Natural Language Processing

Info 159/259

Lecture 12: Neural sequence labeling (Oct 2, 2018)

David Bamman, UC Berkeley

MEMM Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

For all training data, we want probability of the true label y_i conditioned on the previous true label y_{i-1} to be high.

This is simply multiclass logistic regression

MEMM Training

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

Locally normalized — at each time step, each conditional distribution sums to 1

Label bias

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

- For a given conditioning context, the probability of a tag (e.g., VBZ) only competes against other tags with that same context (e.g., NN)

Bottou 2001; Lafferty et al. 2001

Label bias

will to fight

	TO
x_i=to	10000000
y_{i-1}=NN	0
y_{i-1}=MD	0

to is relatively deterministic (almost always TO) so it doesn't matter what tag precedes it.

Toutanova et al. 2003

Label bias

will to fight

$$\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

Because of this local normalization, P(TO | context) will always be 1 if x="to"

Toutanova et al. 2003

Conditional random fields

- We can solve this problem using global normalization (over the entire sequences) rather than locally normalized factors.

MEMM

$$P(y \mid x, \beta) = \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)$$

CRF

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

Conditional random fields

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

Feature vector scoped over the entire input and label sequence

$$\Phi(x, y) = \sum_{i=1}^{n} \phi(x, i, y_i, y_{i-1})$$

ϕ is the same feature vector we used for local predictions using MEMMs

Features

$$\phi(x, i, y_i, y_{i-1})$$

Features are scoped over the previous predicted tag and the entire observed input

feature	example
x_i = man	1
y_{i-1} = JJ	1
i=n (last word of sentence)	1
x_i ends in -ly	0

In an MEMM, we estimate $P(y_t \mid y_{t-1}, x, \beta)$ from each $\phi(x, t, y_t, y_{t-1})$ independently

	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$
x_i=will ^ y_i = NN	1	0	0
y_{i-1}=START ^ y_i = NN	1	0	0
x_i=will ^ y_i = MD	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0
…			
x_i=to ^ y_i = TO	0	1	0
y_{i-1}=NN ^ y_i = TO	0	1	0
y_{i-1}=MD ^ y_i = TO	0	0	0
…			
x_i=fight ^ y_i = VB	0	0	1
y_{i-1}=TO ^ y_i = VB	0	0	1

In a CRF, we use features from the entire sequence (by summing the individual features at each time step)

	will $\phi(x, 1, y_1, y_0)$	to $\phi(x, 2, y_2, y_1)$	fight $\phi(x, 3, y_3, y_2)$	$\mathbf{\Phi}$(x, NN TO VB)
x_i=will ^ y_i = NN	1	0	0	1
y_{i-1}=START ^ y_i = NN	1	0	0	1
x_i=will ^ y_i = MD	0	0	0	0
y_{i-1}=START ^ y_i = MD	0	0	0	0
…				
x_i=to ^ y_i = TO	0	1	0	1
y_{i-1}=NN ^ y_i = TO	0	1	0	1
y_{i-1}=MD ^ y_i = TO	0	0	0	0
…				
x_i=fight ^ y_i = VB	0	0	1	1
y_{i-1}=TO ^ y_i = VB	0	0	1	1

This lets us isolate the global sequence features that separate good sequences (in our training data) from bad sequences (not in our training data)

	Φ(x, NN TO VB) GOOD	Φ(x, MD TO VB) BAD
x_i=will ^ y_i = NN	1	0
y_{i-1}=START ^ y_i = NN	1	0
x_i=will ^ y_i = MD	0	1
y_{i-1}=START ^ y_i = MD	0	1
…		
y_{i-1}=NN ^ y_i = TO	1	0
y_{i-1}=MD ^ y_i = TO	0	1
x_i=to ^ y_i = TO	1	1
x_i=fight ^ y_i = VB	1	1
y_{i-1}=TO ^ y_i = VB	1	1

these are the different (and so are potentially predictive of a good label sequence)

these are the same (and so are not)

Conditional random fields

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

- In MEMMs, we normalize over the set of 45 POS tags
- CRFs are globally normalized, but the normalization complexity is huge — every possible sequence of labels of length n.

Forward algorithm (CRF)

$$P(y \mid x, \beta) = \frac{\exp(\Phi(x, y)^\top \beta)}{\sum_{y' \in \mathcal{Y}} \exp(\Phi(x, y')^\top \beta)}$$

- Calculating the denominator naively would involve a summation over K^N terms
- But we can do this efficiently in NK^2 time using the forward algorithm

For details, see: Collins, “The Forward-Backward Algorithm”

Forward algorithm (CRF)

END							
DT							
NNP							
VB							
NN							
MD							
START							
	^	Janet	will	back	the	bill	$

$$\alpha(1, y) = \exp\left(\phi(x, i, y, \text{START})^\top \beta\right)$$

$$\alpha(i, y) = \sum_{y' \in \mathcal{S}} \alpha(i-1, y') \times \exp\left(\phi(x, i, y, y')^\top \beta\right)$$

Forward algorithm (CRF)

END							
DT							
NNP							
VB							
NN							
MD							
START							
	^	Janet	will	back	the	bill	$

$$Z = \sum_{y' \in \mathcal{Y}} \exp\left(\Phi(x, y')^\top \beta\right) = \sum_{s \in \mathcal{S}} \alpha(n, s)$$

Conditional random fields

With a CRF, we have exactly the same parameters as we do with an equivalent MEMM; but we learn the best values of those parameters that leads to the best probability of the sequence overall (in our training data)

MEMM

	TO
x_i=to ^ y_i = TO	10000000
y_{i-1}=NN ^ y_i = TO	0
y_{i-1}=MD ^ y_i = TO	0

CRF

	TO
x_i=to ^ y_i = TO	7.8
y_{i-1}=NN ^ y_i = TO	1.4
y_{i-1}=MD ^ y_i = TO	-5.8

Parameter estimation

- Just like logistic regression/MEMM, we can find the optimal values for the parameters using stochastic gradient descent for a given sequence x and true labels y.

$$\frac{\partial L}{\partial \beta_k} = \Phi_k(x, y) - \sum_{y' \in \mathcal{Y}} P(y' \mid x, \beta)\Phi_k(x, y')$$

Features for the true label y

Expected feature counts under the probability for a sequence assigned by current β

Parameter estimation

- Just like logistic regression/MEMM, we can find the optimal values for the parameters using stochastic gradient descent for a given sequence x and true labels y.

$$\frac{\partial L}{\partial \beta_k} = \Phi_k(x, y) - \sum_{y' \in \mathcal{Y}} P(y' \mid x, \beta)\Phi_k(x, y')$$

If current model assigns probability of 1 to true sequence and 0 to all other K^N sequences, then gradient = 0

Parameter estimation

$$\sum_{y' \in \mathcal{Y}} P(y' \mid x, \beta) \Phi_k(x, y')$$

$$= \sum_{i=1}^{n} \sum_{a \in \mathcal{S}, b \in \mathcal{S}} \phi_k(x, i, a, b) \sum_{y' \in \mathcal{Y} : y'_{i-1} = a, y'_i = b} P(y' \mid x, \beta)$$

This entire sum can be found in NK^2 time using the forward-backward algorithm

For details, see: Collins, “The Forward-Backward Algorithm”

Viterbi Decoding

$$v_1(y) = \exp\left(\phi(x, 1, START, y)^\top \beta\right)$$

$$v_t(y) = \max_{u \in \mathcal{S}} [v_{t-1}(u) \times \exp\left(\phi(x, t, y, u)^\top \beta\right)]$$

(equivalently)

$$v_1(y) = \phi(x, 1, START, y)^\top \beta$$

$$v_t(y) = \max_{u \in \mathcal{S}} [v_{t-1}(u) + \phi(x, t, y, u)^\top \beta]$$

In practice

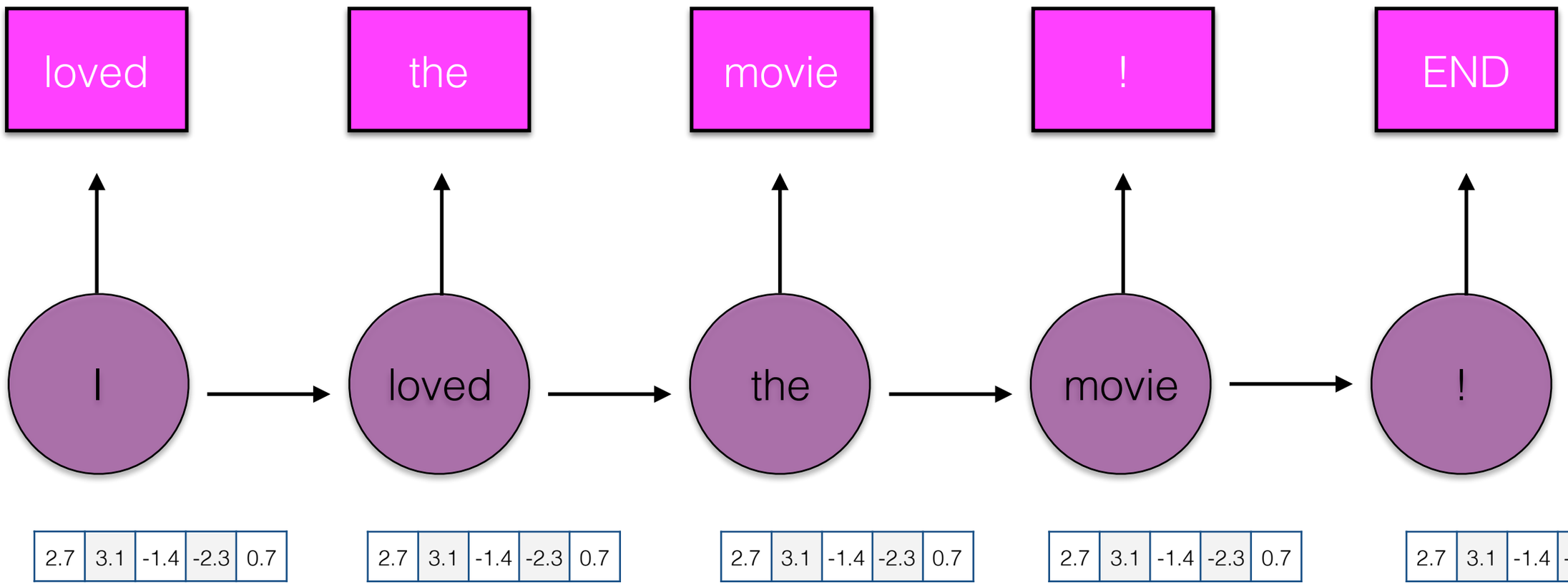
- CRF training is slow! NK^2 complexity for each sequence at each gradient step.

