

Recurrent Neural Networks

(Content by Yoav Goldberg, Silviu Pitis)

Dealing with Sequences

- For an input sequence x_1, \dots, x_n , we can:
 - If n is **fixed**: *concatenate* and feed into an MLP.
 - *sum* the vectors (*CBOW*) and feed into an MLP.
 - Break the sequence into *windows*. Find n-gram embedding, sum into an MLP.
 - Find good ngrams using ConvNet, using *pooling* (either sum/avg or max) to combine to a single vector.

Dealing with Sequences

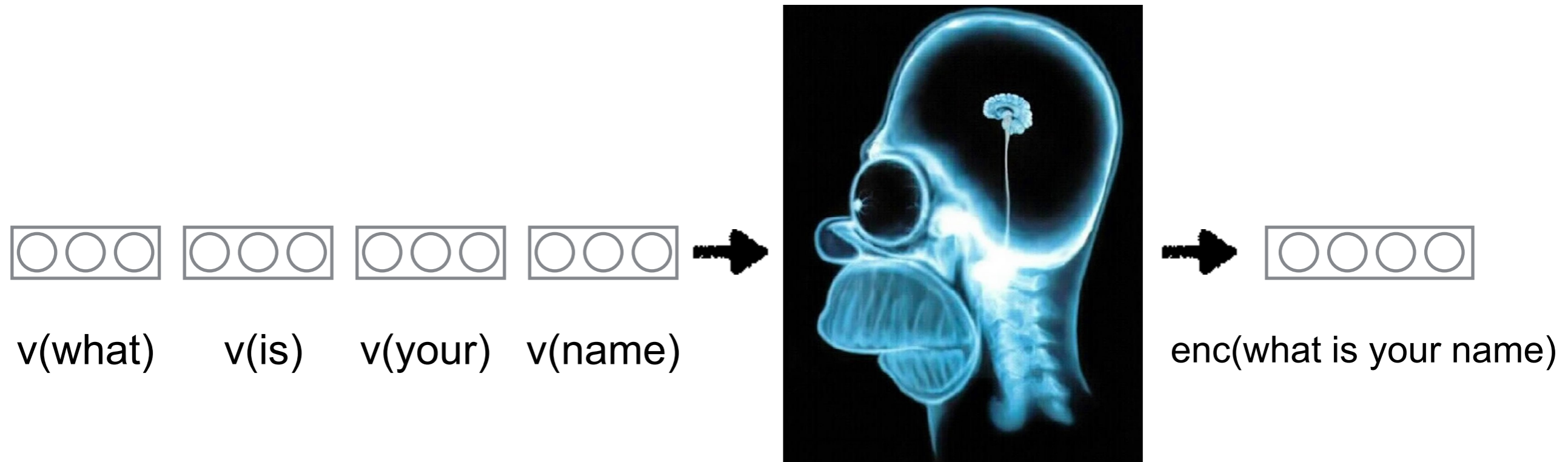
- For an input sequence x_1, \dots, x_n , we can:

Some of these approaches consider **local** word order (which ones?).

How can we consider **global** word order?

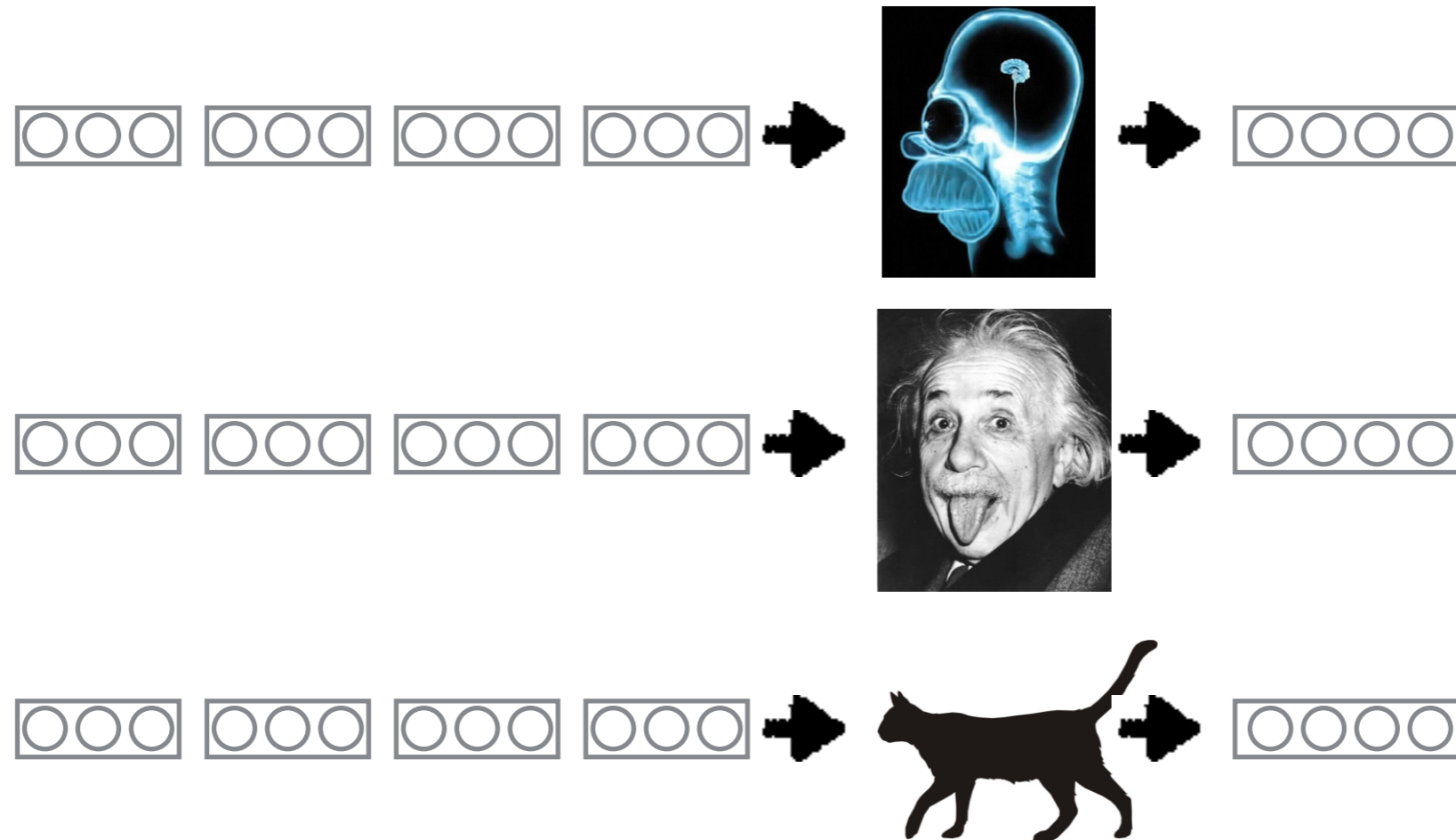
- Find good ngrams using ConvNet, using *pooling* (either sum/avg or max) to combine to a single vector.

Recurrent Neural Networks



- Very strong models of sequential data.
- **Trainable** function from n vectors to a single vector.

Recurrent Neural Networks



- There are different variants (implementations).
- So far, we focused on the interface level.

Recurrent Neural Networks

$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_n, \mathbf{y}_n$$

$$\mathbf{x}_i \in \mathbb{R}^{d_{in}}, \mathbf{y}_i \in \mathbb{R}^{d_{out}}, \mathbf{s}_i \in \mathbb{R}^{f(d_{out})}$$

- Very strong models of sequential data.
- **Trainable** function from n vectors to a single* vector.

Recurrent Neural Networks

$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_n, \mathbf{y}_n$$

*this one is internal. we only care about the \mathbf{y}

$$\mathbf{x}_i \in \mathbb{R}^{d_{in}}, \mathbf{y}_i \in \mathbb{R}^{d_{out}}, \mathbf{s}_i \in \mathbb{R}^{f(d_{out})}$$

- Very strong models of sequential data.
- **Trainable** function from n vectors to a single* vector.

Recurrent Neural Networks

$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_n, \mathbf{y}_n$$

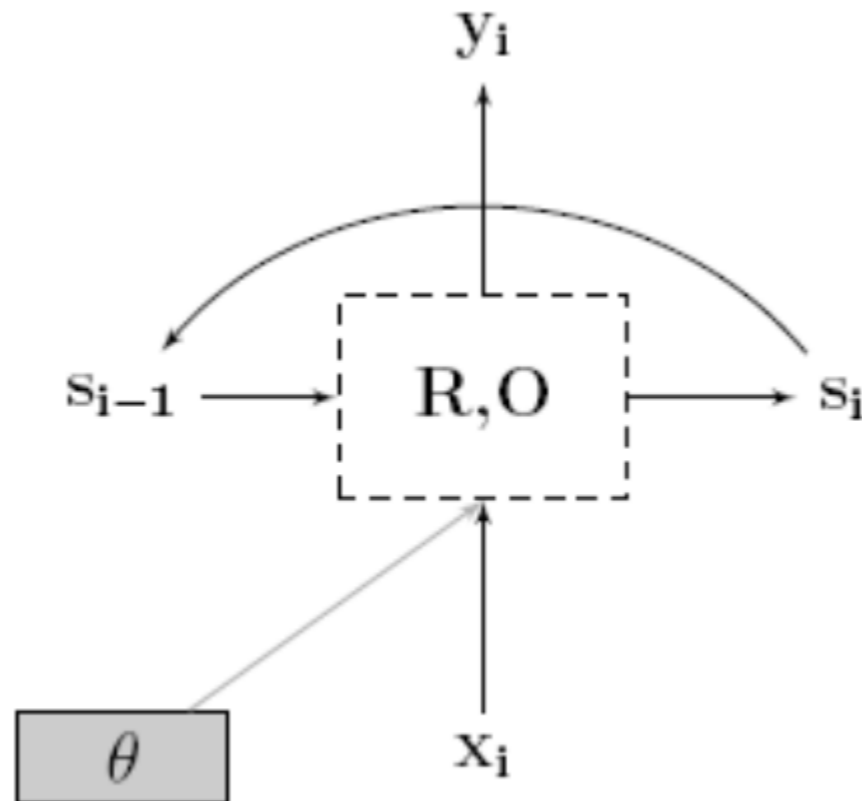
$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i)$$

$$\mathbf{y}_i = O(\mathbf{s}_i)$$

$$\mathbf{x}_i \in \mathbb{R}^{d_{in}}, \mathbf{y}_i \in \mathbb{R}^{d_{out}}, \mathbf{s}_i \in \mathbb{R}^{f(d_{out})}$$

- **Recursively defined.**
- There's a vector \mathbf{y}_i for every prefix $\mathbf{x}_{1:i}$

Recurrent Neural Networks



$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_n, \mathbf{y}_n$$

$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i)$$

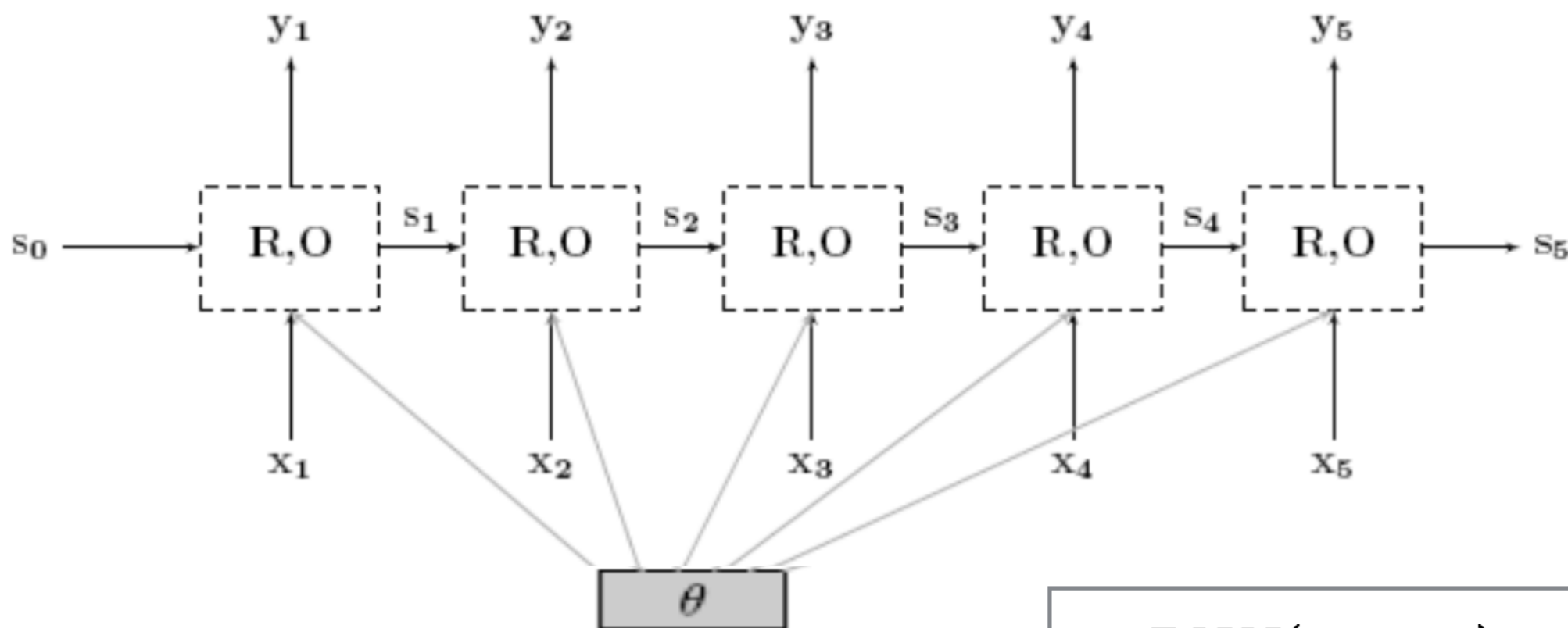
$$\mathbf{y}_i = O(\mathbf{s}_i)$$

$$\mathbf{x}_i \in \mathbb{R}^{d_{in}}, \mathbf{y}_i \in \mathbb{R}^{d_{out}}, \mathbf{s}_i \in \mathbb{R}^{f(d_{out})}$$

- **Recursively defined.**

- There's a vector \mathbf{y}_i for every prefix $\mathbf{x}_{1:i}$

Recurrent Neural Networks



for every finite input sequence,
can unroll the recursion.

