Recurrent Neural Networks

Mausam
IIT Delhi

(some slides by Yoav Goldberg, Silviu Pitis)

Common NLP Tasks

- Word-level Tasks
  - Understanding word synonyms, word senses…
- Sentence/Document Classification
  - Sentiment Mining, Fake news detection, Racist tweet classification
- Sequence Labeling
  - POS Tagging, Noun Phrase Chunking, Named Entity Recognition
- Parsing: converting sentence to its syntactic structure
- Generation Tasks
  - Machine Translation, Summarization, Dialogue Systems

Common NLP Tasks

- Word-level Tasks
  - Understanding word synonyms, word senses…
- Sentence/Document Classification
  - Sentiment Mining, Fake news detection, Racist tweet classification
- Sequence Labeling
  - POS Tagging, Noun Phrase Chunking, Named Entity Recognition
- Parsing: converting sentence to its syntactic structure
- Generation Tasks
  - Machine Translation, Summarization, Dialogue Systems

Main Challenge in Text Data

- Input (sentence) is *variable* length
- Classification: Output may be a single bit

This book is a fantastic read.
This movie should never have been made. → Model → Positive
Negative

- Sequence Labeling: Output may be a sequence of same length as input

Rama killed Ravana with an arrow. → Model → NNP VBD NNP PREP DT NN

- Generation: Output may be sequence of length different from input

Rama killed Ravana with an arrow. → Model → राम ने एक तीर से रावण की हत्या की

Dealing with Sequences

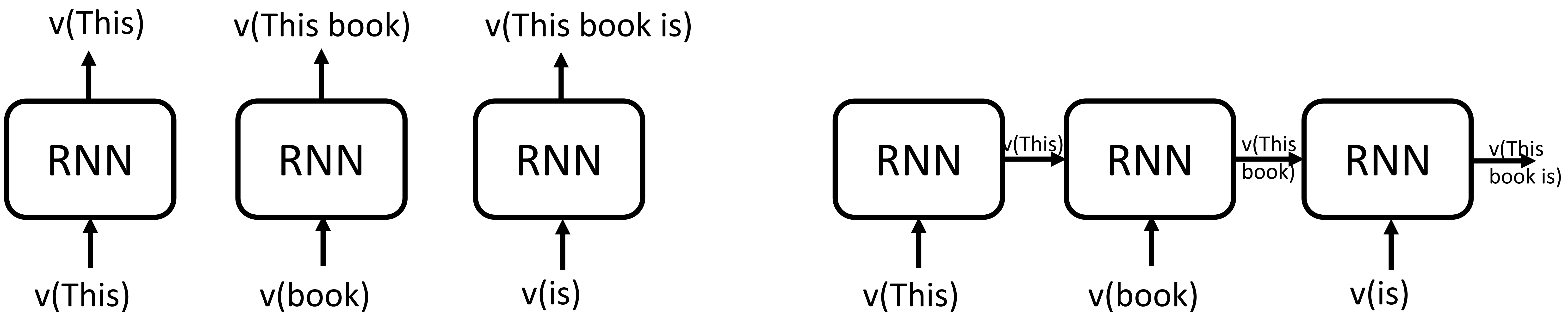
- For an input sequence **x1**,...,**xn**, we can:
  - If *n* is **fixed**: *concatenate* and feed into an MLP.
  - *su*
  - Br
    co
  - Fir bine to
    a single vector.

Some of these approaches consider **local** word order

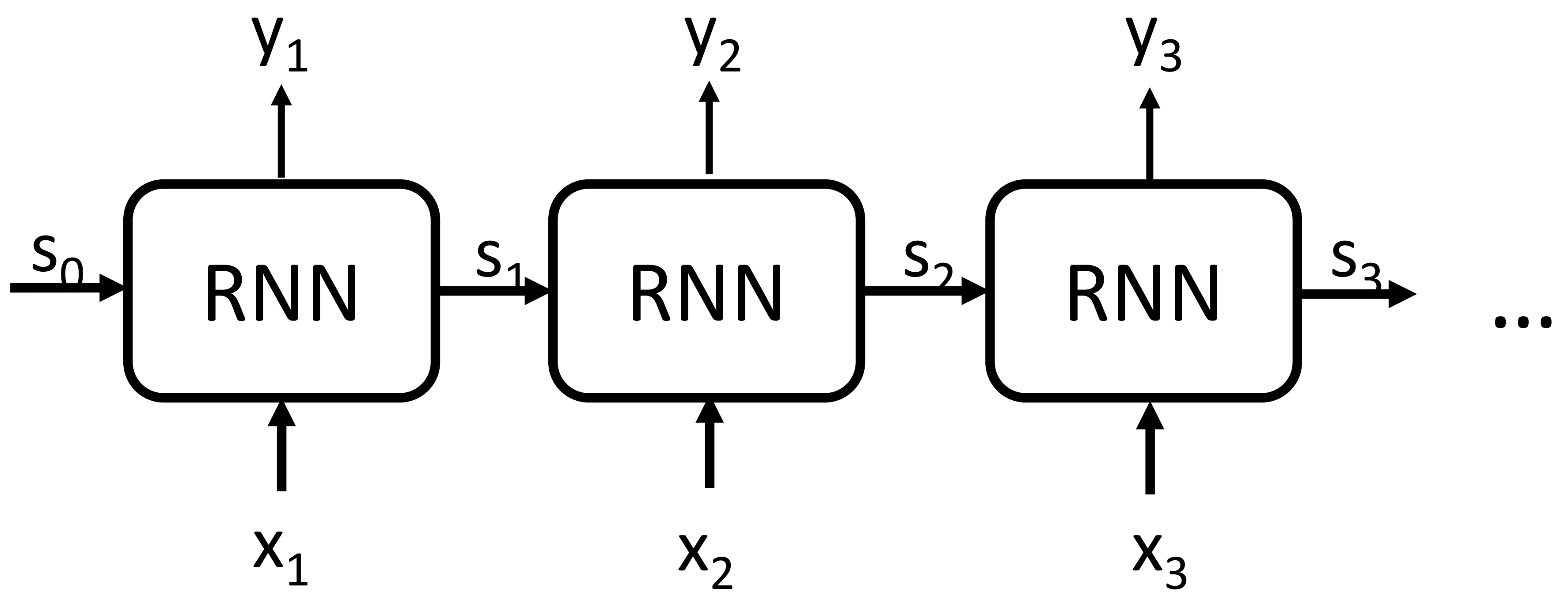
How can we consider **global** word order?

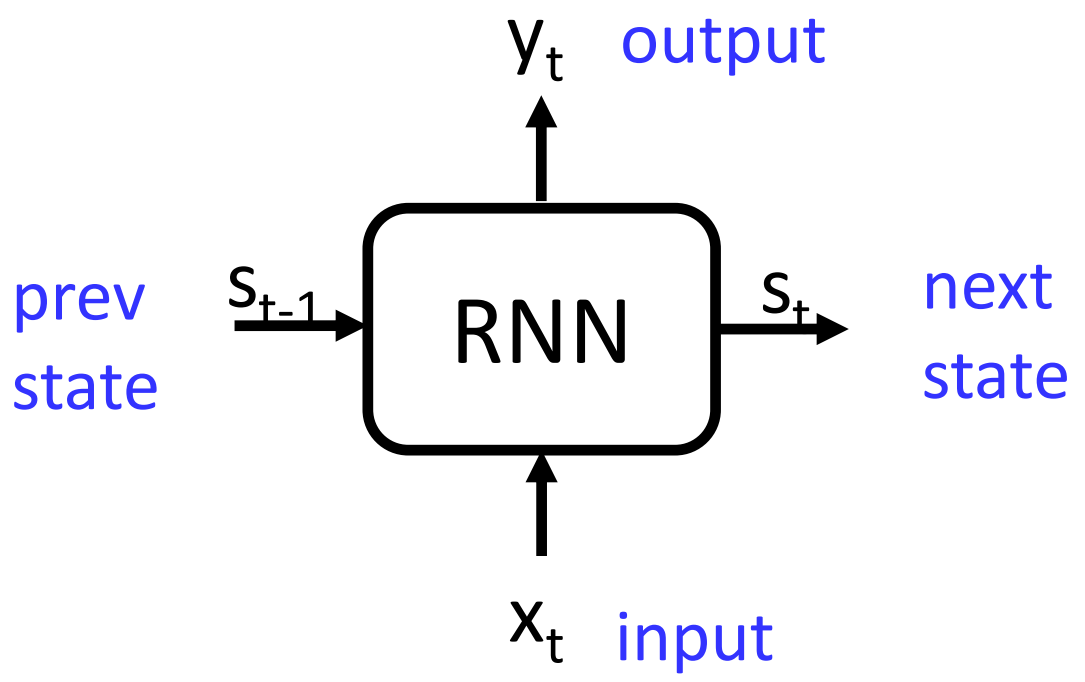
Recurrent Neural Networks (Encoder)

- Model to handle variable length input
  - Parameters/model cannot be position dependent
  - Same computation will be repeated at every position



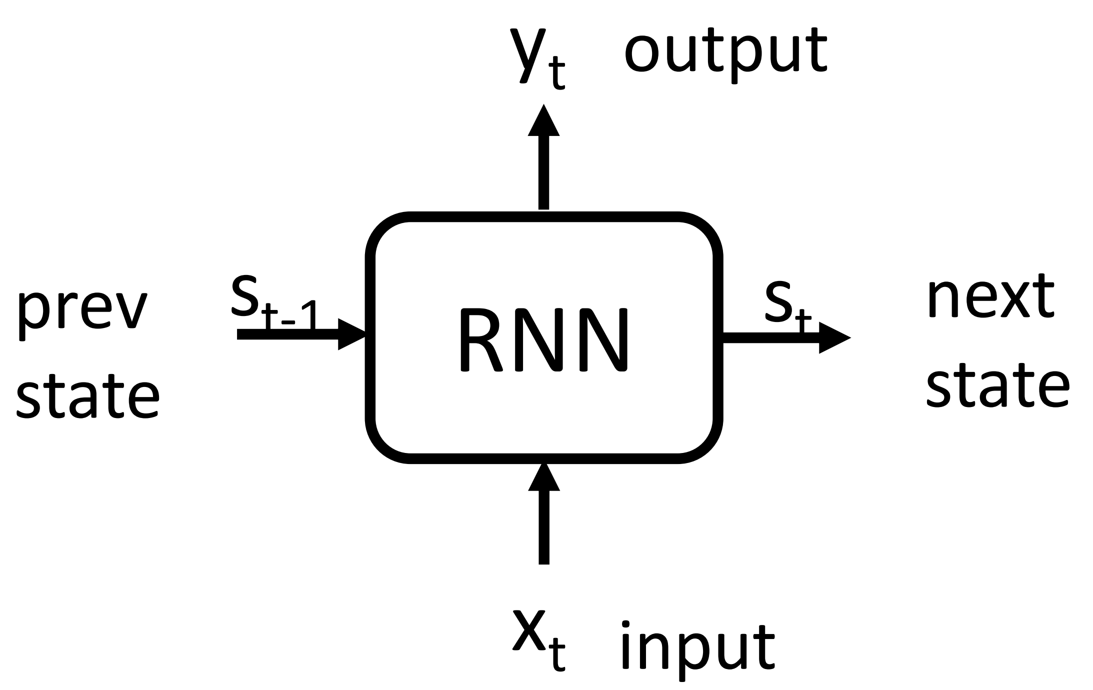
Recurrent Neural Networks (Encoder)