- Accuracy is typically better than MEMMs

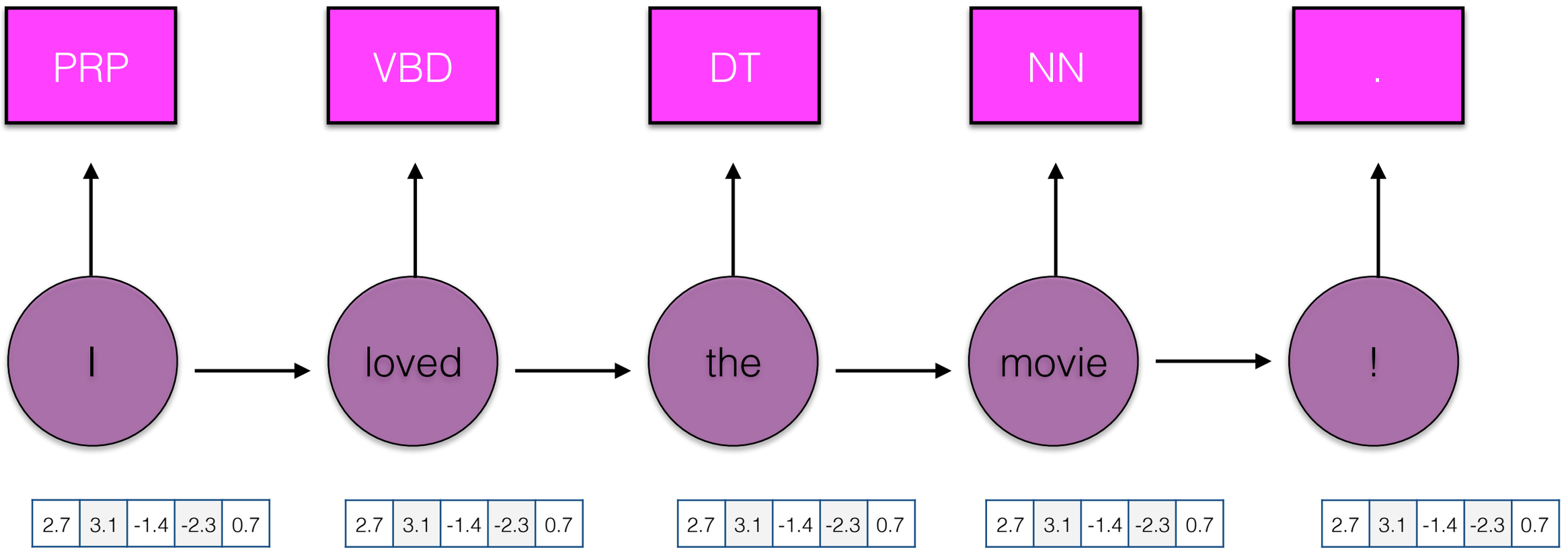
Recurrent neural network

- RNN allow arbitarily-sized conditioning contexts; condition on the entire sequence history.

RNNs for language modeling are already performing a kind of sequence labeling: at each time step, predict the word from $\boldsymbol{\mathcal{V}}$ conditioned on the context



For POS tagging, predict the tag from $\boldsymbol{y}$ conditioned on the context



RNNs for POS

will to fight

- To make a prediction for y_t, RNNs condition on all input seen through time t (x_1, …, x_t)

- But knowing something about the future can help (x_{t+1}, …, x_n)

Bidirectional RNN

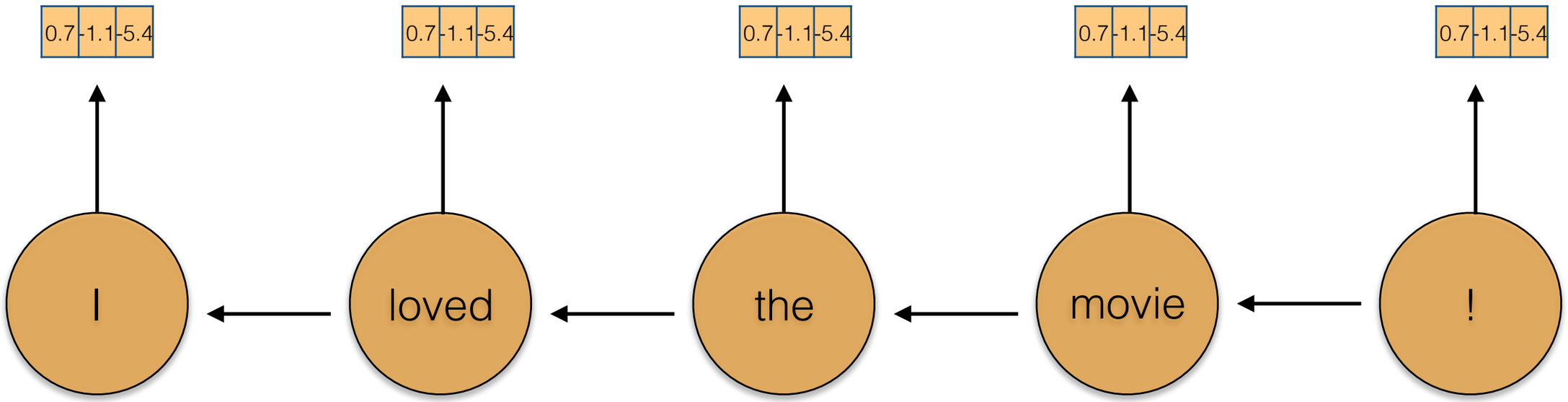
- A powerful alternative is make predictions conditioning both on the past and the future.

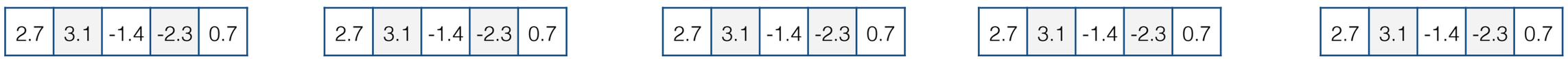
- Two RNNs
    - One running left-to-right
    - One right-to-left

- Each produces an output vector at each time step, which we concatenate

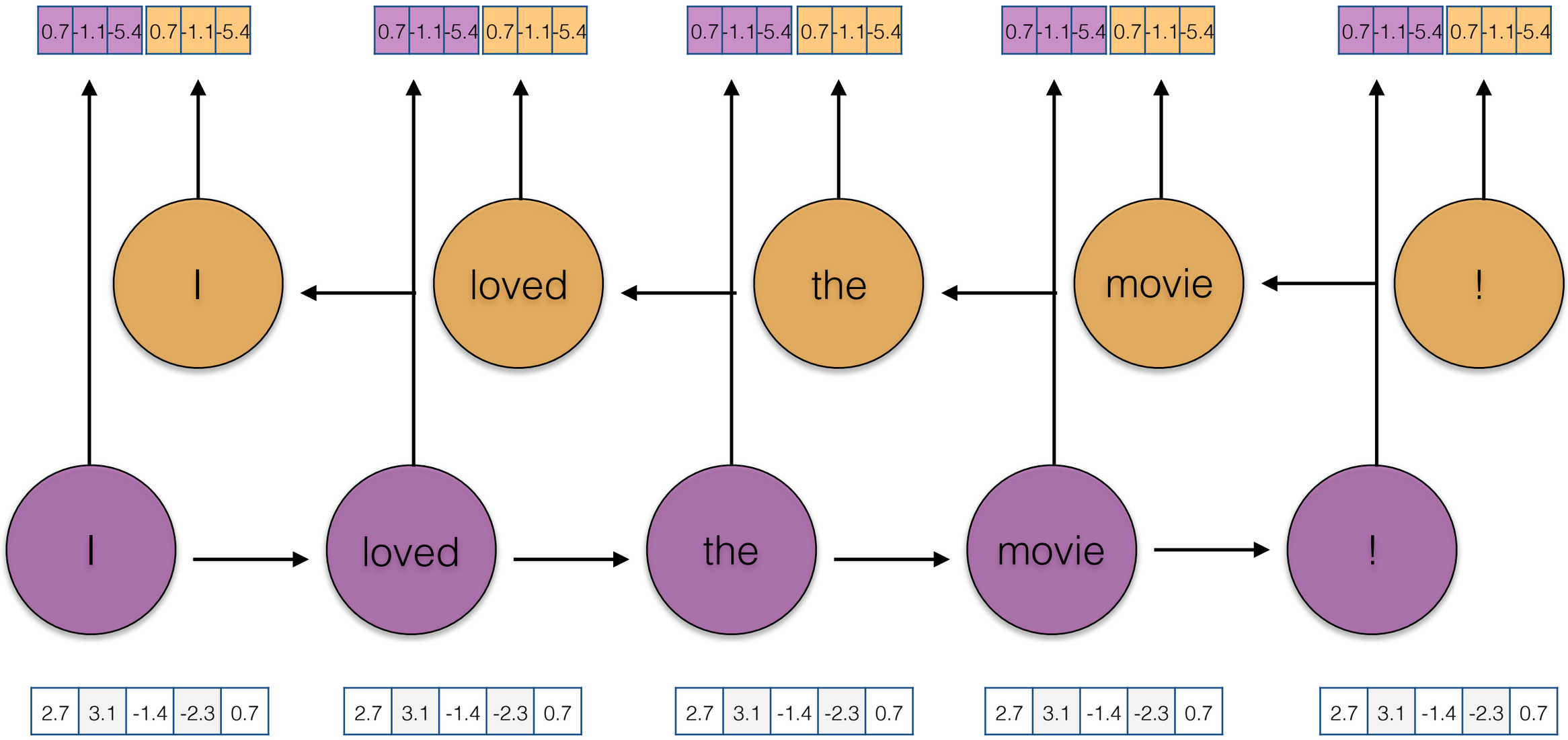
Bidirectional RNN

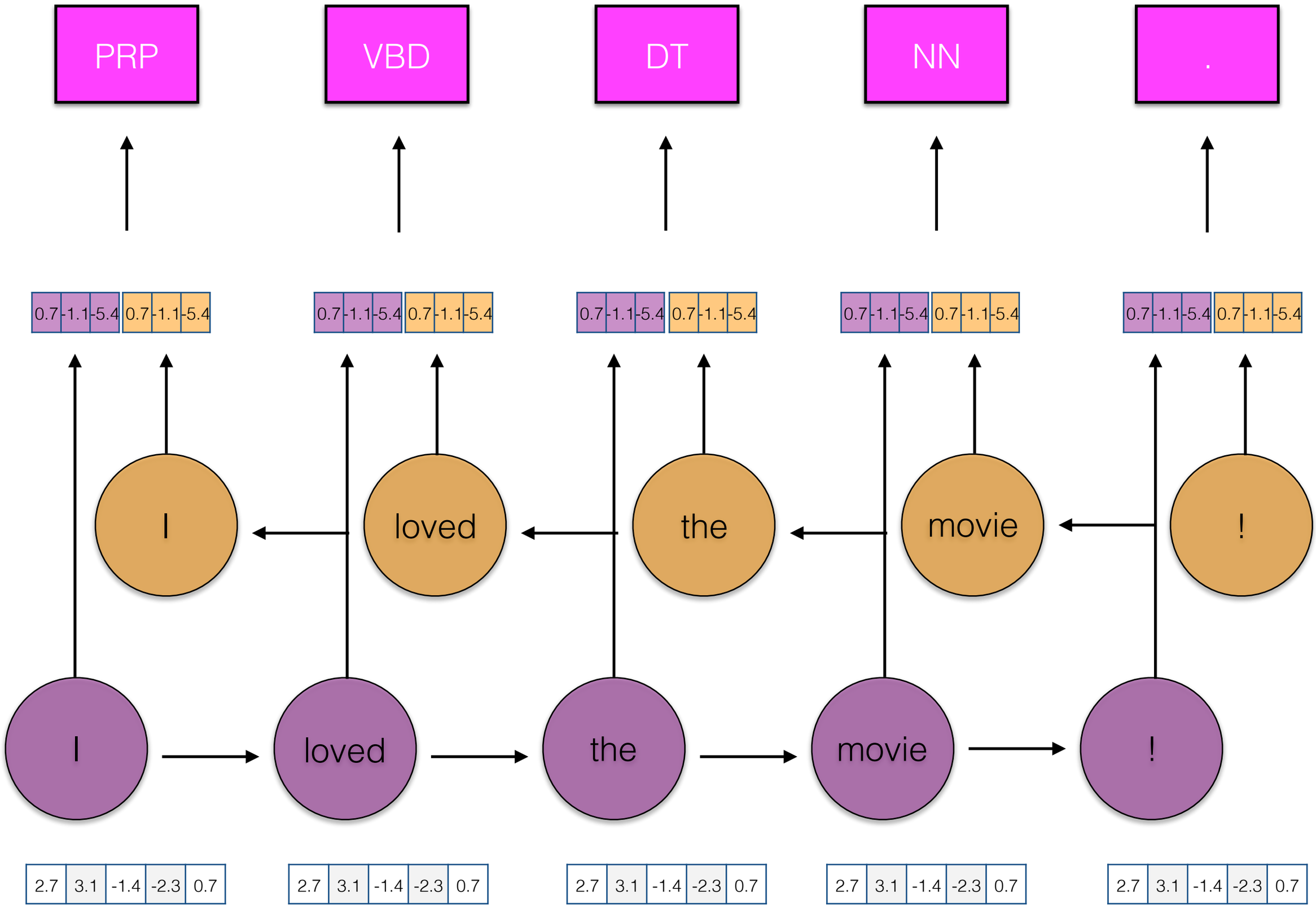
backward RNN





Bidirectional RNN




PRP
VBD
DT
NN
.
0.7 -1.1 -5.4 0.7 -1.1 -5.4
I
loved
the
movie
!
2.7 3.1 -1.4 -2.3 0.7