- Recursively defined.

- There's a vector \mathbf{y}_i for every prefix $\mathbf{x}_{1:i}$

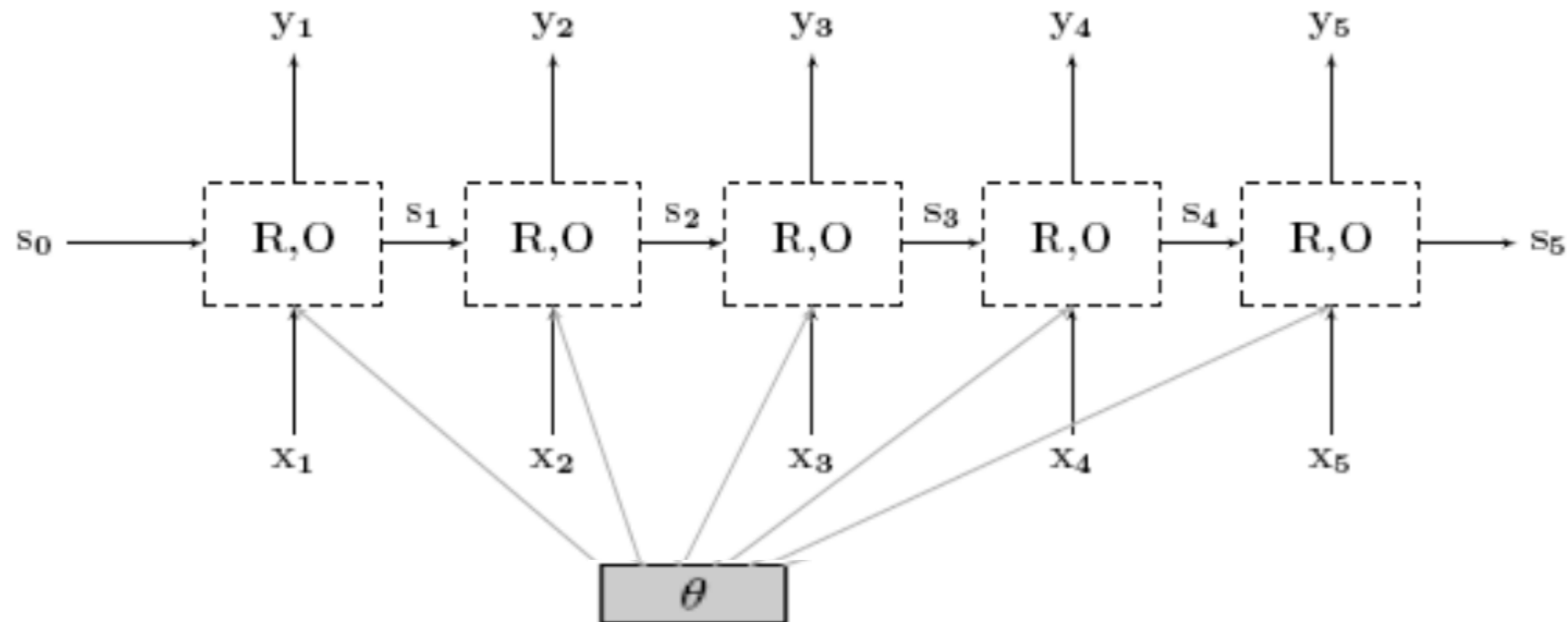
$$RNN(\mathbf{s}_0, \mathbf{x}_{1:n}) = \mathbf{s}_n, \mathbf{y}_n$$

$$\mathbf{s}_i = R(\mathbf{s}_{i-1}, \mathbf{x}_i)$$

$$\mathbf{y}_i = O(\mathbf{s}_i)$$

$$\mathbf{x}_i \in \mathbb{R}^{d_{in}}, \mathbf{y}_i \in \mathbb{R}^{d_{out}}, \mathbf{s}_i \in \mathbb{R}^{f(d_{out})}$$

Recurrent Neural Networks



for every finite input sequence,
can unroll the recursion.



Recurrent Neural Networks

$$y_4 = O(s_4)$$

$$s_4 = R(s_3, x_4)$$

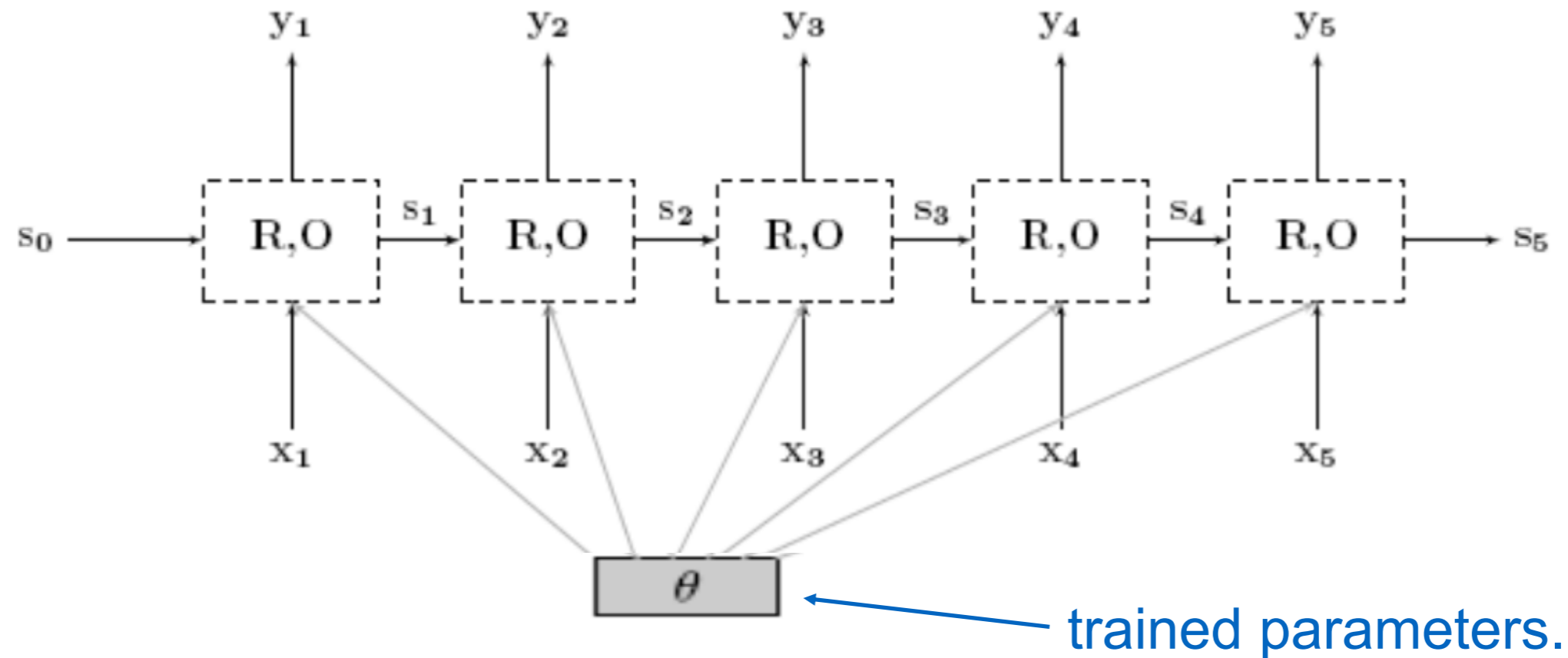
$$= R(\overbrace{R(s_2, x_3)}^{s_3}, x_4)$$

$$= R(R(\overbrace{R(s_1, x_2)}^{s_2}), x_3), x_4)$$

$$= R(R(R(\overbrace{R(s_0, x_1)}^{s_1}), x_2), x_3), x_4)$$

- The output vector y_i depends on **all** inputs $x_{1:i}$

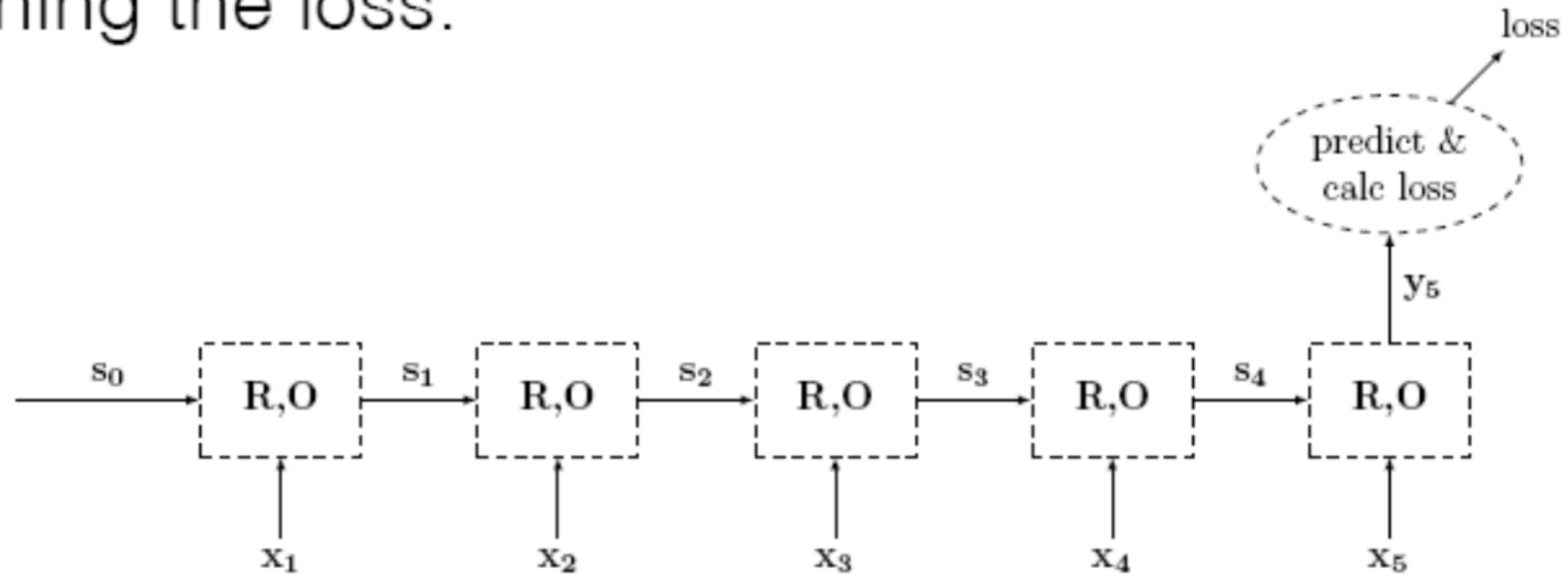
Recurrent Neural Networks



- **But we can train them.**
 - define function form
 - define loss

Recurrent Neural Networks for Text Classification

Defining the loss.



Acceptor: predict something from end state.

Backprop the error all the way back.

Train the network to capture meaningful information

CBOW as an RNN

$$R_{CBOW}(\mathbf{s}_{i-1}, \mathbf{x}_i) = \mathbf{s}_{i-1} + \mathbf{x}_i$$

(what are the parameters?)