Recurrent Neural Networks (Encoder)

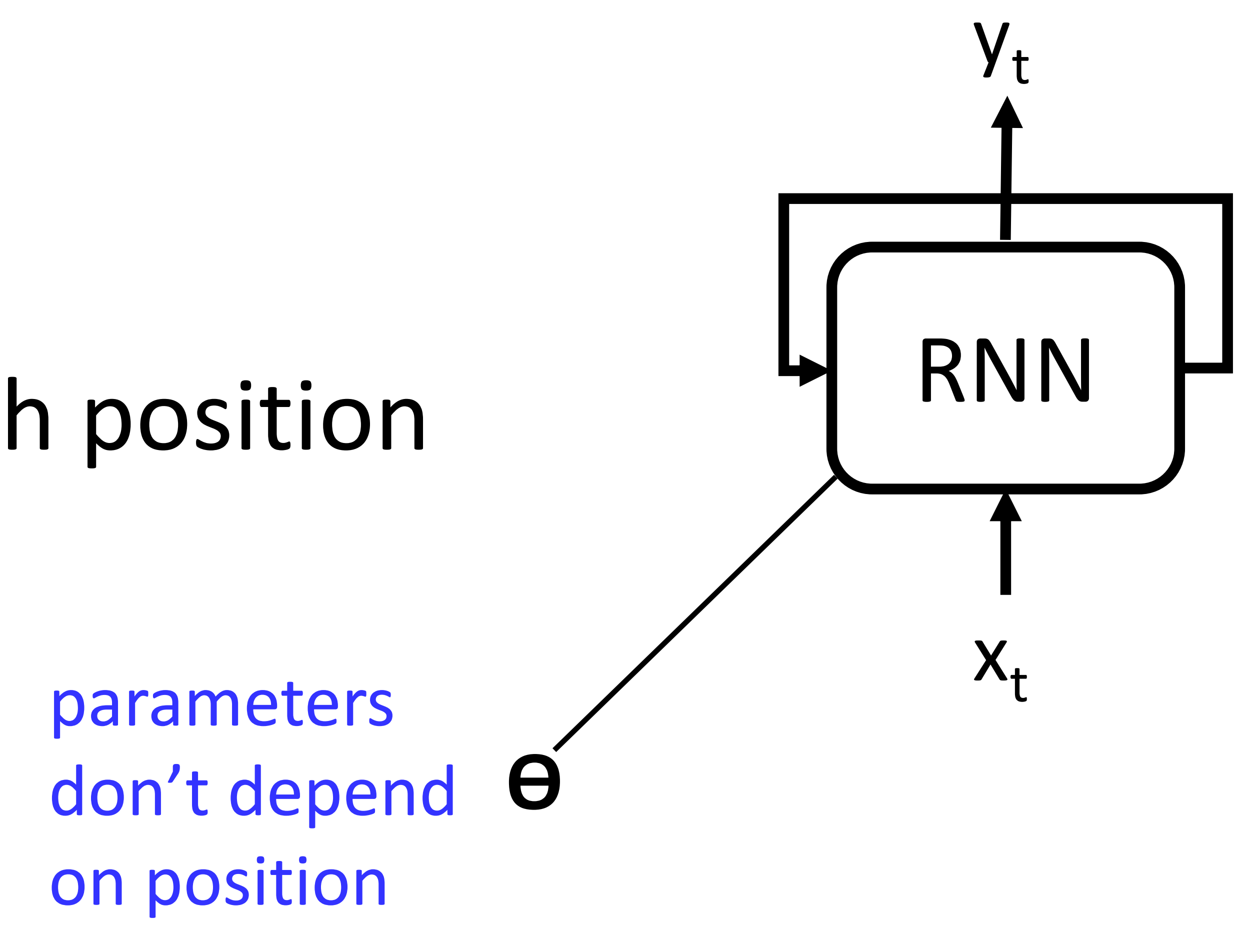
$$RNN(s_{t-1}, x_t) = s_t, y_t$$



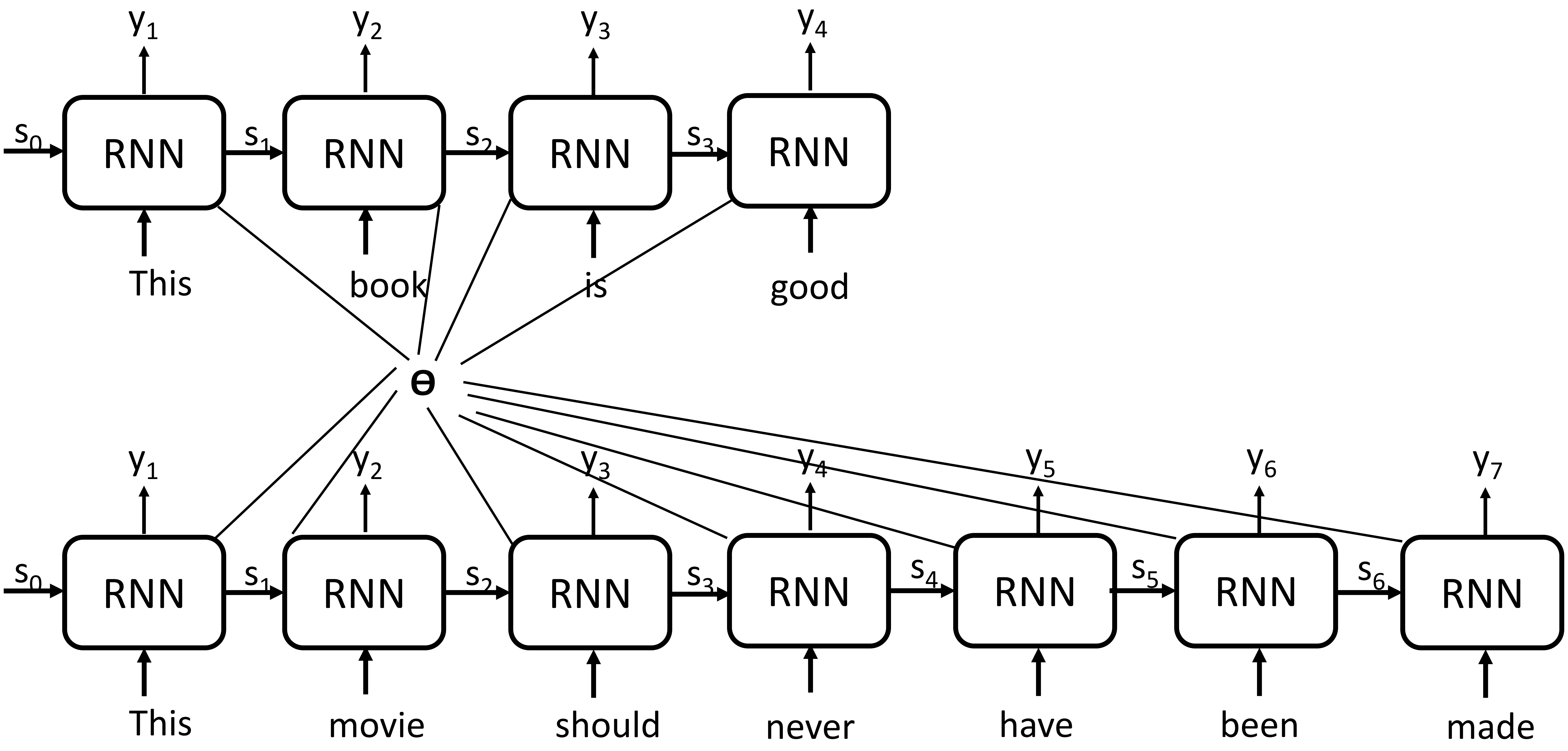
$$x_t \in \mathbb{R}^{din}$$
$$y_t \in \mathbb{R}^{dout}$$
$$s_t \in \mathbb{R}^{dstate}$$

$$s_t = R(s_{t-1}, x_t)$$
$$y_t = O(s_t)$$

- They are called recurrent nets
  - because the same computation recurs at each position
- There's a vector y_t for every prefix $x_{1:t}$



Unrolling an RNN



y_t depends on $x_{1:t}$

$$
\begin{aligned}
y_t &= O(s_t) \\
s_t &= R(s_{t-1}, x_t) \\
&= R(R(s_{t-2}, x_{t-1}), x_t) \\
&= R(R(R(s_{t-3}, x_{t-2}), x_{t-1}), x_t) \\
&\quad \text{....} \\
&= R(R(R \ldots R(s_0, x_1), x_2), \ \ldots), x_t)
\end{aligned}
$$

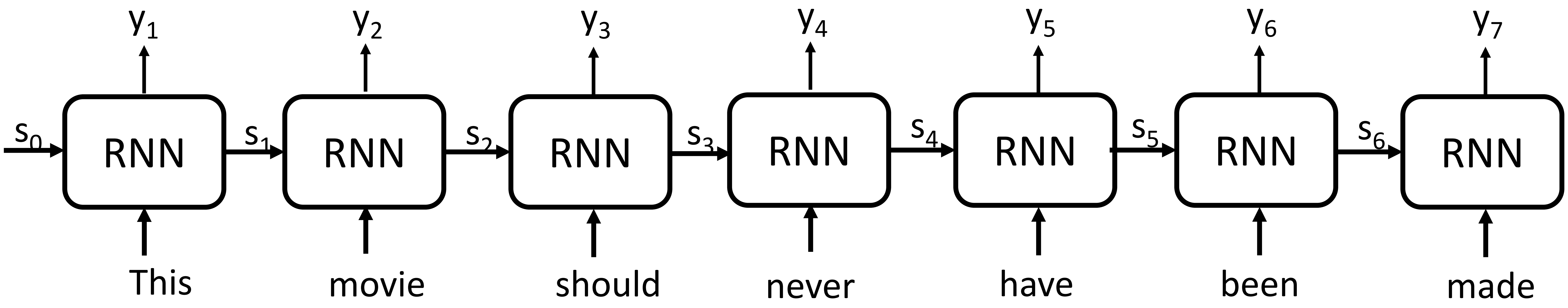
y_t depends on $x_{1:t}$

$$
\begin{aligned}
y_t &= O(s_t) \\
s_t &= R(s_{t-1}, x_t) \\
&= R(R(s_{t-2}, x_{t-1}), x_t) \\
&= R(R(R(s_{t-3}, x_{t-2}), x_{t-1}), x_t) \\
&\quad \ldots. \\
&= R(R(R \ldots R(s_0, x_1), x_2), \ \ldots), x_t)
\end{aligned}
$$