Training BiRNNs

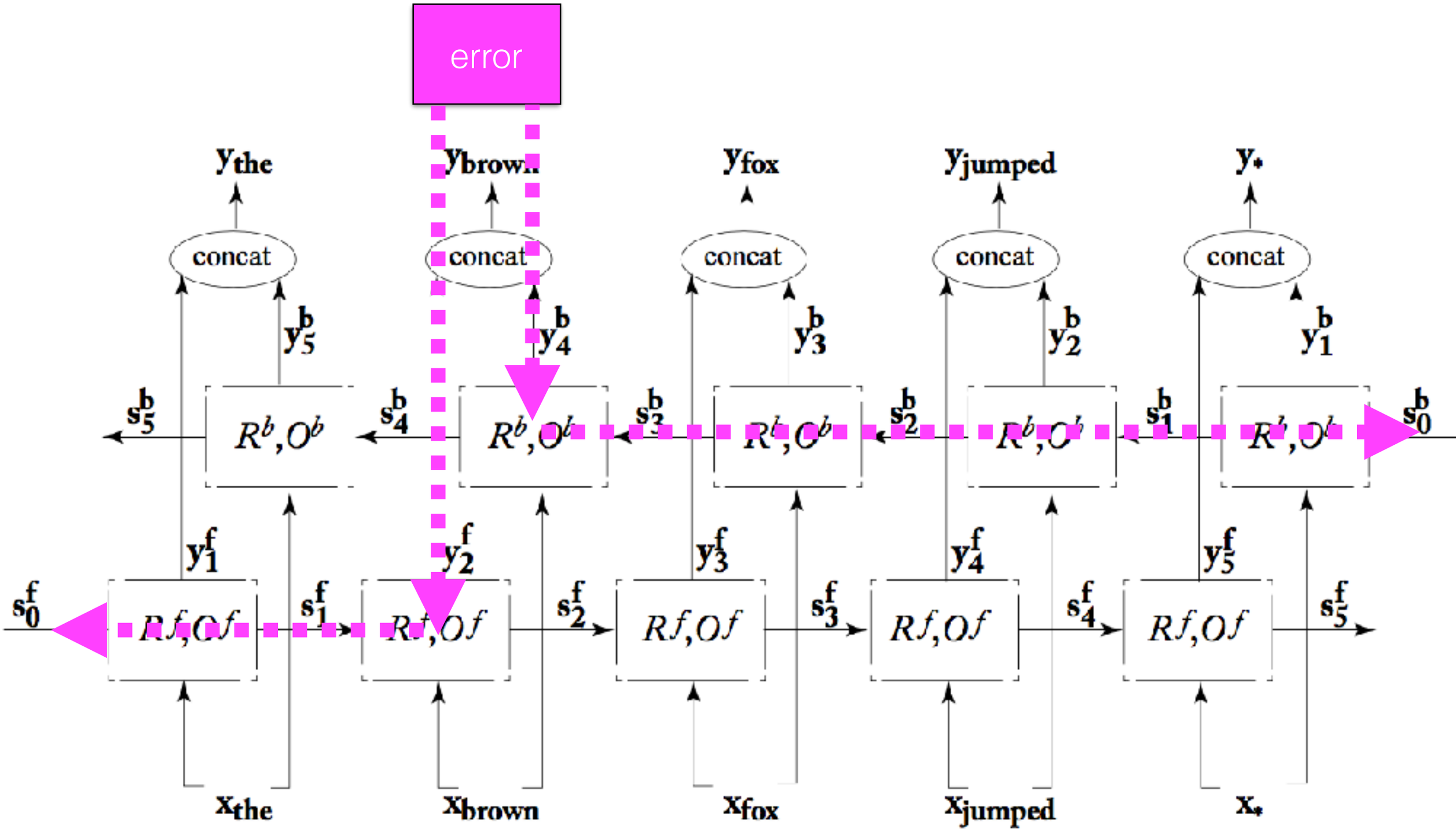
- Given this definition of an BiRNN:

$$s_b^i = R_b(x^i, s_b^{i+1}) = g(s_b^{i+1} W_b^s + x^i W_b^x + b_b)$$

$$s_f^i = R_f(x^i, s_f^{i-1}) = g(s_f^{i-1} W_f^s + x^i W_f^x + b_f)$$

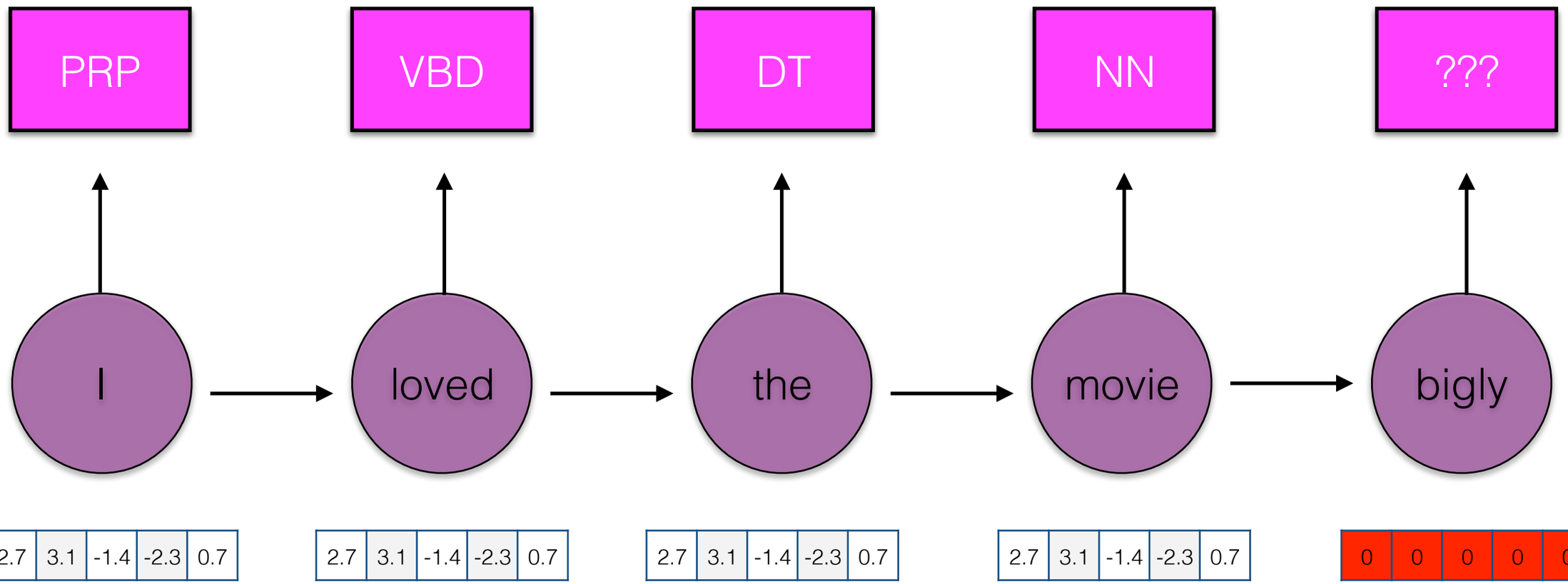
$$y_i = \text{softmax}\left([s_f^i; s_b^i] W^o + b^o\right)$$

- We have 8 sets of parameters to learn (3 for each RNN + 2 for the final layer)



Goldberg 2017

How do we fix this?



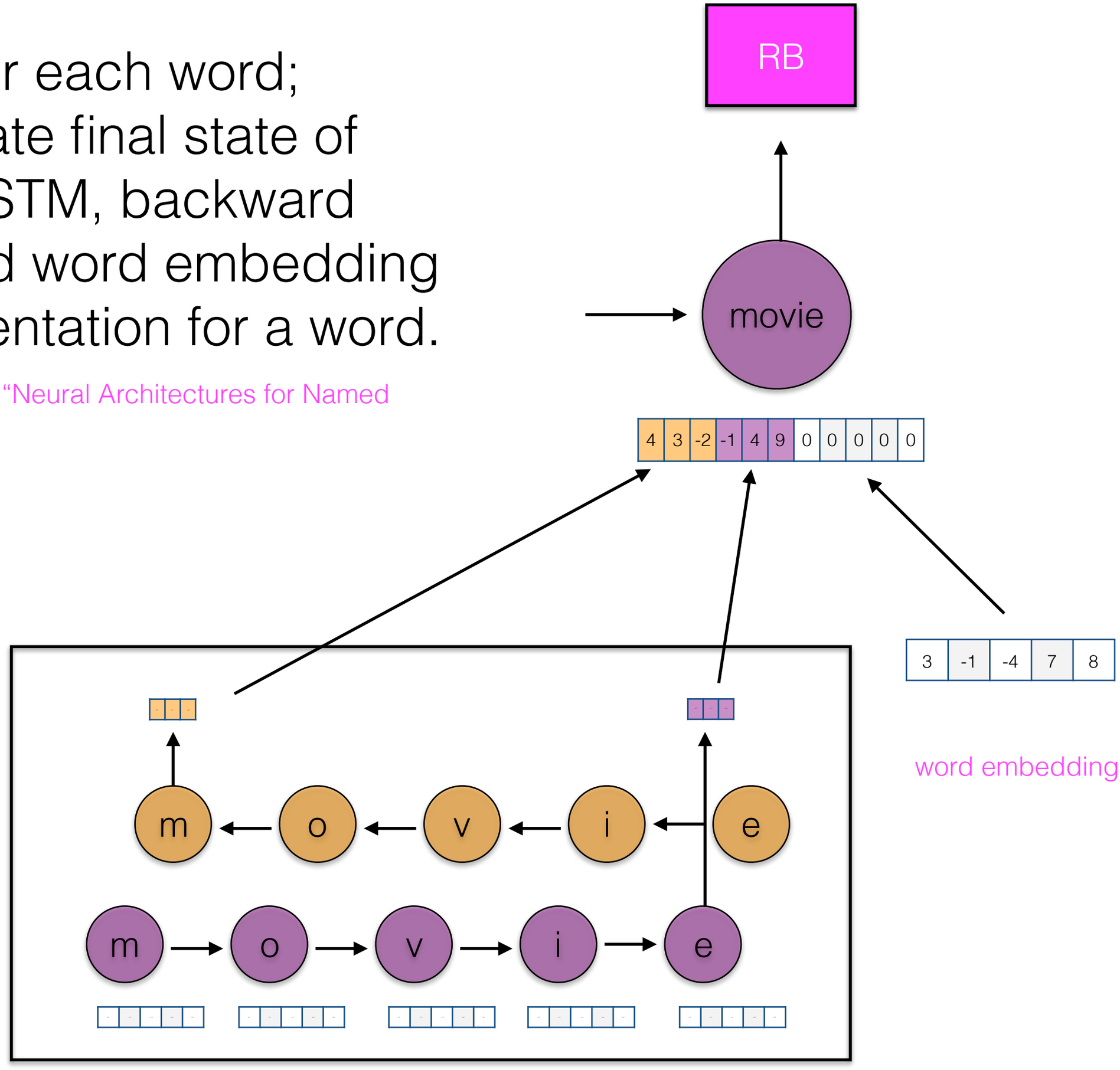
Subword information

- We saw subword information used for creating embeddings (FastText)

- Another alternative is to use standard word embeddings and reason about subword information within a model.