CBOW as an RNN

$$R_{CBOW}(\mathbf{s}_{i-1}, \mathbf{x}_i) = \mathbf{s}_{i-1} + \mathbf{x}_i$$

(what are the parameters?)

$$R_{CBOW}(\mathbf{s}_{i-1}, x_i) = \mathbf{s}_{i-1} + \mathbf{E}_{[x_i]}$$

CBOW as an RNN



$$R_{CBOW}(\mathbf{s}_{i-1}, x_i) = \mathbf{s}_{i-1} + \mathbf{E}_{[x_i]}$$

CBOW as an RNN



$$R_{CBOW}(\mathbf{s}_{i-1}, x_i) = \underline{\tanh}(\mathbf{s}_{i-1} + \mathbf{E}_{[x_i]})$$

Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s}_{i-1}, \mathbf{x}_i) = \tanh(\mathbf{W}^s \cdot \mathbf{s}_{i-1} + \mathbf{W}^x \cdot \mathbf{x}_i)$$

Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s}_{i-1}, \mathbf{x}_i) = \tanh(\mathbf{W}^s \cdot \mathbf{s}_{i-1} + \mathbf{W}^x \cdot \mathbf{x}_i)$$

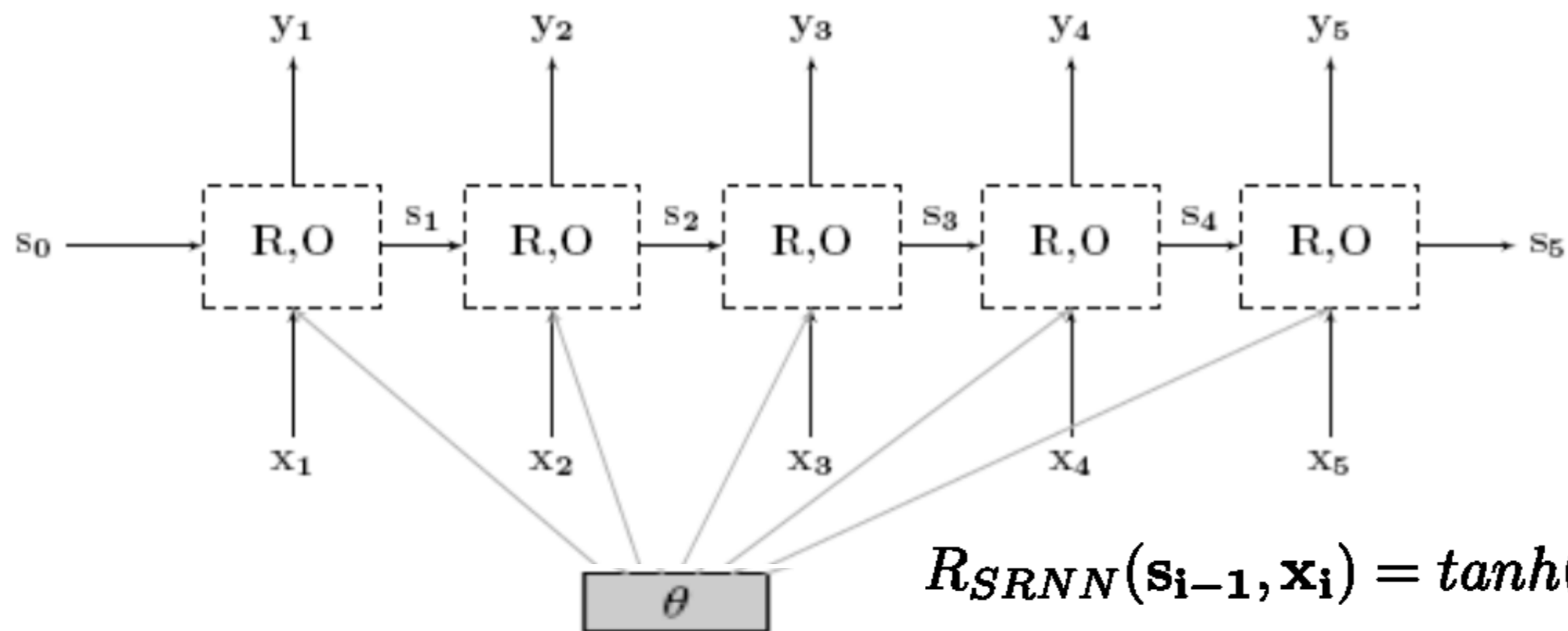
- Looks very simple.
- Theoretically very powerful.
- In practice not so much (hard to train).
- Why? Vanishing gradients.

Simple RNN (Elman RNN)

$$R_{SRNN}(\mathbf{s}_{i-1}, \mathbf{x}_i) = \tanh(\mathbf{W}^s \cdot \mathbf{s}_{i-1} + \mathbf{W}^x \cdot \mathbf{x}_i)$$

Another view on behavior:

- RNN as a "computer":
input \mathbf{x}_i arrives, memory \mathbf{s} is updated.
- In the Elman RNN, **entire memory is written** at each time-step.
- entire memory \rightarrow output



d_i

$$R_{SRNN}(s_{i-1}, x_i) = \tanh(\mathbf{W}^s \cdot s_{i-1} + \mathbf{W}^x \cdot x_i)$$

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^T \frac{\partial L_t}{\partial \theta}$$

$$\frac{\partial L_t}{\partial W^s} = \sum_{k=1}^t \left(\frac{\partial L_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W^s} \right)$$

$$\frac{\partial s_t}{\partial s_k} = \prod_{i=k+1}^t \frac{\partial s_i}{\partial s_{i-1}} =$$



LSTM RNN

(Long Short Term Memory)

better controlled memory access

continuous gates

Differentiable "Gates"

- The main idea behind the LSTM is that you want to somehow control the "memory access".
- In a SimpleRNN:

$$R_{SRNN}(s_{i-1}, x_i) = \tanh(\mathbf{W}^s \cdot s_{i-1} + \mathbf{W}^x \cdot x_i)$$

read previous state memory



write new input



- All the memory gets overwritten

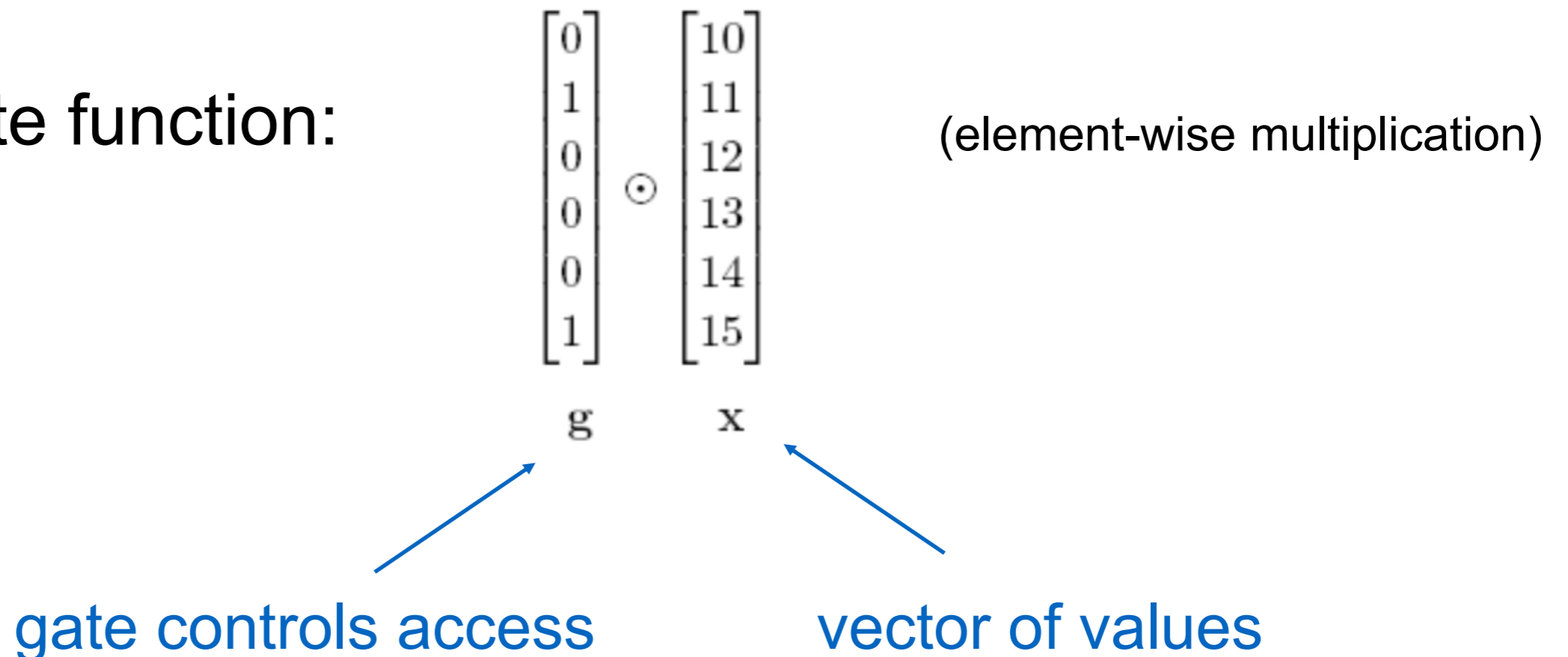
Vector "Gates"

- We'd like to:
 - * Selectively read from some memory "cells".
 - * Selectively write to some memory "cells".

Vector "Gates"

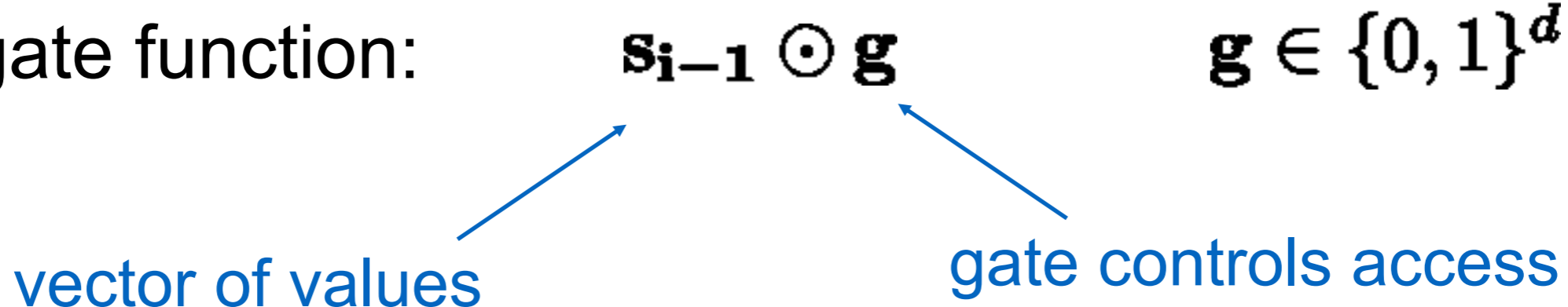
- We'd like to:
 - * Selectively read from some memory "cells".
 - * Selectively write to some memory "cells".

- A gate function:



Vector "Gates"

- We'd like to:
 - * Selectively read from some memory "cells".
 - * Selectively write to some memory "cells".

- A gate function: $\mathbf{s}_{i-1} \odot \mathbf{g}$ $\mathbf{g} \in \{0, 1\}^d$


The diagram shows the equation $\mathbf{s}_{i-1} \odot \mathbf{g}$ with $\mathbf{g} \in \{0, 1\}^d$ to its right. A blue arrow points from the text "vector of values" below to the variable \mathbf{s}_{i-1} . Another blue arrow points from the text "gate controls access" below to the variable \mathbf{g} .

Vector "Gates"

- Using the gate function to control access:

$$\mathbf{s}_i \leftarrow \mathbf{s}_{i-1} \odot \mathbf{g}^r + \mathbf{x}_i \odot \mathbf{g}^w \quad \mathbf{g} \in \{0, 1\}^d$$

which cells to read

which cells to write

Vector "Gates"

- Using the gate function to control access:

$$\mathbf{s}_i \leftarrow \mathbf{s}_{i-1} \odot \mathbf{g}^r + \mathbf{x}_i \odot \mathbf{g}^w \quad \mathbf{g} \in \{0, 1\}^d$$

which cells to read

which cells to write

- (can also tie them: $\mathbf{g}^r = 1 - \mathbf{g}^w$)

Vector "Gates"

$$\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix} \leftarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}$$

s' \mathbf{g} \mathbf{x} $(\mathbf{1} - \mathbf{g})$ \mathbf{s}

Differentiable "Gates"

- **Problem with the gates:**
 - * they are fixed.
 - * they don't depend on the input or the output.

Differentiable "Gates"

- **Problem with the gates:**
 - * they are fixed.
 - * they don't depend on the input or the output.
- Solution: make them smooth, input dependent, and trainable.

$$\mathbf{g}^r = \sigma(\mathbf{W} \cdot \mathbf{x}_i + \mathbf{U} \cdot \mathbf{s}_{i-1})$$

"almost 0"

or

"almost 1"

function of input and state

Goal 1

$$s_t = \phi(W s_{t-1} + U x_t + b)$$

Uncontrolled and uncoordinated writes, particularly at the start of training when writes are completely random, create a chaotic state that leads to bad results and from which it can be difficult to recover.

Selectivity to Control Writing

- Write Selectively: when taking class notes, we only record the most important points; we certainly don't write our new notes on top of our old notes
- Read Selectively: apply the most relevant knowledge
- Forget Selectively: in order to make room for new information, we need to selectively forget the least relevant old information

Gates for Selectivity

$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$$

- Our three gates at time step t are denoted i_t , the input gate (for writing), o_t , the output gate (for reading) and f_t , the forget gate (for remembering)
- Notice:
 - (1) forget gate should be called remember gate
 - (2) names/roles are confusing. Will get clear slowly

Candidate Write

$$s_{t+1} = s_t + \Delta s_{t+1}.$$

- Goal is to compute change in state value
- We calculate candidate write the same way we would calculate the state in a vanilla RNN, except that instead of using the prior state, s_{t-1} , we first multiply the prior state element-wise by the read gate to get the *gated prior state*,

$$\tilde{s}_t = \phi(W(o_t \odot s_{t-1}) + Ux_t + b)$$

New State

\tilde{s}_t is only a candidate write because we are applying selective writing and have a write gate. Thus, we multiply \tilde{s}_t element-wise by our write gate, i_t , to obtain our true write, $i_t \odot \tilde{s}_t$.