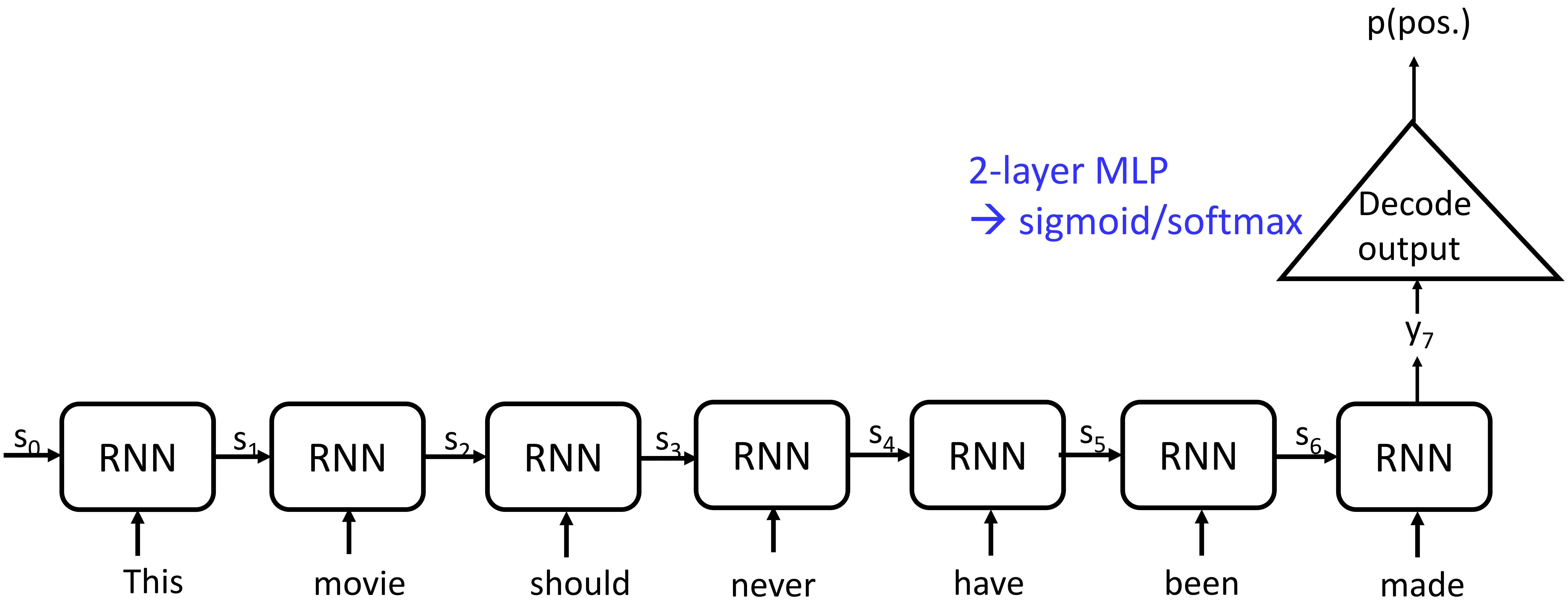
$$
\begin{aligned}
y_t &= O(s_t) \\
s_t &= RNN(s_0, x_{1:t})
\end{aligned}
$$

Classification: To make a single bit prediction for the full sentence decode y_t

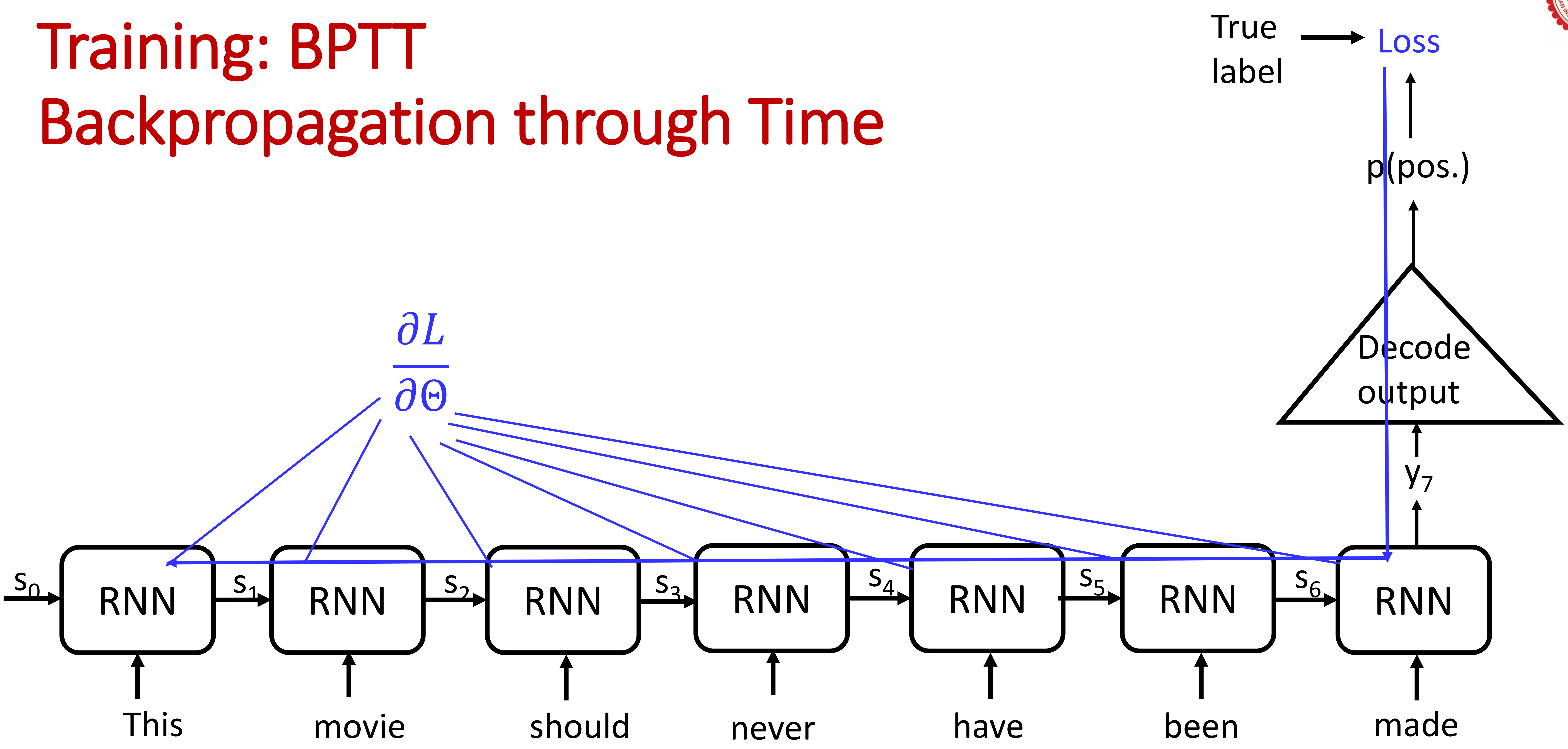
Sentiment Classification



Sentence Classification (Sentiment Mining)



Training: BPTT
Backpropagation through Time



Building a Simple RNN

$$s_t = R(s_{t-1}, x_t)$$

$$y_t = O(s_t)$$

- What are good functions for R and O ?

- Suggestion 1: $s_t = s_{t-1} + x_t$
- What are the parameters?
- Problem?

- Suggestion2: $s_t = \tanh(s_{t-1} + x_t + b^s)$
- Problem?

Building a Simple RNN

- What are good functions for R and O ?

$$s_t = R(s_{t-1}, x_t)$$
$$y_t = O(s_t)$$

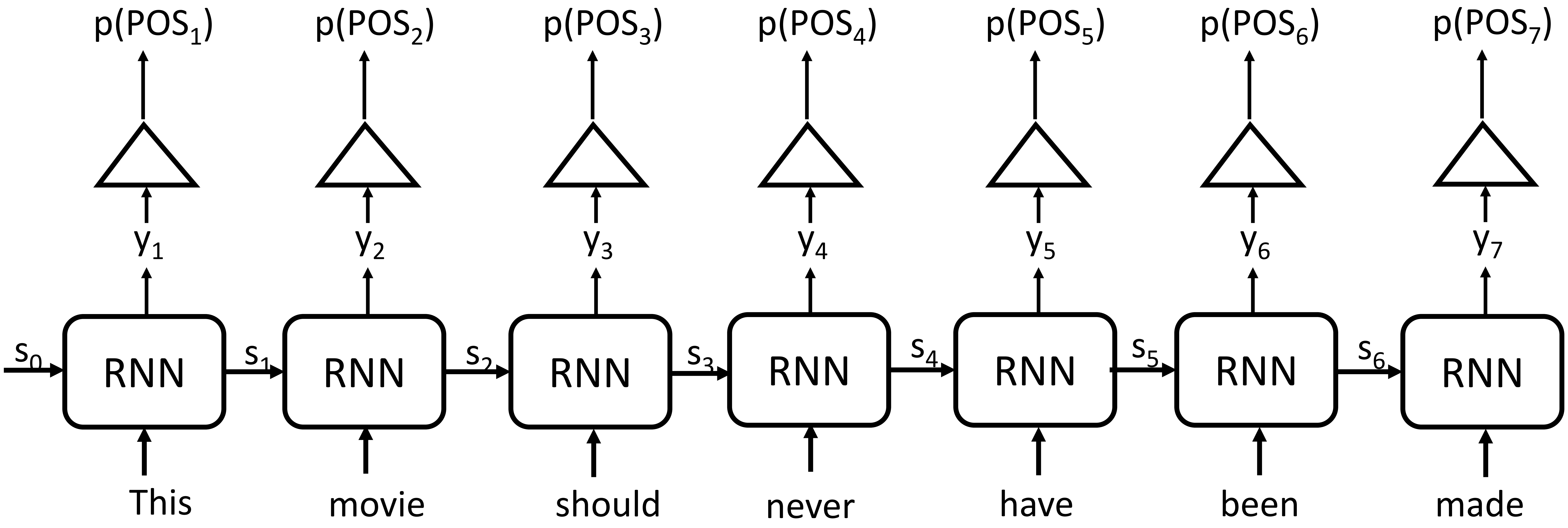
- Suggestion 1: $s_t = s_{t-1} + x_t$
- Problem?

- Suggestion2: $s_t = \tanh(s_{t-1} + x_t + b^s)$
- Problem?