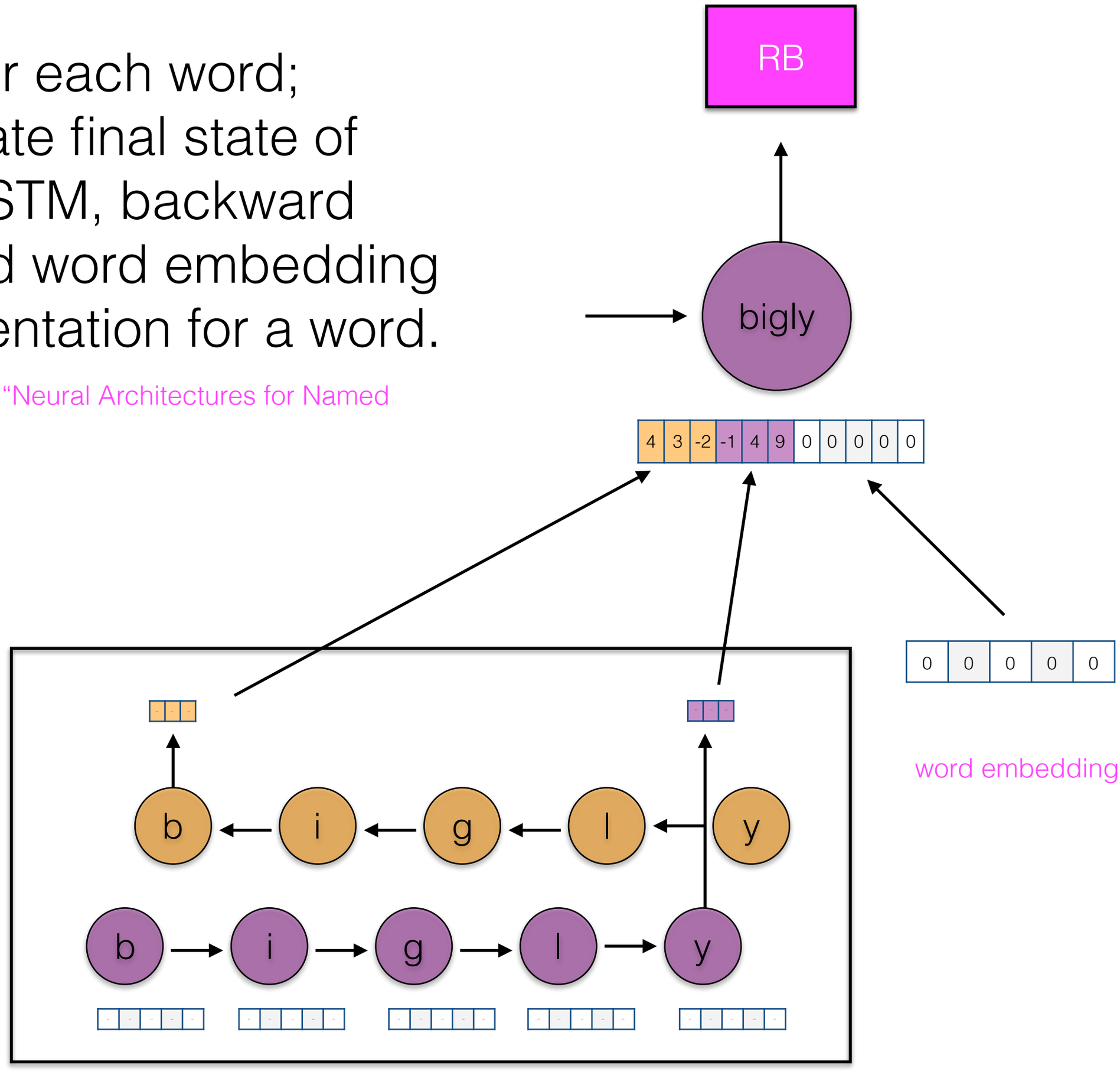
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

Lample et al. (2016), “Neural Architectures for Named Entity Recognition”



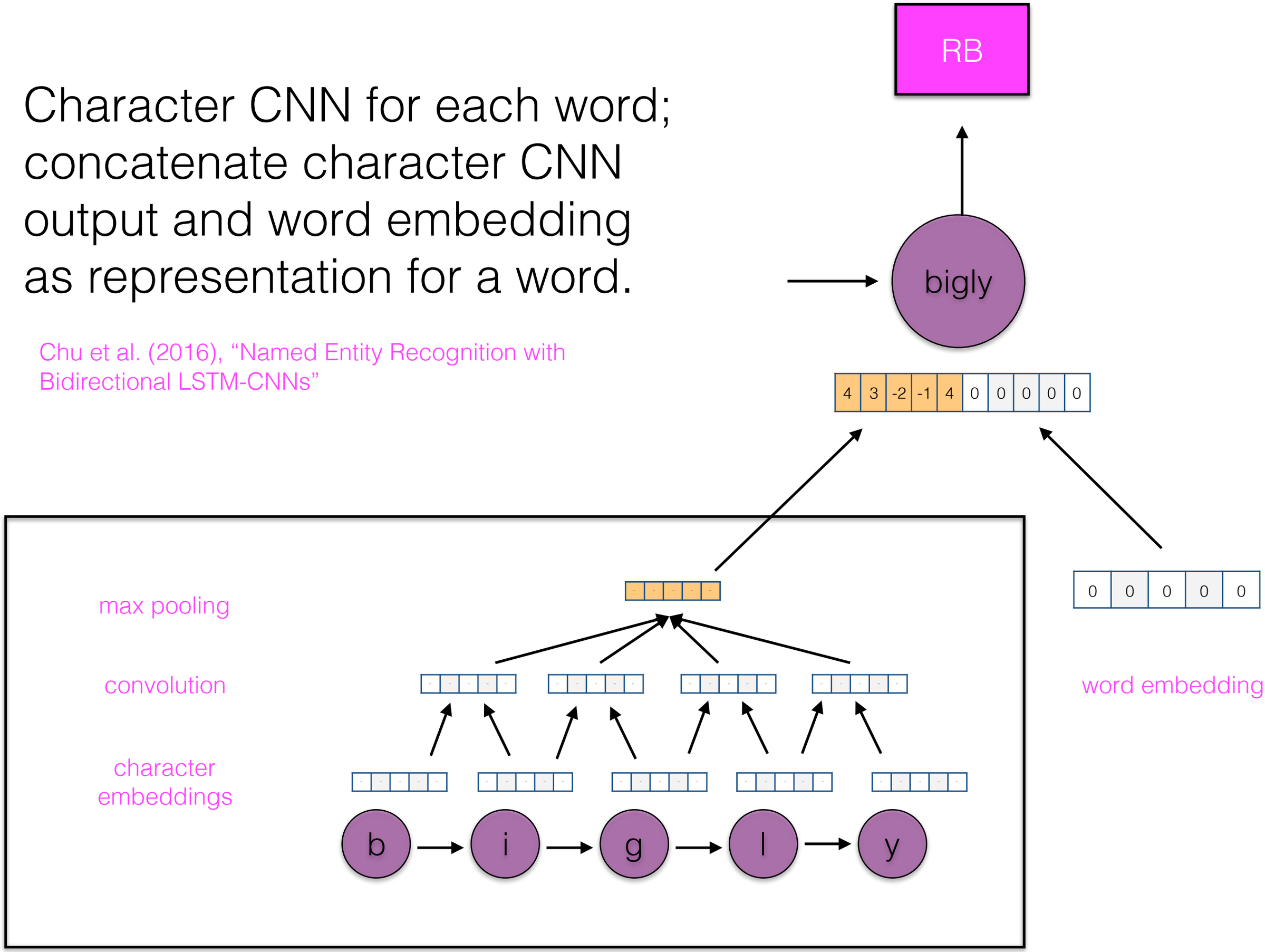
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

Lample et al. (2016), “Neural Architectures for Named Entity Recognition”



Character CNN for each word; concatenate character CNN output and word embedding as representation for a word.

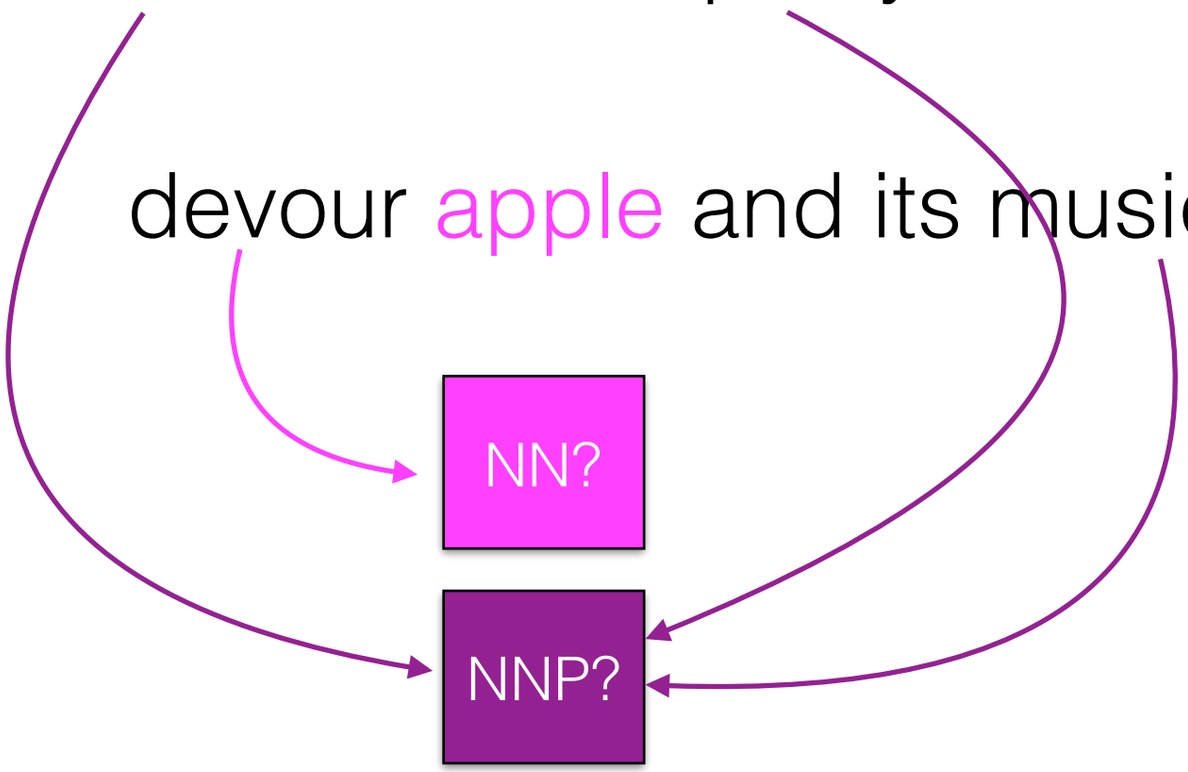
Chu et al. (2016), “Named Entity Recognition with Bidirectional LSTM-CNNs”



RNNs for POS

amazon and spotify's streaming services are going to

devour apple and its music purchasing model



RNNs for POS

amazon and spotify's streaming services are going to devour apple and its music purchasing model

Prediction: Can the information from far away get to the time step that needs it?

Training: Can error reach that far back during backpropagation?