The final step is to add this to our prior state, but forget selectivity says that we need to have a mechanism for forgetting. So before we add anything to our prior state, we multiply it (element-wise) by the forget gate (which actually operates as a remember gate). Our final prototype LSTM equation is:

$$\mathbf{s}_t = \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{s}}_t$$

Prototype LSTM

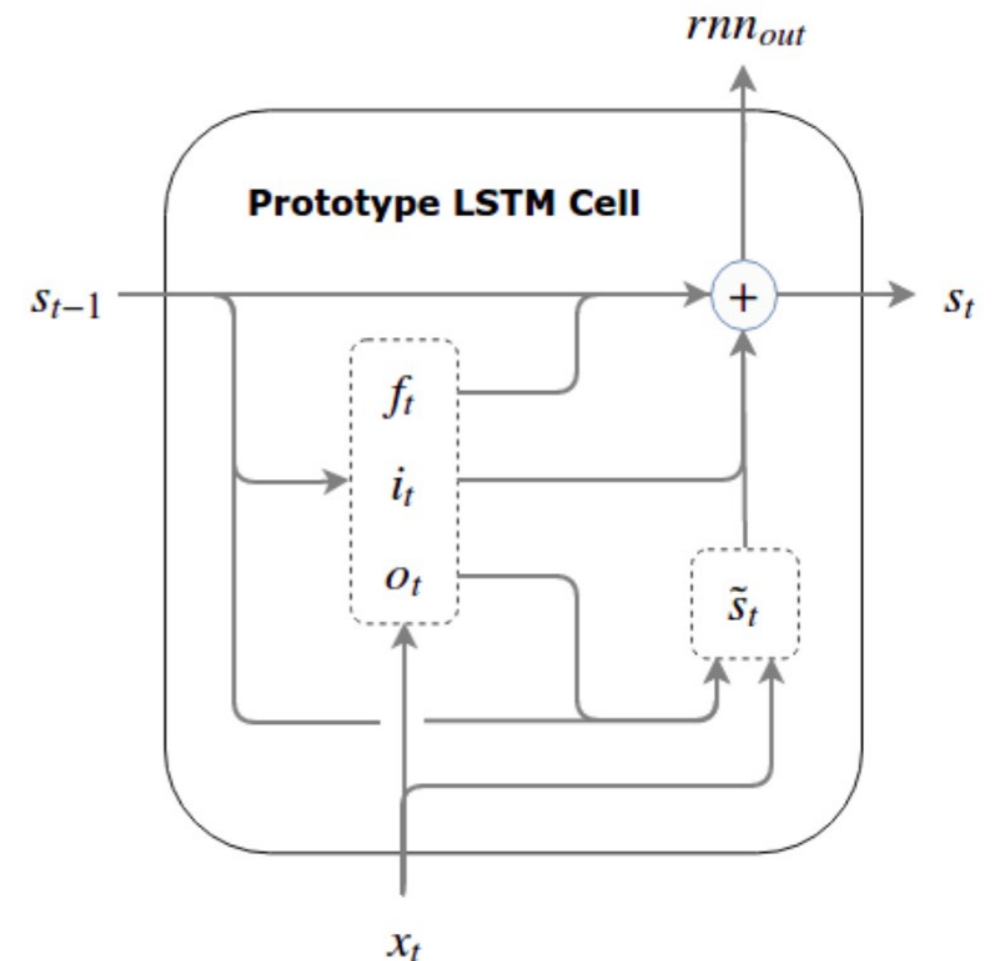
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$$

$$\tilde{s}_t = \phi(W(o_t \odot s_{t-1}) + Ux_t + b)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$



Problem

- the selective forgets and the selective writes are not coordinated at the start of training which can cause the state to quickly become large and chaotic.
- since the state is potentially unbounded, the gates and the candidate write will often become saturated, which causes problems for training.
- This problem is real and often happens in practice!
- (Hochreiter & Schmidhuber 97) “if the [writes] are mostly positive or mostly negative, then the internal state will tend to drift away over time”

Solution 1: GRU

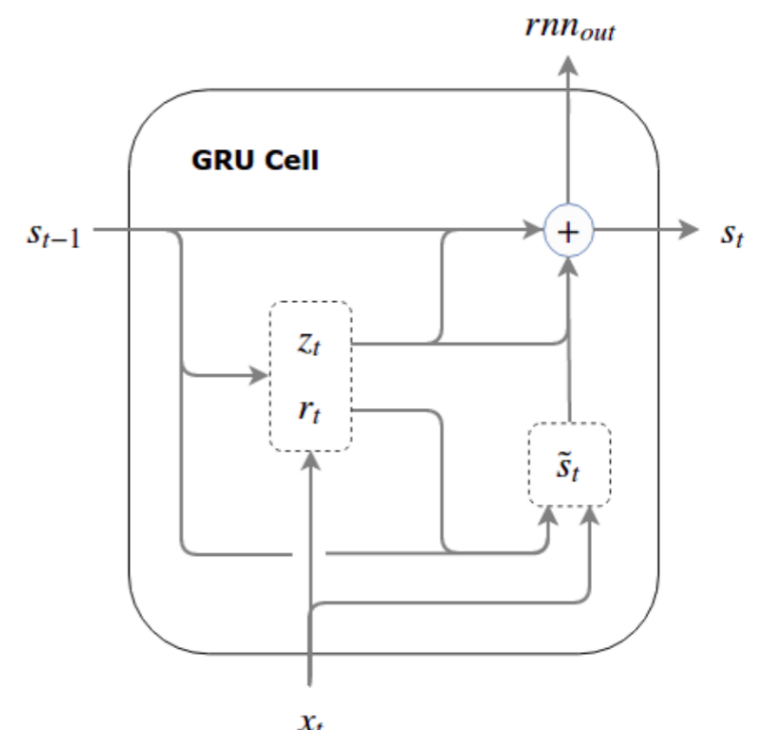
- Impose a hard bound on the state & coordinate writes and forgets by explicitly linking them
- instead of selective writes and selective forgets, we forego some expressiveness and do selective overwrites by setting our forget gate equal to 1 minus our write gate

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

$$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$$

$$\tilde{s}_t = \phi(W(r_t \odot s_{t-1}) + U x_t + b)$$

$$s_t = z_t \odot s_{t-1} + (1 - z_t) \odot \tilde{s}_t$$

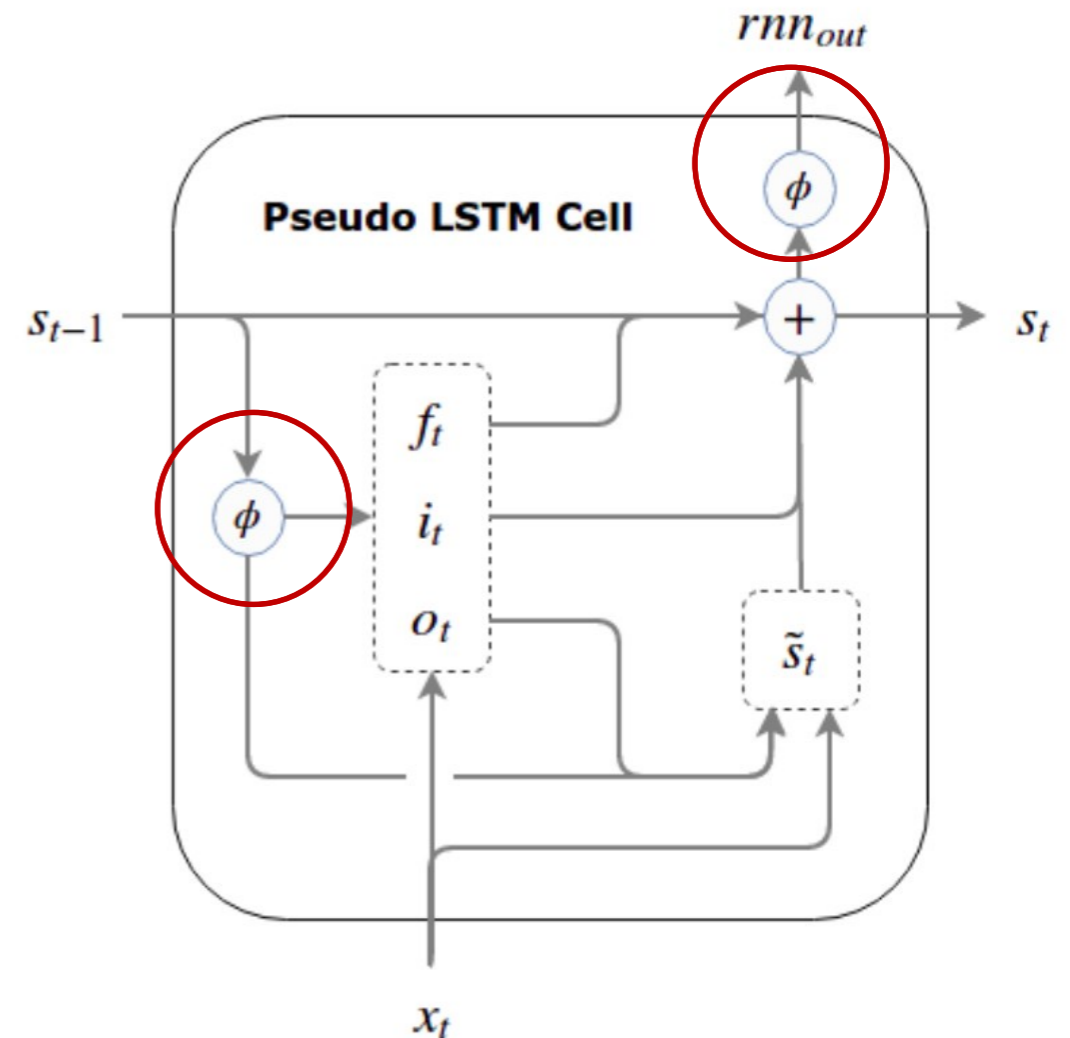


Solution 2: Pseudo-LSTM

$$i_t = \sigma(W_i(\phi(s_{t-1})) + U_i x_t + b_i)$$
$$o_t = \sigma(W_o(\phi(s_{t-1})) + U_o x_t + b_o)$$
$$f_t = \sigma(W_f(\phi(s_{t-1})) + U_f x_t + b_f)$$

$$\tilde{s}_t = \phi(W(o_t \odot \phi(s_{t-1})) + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

$$\text{rnn}_{out} = \phi(s_t)$$



- Why not change the state s_t by applying non-linearity?
Information morphing:
state shouldn't change unless new external influence
- we pass the state through the squashing function every time we need to use it for anything except making incremental writes to it

Pseudo LSTM \rightarrow LSTM

$$\begin{aligned}i_t &= \sigma(W_i(\phi(s_{t-1})) + U_i x_t + b_i) \\o_t &= \sigma(W_o(\phi(s_{t-1})) + U_o x_t + b_o) \\f_t &= \sigma(W_f(\phi(s_{t-1})) + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{s}_t &= \phi(W(o_t \odot \phi(s_{t-1})) + U x_t + b) \\s_t &= f_t \odot s_{t-1} + i_t \odot \tilde{s}_t\end{aligned}$$

$$\text{rnn}_{out} = \phi(s_t)$$



(1) Read happens after write! Why!!!

\rightarrow How can we write when we can't read!

\rightarrow Send a "shadow" state that has the read information.

Pseudo LSTM \rightarrow LSTM

$$\begin{aligned}i_t &= \sigma(W_i(\phi(s_{t-1})) + U_i x_t + b_i) \\o_t &= \sigma(W_o(\phi(s_{t-1})) + U_o x_t + b_o) \\f_t &= \sigma(W_f(\phi(s_{t-1})) + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{s}_t &= \phi(W(o_t \odot \phi(s_{t-1})) + U x_t + b) \\s_t &= f_t \odot s_{t-1} + i_t \odot \tilde{s}_t\end{aligned}$$

$$\text{rnn}_{out} = \phi(s_t)$$

Rewriting s_t to c_t

Defining

$$h_{t-1} = o_{t-1} \odot \phi(c_{t-1}).$$

$$\begin{aligned}i_t &= \sigma(W_i(\phi(c_{t-1})) + U_i x_t + b_i) \\o_t &= \sigma(W_o(\phi(c_{t-1})) + U_o x_t + b_o) \\f_t &= \sigma(W_f(\phi(c_{t-1})) + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{c}_t &= \phi(W h_{t-1} + U x_t + b) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\end{aligned}$$

$$\begin{aligned}h_t &= o_t \odot \phi(c_t) \\rnn_{out} &= \phi(c_t)\end{aligned}$$

Pseudo LSTM → LSTM

$$i_t = \sigma(W_i(\phi(c_{t-1})) + U_i x_t + b_i)$$
$$o_t = \sigma(W_o(\phi(c_{t-1})) + U_o x_t + b_o)$$
$$f_t = \sigma(W_f(\phi(c_{t-1})) + U_f x_t + b_f)$$

$$\tilde{c}_t = \phi(W h_{t-1} + U x_t + b)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \phi(c_t)$$
$$rnn_{out} = \phi(c_t)$$

(2) Gates are computed using shadow state

→ information that is irrelevant for the candidate write is also irrelevant for gate computations



Pseudo LSTM \rightarrow LSTM

$$\begin{aligned}i_t &= \sigma(W_i(\phi(c_{t-1})) + U_i x_t + b_i) \\o_t &= \sigma(W_o(\phi(c_{t-1})) + U_o x_t + b_o) \\f_t &= \sigma(W_f(\phi(c_{t-1})) + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{c}_t &= \phi(W h_{t-1} + U x_t + b) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\end{aligned}$$

$$\begin{aligned}h_t &= o_t \odot \phi(c_t) \\rnn_{out} &= \phi(c_t)\end{aligned}$$

$$\begin{aligned}i_t &= \sigma(W_i h_{t-1} + U_i x_t + b_i) \\o_t &= \sigma(W_o h_{t-1} + U_o x_t + b_o) \\f_t &= \sigma(W_f h_{t-1} + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{c}_t &= \phi(W h_{t-1} + U x_t + b) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\end{aligned}$$

$$\begin{aligned}h_t &= o_t \odot \phi(c_t) \\rnn_{out} &= \phi(c_t)\end{aligned}$$



Pseudo LSTM \rightarrow LSTM

$$\begin{aligned}i_t &= \sigma(W_i(\phi(c_{t-1})) + U_i x_t + b_i) \\o_t &= \sigma(W_o(\phi(c_{t-1})) + U_o x_t + b_o) \\f_t &= \sigma(W_f(\phi(c_{t-1})) + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{c}_t &= \phi(W h_{t-1} + U x_t + b) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\end{aligned}$$

$$\begin{aligned}h_t &= o_t \odot \phi(c_t) \\rnn_{out} &= \phi(c_t)\end{aligned}$$

(3) LSTM's external output is the shadow state, instead of the squished main state.