- Elman's RNN: $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s) \longleftarrow s_t = R(s_{t-1}, x_t)$

  $y_t = \tanh(W^y s_t + b^y) \longleftarrow y_t = O(s_t)$

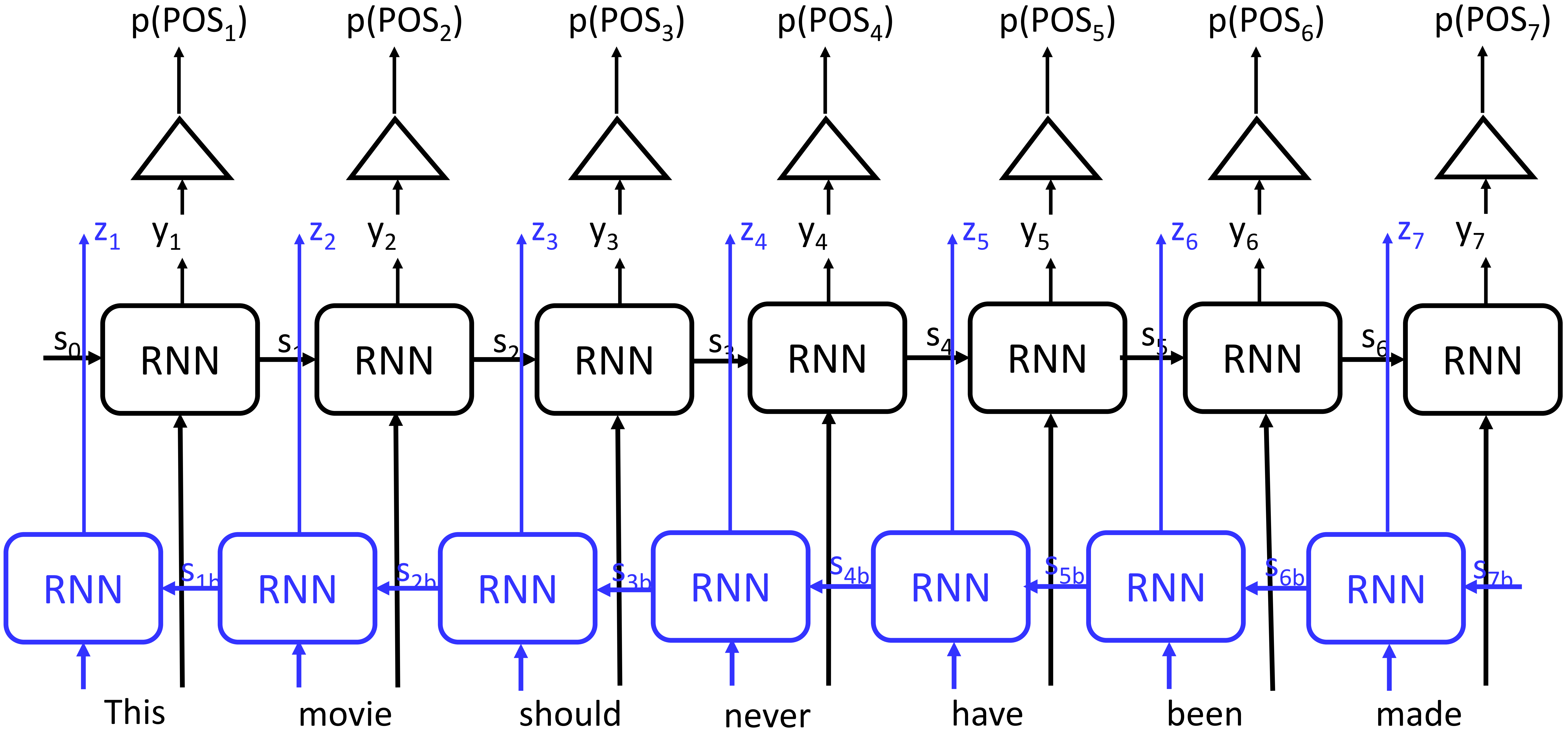
RNN Transducer for Sequence Labeling (POS Tagging)



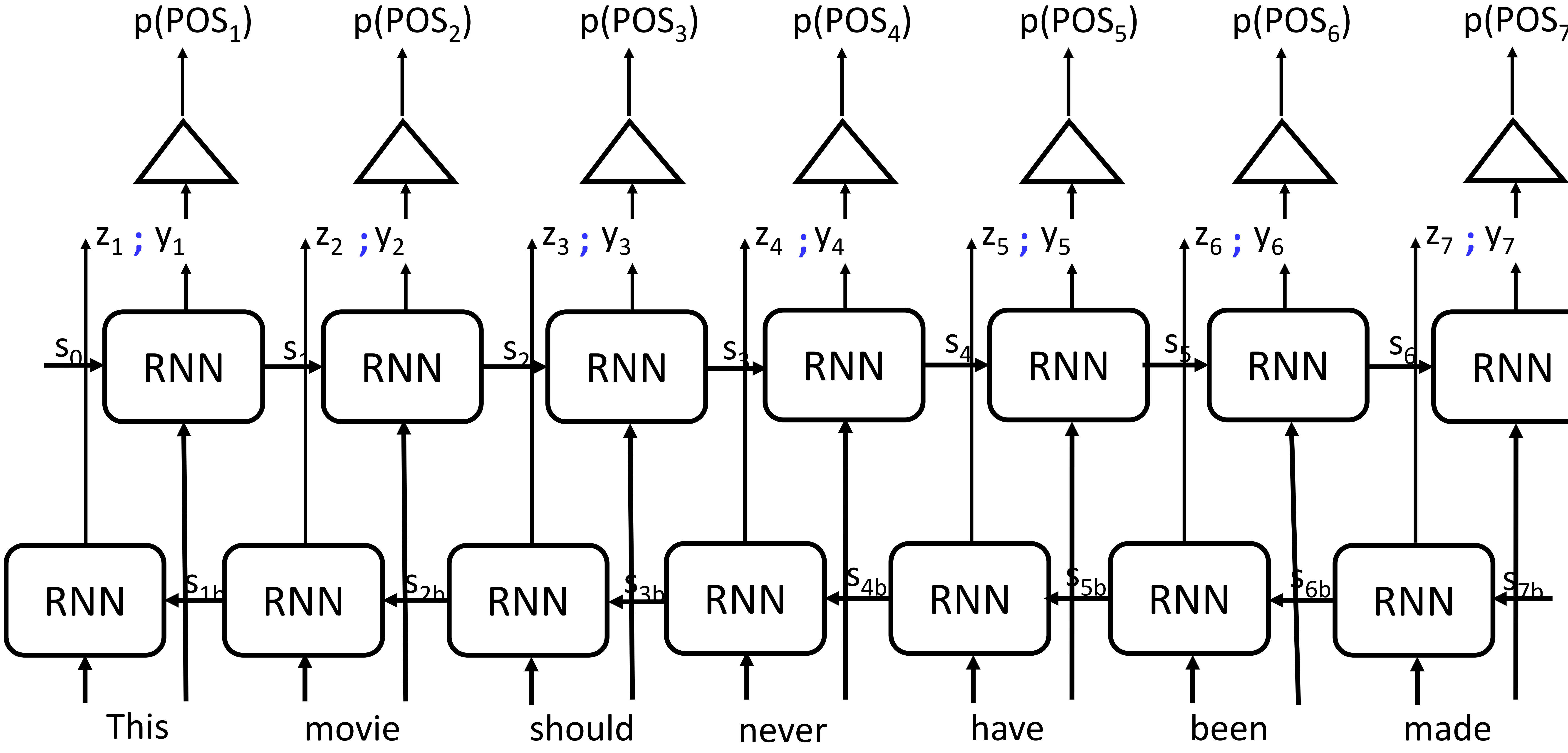
RNN → Bidirectional RNN

- An RNN s_t encodes all history $x_{1:t}$.
- But, future can also help in making a prediction
- Example: “the length is 6 hours” vs. “the length is 6 metres”
- A bidirectional RNN runs two unidirectional RNNs
- The final state encodes $x_{1:t}$ and $x_{t:T}$