RNNs

- Recurrent networks are deep in that they involve one “layer” for each time step (e.g., words in a sentence)

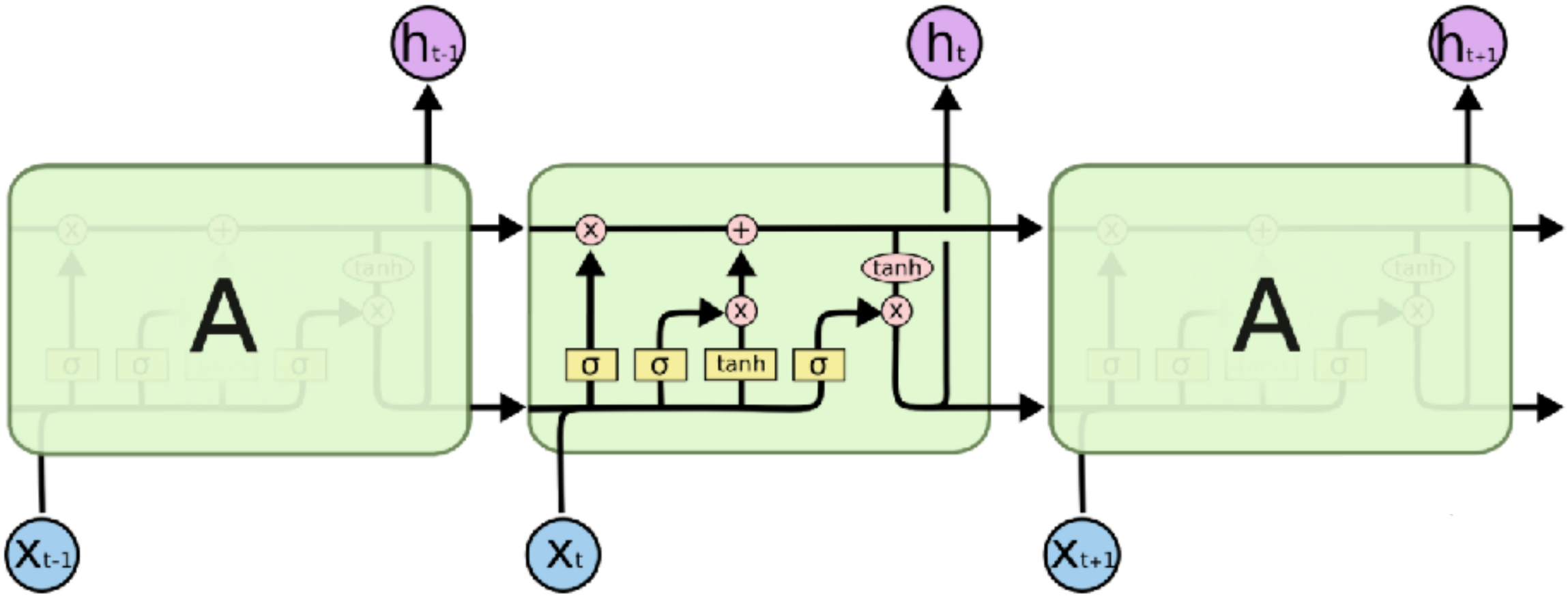
- Vanishing gradient problem: as error is back propagated through the layers of a deep network, they tend toward 0.

Long short-term memory network (LSTM)

- Designed to account for the vanishing gradient problem

- Basic idea: split the *s* vector propagated between time steps into a memory component and a hidden state component

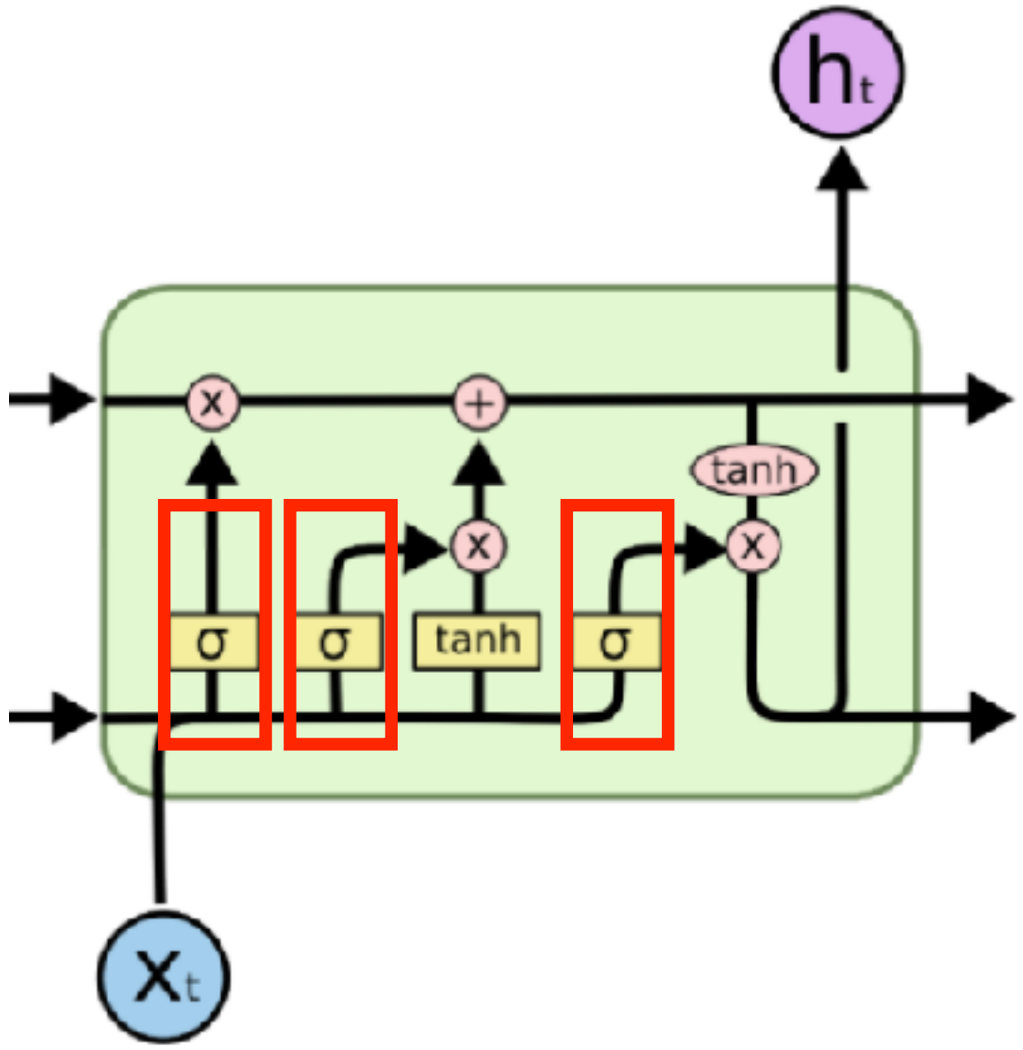
LSTMs



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gates

- LSTMs gates control the flow of information



- A sigmoid squashes its input to between 0 and 1
- By multiplying the output of a sigmoid elementwise with another vector, we forget information in the vector (if multiplied by 0) or allow it to pass (if multiplied by 1)

Forget gate: as a function of the previous hidden state and current input, forget information in the memory

3.7	1.4	-0.7	-1.4	7.8

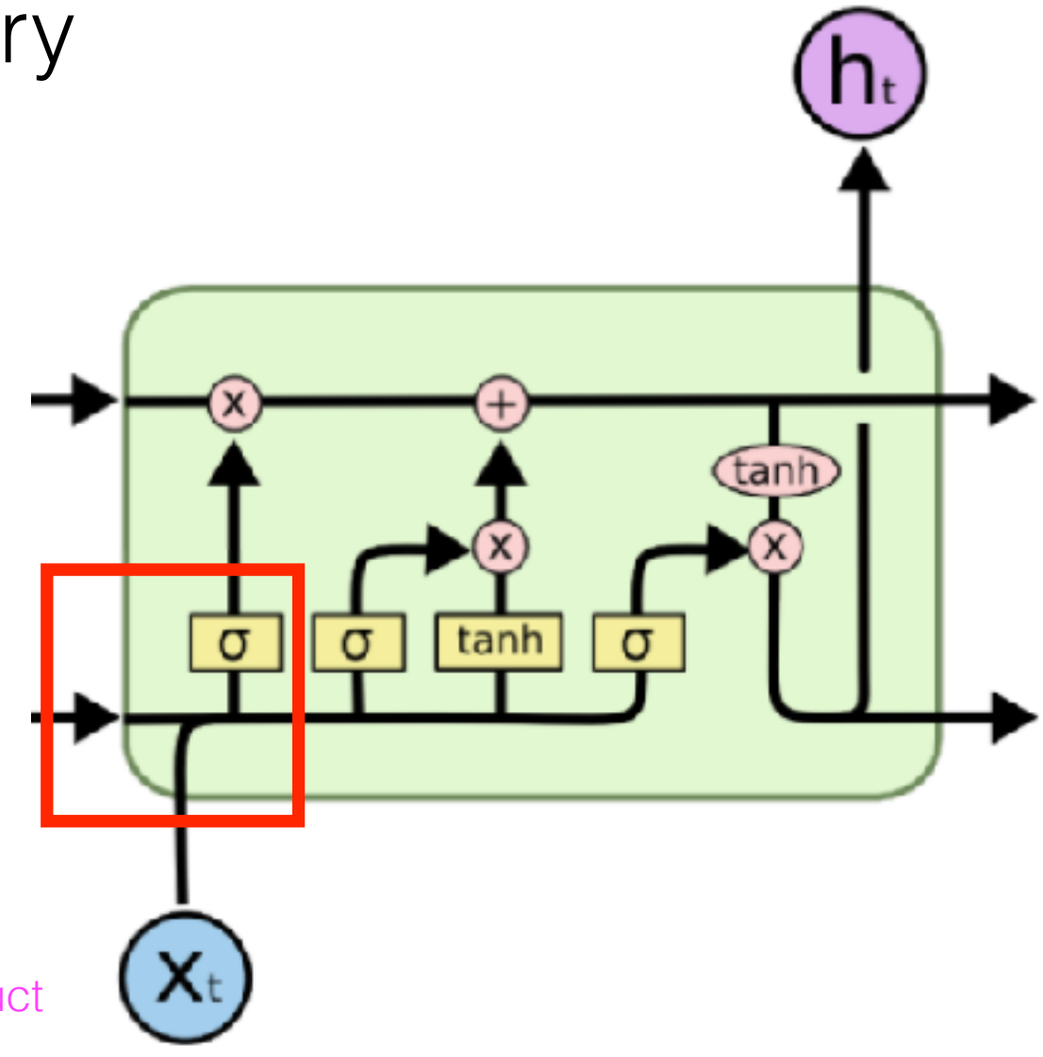
memory

0.01	0.99	0.5	0.98	0.01

hidden state

0.03	1.4	-0.35	-1.38	0.08

elementwise product



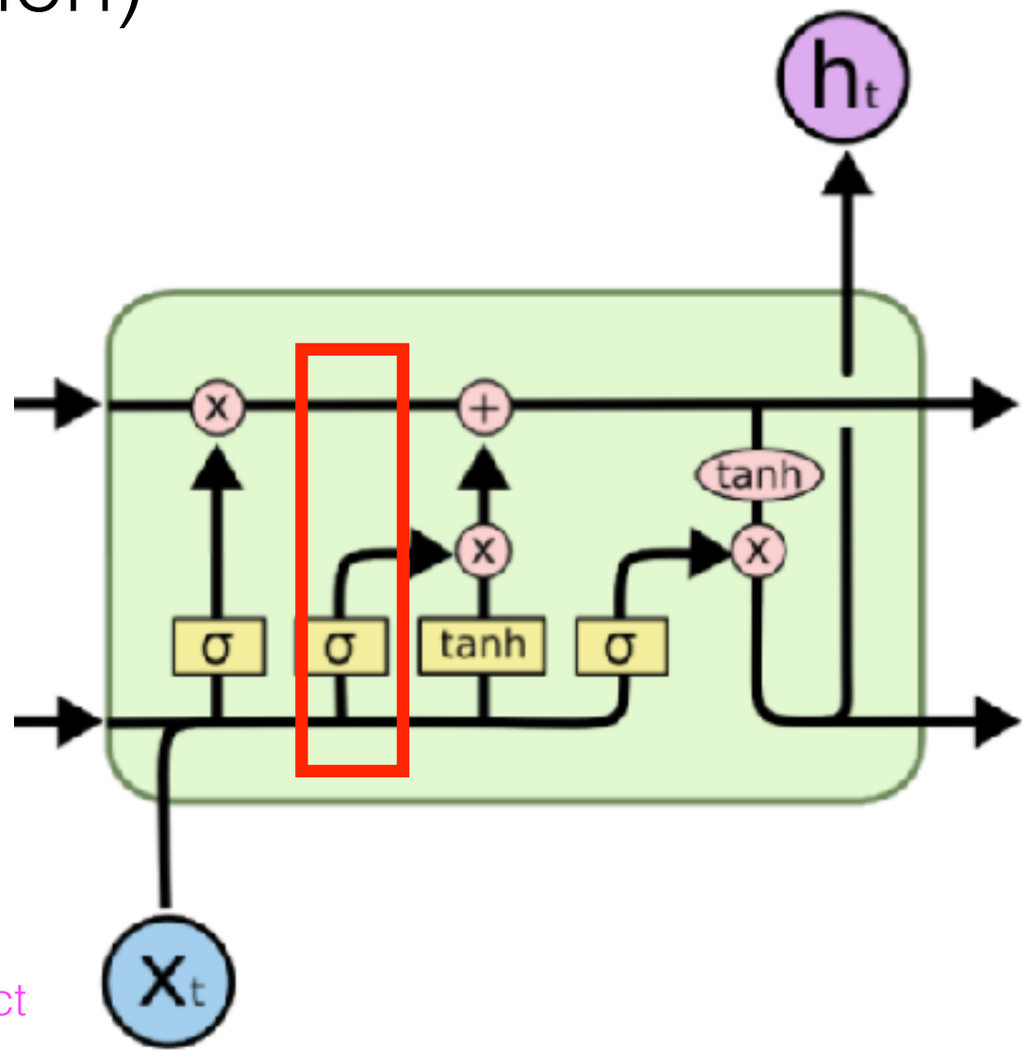
Input gate (but forget some information about the current observation)

