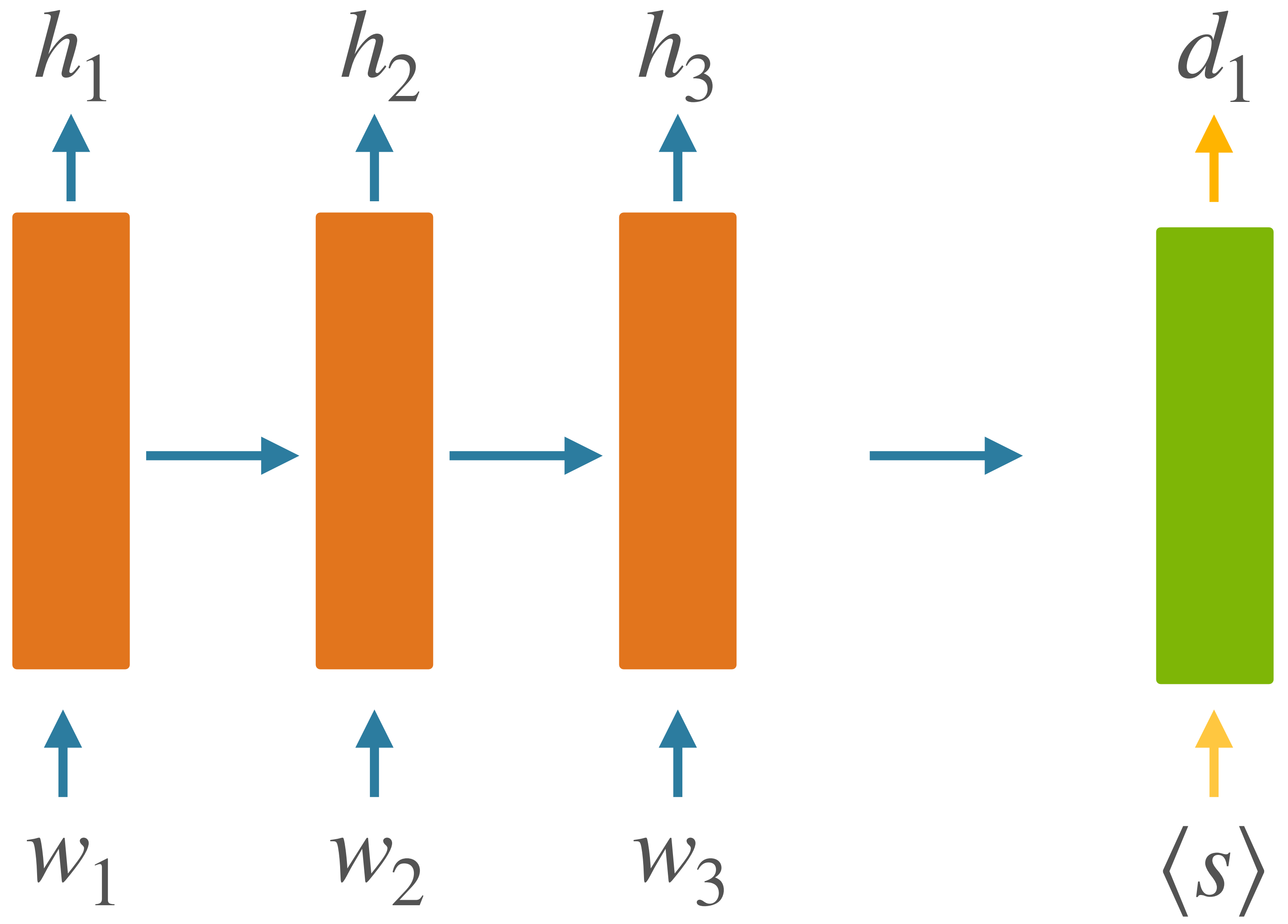
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