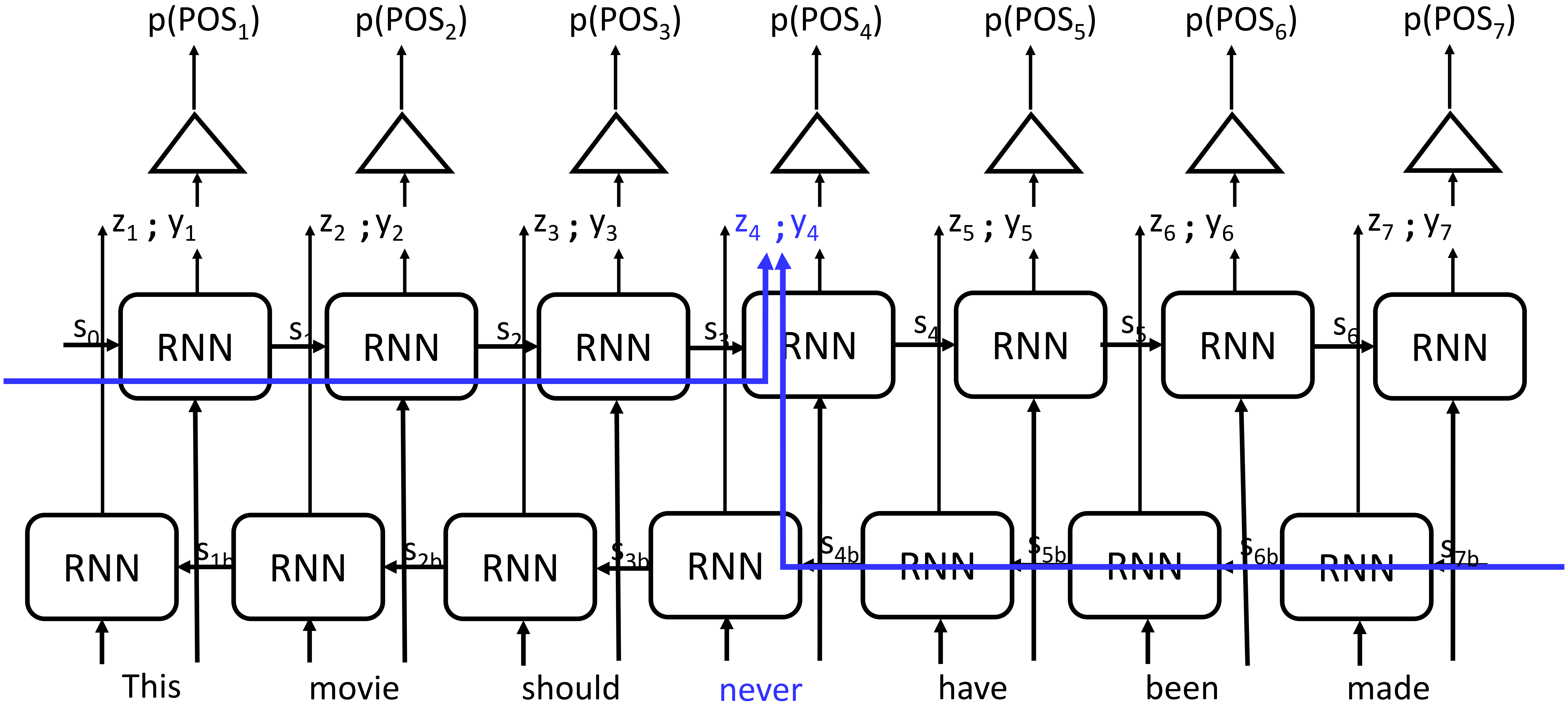
Bidirectional RNN



Bidirectional RNN

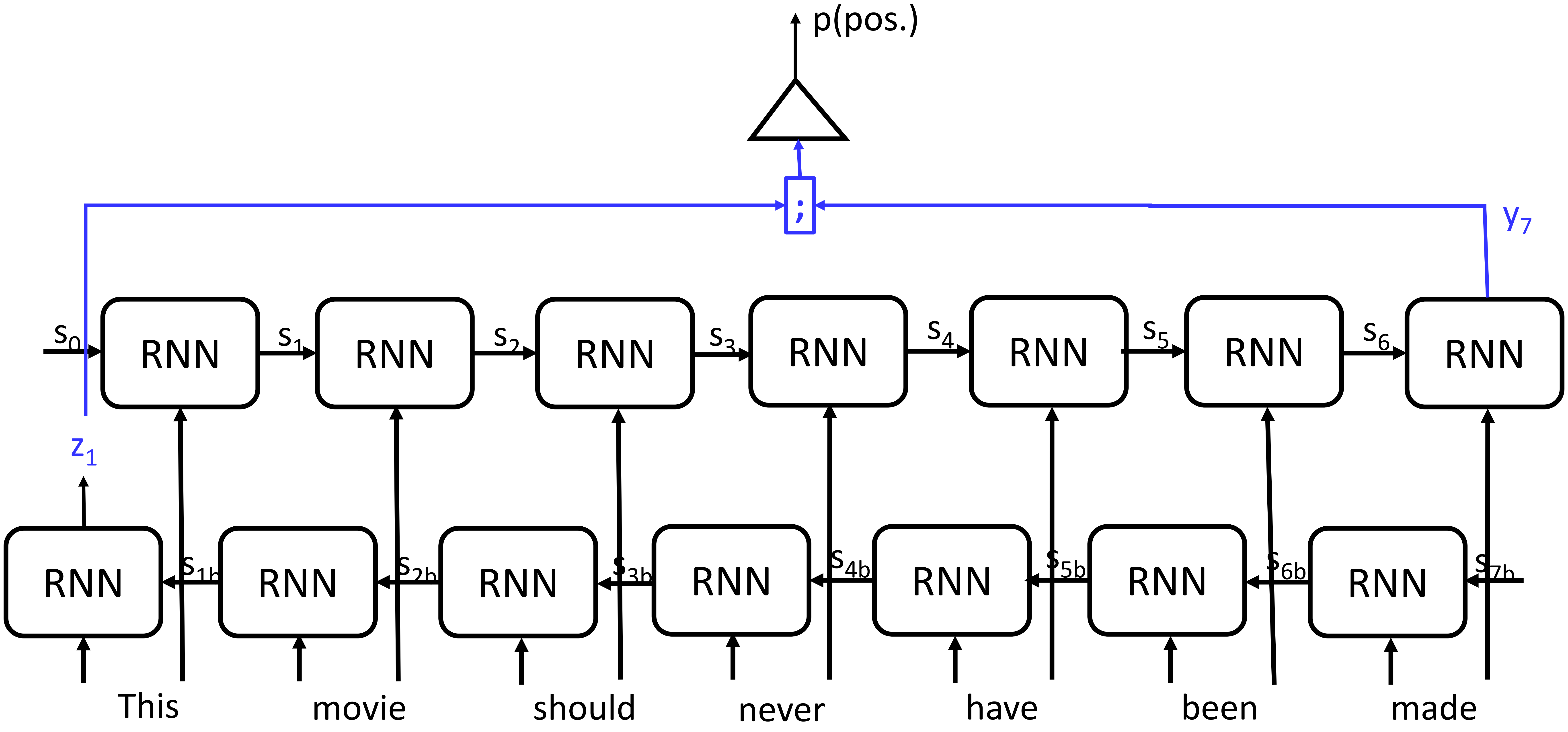


Bidirectional RNN



Infinite window around a word

Bidirectional RNN for Classification



Elman's RNN

- $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$
- $y_t = \tanh(W^y s_t + b^y)$

- Theorem: Any non-linear dynamical system can be approximated to any accuracy by an Elman's RNN, provided that the network has enough hidden units.

- Just because it can approximate it, doesn't mean it knows how to!
  - In practice: Elman's RNN is very hard to train
  - This is because of vanishing/exploding gradients!

$$\frac{\partial L}{\partial W^s} = \sum_{k=1}^{T} \left(\frac{\partial L}{\partial s_T} \frac{\partial s_k}{\partial W^s} \prod_{i=k+1}^{T} \frac{\partial \mathrm{R}(s_{i-1}, x_i)}{\partial d_i} W^s \right)$$

Vanishing Gradients

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

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

$$\frac{\partial L}{\partial W^s} = \sum_{k=1}^{T} \left(\frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_k} \frac{\partial s_k}{\partial W^s} \right)$$

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

$$\frac{\partial L}{\partial W^s} = \sum_{k=1}^{T} \left(\frac{\partial L}{\partial s_T} \frac{\partial s_k}{\partial W^s} \prod_{i=k+1}^{T} \frac{\partial \mathrm{R}(s_{i-1}, x_i)}{\partial d_i} W^s \right)$$

A Memory View of Elman's RNN

- $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$
- $y_t = \tanh(W^y s_t + b^y)$

- Think of RNN as a computer. Input (x_t) arrives. Memory s gets updated

- In Elman RNN entire memory is rewritten at every time step!
  - There is no explicit inertia!

- Memory predicts the output PLUS maintains the history
  - Ideally those two calculations should be separated.

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 new knowledge
- Forget Selectively: in order to make room for new information, we need to selectively forget the least relevant old information

Building Towards LSTM

- Main Idea: control the reading and writing of memory

$$s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$$

write (current) memory   read (previous) memory   write (current) input

We'd like to:

- Selectively read from some memory "cells".
- Selectively write to some memory "cells".
- Selectively write from the “input”.

Vector of Gates

- Read/write selectivity

(element-wise multiplication)
Hadamard product

$$\begin{bmatrix} 5 \\ -2 \\ 12 \\ 4.2 \\ -7 \\ 11 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 0 \\ 4.2 \\ -7 \\ 0 \end{bmatrix}$$

vector of values

gate controls access

$$\boldsymbol{s_{t-1}} \odot \mathbf{g}$$

$$s_{t-1} \in \mathbb{R}^{dstate}$$

$$g \in \{0,1\}^{dstate}$$

Gating to Control Access in an LSTM

- Main Idea: control the reading and writing of memory

$$s_t = s_{t-1} \odot f + x_t \odot i$$

$f \in \{0,1\}^{dstate}$

$i \in \{0,1\}^{dstate}$

forget gate

what to forget/remember?

input gate

what to write from the input?

Problem with 0-1 Gates

- They are fixed
- They don't depend on inputs or outputs
- We need to make them differentiable!

- Solution: make the gates "soft" and "input dependent"
- Instead of $f \in \{0,1\}^{dstate}$, use $f \in [0,1]^{dstate}$
- Moreover, compute $f = \sigma(Ws_{t-1} + W'x_t + b)$

sigmoid →
number between 0 and 1

dependent on state & input

Differentiable Gating to Control Access in an LSTM

- Main Idea: control the reading and writing of memory

$$s_t = s_{t-1} \odot f_t + x_t \odot i_t$$

$$f_t \in [0,1]^{dstate}$$
$$i_t \in [0,1]^{dstate}$$

time-dependent soft forget gate

time-dependent soft input gate

$$f_t = \sigma\left(W^{sf} s_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} s_{t-1} + W^{xi} x_t + b^i\right)$$

Differentiable Gating to Control Access in an LSTM

- Not a good idea adding input to state

$$\cancel{s_t = s_{t-1} \odot f_t + x_t \odot i_t}$$

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

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

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

$$\tilde{s}_t = \phi(s_{t-1}, x_t)$$

proposal for new state

From Elman RNN to Prototype LSTM

- RNN: $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$

  $y_t = \tanh(W^y s_t + b^y)$

- Prototype LSTM:

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

  $\tilde{s}_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$

  $f_t = \sigma(W^{sf} s_{t-1} + W^{xf} x_t + b^f)$

  $i_t = \sigma(W^{si} s_{t-1} + W^{xi} x_t + b^i)$

Problem: same s_t will be used for output and maintaining state

Prototype LSTM → LSTM by Splitting the State

- Prototype LSTM:

$$\tilde{s}_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$$
$$s_t = s_{t-1} \odot f_t + \tilde{s}_t \odot i_t$$

$$f_t = \sigma\left(W^{sf} s_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} s_{t-1} + W^{xi} x_t + b^i\right)$$

$$s_t \to c_t\text{: internal/cell state}$$
$$s_t \to h_t\text{: output state/state}$$

- LSTM:

$$\tilde{c}_t = \tanh(W^s h_{t-1} + W^x x_t + b^s)$$
$$c_t = c_{t-1} \odot f_t + \tilde{c}_t \odot i_t$$
$$h_t = \tanh(c_t) \odot o_t$$

$$f_t = \sigma\left(W^{sf} h_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} h_{t-1} + W^{xi} x_t + b^i\right)$$
$$o_t = \sigma(W^{so} h_{t-1} + W^{xo} x_t + b^o)$$

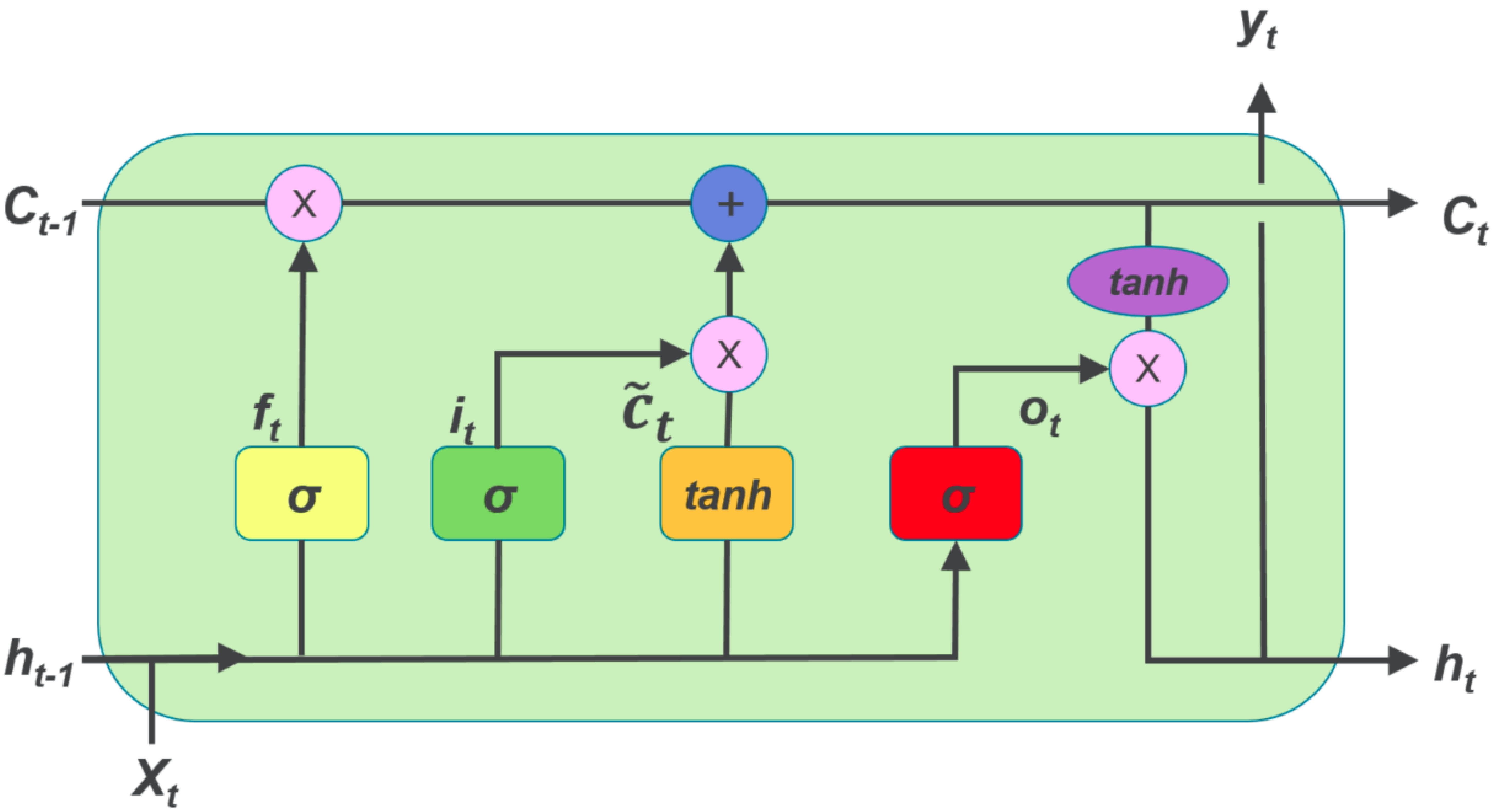
Asssumption: information irrelevant for previous output is irrelevant for gate computation