0.03	1.4	-0.35	-1.38	0.08

memory

hidden state

elementwise product



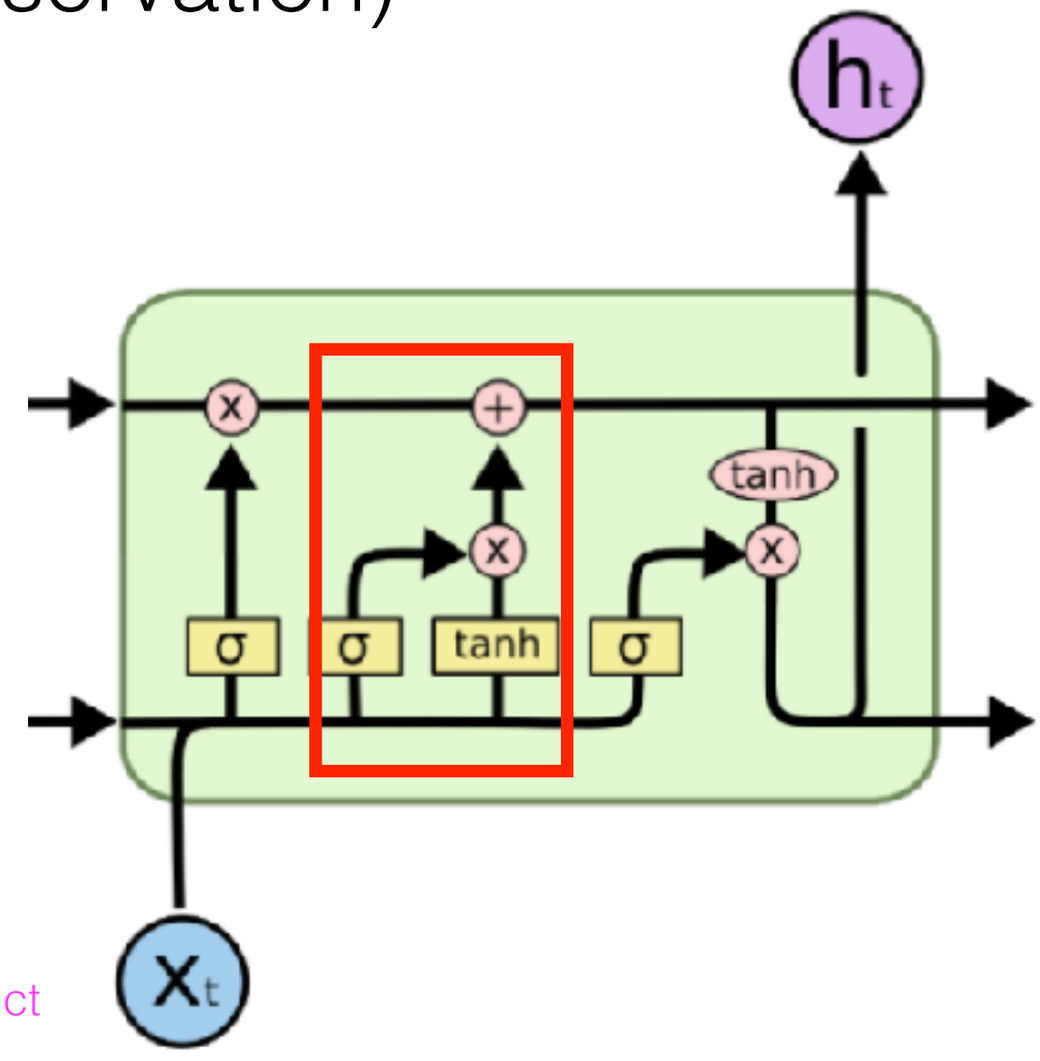
Update the memory (but forget some information about the current observation)

0.03	1.4	-0.35	-1.38	0.08

memory

hidden state

elementwise product



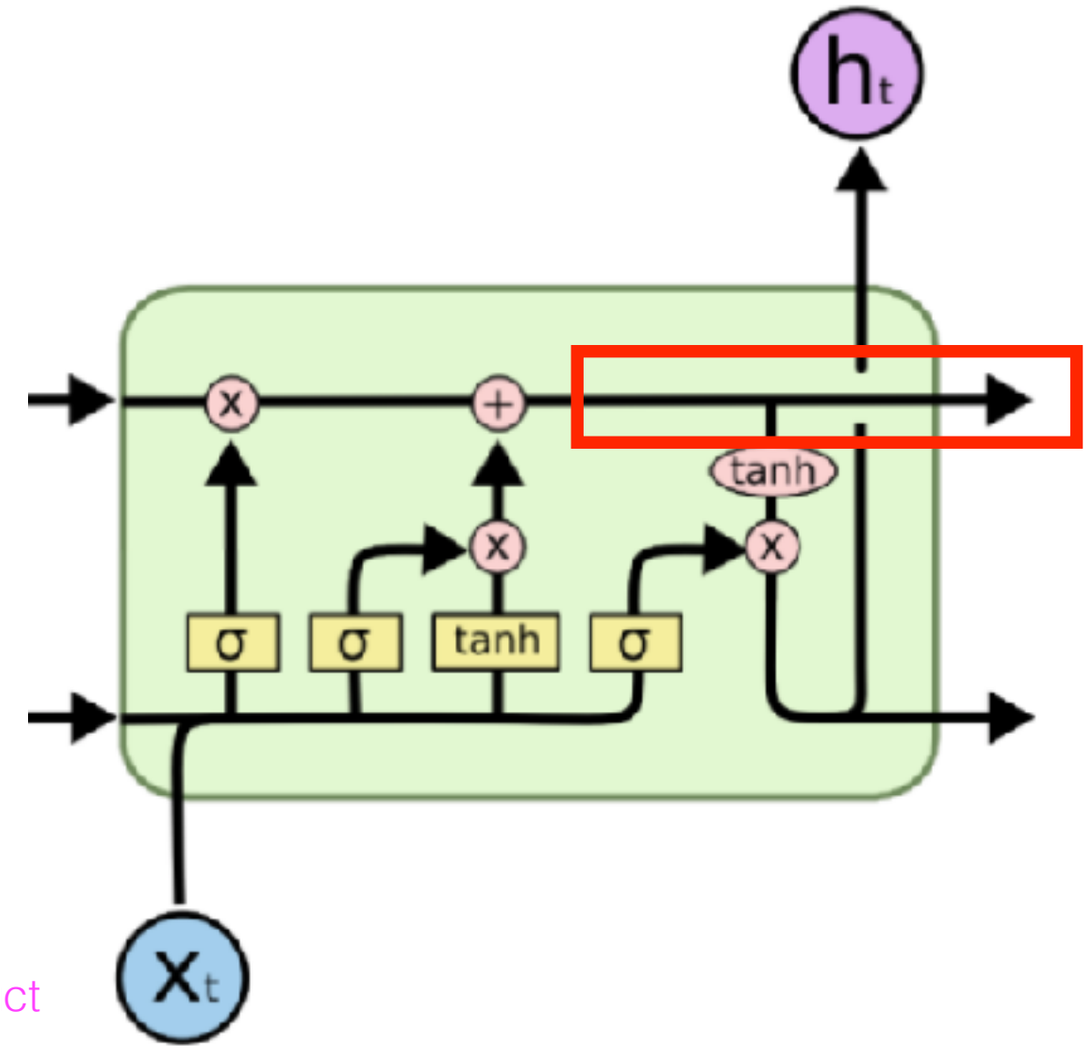
The memory passes directly to the next state