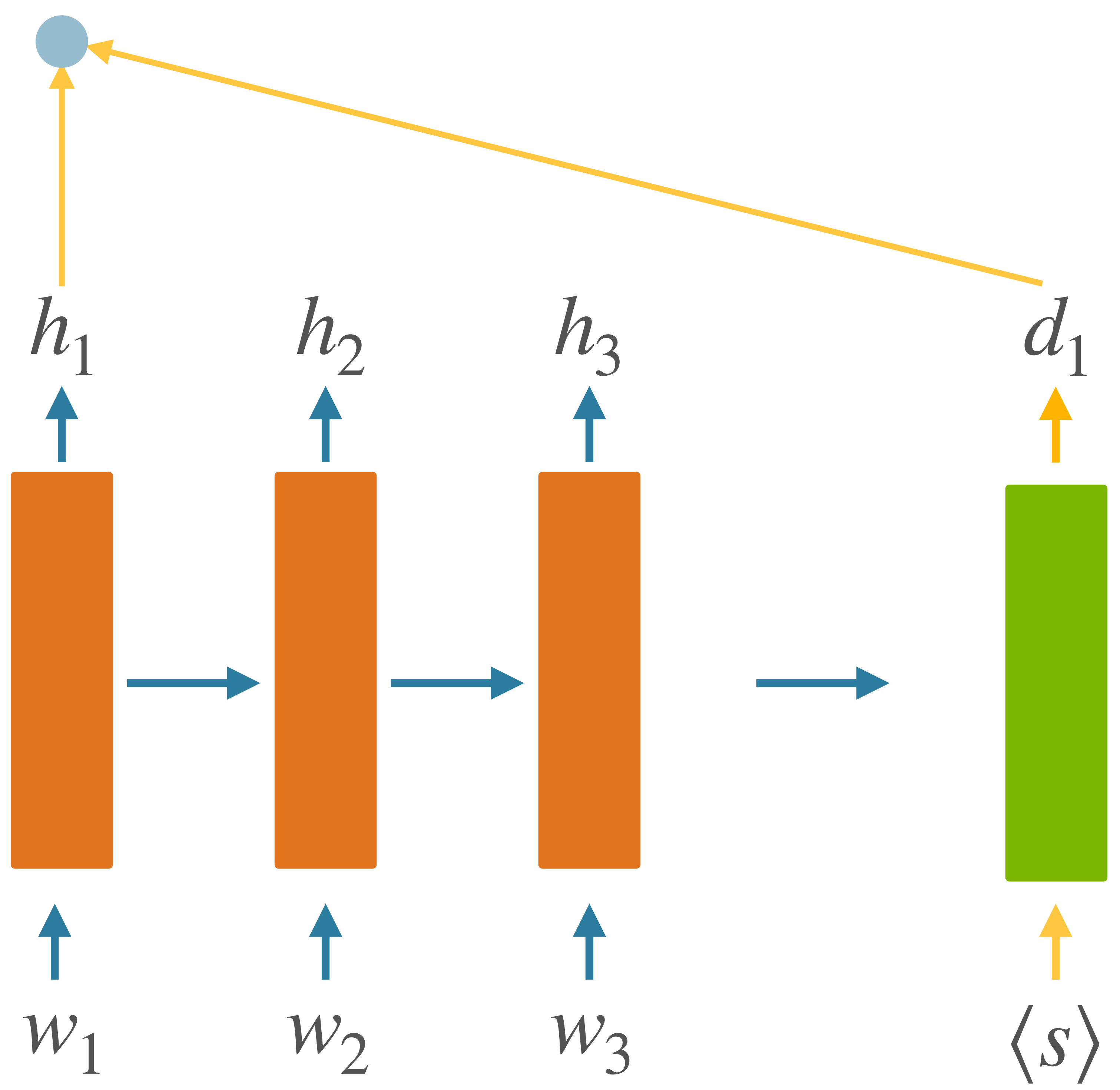
source

Adding Attention



Badhanau et al 2014

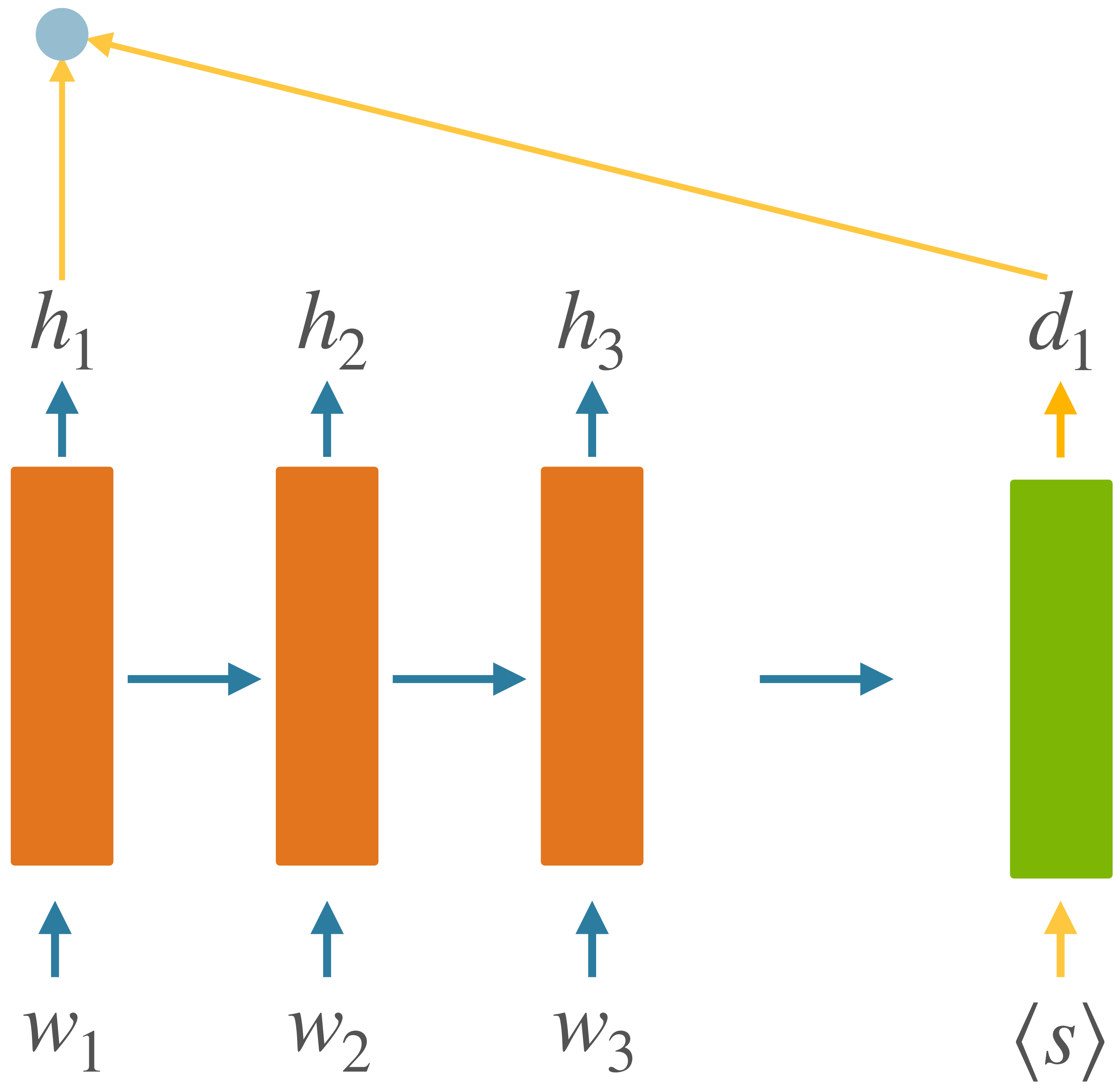
Adding Attention



Badhanau et al 2014

Adding Attention

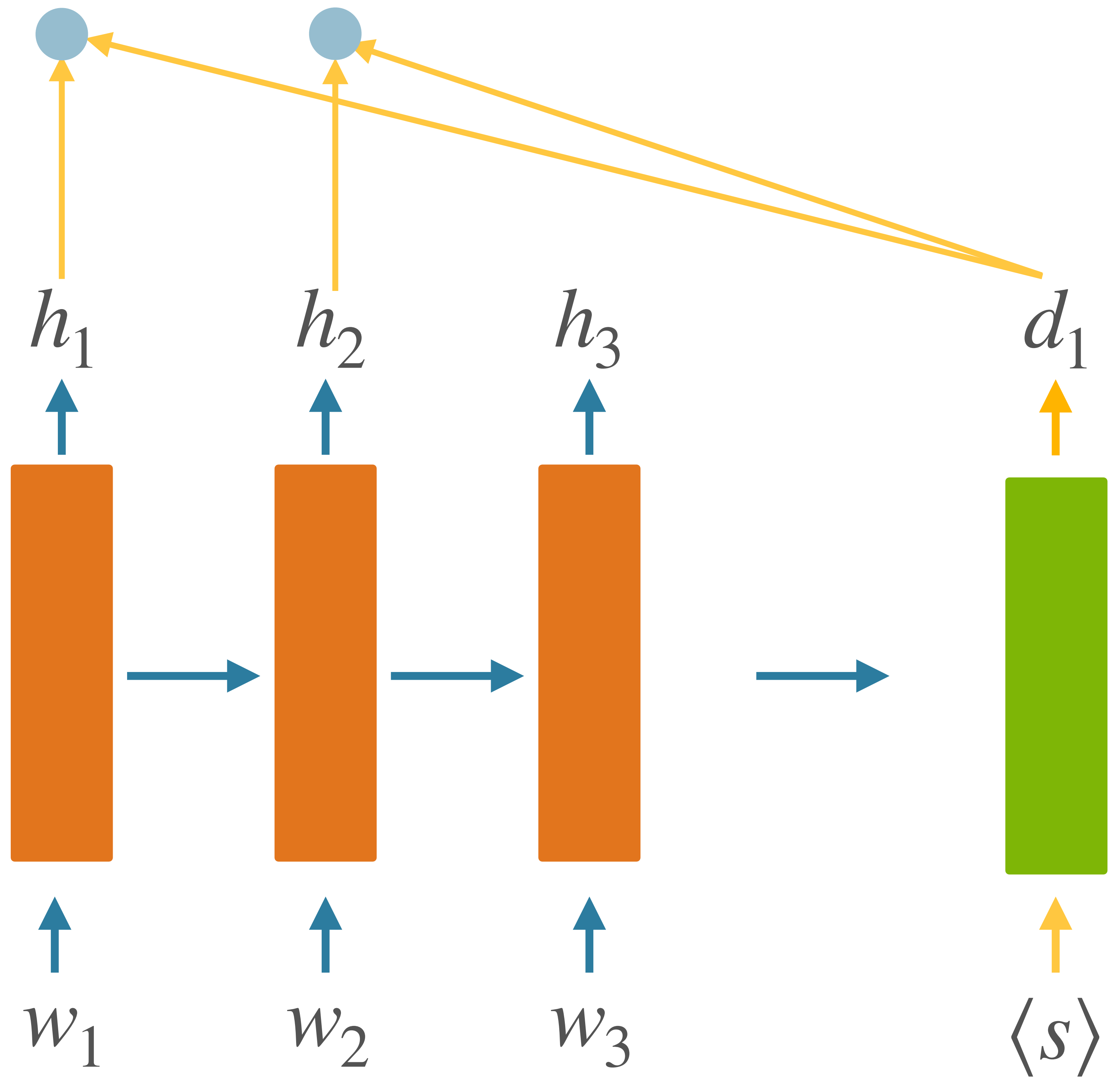