LSTM

$$\tilde{c}_t = \tanh(W^s h_{t-1} + W^x x_t + b^s)$$
$$c_t = c_{t-1} \odot f_t + \tilde{c}_t \odot i_t$$
$$h_t = \tanh(c_t) \odot o_t$$

$$f_t = \sigma\left(W^{sf} h_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} h_{t-1} + W^{xi} x_t + b^i\right)$$
$$o_t = \sigma(W^{so} h_{t-1} + W^{xo} x_t + b^o)$$

LSTM



Less Problem of Vanishing Gradient

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

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

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial f_t}{\partial c_{t-1}} c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} f_t + \frac{\partial i_t}{\partial c_{t-1}} \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} i_t$$

Initialize such that $f_t \rightarrow 1$
=> b^f = 1 or more

GRU (Gated Recurrent Unit)

- Impose a hard bound on the state & coordinate writes and forgets by explicitly linking them

- instead of selective writes and selective forgets, we do selective overwrites
  - by setting our forget gate equal to 1 minus our write gate

GRU (Gated Recurrent Unit)

- The GRU formulation:

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =$$

Proposal state: $\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j}\mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$

GRU (Gated Recurrent Unit)

- The GRU formulation:

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =$$

gate controlling effect of prev on proposal:

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU (Gated Recurrent Unit)

blend of old state and proposal state

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (1 - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}}$$

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU (Gated Recurrent Unit)

$$\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) = (1 - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}}$$

gate for controlling the blend

$$\mathbf{z} = \sigma(\mathbf{x_j W^{xz}} + \mathbf{s_{j-1} W^{sz}})$$

$$\mathbf{r} = \sigma(\mathbf{x_j W^{xr}} + \mathbf{s_{j-1} W^{sr}})$$

$$\tilde{\mathbf{s_j}} = \tanh(\mathbf{x_j W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}})\mathbf{W^{sg}})$$

GRU (Gated Recurrent Unit)

- The GRU formulation.

$$
\begin{aligned}
\mathbf{s_j} = R_{\mathrm{GRU}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \tilde{\mathbf{s_j}} \\
\mathbf{z} =& \sigma(\mathbf{x_j} \mathbf{W^{xz}} + \mathbf{s_{j-1}} \mathbf{W^{sz}}) \\
\mathbf{r} =& \sigma(\mathbf{x_j} \mathbf{W^{xr}} + \mathbf{s_{j-1}} \mathbf{W^{sr}}) \\
\tilde{\mathbf{s_j}} =& \tanh(\mathbf{x_j} \mathbf{W^{xs}} + (\mathbf{r} \odot \mathbf{s_{j-1}}) \mathbf{W^{sg}})
\end{aligned}
$$

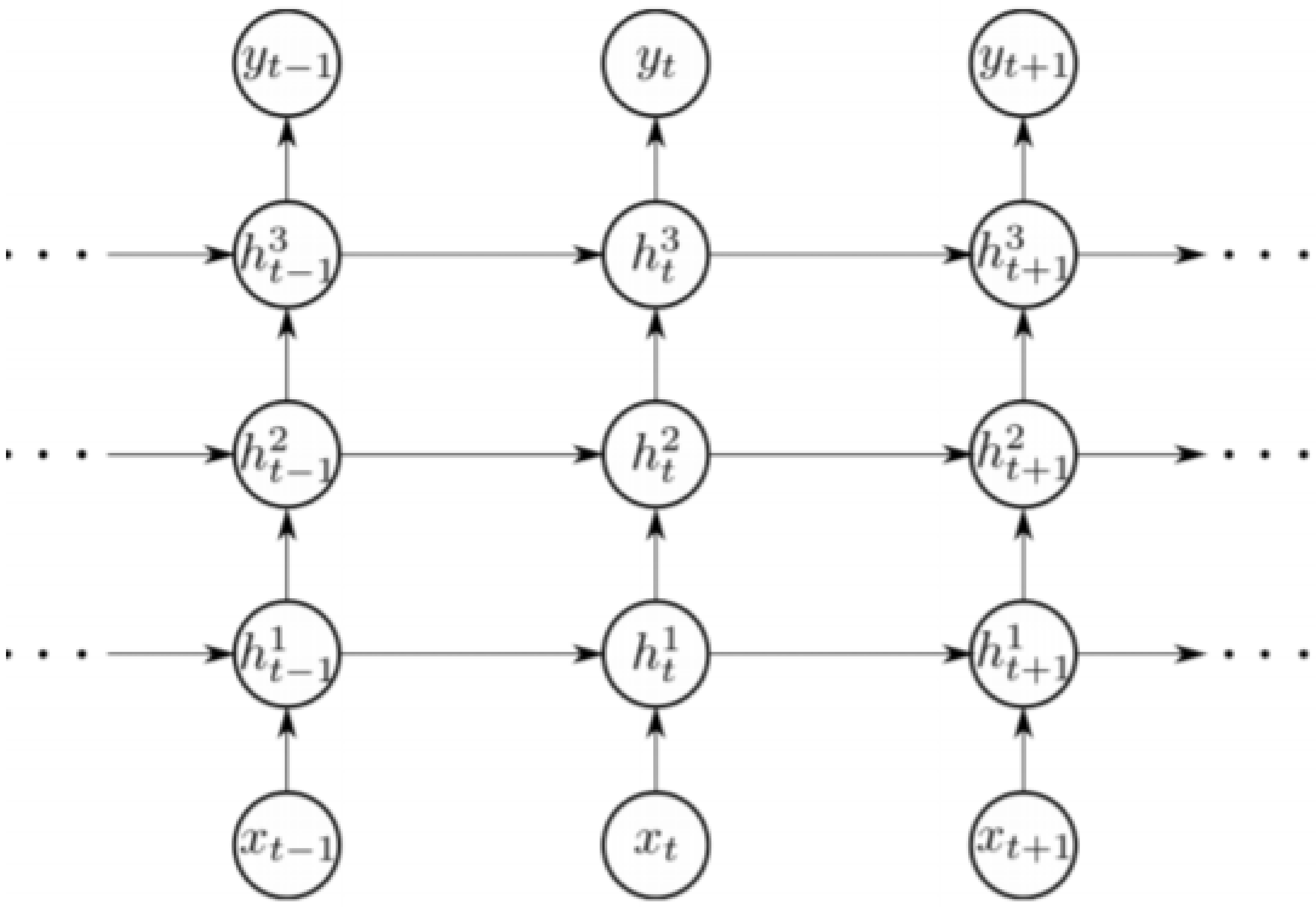
Other Variants

- Many other variants exist.

- Mostly perform similarly to each other.
  - Different tasks may work better with different variants.

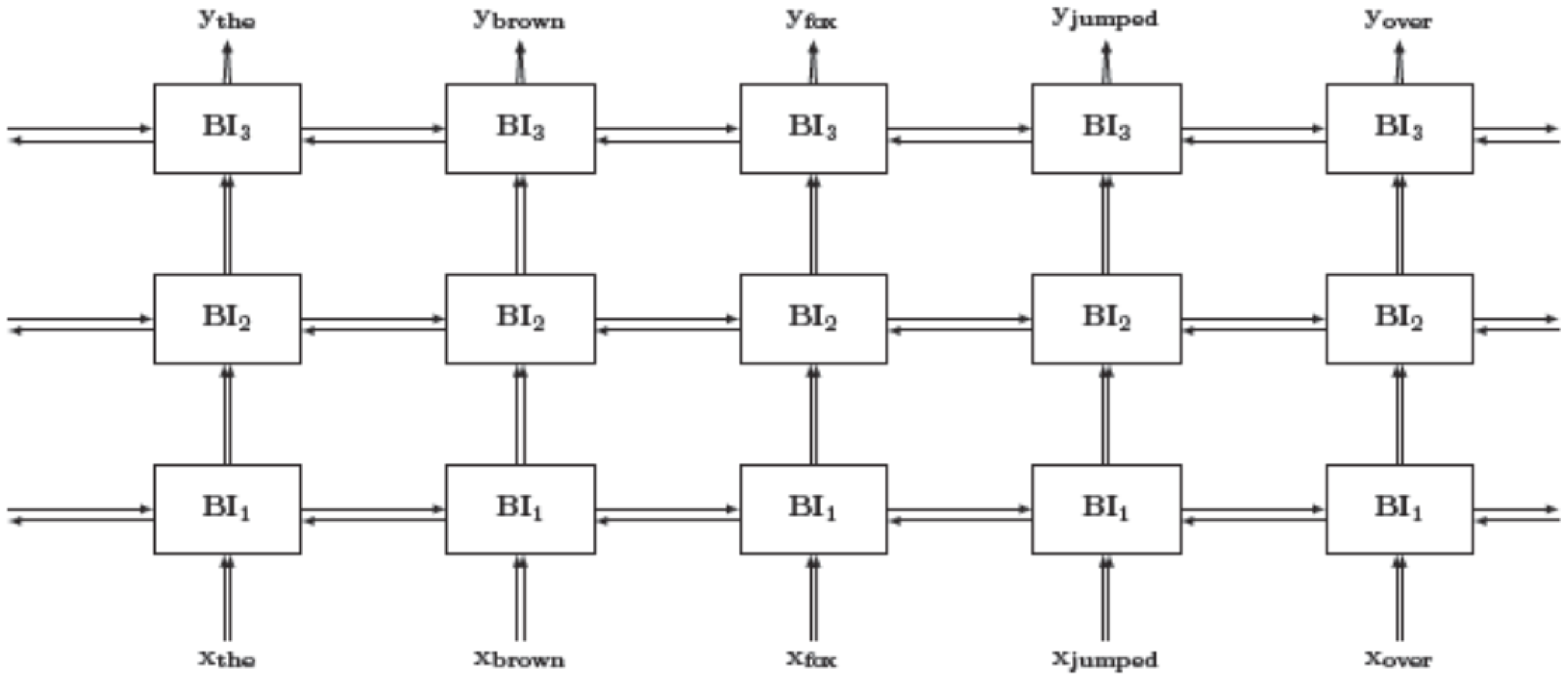
- **The important idea is the differentiable gates.**