0.8	0.4	-1.4	9.8	3.4

memory

hidden state

elementwise product

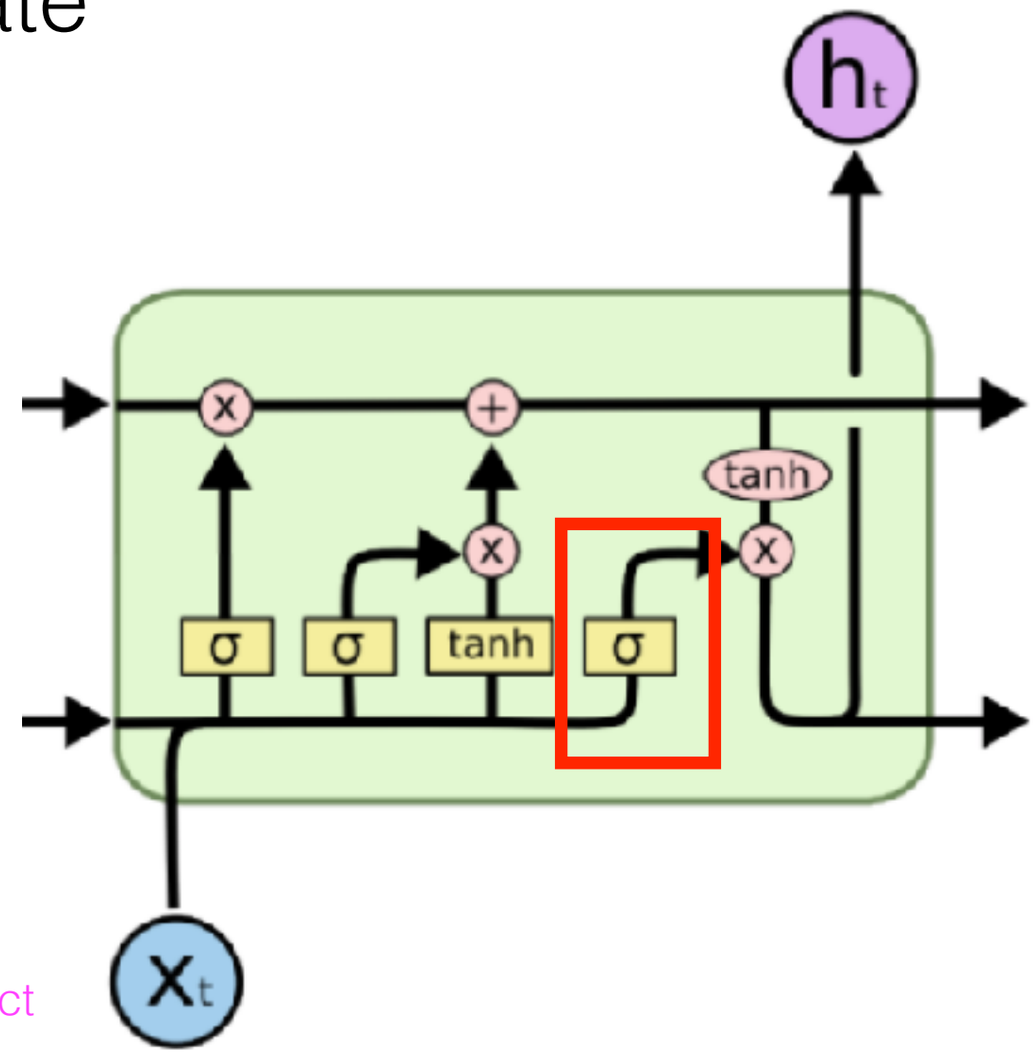


Output gate: forget some information
to send to the hidden state

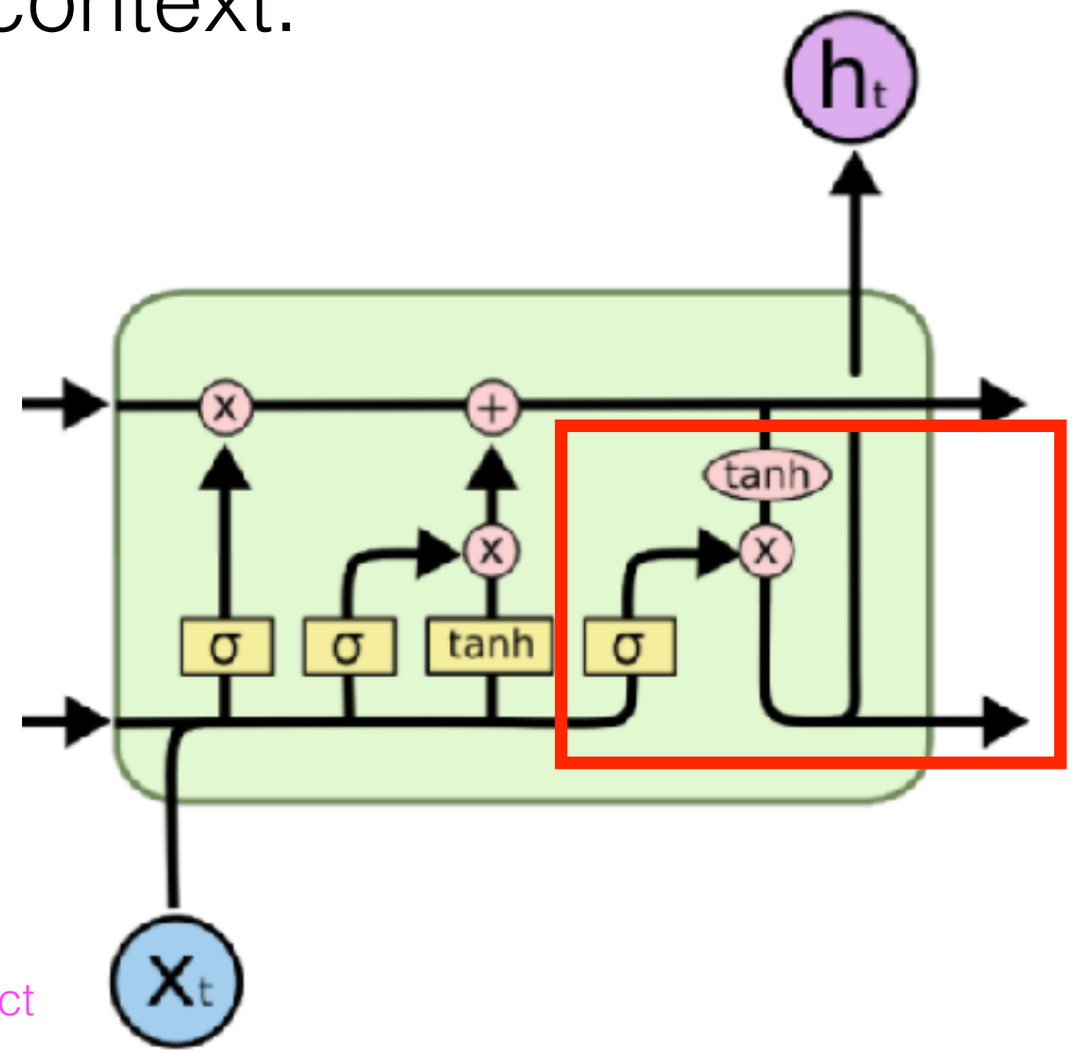
memory

hidden state

elementwise product



The hidden state is updated with the current observation and new context.



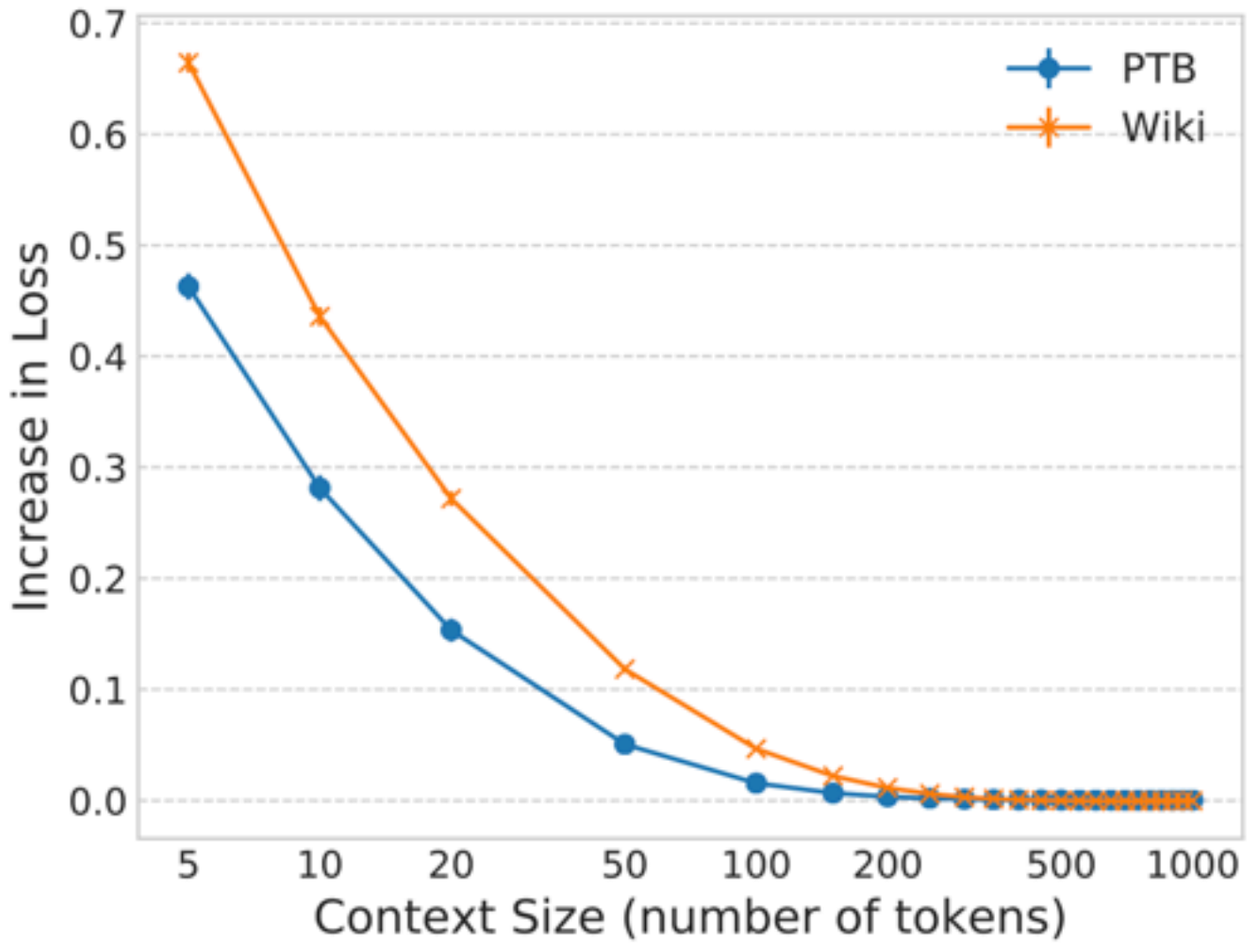
How much context?

- How would we design an experiment to measure how context an LSTM is using?

How much context?

- For language modeling, LSTMs are aware of about 200 words of context

- Ignores word order beyond 50 words



Khandelwal et al. (2018), “Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context” (ACL)

LSTM/RNN

- Is an RNN the same kind of sequence labeling model as an MEMM or CRF?

- It doesn't use nearby labels in making predictions! (More like logistic regression in this respect)

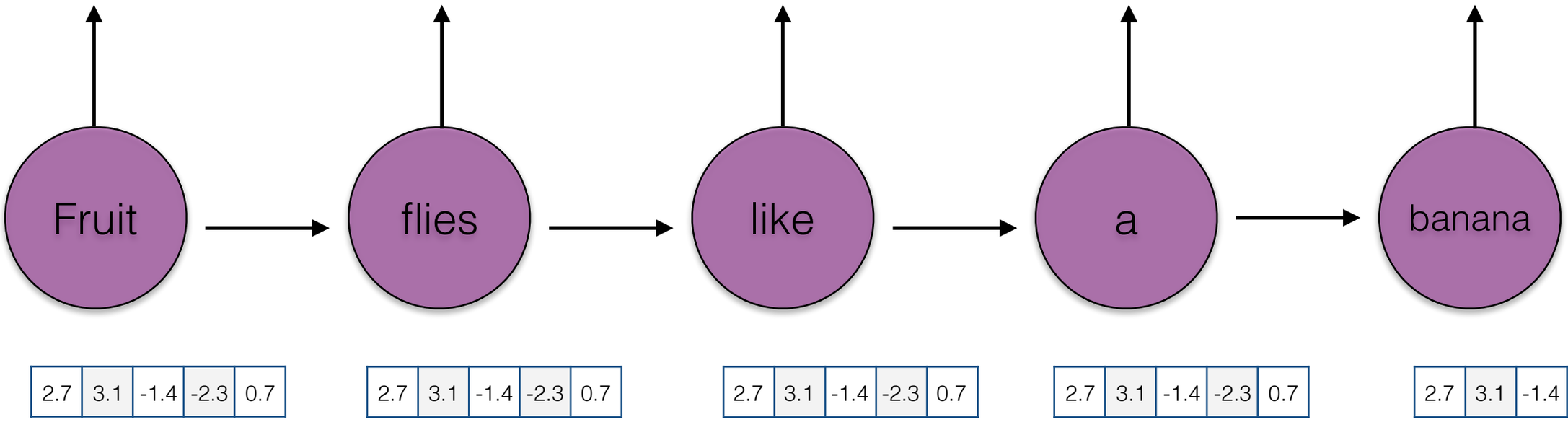
NN

VBZ

VB

VBZ	0.51
NNS	0.48
JJ	0.01
NN	0
…	…

The information that's passed between states is not the categorical choice (VBZ) but a hidden state that generated the distribution.