$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



Badhanau et al 2014

Adding Attention

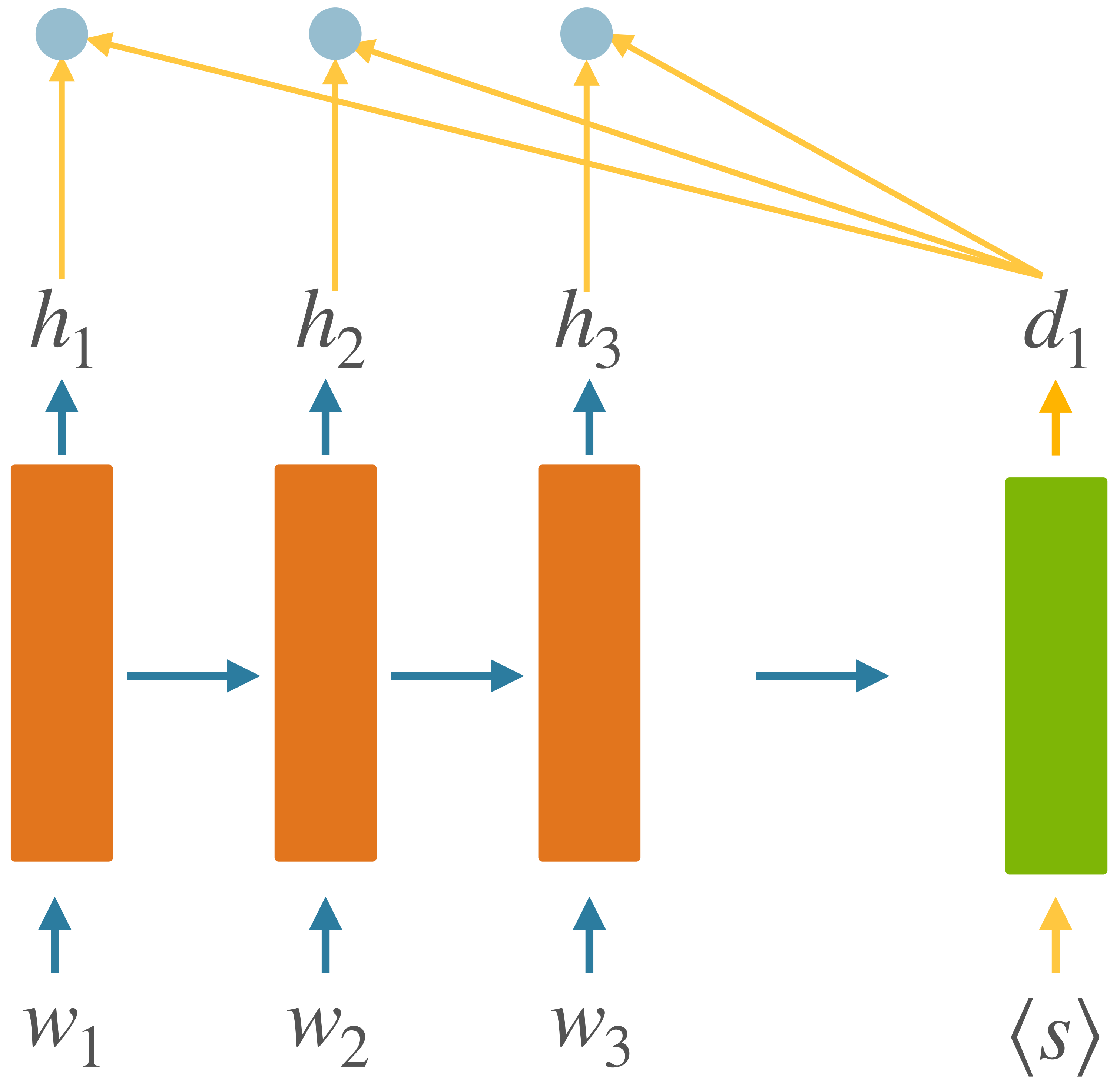
$\alpha_{ij} = a(h_j, d_i)$
(dot product usually)



Badhanau et al 2014

Adding Attention

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

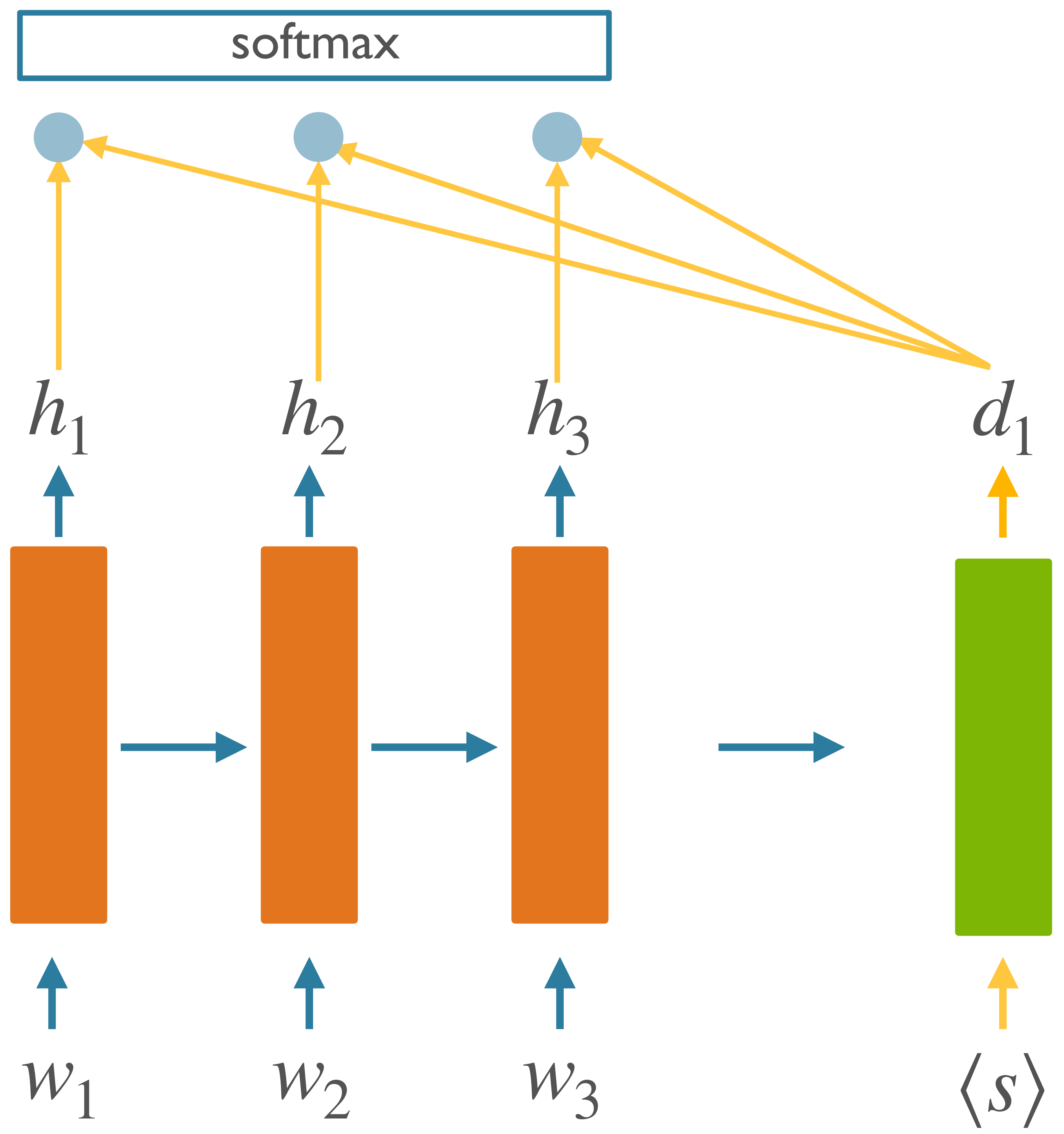


Badhanau et al 2014

Adding Attention

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

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



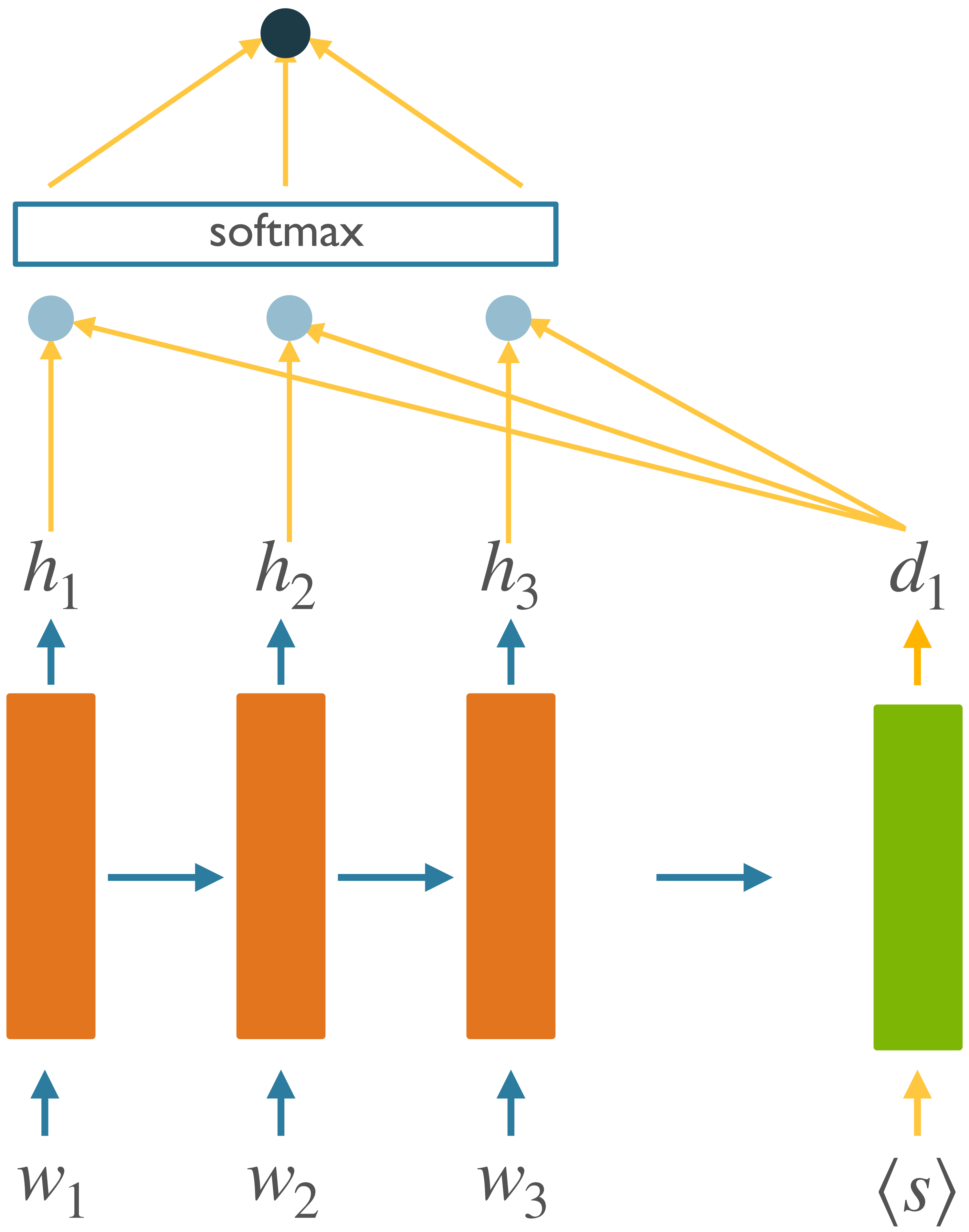
[Badhanau et al 2014]

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)



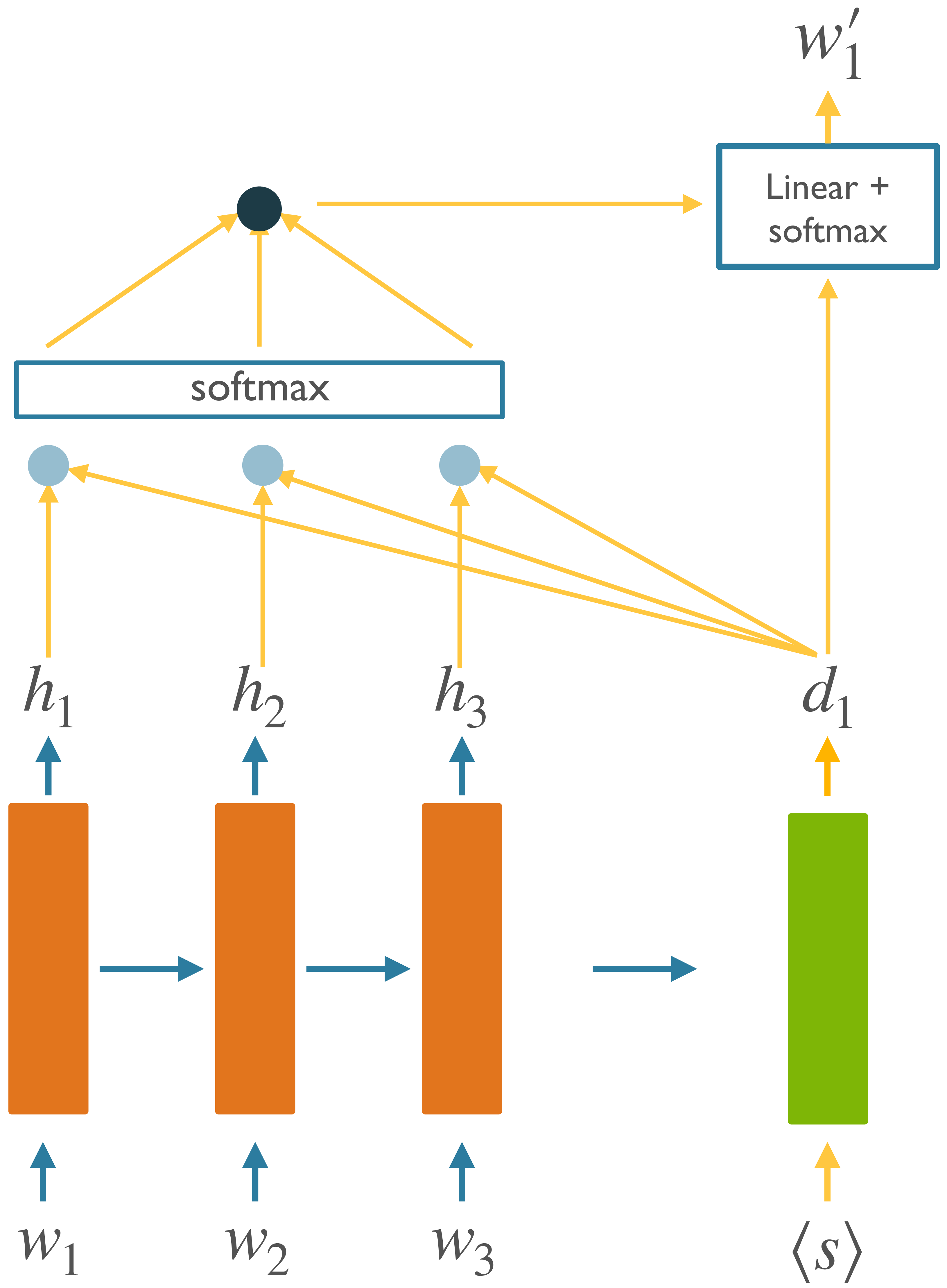
Badhanau et al 2014

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)



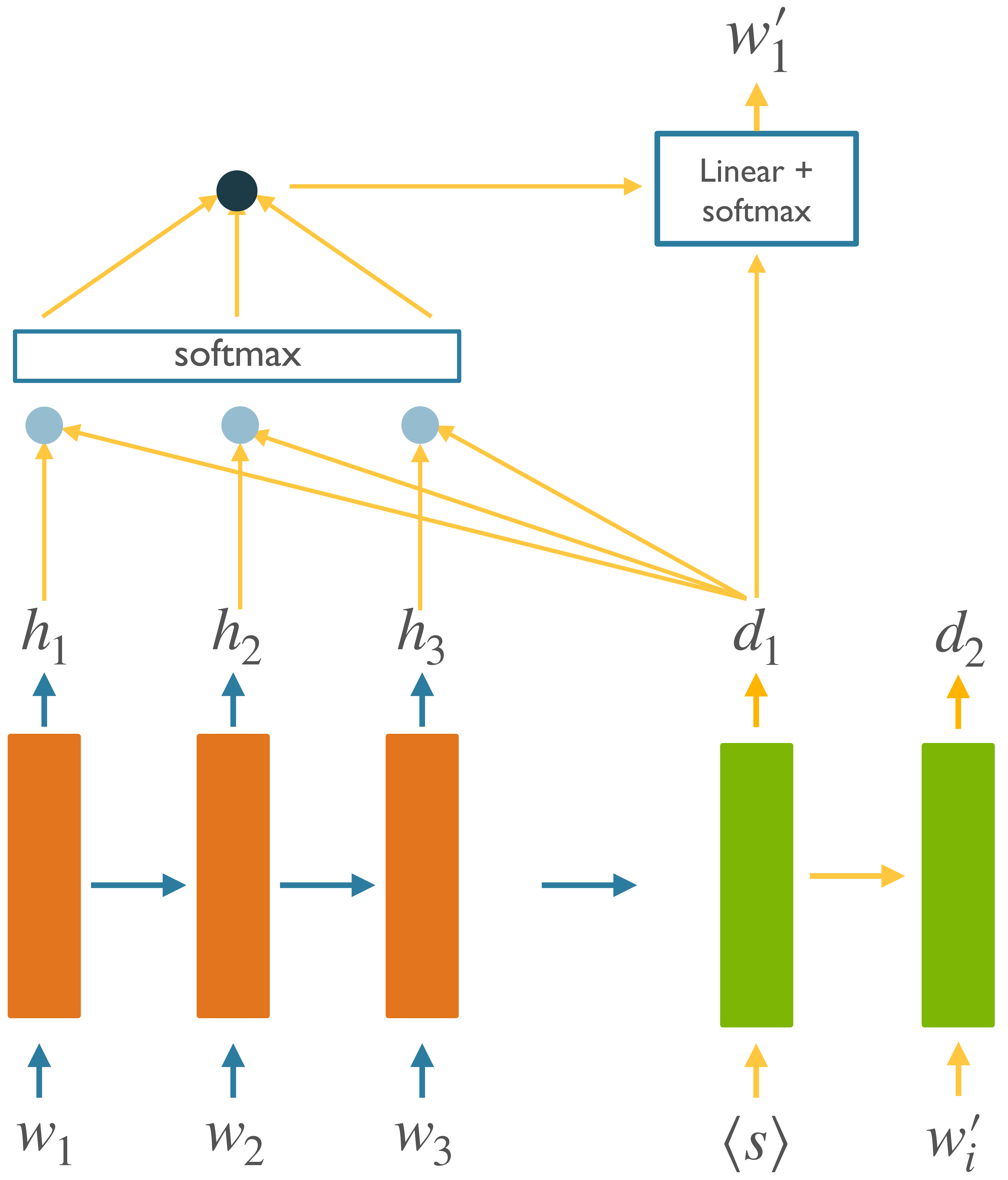
Badhanau et al 2014

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

Attention, Generally

Attention, Generally

- A *query* q pays attention to some *values* $\{v_k\}$ based on similarity with some *keys* $\{k_v\}$.

Attention, Generally

- A *query* q pays attention to some *values* $\{v_k\}$ based on similarity with some *keys* $\{k_v\}$.
- Dot-product attention:

$$\alpha_j = q \cdot k_j$$
$$e_j = e^{\alpha_j} / \Sigma_j e^{\alpha_j}$$
$$c = \Sigma_j e_j v_j$$

Attention, Generally

- A *query* q pays attention to some *values* $\{v_k\}$ based on similarity with some *keys* $\{k_v\}$.
- Dot-product attention:

$$\alpha_j = q \cdot k_j$$

$$e_j = e^{\alpha_j} / \Sigma_j e^{\alpha_j}$$

$$c = \Sigma_j e_j v_j$$

- In the previous example: encoder hidden states played *both* the keys and the values roles.

Why attention?

Why attention?

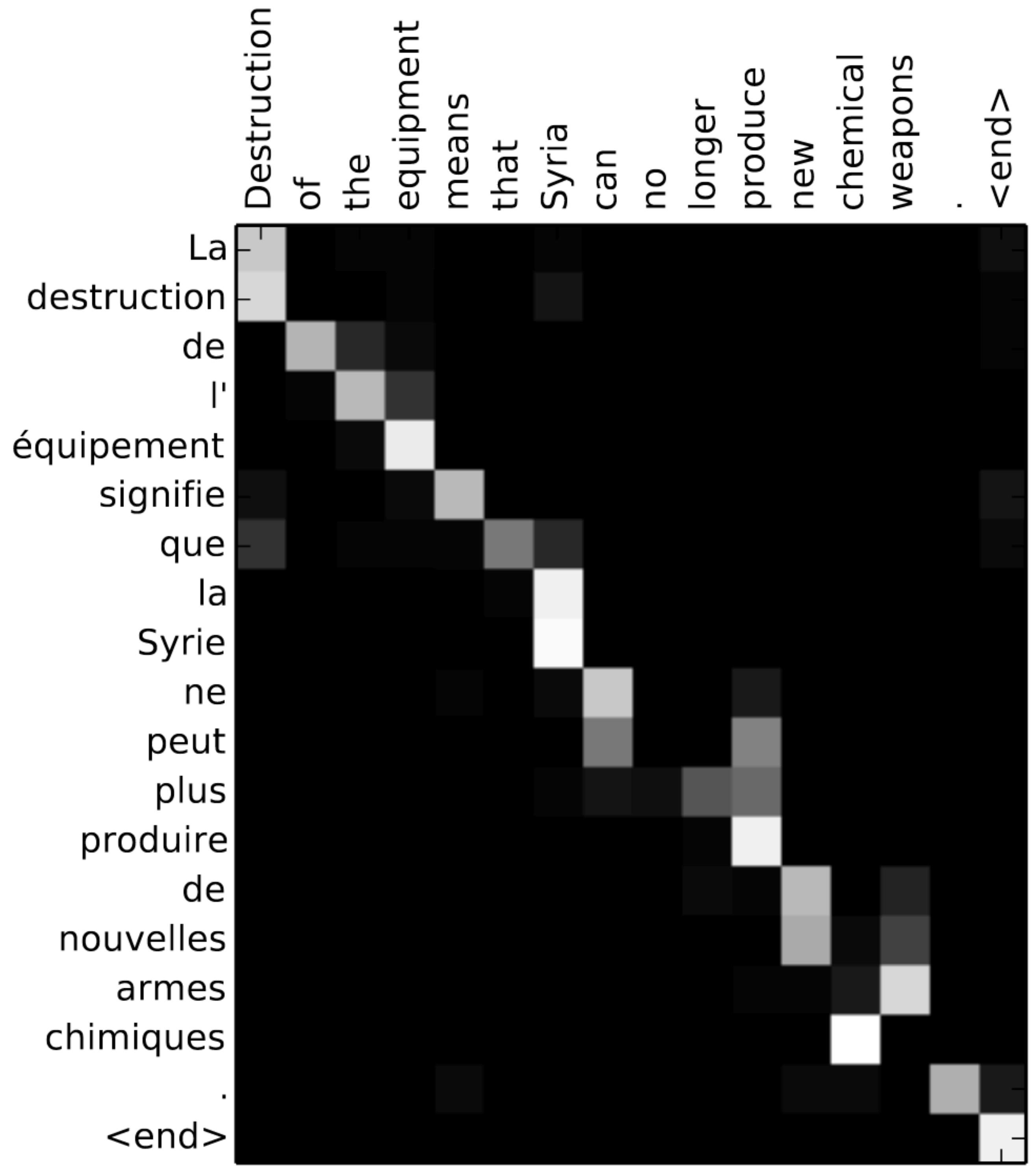
- Incredibly useful (for performance)
  - By “solving” the bottleneck issue

Why attention?

- Incredibly useful (for performance)
  - By “solving” the bottleneck issue
- Aids interpretability (maybe)

Why attention?

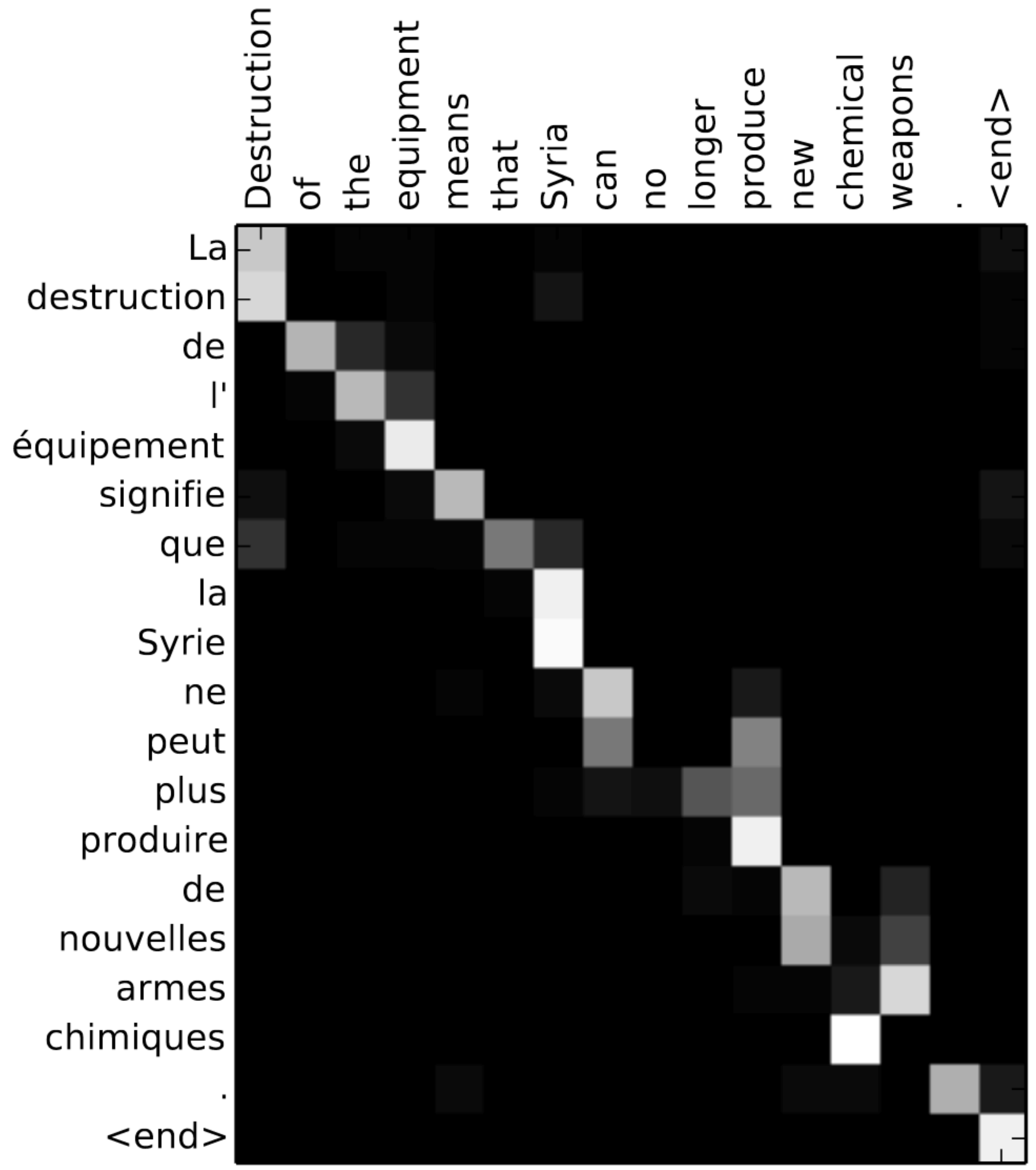
- Incredibly useful (for performance)
  - By “solving” the bottleneck issue
- Aids interpretability (maybe)



[Badhanau et al 2014]

Why attention?

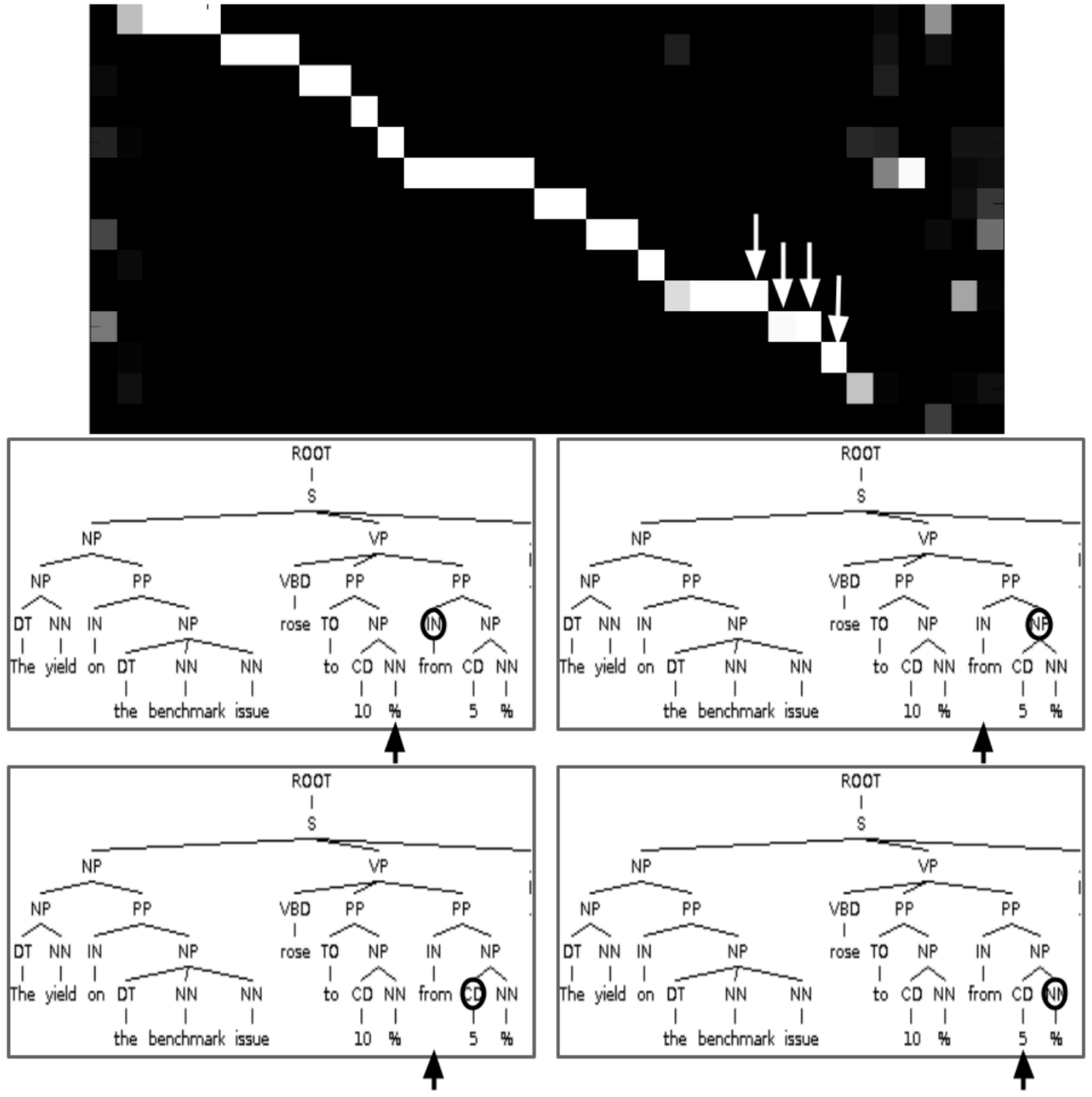
- Incredibly useful (for performance)
  - By "solving" the bottleneck issue
- Aids interpretability (maybe)
- A general technique for combining representations, applications in:
  - NMT, parsing, image/video captioning, …, everything



Badhanau et al 2014

Why attention?

- Incredibly useful (for performance)
  - By "solving" the bottleneck issue
- Aids interpretability (maybe)
- A general technique for combining representations, applications in:
  - NMT, parsing, image/video captioning, …, everything



Vinyals et al 2015

Next Time

- We will introduce a new type of large neural model: the *Transformer*
  - Hint: “Attention is All You Need” is the original paper
- Introduce the idea of transfer learning and pre-training language models
  - Canvas recent developments and trends in that approach
  - What we might call “The Transformer Hegemony” or “The Muppet Hegemony”