Deep LSTMs



(a) Conventional stacked RNN

Deep Bi-LSTMs



Pooling in RNNs (2020)

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

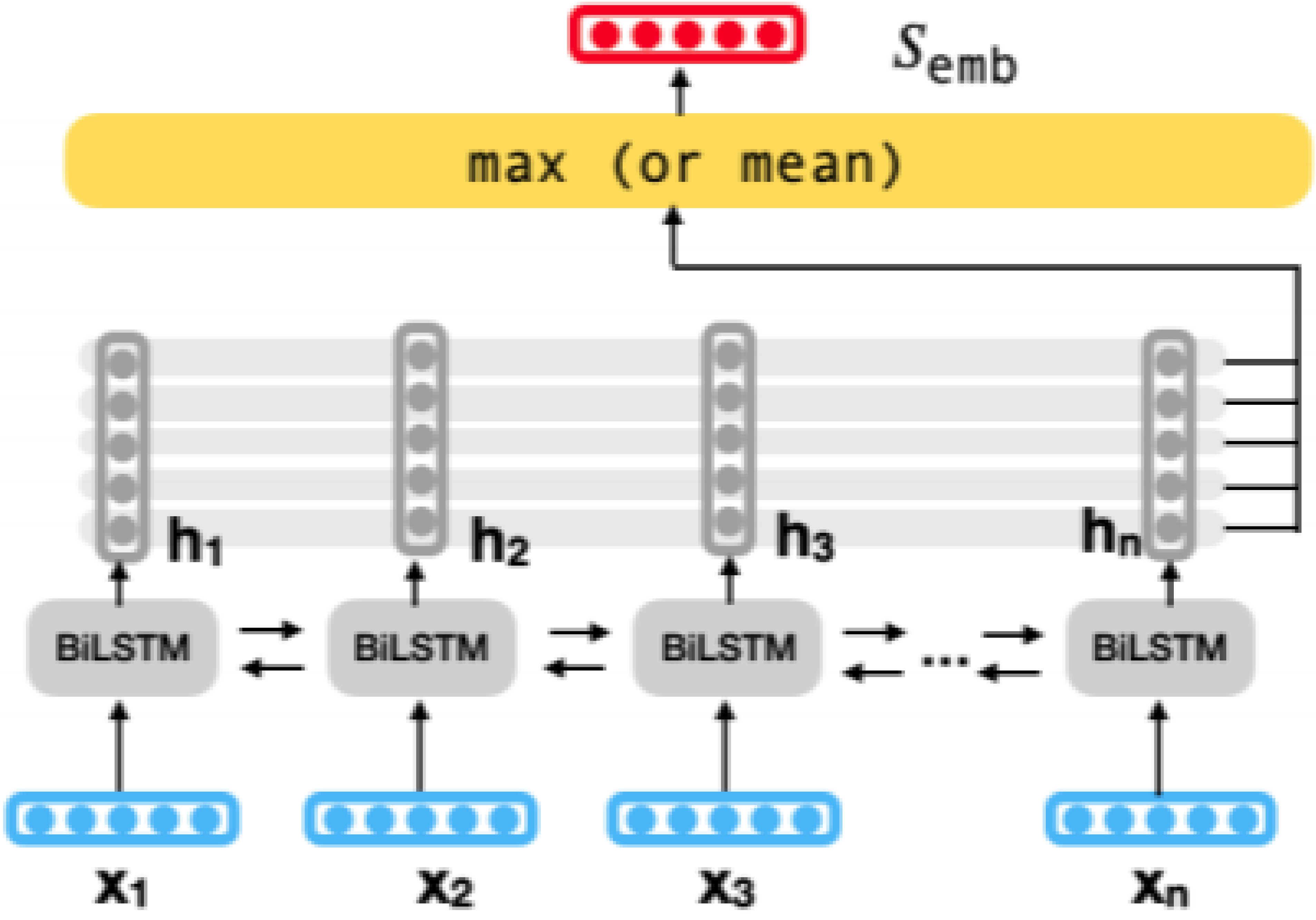
Pratyush Maini[†], **Keshav Kolluru**[†], **Danish Pruthi**[‡], **Mausam**[†]
[†]Indian Institute of Technology, Delhi, India
[‡]Carnegie Mellon University, Pittsburgh, USA
{pratyush.maini, keshav.kolluru}@gmail.com,
ddanish@cs.cmu.edu, mausam@cse.iitd.ac.in

Sentence Representation: Pooling in RNNs

*You can't cram the meaning of the whole *%#@ing sentence in a single *%#@ing vector.*

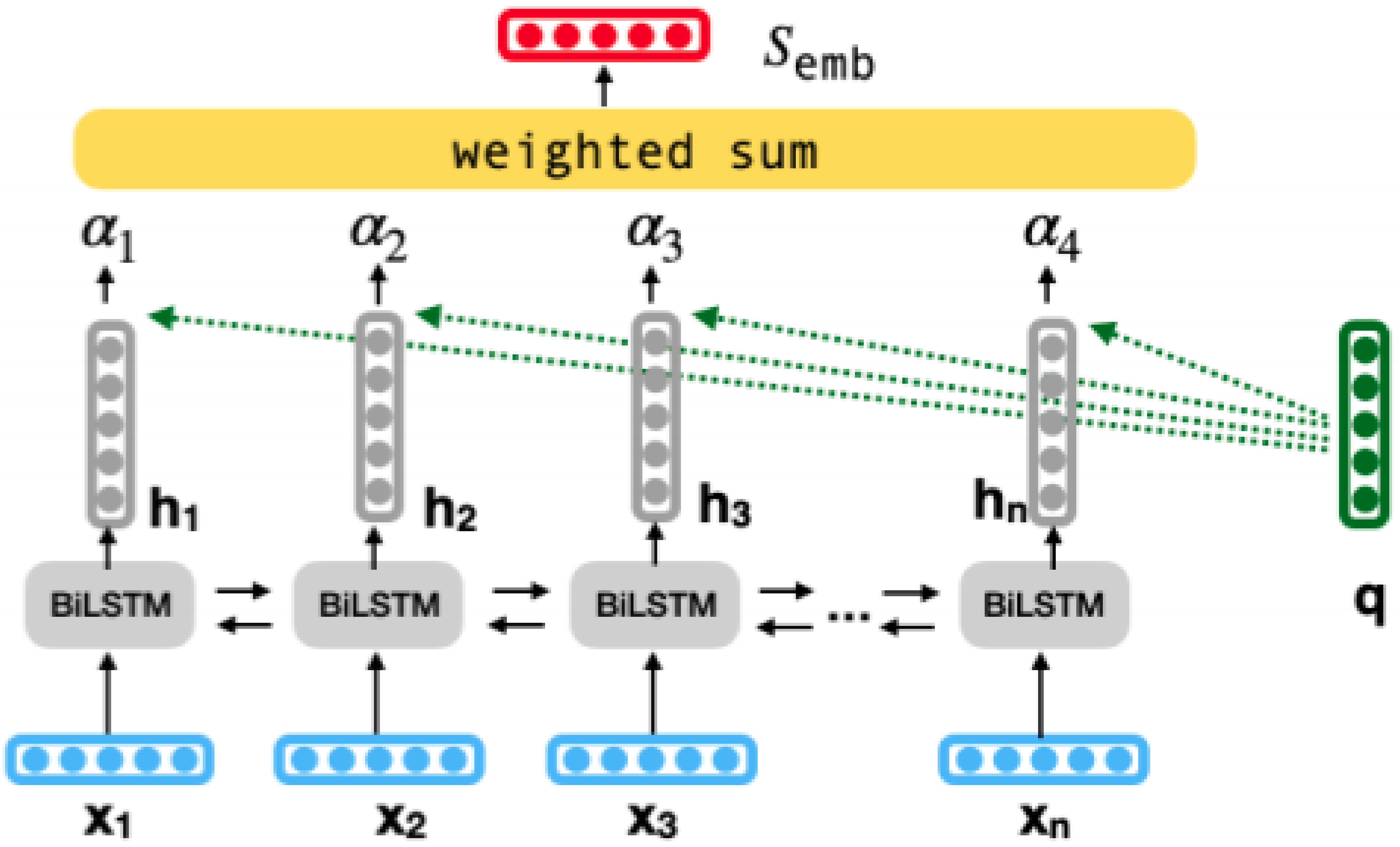
- Encoding a single vector is too restrictive.
  Instead of producing a single vector for the sentence, produce one vector for each word.
- But, eventually need 1 vector.
  Multiple vectors → Single vector
  → Pooling

Pooling



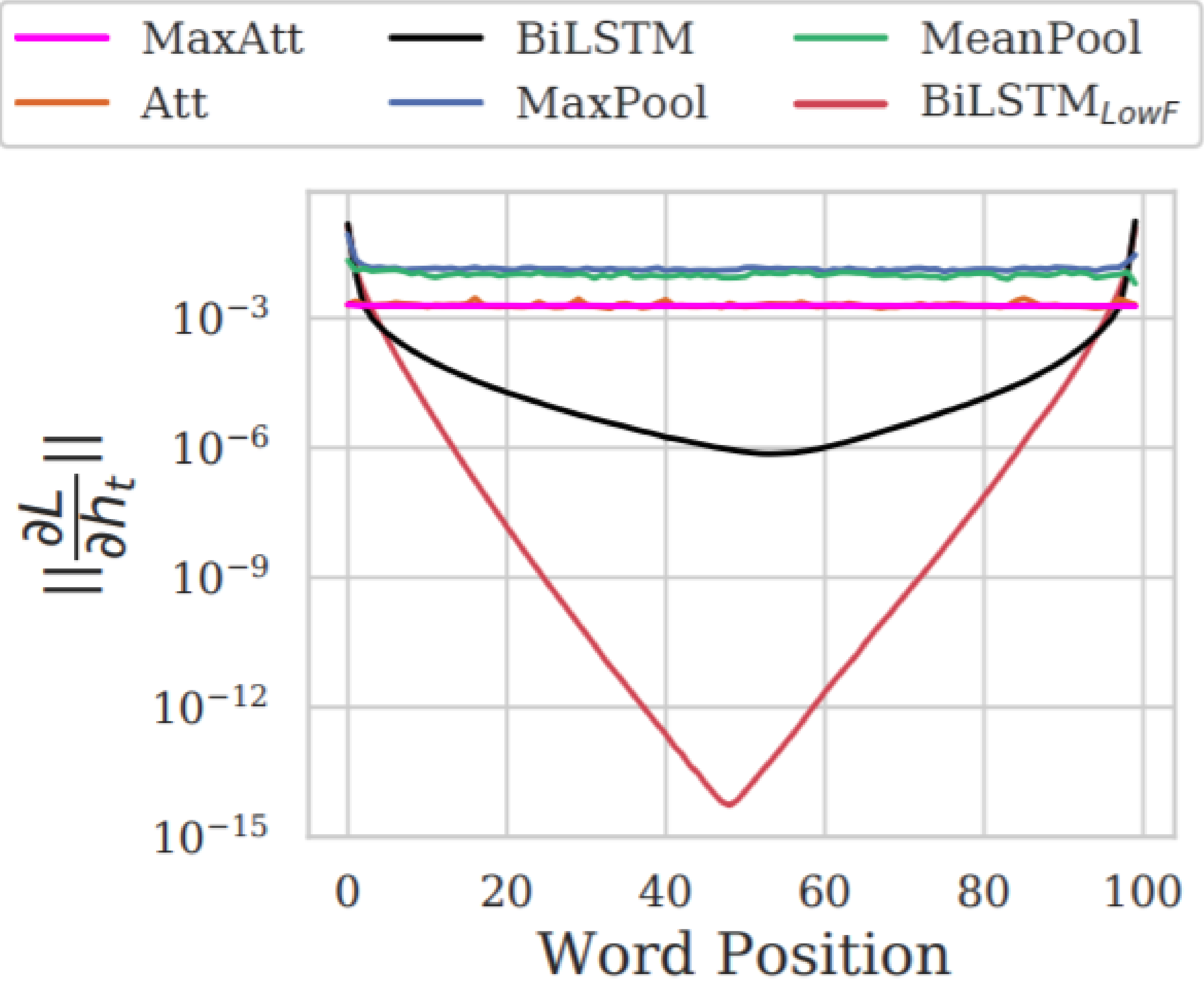
s_{emb}
max (or mean)
h_1
h_2
h_3
h_n
BiLSTM
BiLSTM
BiLSTM
...
BiLSTM
x_1
x_2
x_3
x_n

Attention



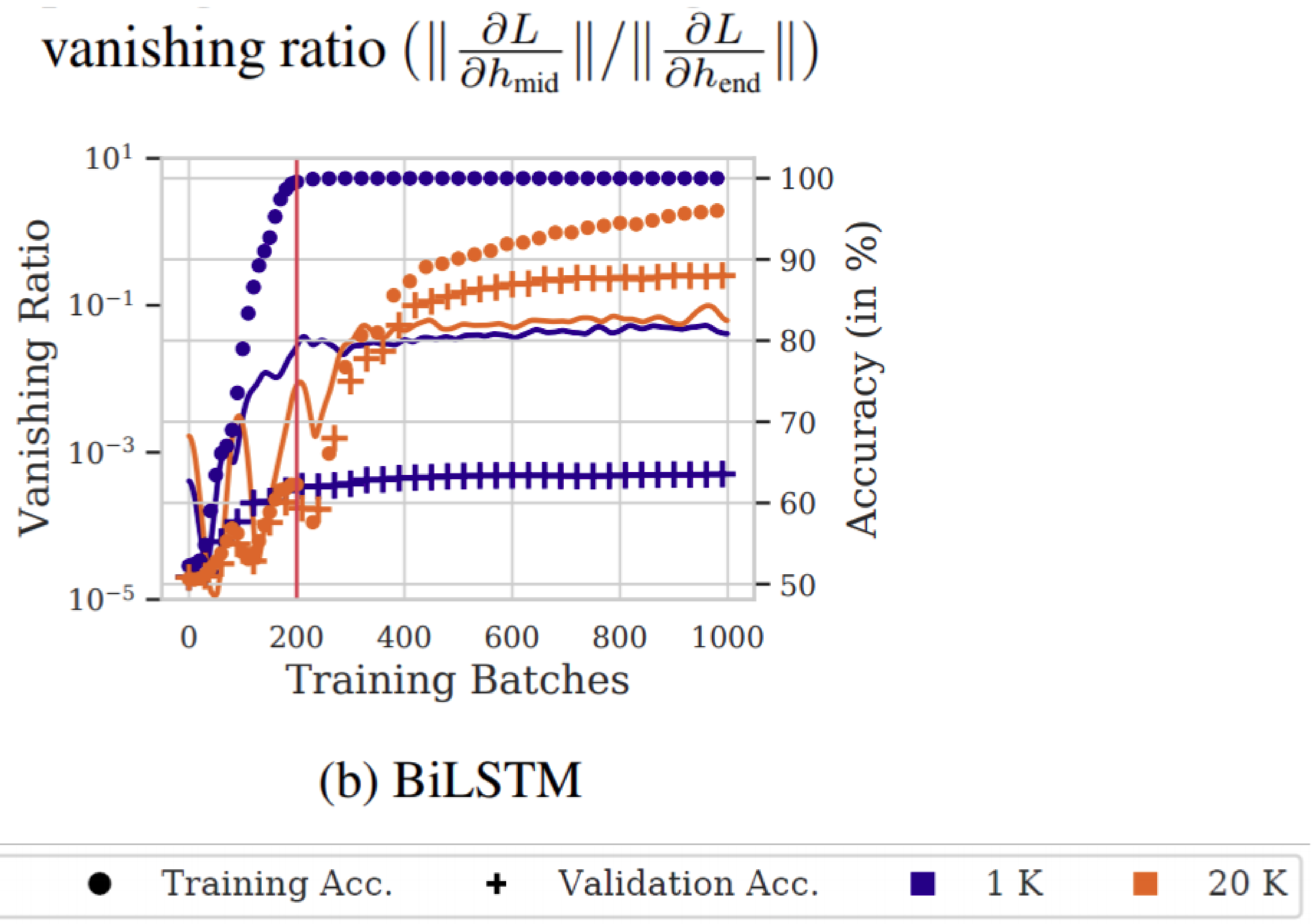
s_{emb}
weighted sum
α_1
α_2
α_3
α_4
$\mathbf{h_1}$
$\mathbf{h_2}$
$\mathbf{h_3}$
$\mathbf{h_n}$
BiLSTM
BiLSTM
BiLSTM
BiLSTM
...
$\mathbf{q}$
$\mathbf{x_1}$
$\mathbf{x_2}$
$\mathbf{x_3}$
$\mathbf{x_n}$

Vanishing Gradients @~Start of Training



(a) Gradient Norms

Vanishing Ratio



(b) BiLSTM

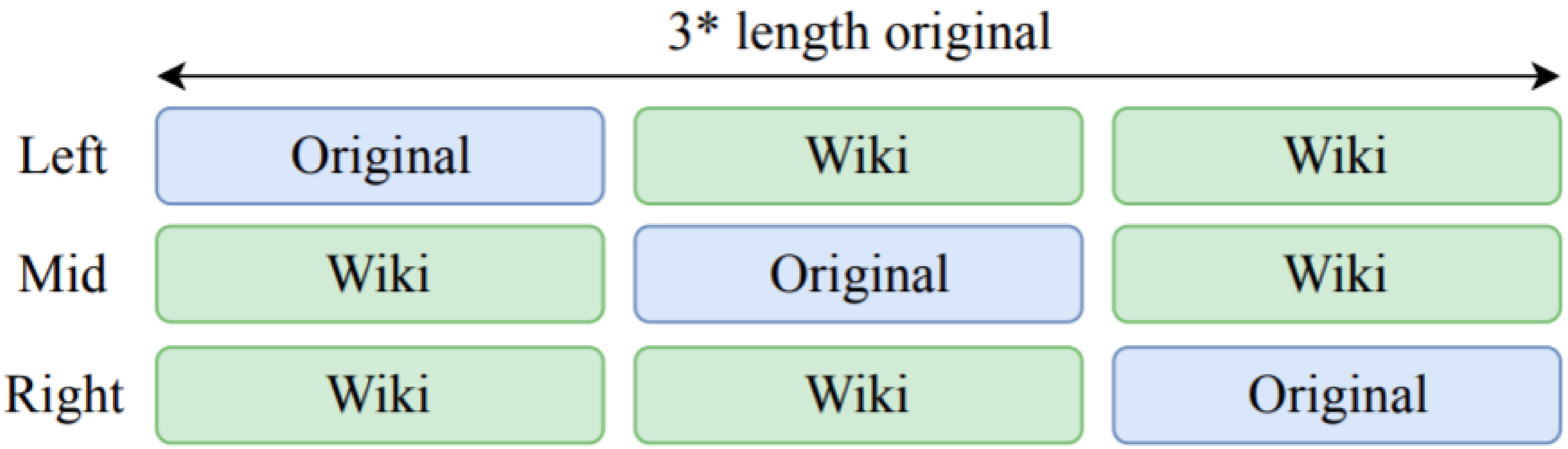
Size-Accuracy-Vanishing

	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 ± 2.3	75.0 ± 0.4	86.6 ± 0.8	49.6 ± 0.7	49.9 ± 0.5	50.3 ± 0.3	53.5 ± 2.5	64.7 ± 2.8	85.9 ± 0.5
MeanPool	73.0 ± 3.0	81.7 ± 0.7	87.1 ± 0.6	69.8 ± 2.1	76.2 ± 1.0	84.1 ± 0.7	70.0 ± 1.1	76.8 ± 1.0	84.8 ± 0.9
MaxPool	69.0 ± 3.9	80.1 ± 0.5	87.8 ± 0.6	64.5 ± 1.8	77.2 ± 2.0	86.0 ± 0.8	65.9 ± 4.6	77.8 ± 0.9	**87.2** ± 0.6
Att	75.7 ± 2.6	**82.8** ± 0.8	**89.0** ± 0.3	75.0 ± 0.8	79.4 ± 0.8	86.7 ± 1.4	74.7 ± 1.4	80.2 ± 1.8	87.1 ± 1.0
MaxAtt	**75.9** ± 2.2	82.5 ± 0.4	88.5 ± 0.5	**75.4** ± 2.4	**80.9** ± 1.8	**86.8** ± 0.5	**77.9** ± 0.9	**81.9** ± 0.5	**87.2** ± 0.5

	Yahoo			Yahoo (mid) + Wiki			Yahoo (right) + Wiki		
	1K	2K	10K	1K	2K	10K	1K	2K	10K
BiLSTM	38.3 ± 4.8	51.4 ± 2.1	63.5 ± 0.6	12.7 ± 1.1	12.7 ± 1.1	11.4 ± 0.8	18.8 ± 2.5	37.3 ± 0.9	60.1 ± 1.5
MeanPool	48.2 ± 2.3	56.6 ± 0.5	64.7 ± 0.6	31.9 ± 2.3	43.1 ± 2.0	58.5 ± 0.6	33.9 ± 2.1	43.2 ± 1.0	58.6 ± 0.4
MaxPool	50.2 ± 2.1	56.3 ± 1.8	63.9 ± 1.1	33.0 ± 1.0	40.1 ± 1.4	58.4 ± 1.2	33.1 ± 2.5	41.2 ± 0.9	60.9 ± 1.0
Att	47.3 ± 2.2	54.2 ± 1.1	**65.1** ± 1.5	39.4 ± 0.5	45.1 ± 1.8	61.5 ± 1.7	37.9 ± 1.4	47.6 ± 2.3	62.2 ± 0.9
MaxAtt	**51.8** ± 1.1	**57.0** ± 1.1	**65.1** ± 1.1	**39.6** ± 0.9	**48.5** ± 0.6	**62.2** ± 1.6	**40.3** ± 1.5	**50.1** ± 1.6	**63.1** ± 0.7

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