Pseudo LSTM \rightarrow LSTM

$$\begin{aligned}i_t &= \sigma(W_i h_{t-1} + U_i x_t + b_i) \\o_t &= \sigma(W_o h_{t-1} + U_o x_t + b_o) \\f_t &= \sigma(W_f h_{t-1} + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{c}_t &= \phi(W h_{t-1} + U x_t + b) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\end{aligned}$$

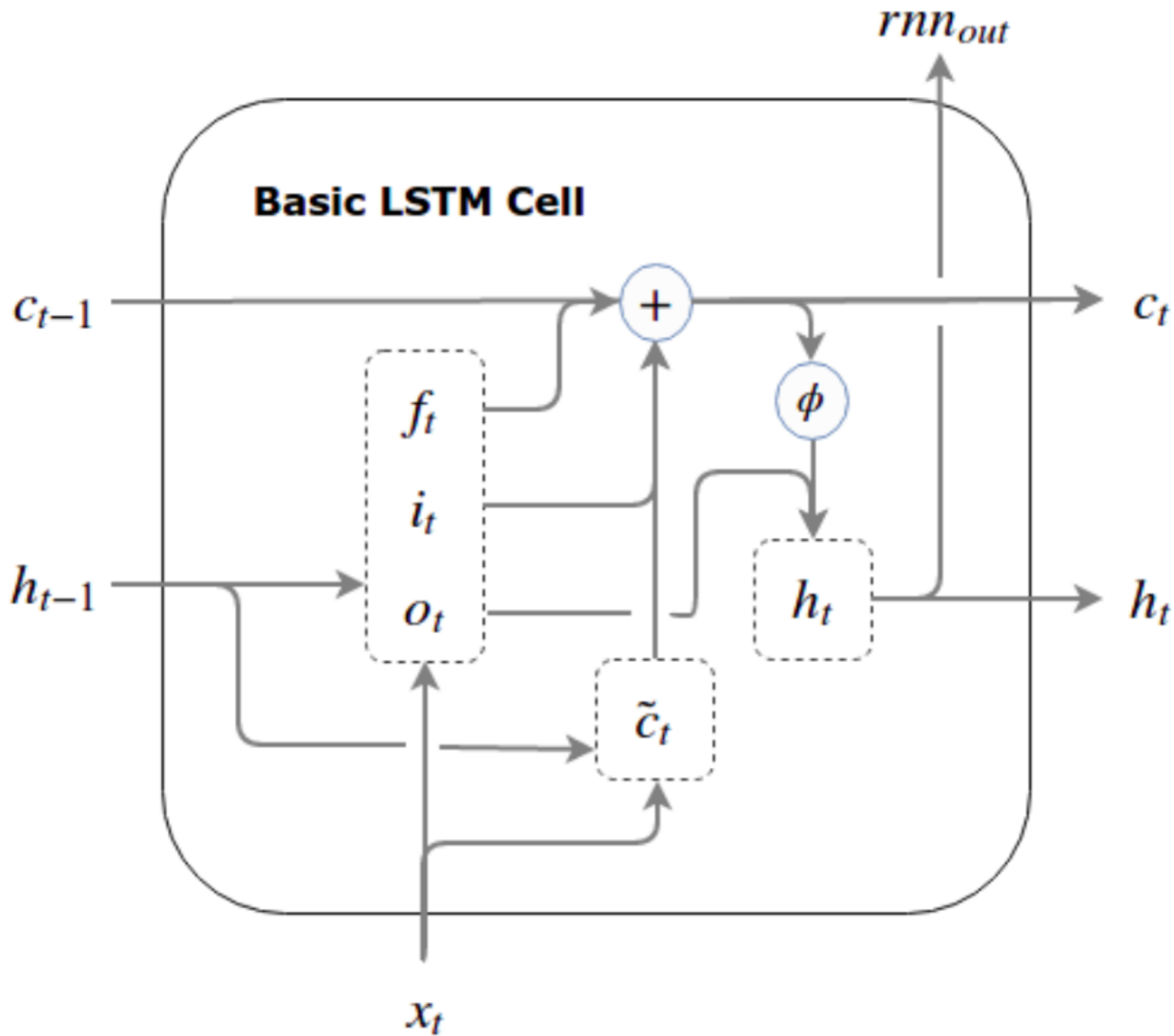
$$\begin{aligned}h_t &= o_t \odot \phi(c_t) \\rnn_{out} &= \phi(c_t)\end{aligned}$$

$$\begin{aligned}i_t &= \sigma(W_i h_{t-1} + U_i x_t + b_i) \\o_t &= \sigma(W_o h_{t-1} + U_o x_t + b_o) \\f_t &= \sigma(W_f h_{t-1} + U_f x_t + b_f)\end{aligned}$$

$$\begin{aligned}\tilde{c}_t &= \phi(W h_{t-1} + U x_t + b) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\end{aligned}$$

$$\begin{aligned}h_t &= o_t \odot \phi(c_t) \\rnn_{out} &= h_t\end{aligned}$$





LSTM

(Long short-term Memory)

- The LSTM is a specific combination of gates.

$$R_{LSTM}(\mathbf{s}_{j-1}, \mathbf{x}_j) = [\mathbf{c}_j; \mathbf{h}_j]$$

$$\mathbf{c}_j = \mathbf{c}_{j-1} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{i} = \sigma(\mathbf{W}^{xi} \cdot \mathbf{x}_j + \mathbf{W}^{hi} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{xf} \cdot \mathbf{x}_j + \mathbf{W}^{hf} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{g} = \tanh(\mathbf{W}^{xg} \cdot \mathbf{x}_j + \mathbf{W}^{hg} \cdot \mathbf{h}_{j-1})$$

LSTM

(Long short-term Memory)

- The LSTM is a specific combination of gates.

$$R_{LSTM}(\mathbf{s}_{j-1}, \mathbf{x}_j) = [\mathbf{c}_j; \mathbf{h}_j]$$

$$\mathbf{c}_j = \mathbf{c}_{j-1} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{h}_j = \tanh(\mathbf{c}_j)$$

$$\mathbf{i} = \sigma(\mathbf{W}^{xi} \cdot \mathbf{x}_j + \mathbf{W}^{hi} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{xf} \cdot \mathbf{x}_j + \mathbf{W}^{hf} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{g} = \tanh(\mathbf{W}^{xg} \cdot \mathbf{x}_j + \mathbf{W}^{hg} \cdot \mathbf{h}_{j-1})$$

LSTM

(Long short-term Memory)

- The LSTM is a specific combination of gates.

$$R_{LSTM}(\mathbf{s}_{j-1}, \mathbf{x}_j) = [\mathbf{c}_j; \mathbf{h}_j]$$

$$\mathbf{c}_j = \mathbf{c}_{j-1} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{h}_j = \tanh(\mathbf{c}_j)$$

$$\mathbf{i} = \sigma(\mathbf{W}^{xi} \cdot \mathbf{x}_j + \mathbf{W}^{hi} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{xf} \cdot \mathbf{x}_j + \mathbf{W}^{hf} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{o} = \sigma(\mathbf{W}^{xo} \cdot \mathbf{x}_j + \mathbf{W}^{ho} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{g} = \tanh(\mathbf{W}^{xg} \cdot \mathbf{x}_j + \mathbf{W}^{hg} \cdot \mathbf{h}_{j-1})$$

LSTM

(Long short-term Memory)

- The LSTM is a specific combination of gates.

$$R_{LSTM}(\mathbf{s}_{j-1}, \mathbf{x}_j) = [\mathbf{c}_j; \mathbf{h}_j]$$

$$\mathbf{c}_j = \mathbf{c}_{j-1} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{h}_j = \tanh(\mathbf{c}_j) \odot \mathbf{o}$$

$$\mathbf{i} = \sigma(\mathbf{W}^{\mathbf{x}\mathbf{i}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{i}} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{\mathbf{x}\mathbf{f}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{f}} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{o} = \sigma(\mathbf{W}^{\mathbf{x}\mathbf{o}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{o}} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{g} = \tanh(\mathbf{W}^{\mathbf{x}\mathbf{g}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{g}} \cdot \mathbf{h}_{j-1})$$

$$R_{LSTM}(\mathbf{s}_{j-1}, \mathbf{x}_j) = [\mathbf{c}_j; \mathbf{h}_j]$$

$$\mathbf{c}_j = \mathbf{c}_{j-1} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i}$$

$$\mathbf{h}_j = \tanh(\mathbf{c}_j) \odot \mathbf{o}$$

$$\mathbf{i} = \sigma(\mathbf{W}^{\mathbf{x}\mathbf{i}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{i}} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{f} = \sigma(\mathbf{W}^{\mathbf{x}\mathbf{f}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{f}} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{o} = \sigma(\mathbf{W}^{\mathbf{x}\mathbf{o}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{o}} \cdot \mathbf{h}_{j-1})$$

$$\mathbf{g} = \tanh(\mathbf{W}^{\mathbf{x}\mathbf{g}} \cdot \mathbf{x}_j + \mathbf{W}^{\mathbf{h}\mathbf{g}} \cdot \mathbf{h}_{j-1})$$

$$\frac{\partial c_i}{\partial c_{i-1}} = \frac{\partial f_i}{\partial c_{i-1}} c_{i-1} + \frac{\partial c_{i-1}}{\partial c_{i-1}} f_i + \frac{\partial i_i}{\partial c_{i-1}} g_i + \frac{\partial c_{i-1}}{\partial c_{i-1}} i_i$$

GRU vs LSTM

- The GRU and the LSTM are very similar ideas.
- Invented independently of the LSTM, almost two decades later.

Other Variants

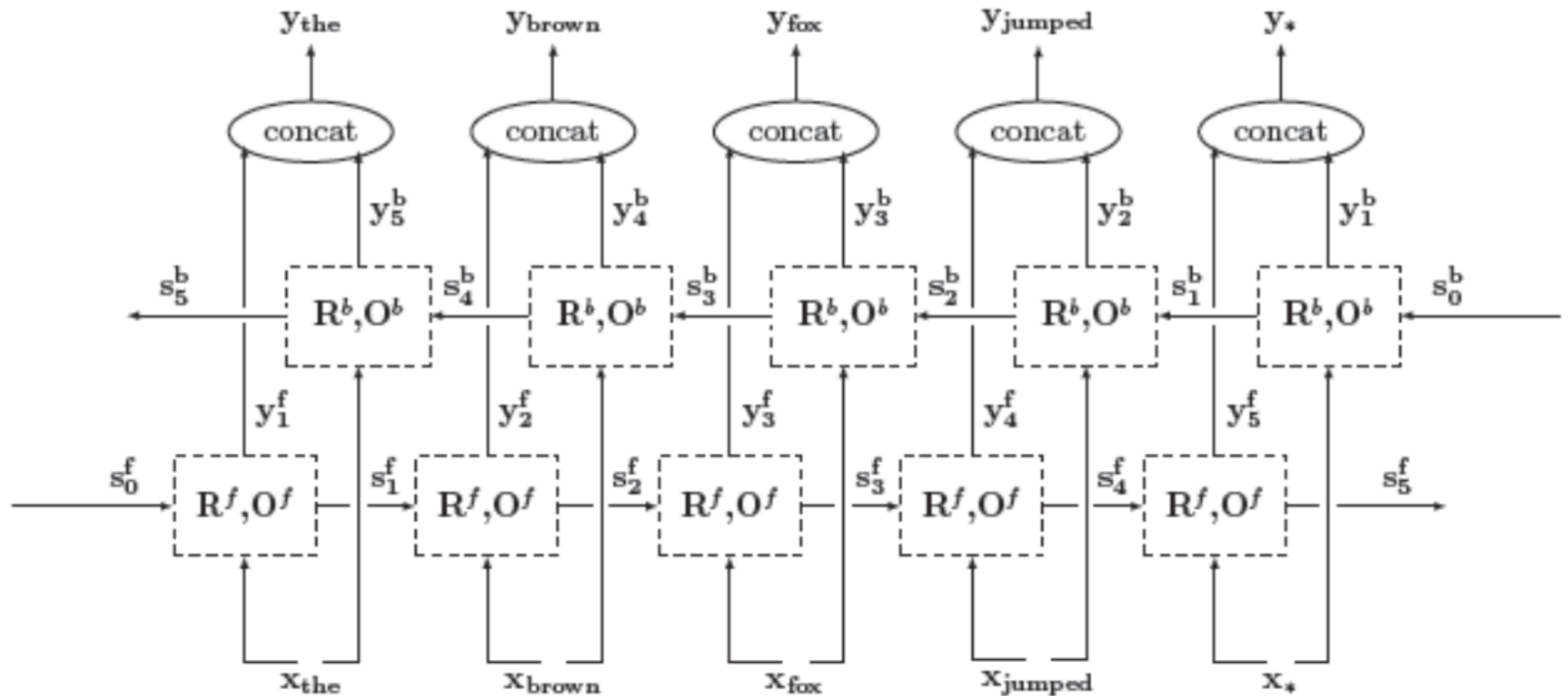
- Many other variants exist.
- Mostly perform similarly to each other.
 - Different tasks may work better with different variants.
- **The important idea is the differentiable gates.**

LSTM: A Search Space Odyssey

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber

- Systematic search over LSTM choices
- Find that (1) forget gate is most important
- (2) non-linearity in output important since cell state can be unbounded
- GRU effective since it doesn't let cell state be unbounded

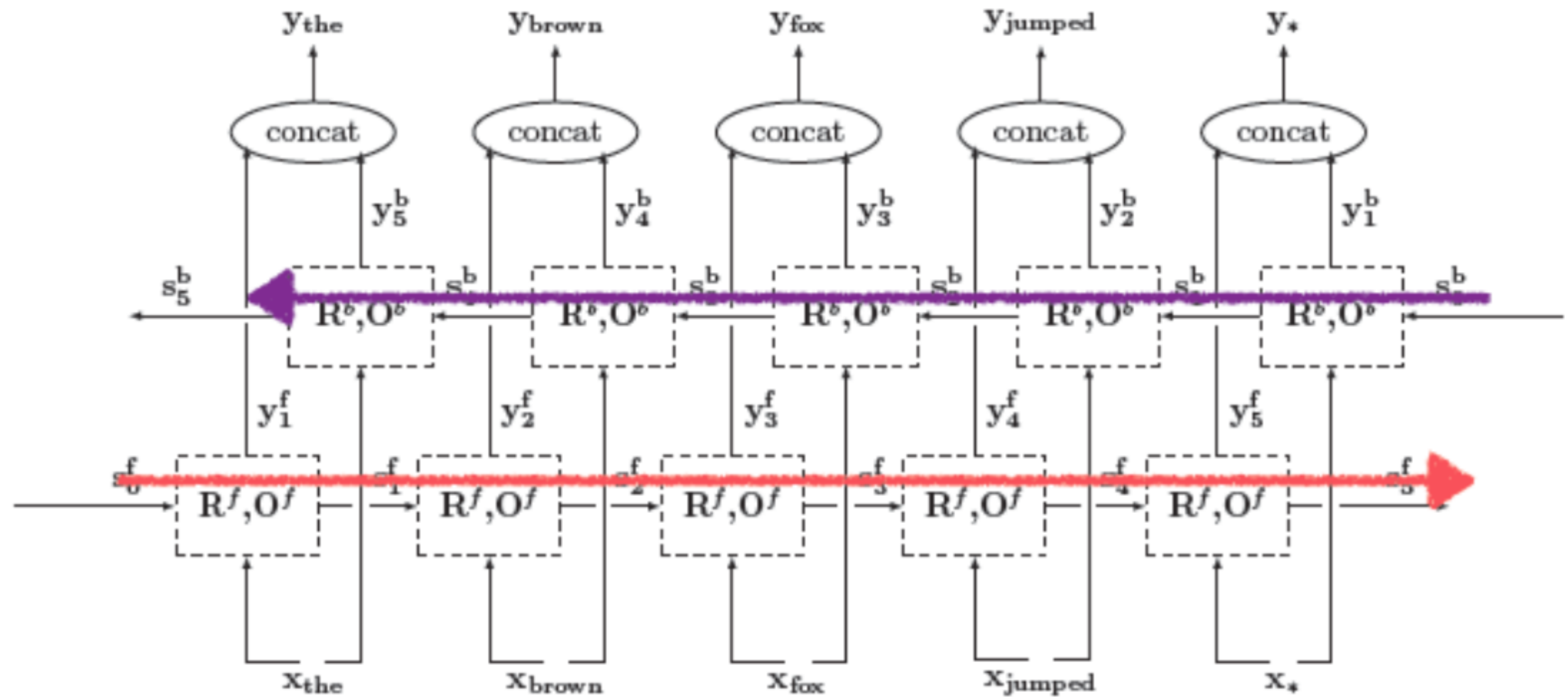
Bidirectional LSTMs



One RNN runs left to right.

Another runs right to left.

Encode **both future and history** of a word.

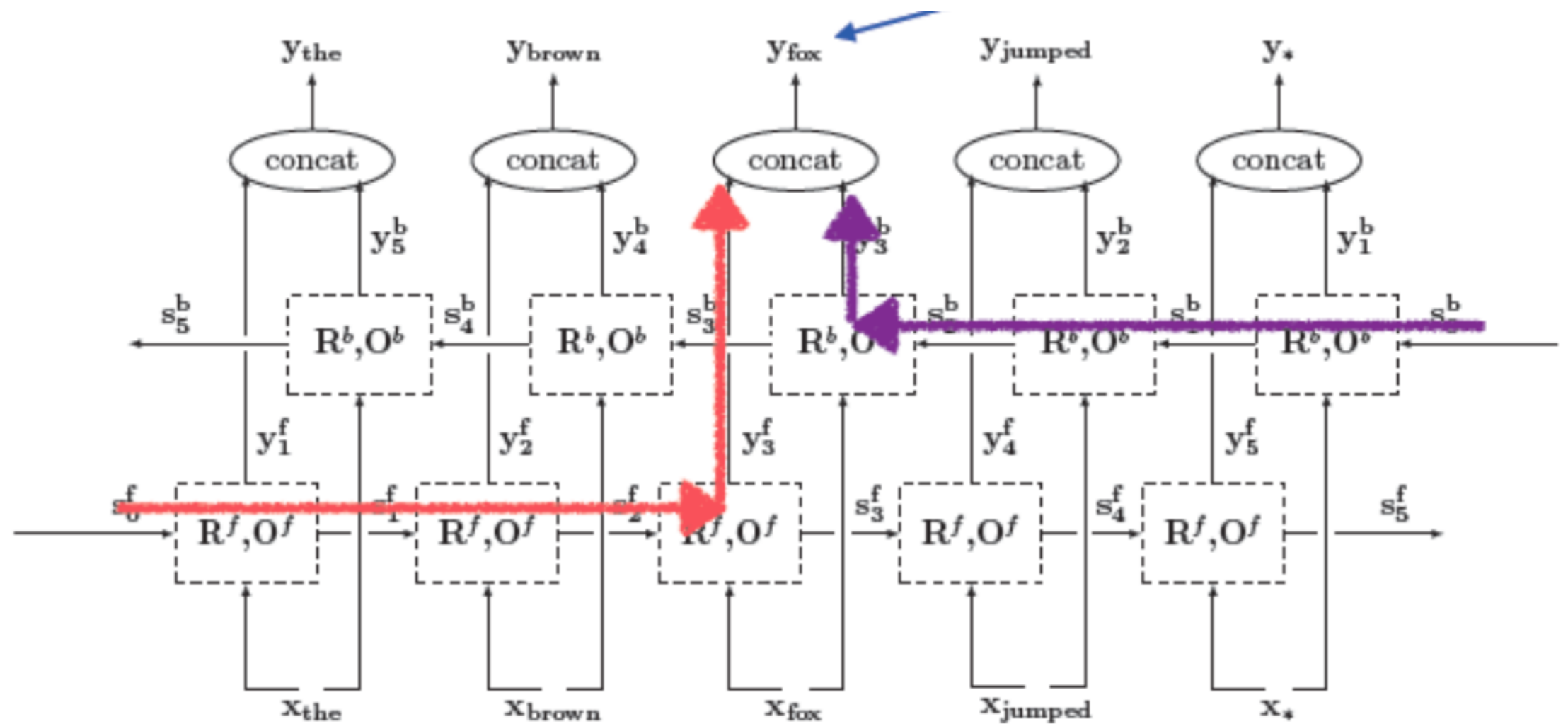


One RNN runs left to right.

Another runs right to left.

Encode **both future and history** of a word.

Infinite window around the word

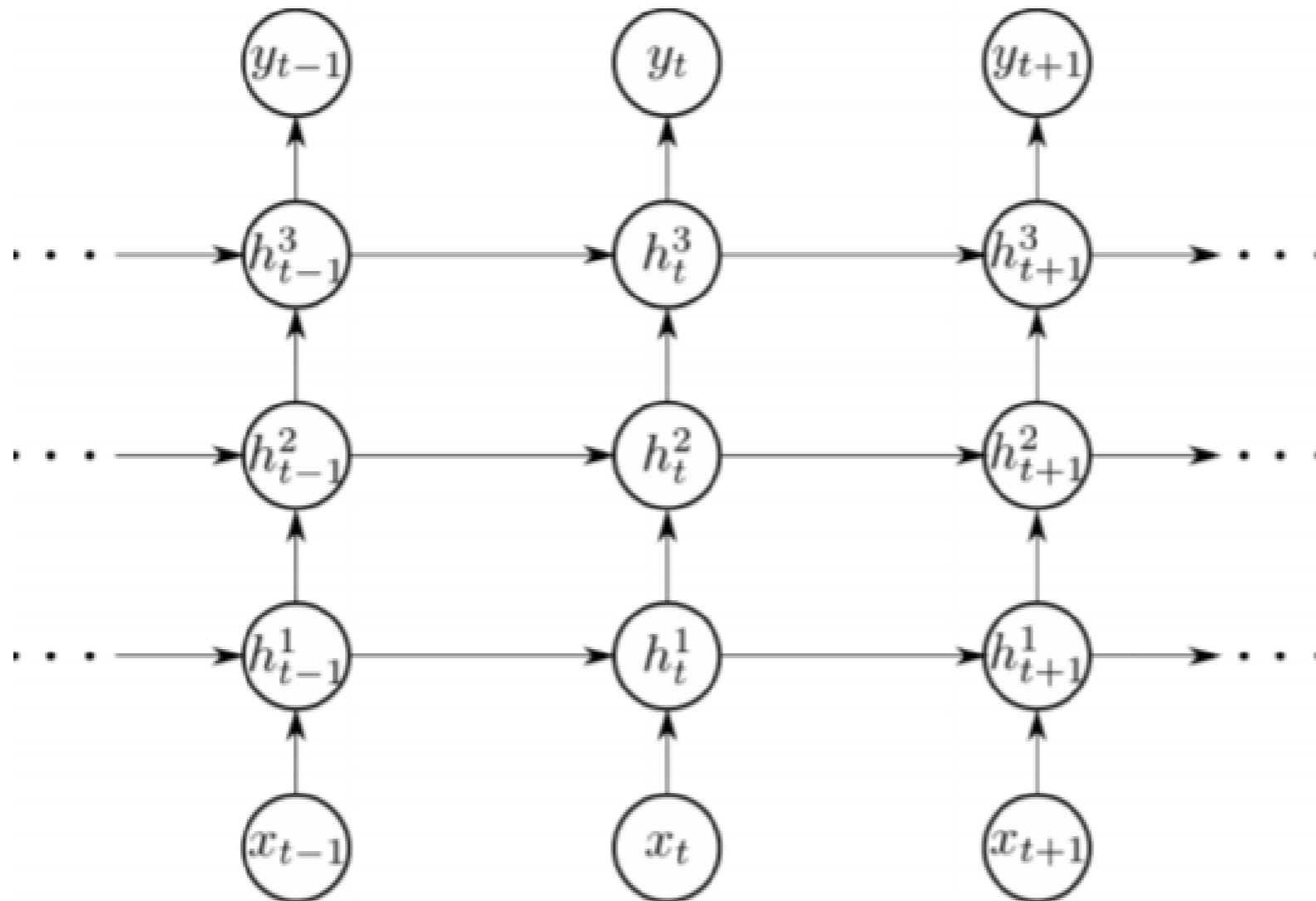


One RNN runs left to right.

Another runs right to left.

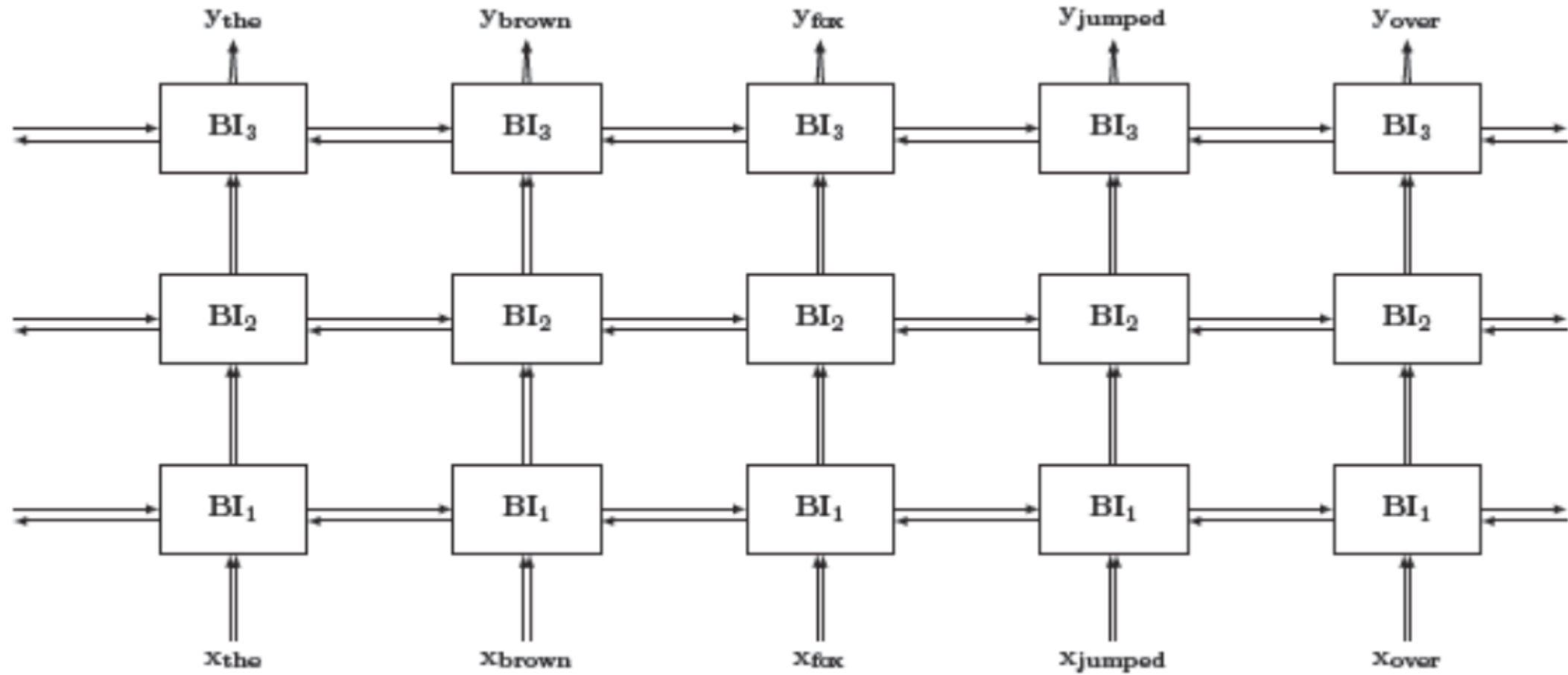
Encode **both future and history** of a word.

Deep LSTMs



(a) Conventional stacked RNN

Deep Bi-LSTMs



Read More

- The gated architecture also helps the vanishing gradients problems.
- For a good explanation, see Kyunghyun Cho's notes:
<http://arxiv.org/abs/1511.07916> sections 4.2, 4.3
- Chris Olah's blog post

Pooling in RNNs (2020)

**Why and when should you pool?
Analyzing Pooling in Recurrent Architectures**

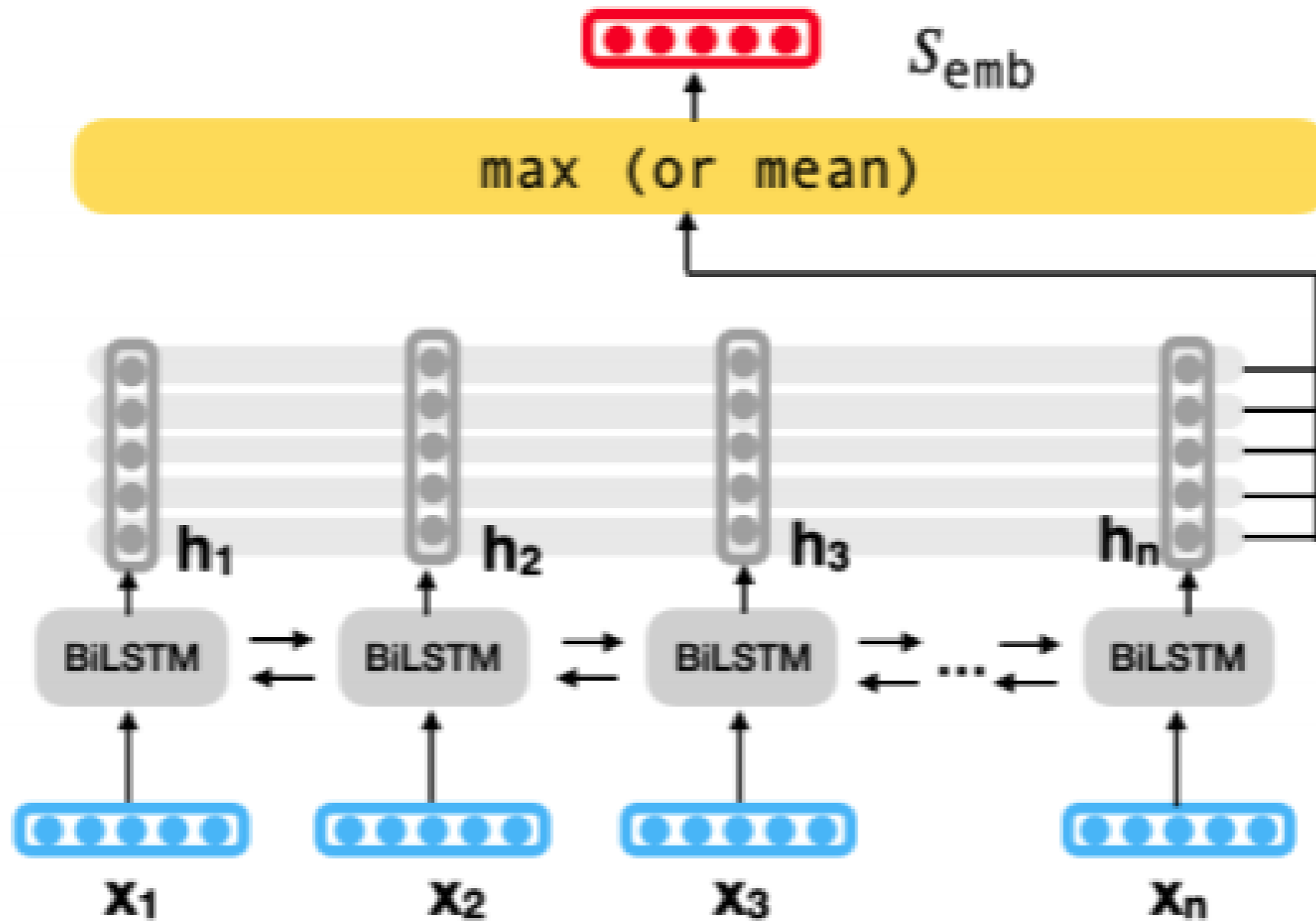
Pratyush Maini[†], Keshav Kolluru[†], Danish Pruthi[‡], Mausam[†]

[†]Indian Institute of Technology, Delhi, India

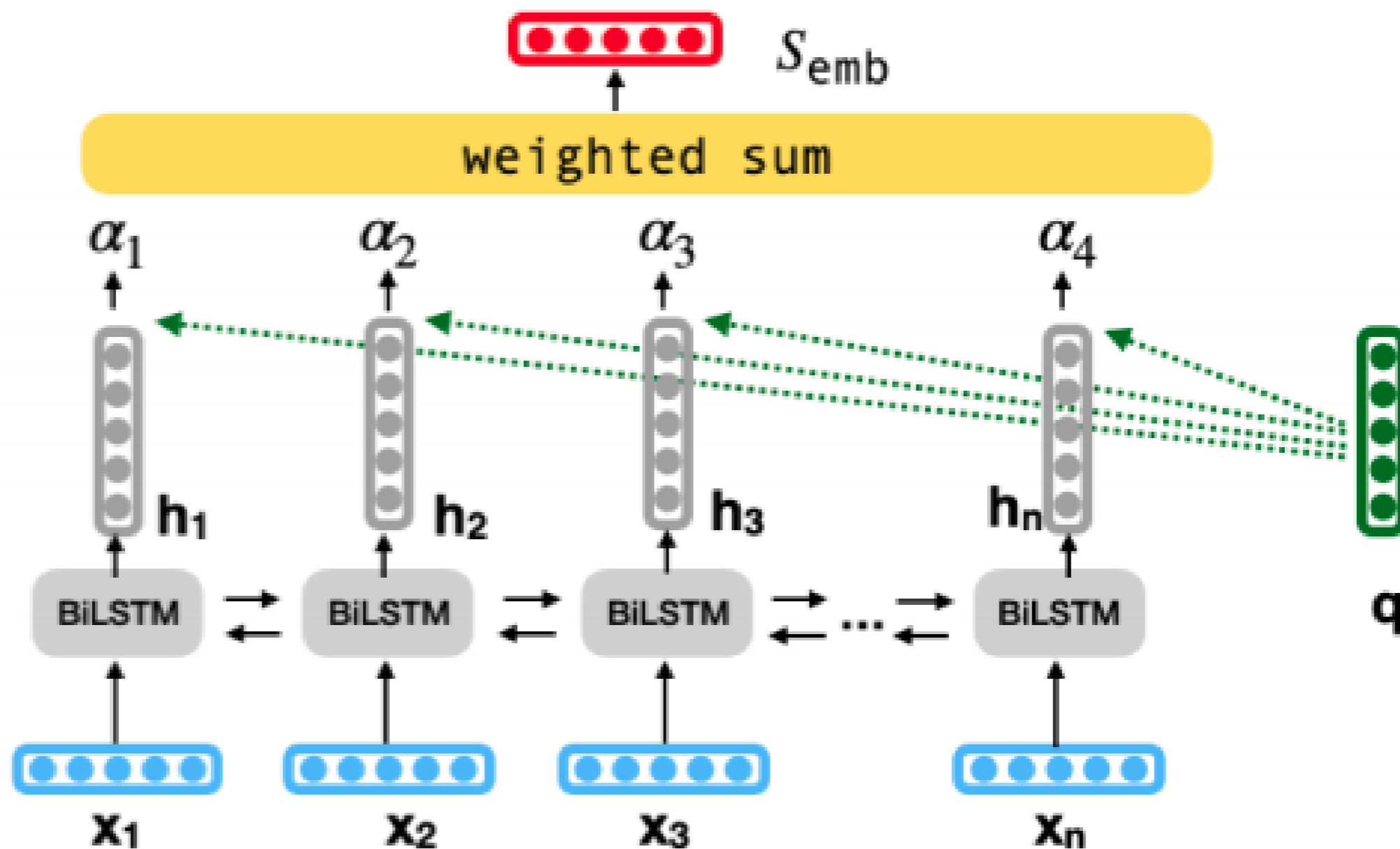
[‡]Carnegie Mellon University, Pittsburgh, USA

{pratyush.maini, keshav.kolluru}@gmail.com,
ddanish@cs.cmu.edu, mausam@cse.iitd.ac.in

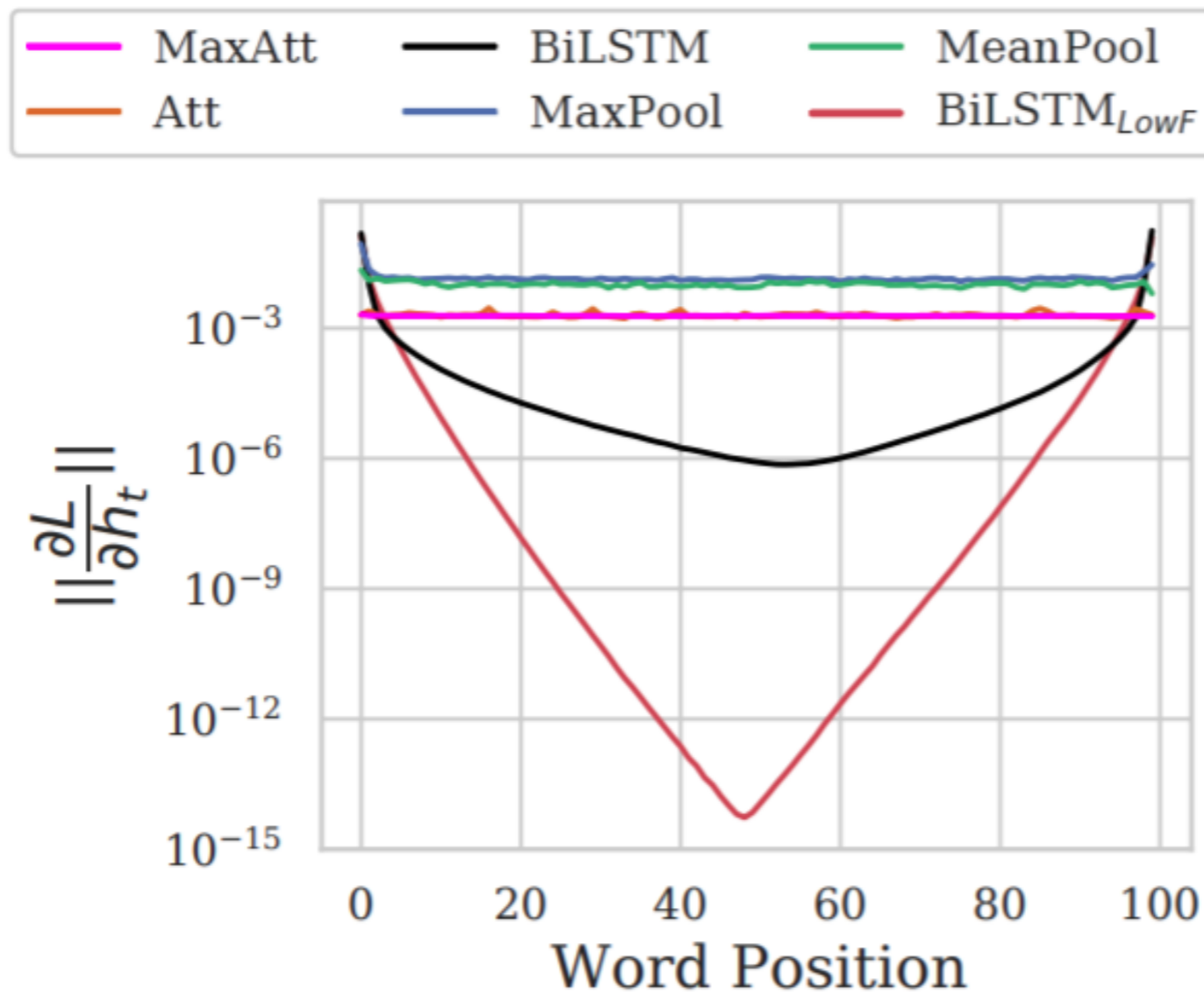
Pooling



Attention



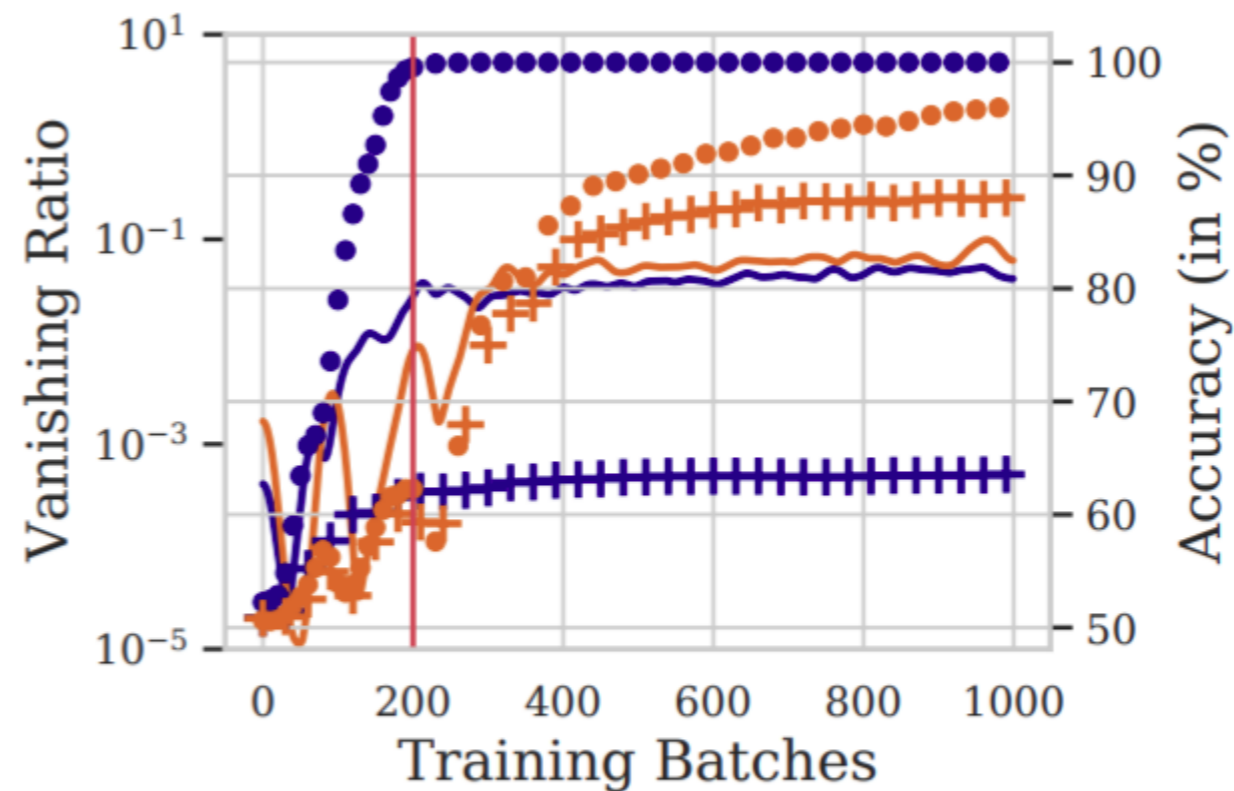
Vanishing Gradients @~Start of Training



(a) Gradient Norms

Vanishing Ratio

$$\text{vanishing ratio } \left(\left\| \frac{\partial L}{\partial h_{\text{mid}}} \right\| / \left\| \frac{\partial L}{\partial h_{\text{end}}} \right\| \right)$$



(b) BiLSTM

— Vanishing Ratio ● Training Acc. + Validation Acc. ■ 1 K ■ 20 K

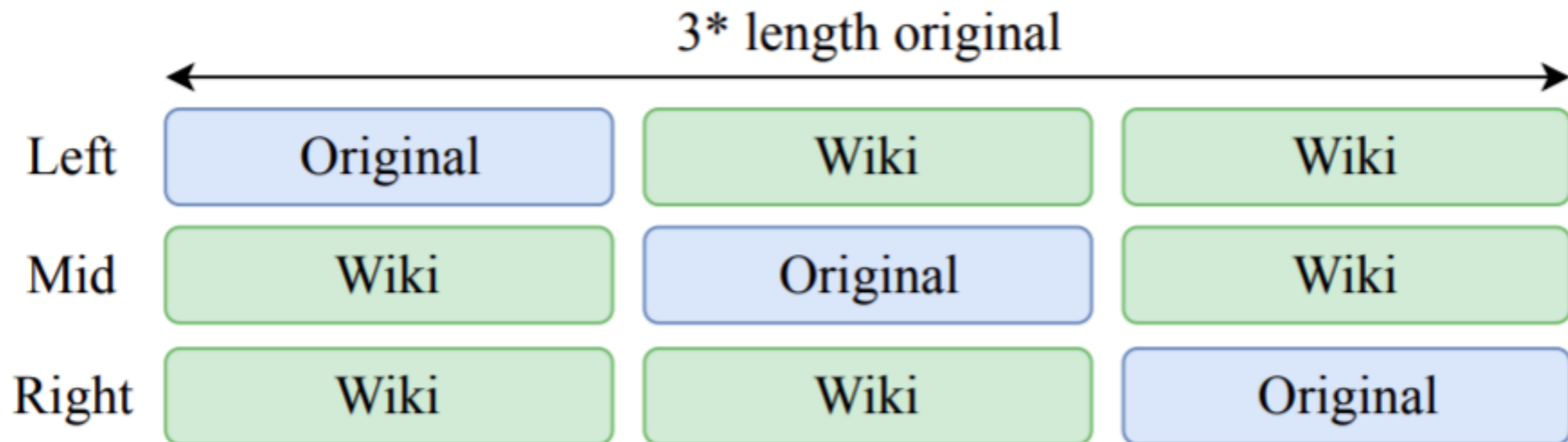
Size-Accuracy-Vanishing

	Vanishing ratio			Validation acc.		
	1K	5K	20K	1K	5K	20K
BiLSTM	5×10^{-3}	0.03	0.06	64.9	82.8	88.4
MEANPOOL	2.5	0.56	1.32	78.4	82.6	88.5
MAXPOOL	0.40	0.42	0.53	78.0	84.7	89.6
ATT	3.87	1.04	1.19	77.1	84.6	90.0
MAXATT	0.69	0.69	0.64	78.1	86.0	90.2

Table 2: Values of vanishing ratio as computed when different models achieve 95% training accuracy, along with the best validation accuracy for that run.

Important Words in Middle?

How well can different models be trained to skip unrelated words?



Results

	IMDb			IMDb (mid) + Wiki			IMDb (right) + Wiki		
	1K	2K	10K	1K	2K	10K	1K	2K	10K
BiLSTM	64.7 \pm 2.3	75.0 \pm 0.4	86.6 \pm 0.8	49.6 \pm 0.7	49.9 \pm 0.5	50.3 \pm 0.3	53.5 \pm 2.5	64.7 \pm 2.8	85.9 \pm 0.5
MEANPOOL	73.0 \pm 3.0	81.7 \pm 0.7	87.1 \pm 0.6	69.8 \pm 2.1	76.2 \pm 1.0	84.1 \pm 0.7	70.0 \pm 1.1	76.8 \pm 1.0	84.8 \pm 0.9
MAXPOOL	69.0 \pm 3.9	80.1 \pm 0.5	87.8 \pm 0.6	64.5 \pm 1.8	77.2 \pm 2.0	86.0 \pm 0.8	65.9 \pm 4.6	77.8 \pm 0.9	87.2 \pm 0.6
ATT	75.7 \pm 2.6	82.8 \pm 0.8	89.0 \pm 0.3	75.0 \pm 0.8	79.4 \pm 0.8	86.7 \pm 1.4	74.7 \pm 1.4	80.2 \pm 1.8	87.1 \pm 1.0
MAXATT	75.9 \pm 2.2	82.5 \pm 0.4	88.5 \pm 0.5	75.4 \pm 2.4	80.9 \pm 1.8	86.8 \pm 0.5	77.9 \pm 0.9	81.9 \pm 0.5	87.2 \pm 0.5
	Yahoo			Yahoo (mid) + Wiki			Yahoo (right) + Wiki		
	1K	2K	10K	1K	2K	10K	1K	2K	10K
BiLSTM	38.3 \pm 4.8	51.4 \pm 2.1	63.5 \pm 0.6	12.7 \pm 1.1	12.7 \pm 1.1	11.4 \pm 0.8	18.8 \pm 2.5	37.3 \pm 0.9	60.1 \pm 1.5
MEANPOOL	48.2 \pm 2.3	56.6 \pm 0.5	64.7 \pm 0.6	31.9 \pm 2.3	43.1 \pm 2.0	58.5 \pm 0.6	33.9 \pm 2.1	43.2 \pm 1.0	58.6 \pm 0.4
MAXPOOL	50.2 \pm 2.1	56.3 \pm 1.8	63.9 \pm 1.1	33.0 \pm 1.0	40.1 \pm 1.4	58.4 \pm 1.2	33.1 \pm 2.5	41.2 \pm 0.9	60.9 \pm 1.0
ATT	47.3 \pm 2.2	54.2 \pm 1.1	65.1 \pm 1.5	39.4 \pm 0.5	45.1 \pm 1.8	61.5 \pm 1.7	37.9 \pm 1.4	47.6 \pm 2.3	62.2 \pm 0.9
MAXATT	51.8 \pm 1.1	57.0 \pm 1.1	65.1 \pm 1.1	39.6 \pm 0.9	48.5 \pm 0.6	62.2 \pm 1.6	40.3 \pm 1.5	50.1 \pm 1.6	63.1 \pm 0.7

More Experiments

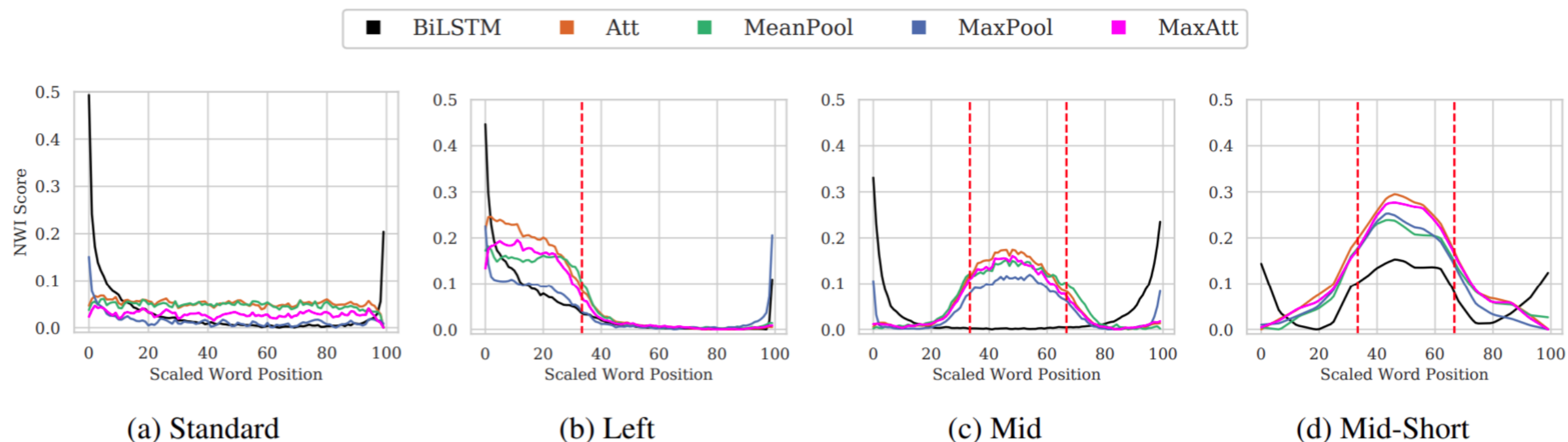


Figure 6: Normalized Word Importance w.r.t. word position averaged over examples of length between 400-500 on the Yahoo (25K) dataset in (a,b,c) using $k = 5$; and NWI for examples of length between 50-60 on the Yahoo Short (25K) dataset in (d) with $k = 3$. Results shown for ‘standard’, ‘left’ & ‘mid’ training settings described in § 6.2. The vertical red line represents a separator between relevant and irrelevant information (by construction).

Conclusions

- pooling mitigates the problem of vanishing gradients
- pooling eliminates positional biases
- gradients in BiLSTM vanish only in initial iterations, recover slowly during further training
- We link the observation with training saturation to provide insights as to why BiLSTMs fail in low resource setups but pooled architectures don't
- BiLSTMs suffer from positional biases even when sentence lengths are short: ~30 words
- pooling makes models significantly more robust to insertions of words on either end of the input regardless of the amount of training data