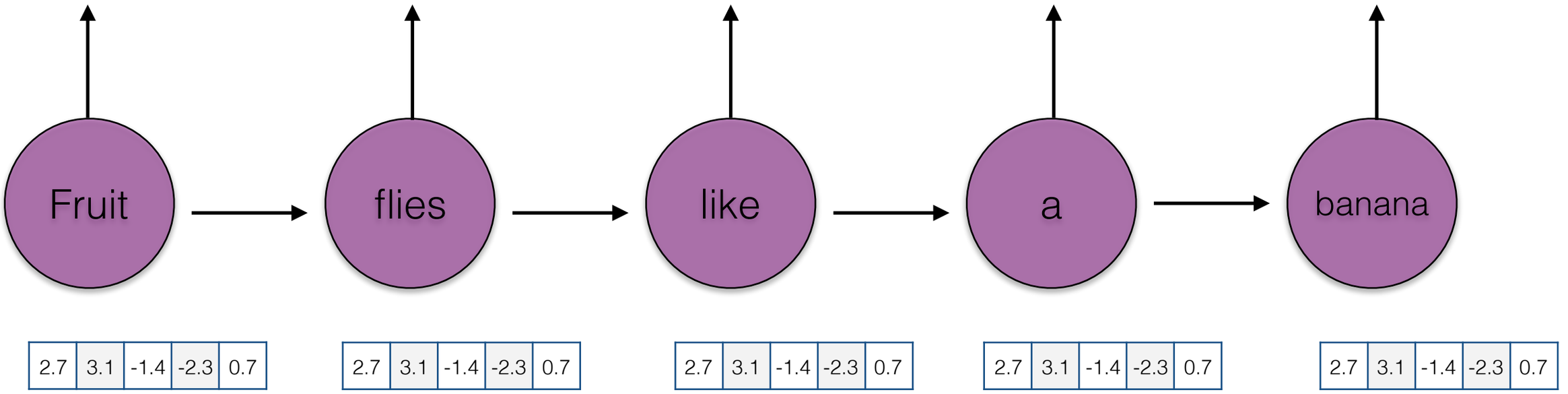
NN

VBZ

VB

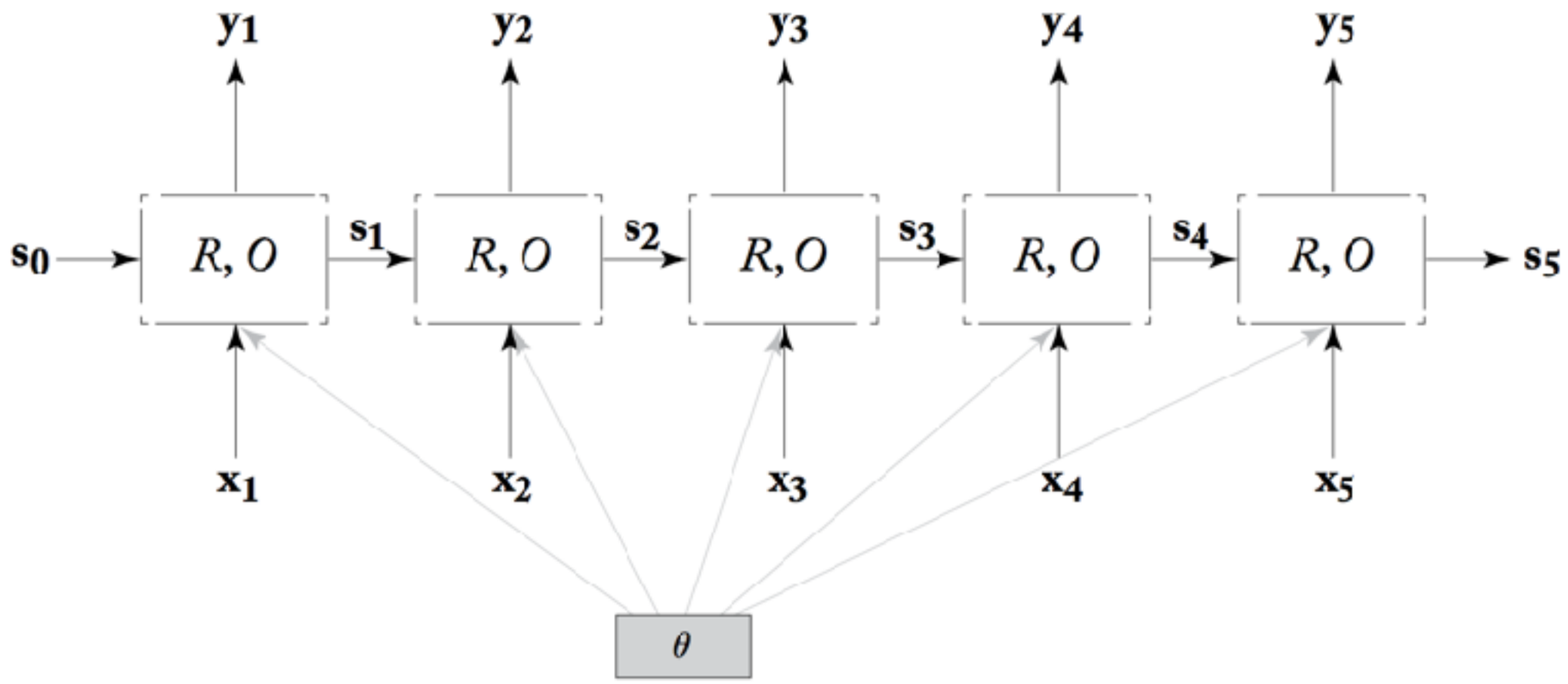
VBZ	0.51
NNS	0.48
JJ	0.01
NN	0
…	…

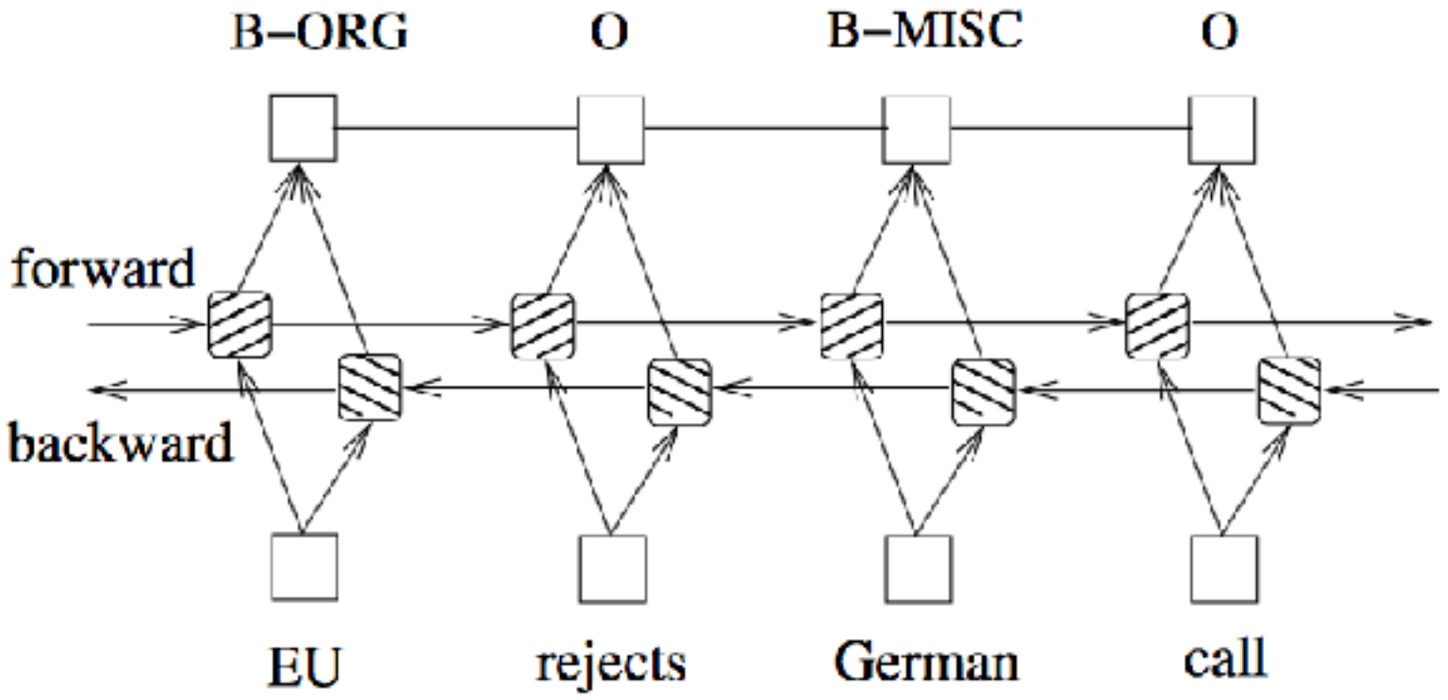
If we knew the categorical choice of VBZ at t_2, P(VB) at t_3 would be much lower.



Recurrent neural network

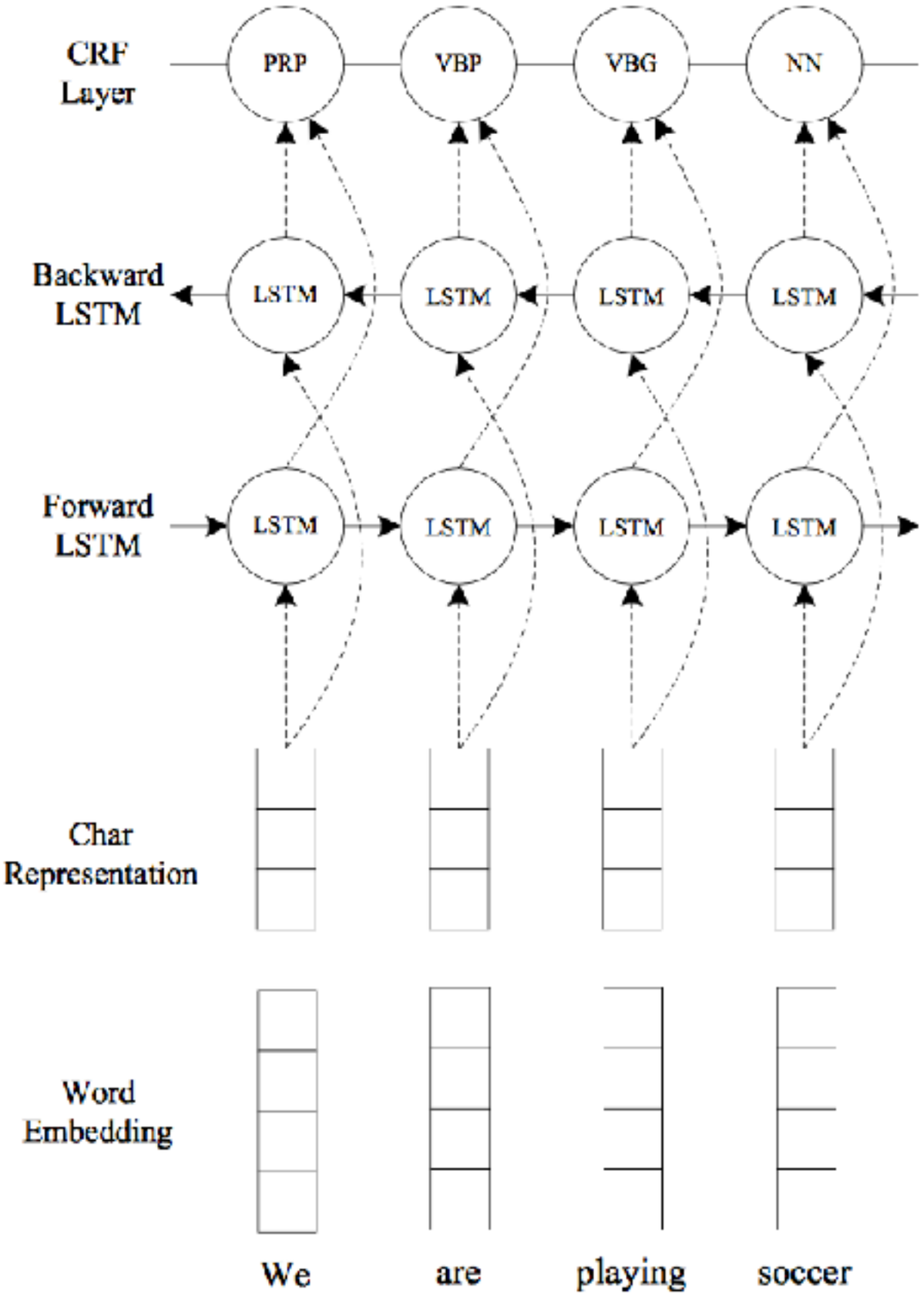
- How could we incorporate nearby label information into a single-direction RNN?





Huang et al. 2015, “Bidirectional LSTM-CRF Models for Sequence Tagging"

Ma and Hovy (2016), “End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF”



	POS		NER					
	Dev	Test	Dev			Test		
Model	Acc.	Acc.	Prec.	Recall	F1	Prec.	Recall	F1
BRNN	96.56	96.76	92.04	89.13	90.56	87.05	83.88	85.44
BLSTM	96.88	96.93	92.31	90.85	91.57	87.77	86.23	87.00
BLSTM-CNN	97.34	97.33	92.52	93.64	93.07	88.53	90.21	89.36
BRNN-CNN-CRF	97.46	97.55	94.85	94.63	94.74	91.35	91.06	91.21

Ma and Hovy (2016), “End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF”