Natural Language Processing

Info 159/259

Lecture 12: Neural sequence labeling (Feb 27, 2020)

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

NN TO VB

will to fight

vs.

MD TO VB

will to fight

$P(y_1 = \text{MD} \mid y_0 = \text{START}, x, \beta)$	0.9
$P(y_1 = \text{NN} \mid y_0 = \text{START}, x, \beta)$	0.1

These probabilities must sum to 1 because the conditioning context is the same (local normalization)

$P(y_2 = \text{TO} \mid y_1 = \text{MD}, x, \beta)$	1.0
$P(y_2 = \text{TO} \mid y_1 = \text{NN}, x, \beta)$	1.0

Different conditioning contexts; $P(y_2 = \text{TO})$ will be 1 no matter the context.

$P(y_3 = \text{VB} \mid y_1 = \text{TO}, x, \beta)$	1.0

$P(\text{NN TO VB} \mid x, \beta) = 0.1$

$P(\text{MD TO VB} \mid x, \beta) = 0.9$

NN TO VB

will to fight

vs.

MD TO VB

will to fight

$P(y_1 = \text{MD} \mid y_0 = \text{START}, x, \beta)$	0.9
$P(y_1 = \text{NN} \mid y_0 = \text{START}, x, \beta)$	0.1

$P(y_2 = \text{TO} \mid y_1 = \text{MD}, x, \beta)$	1.0
$P(y_2 = \text{TO} \mid y_1 = \text{NN}, x, \beta)$	1.0

$P(y_3 = \text{VB} \mid y_1 = \text{TO}, x, \beta)$	1.0

Here, the information that TO *almost never* follows MD is lost and can't influence the tagging decision. We end up with an incorrect tagging.

$P(\text{NN TO VB} \mid x, \beta) = 0.1$

$P(\text{MD TO VB} \mid x, \beta) = 0.9$

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

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. But we can do this efficiently in NK^2 time using the forward algorithm.

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

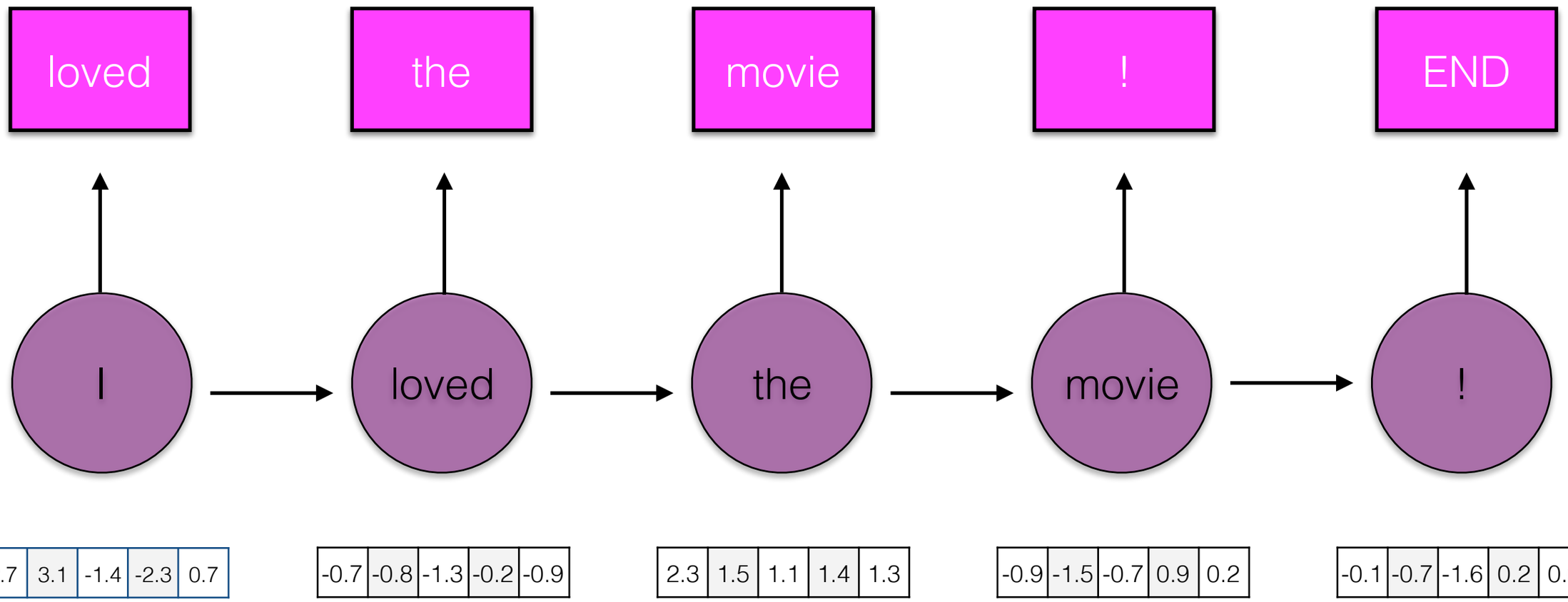
CRF

	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

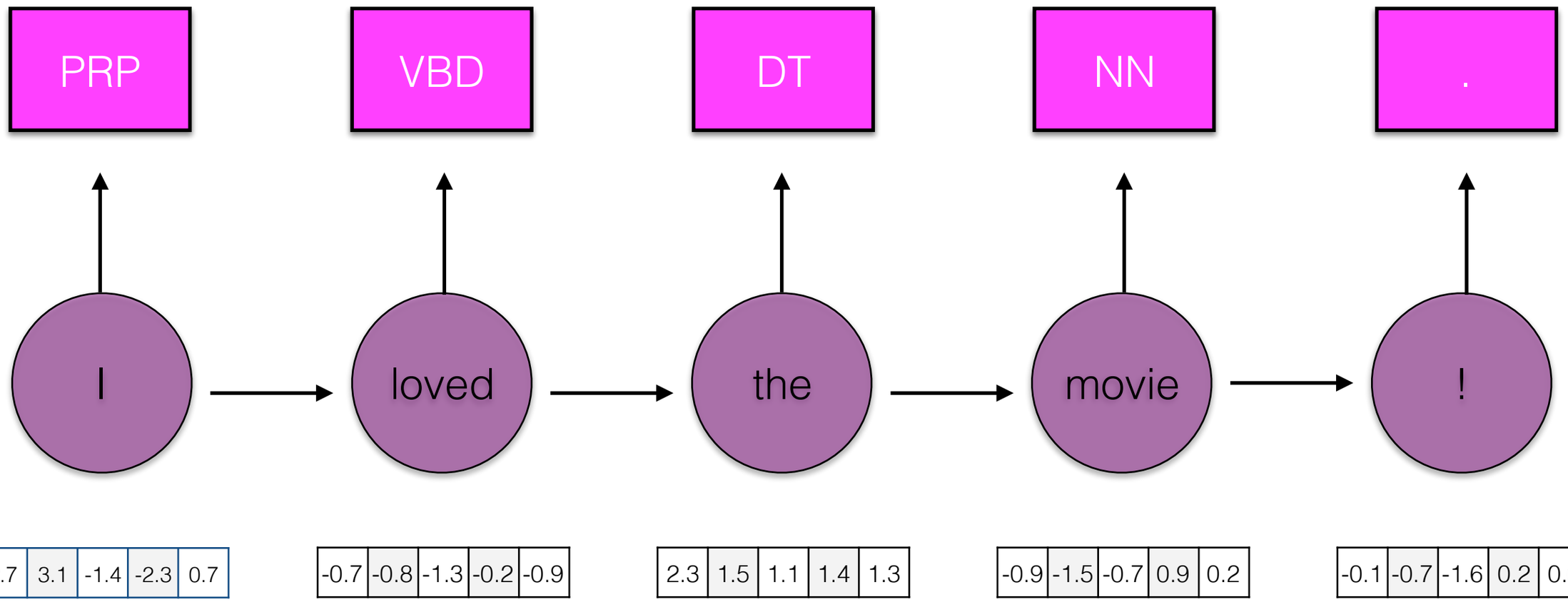
Recurrent neural network

- RNNs allow arbitarily-sized conditioning contexts and 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

- 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

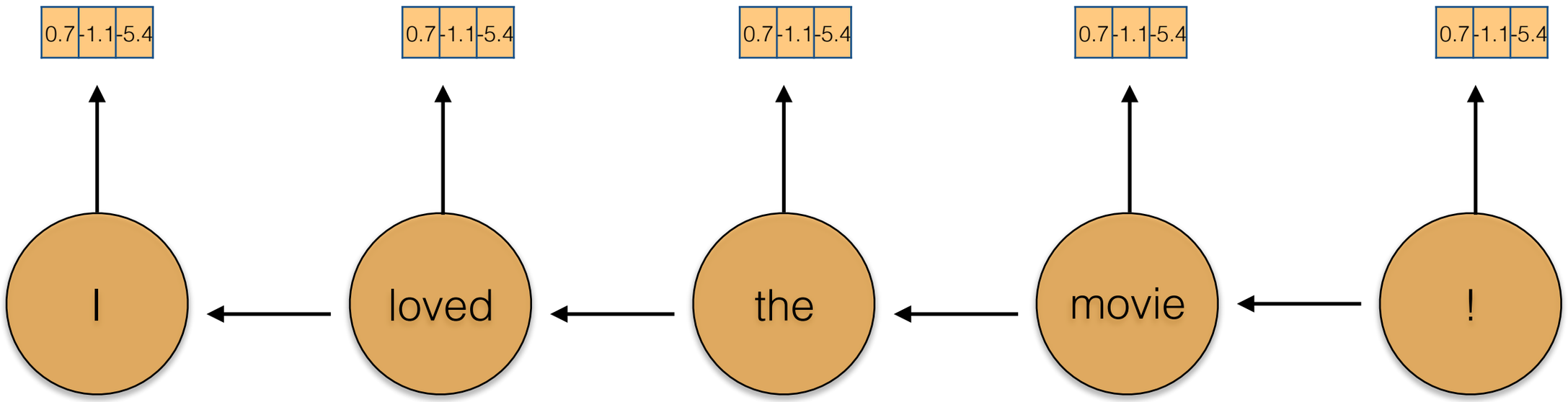
- A powerful alternative is make predictions conditioning both on the past and the future.

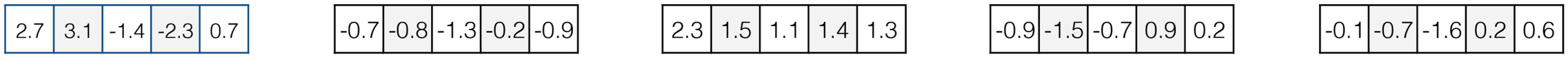
- Two RNNs
    - One running left-to-right
    - One right-to-left

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

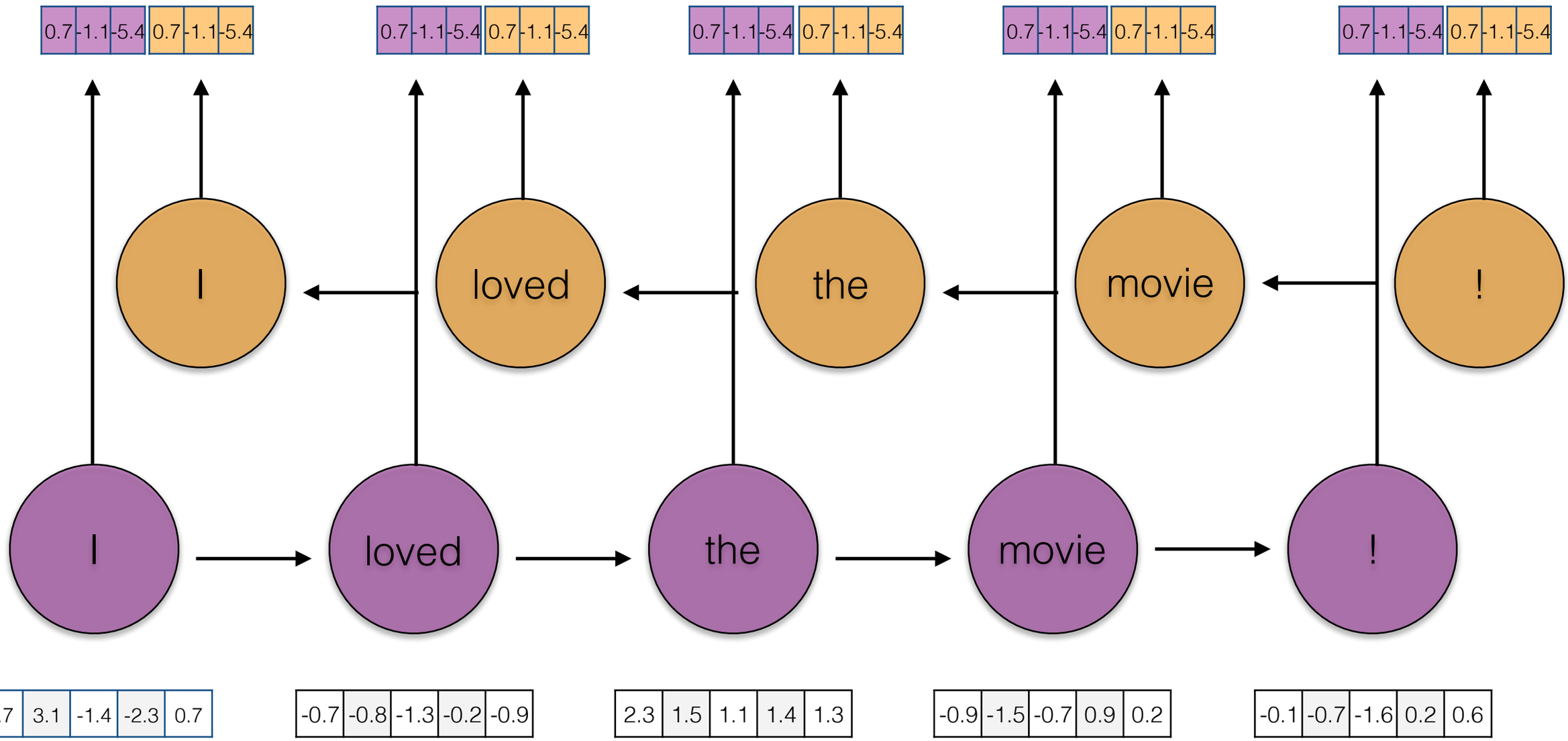
Bidirectional RNN

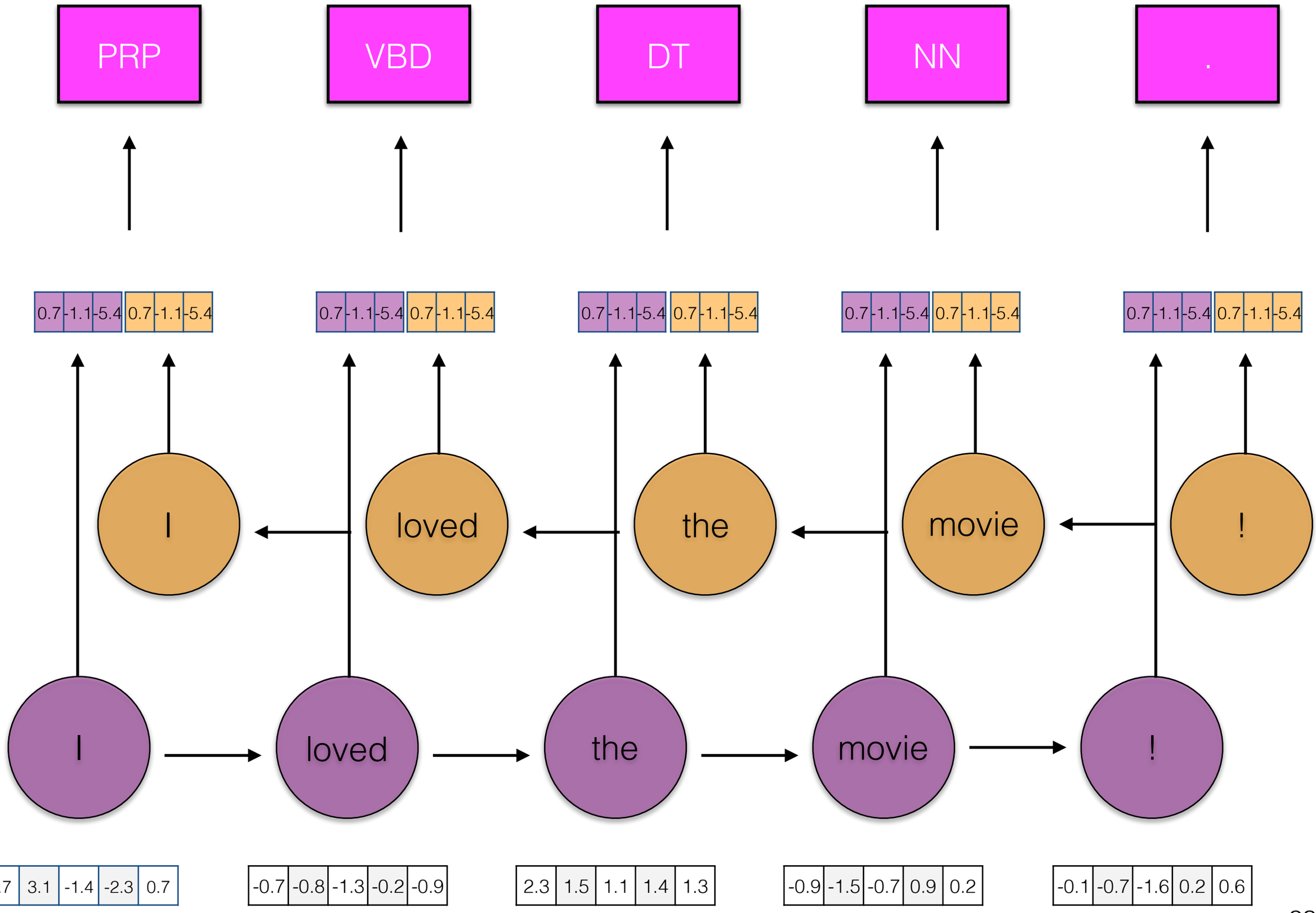
backward RNN





Bidirectional RNN




PRP
VBD
DT
NN
.
0.7 -1.1 -5.4 0.7 -1.1 -5.4
I
loved
the
movie
!
I
loved
the
movie
!
2.7 3.1 -1.4 -2.3 0.7
-0.7 -0.8 -1.3 -0.2 -0.9
2.3 1.5 1.1 1.4 1.3
-0.9 -1.5 -0.7 0.9 0.2
-0.1 -0.7 -1.6 0.2 0.6

Training BiRNNs

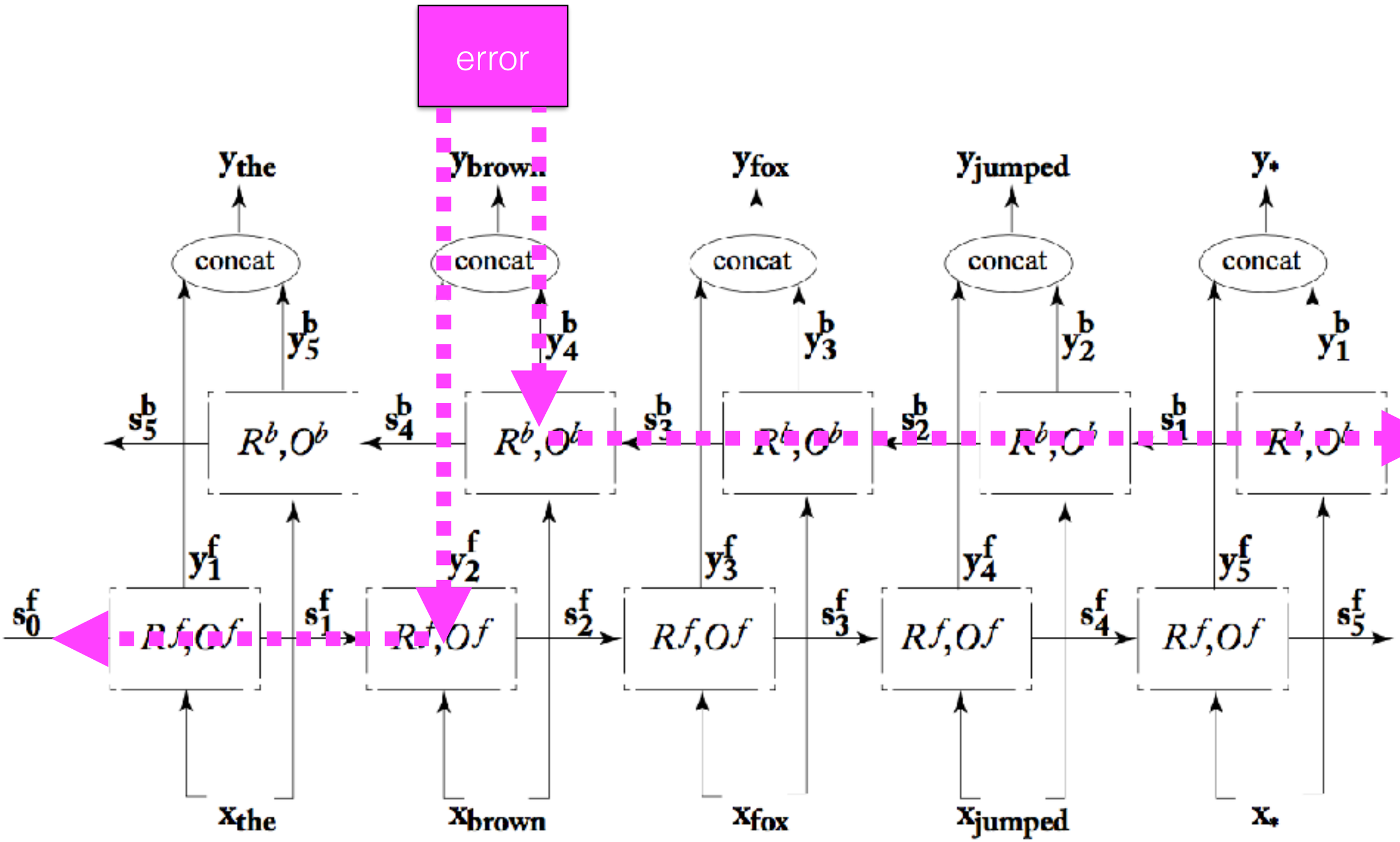
- Given this definition of an BiRNN:

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

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

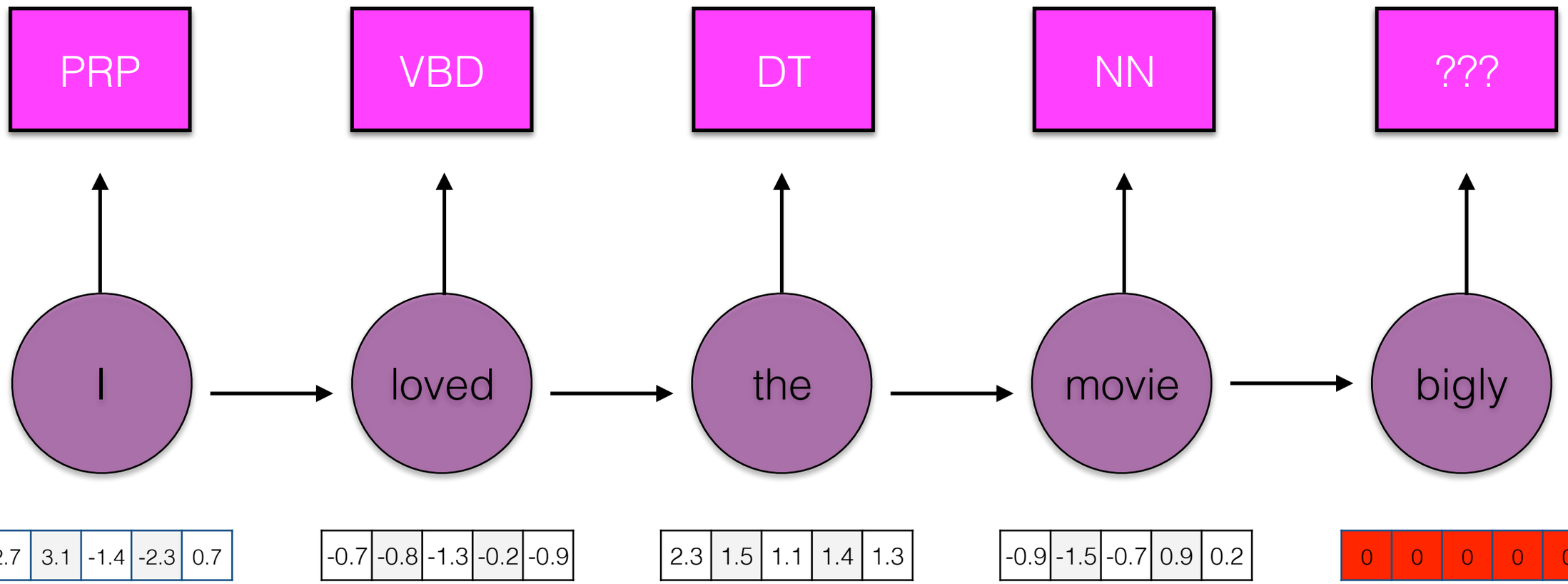
$$y_i = \text{softmax}\left([s_f^i; s_b^i] W^o + b^o\right)$$

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



Goldberg 2017

How do we fix this?



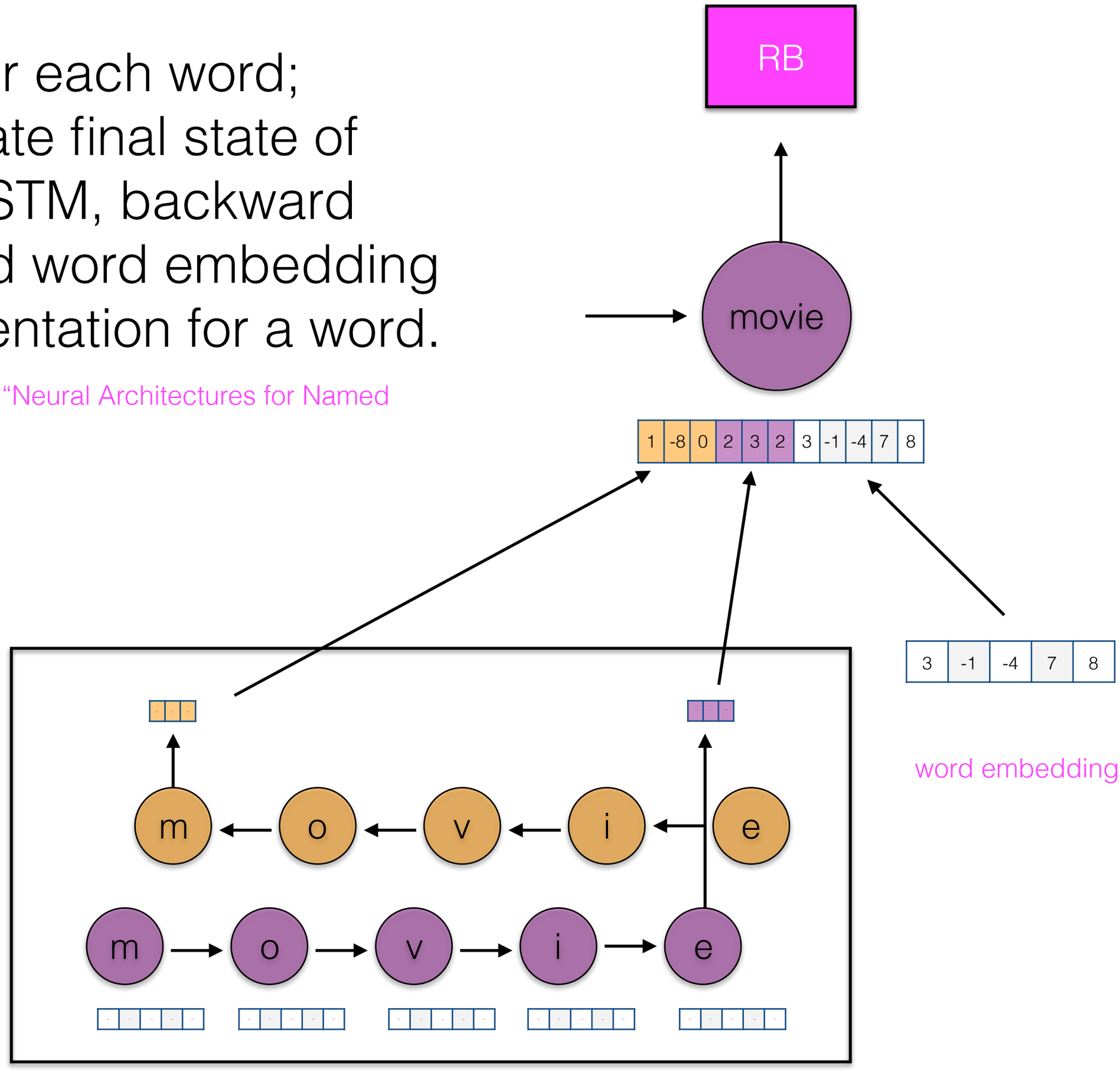
Subword information

- 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.

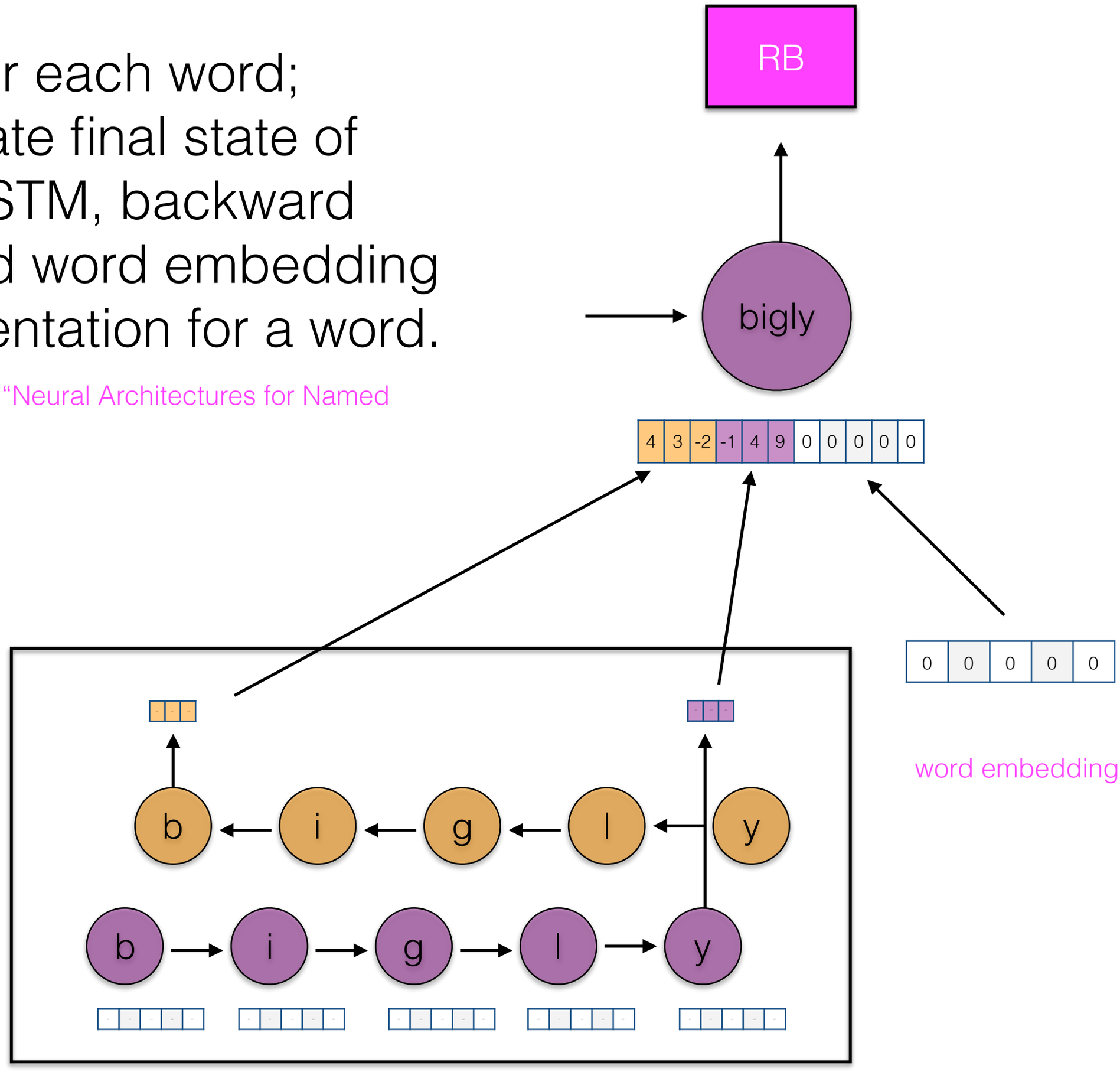
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

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



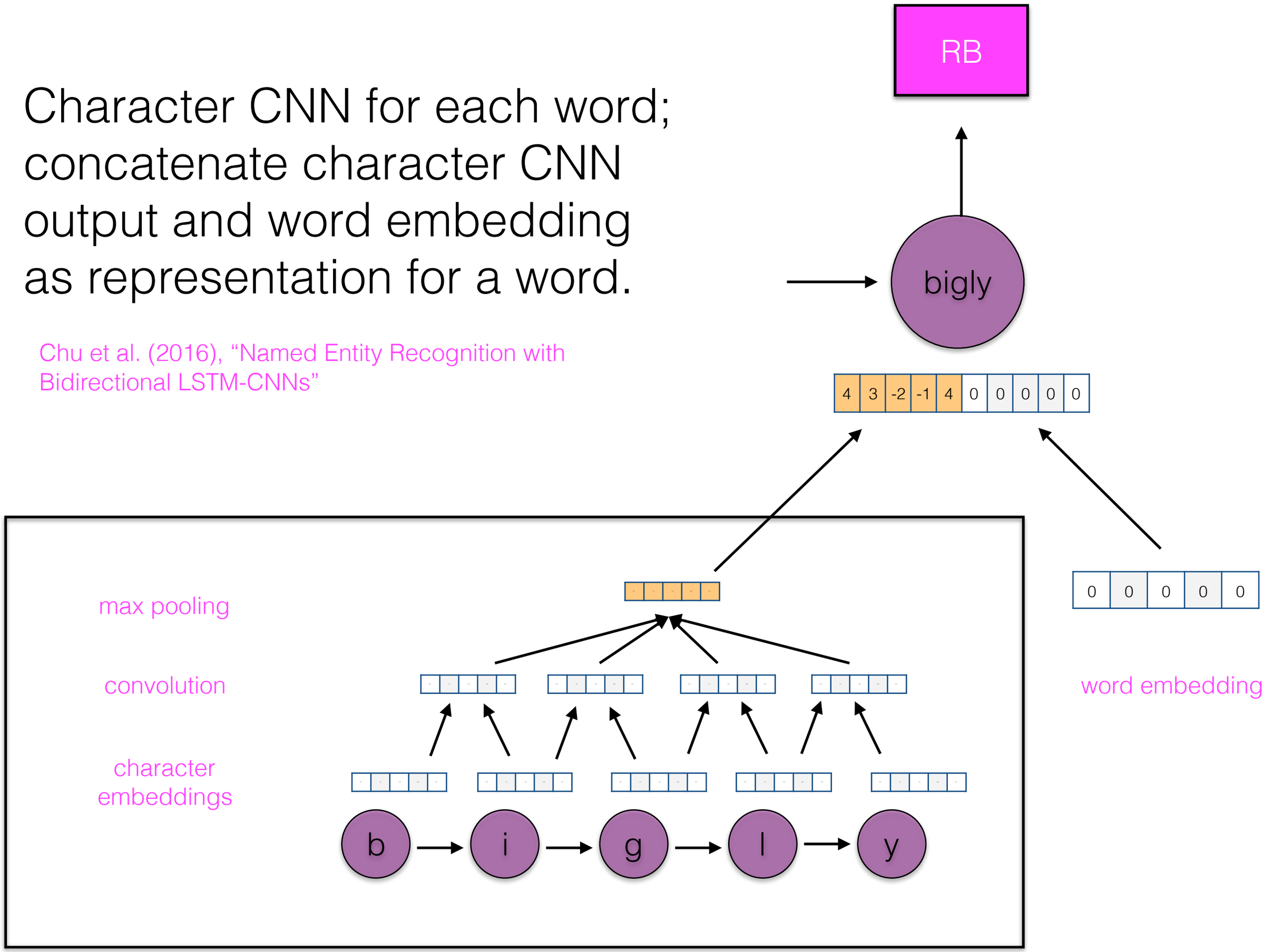
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

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.

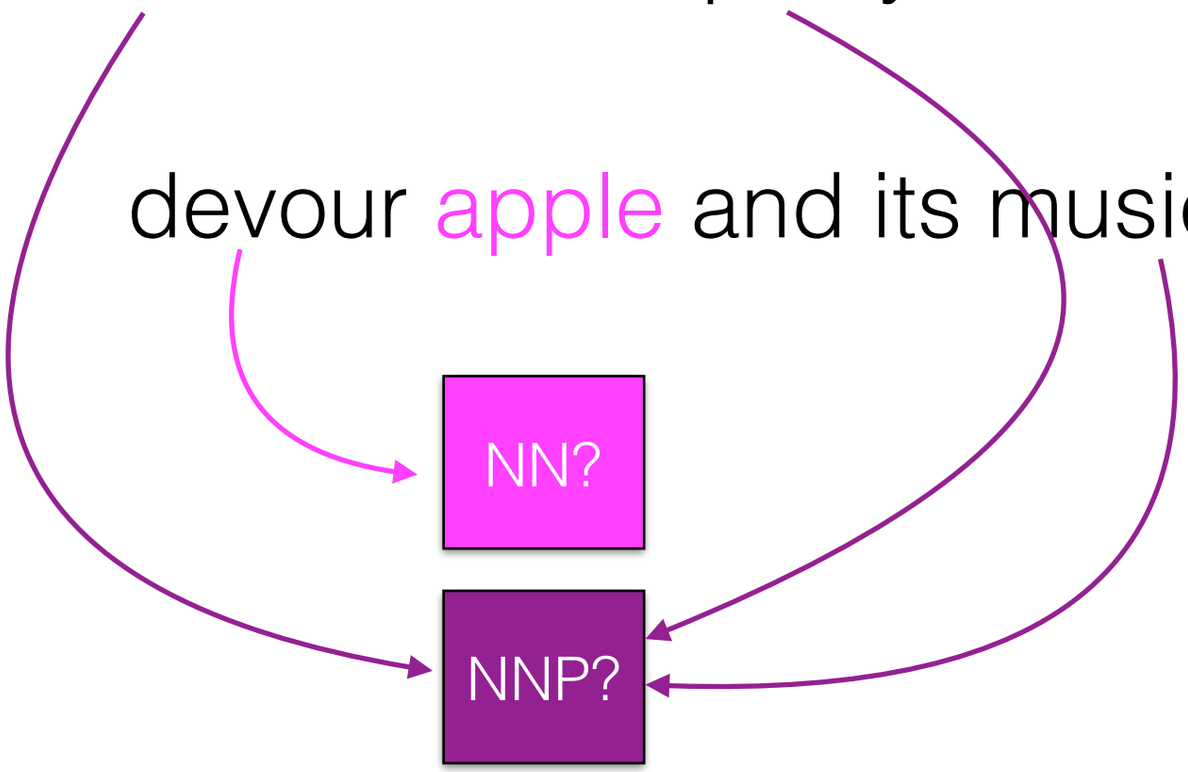
Chu et al. (2016), “Named Entity Recognition with Bidirectional LSTM-CNNs”



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)

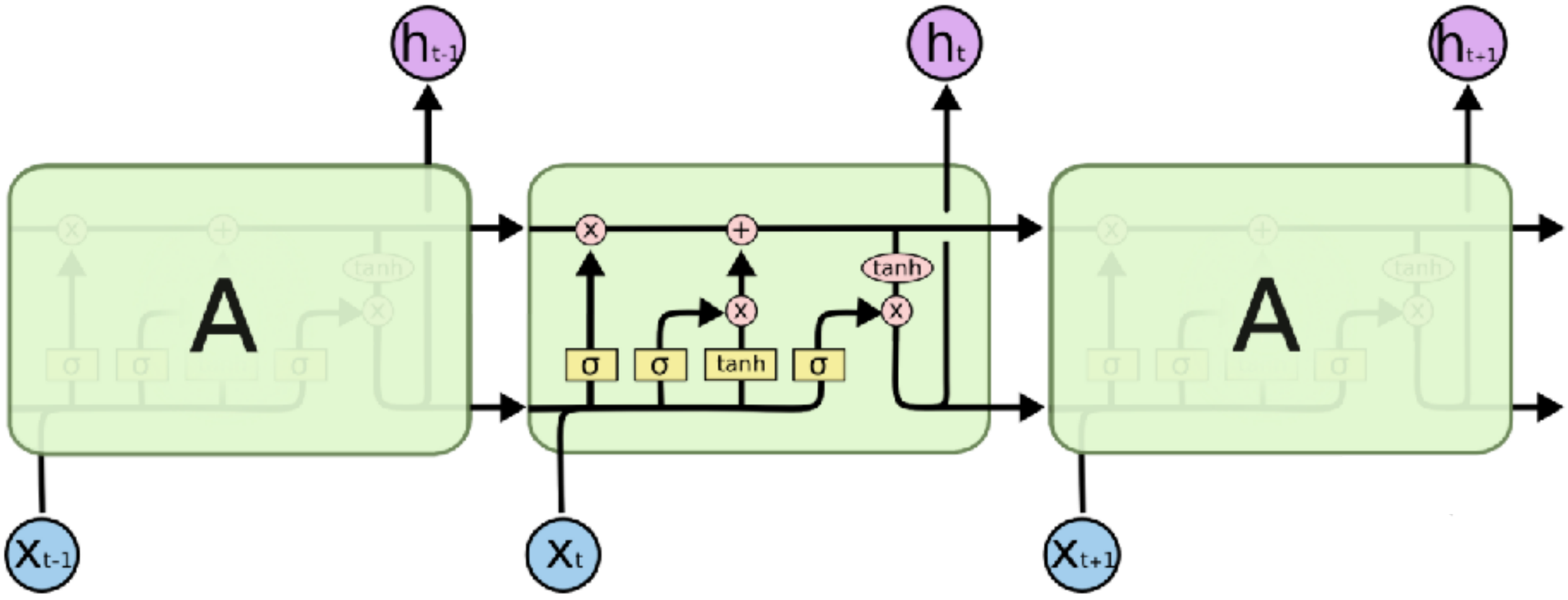
- Vanishing gradient problem: as error is back propagated through the layers of a deep network, they tend toward 0.

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

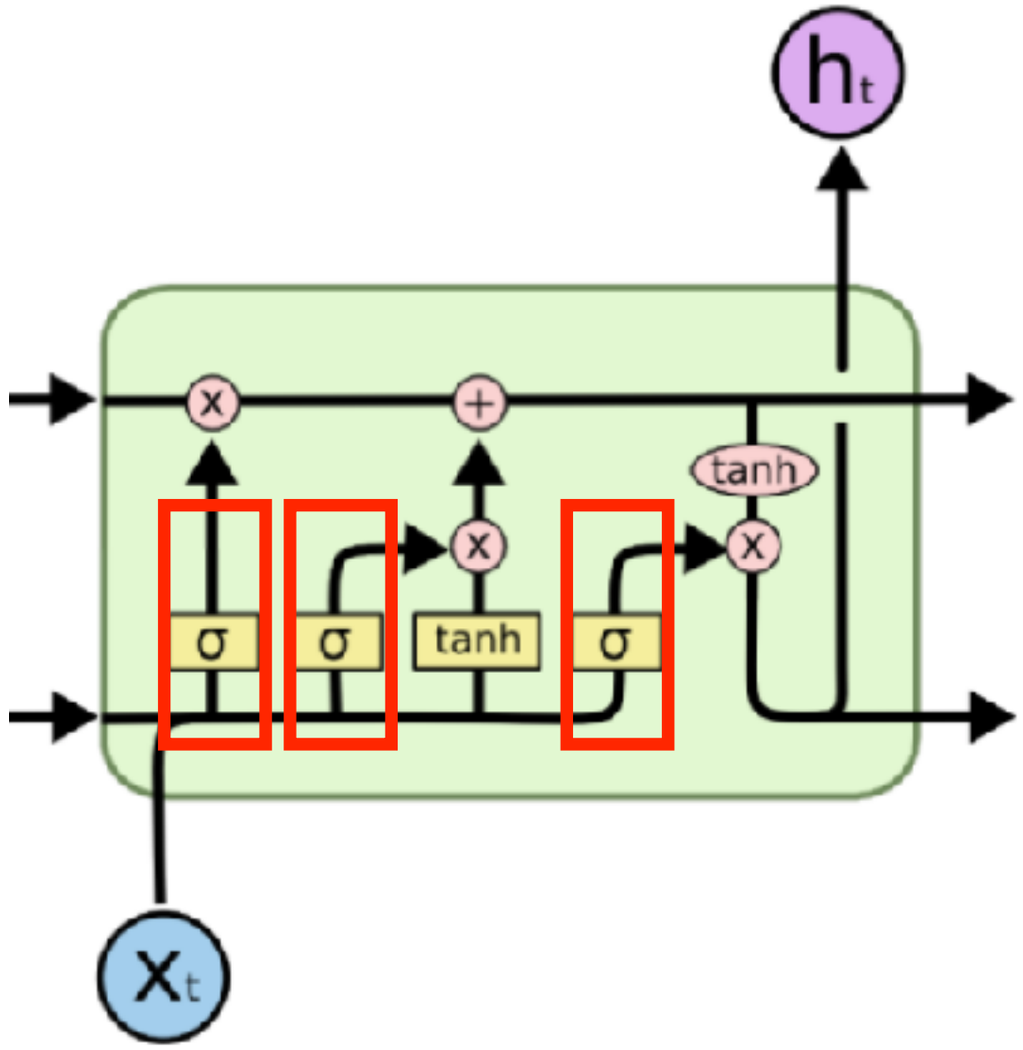
LSTMs



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)

input

3.7	1.4	-0.7	-1.4	7.8

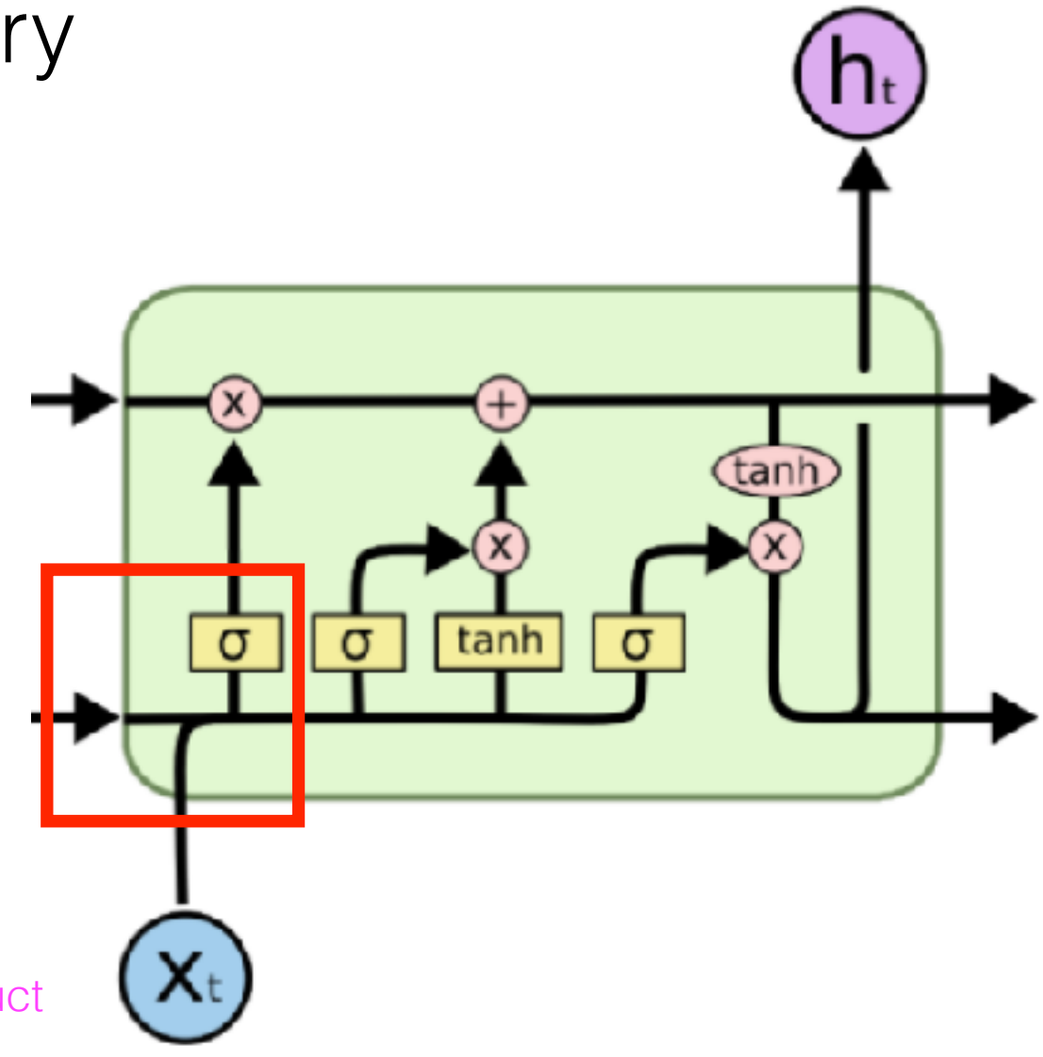
gate

0.01	0.99	0.5	0.98	0.01

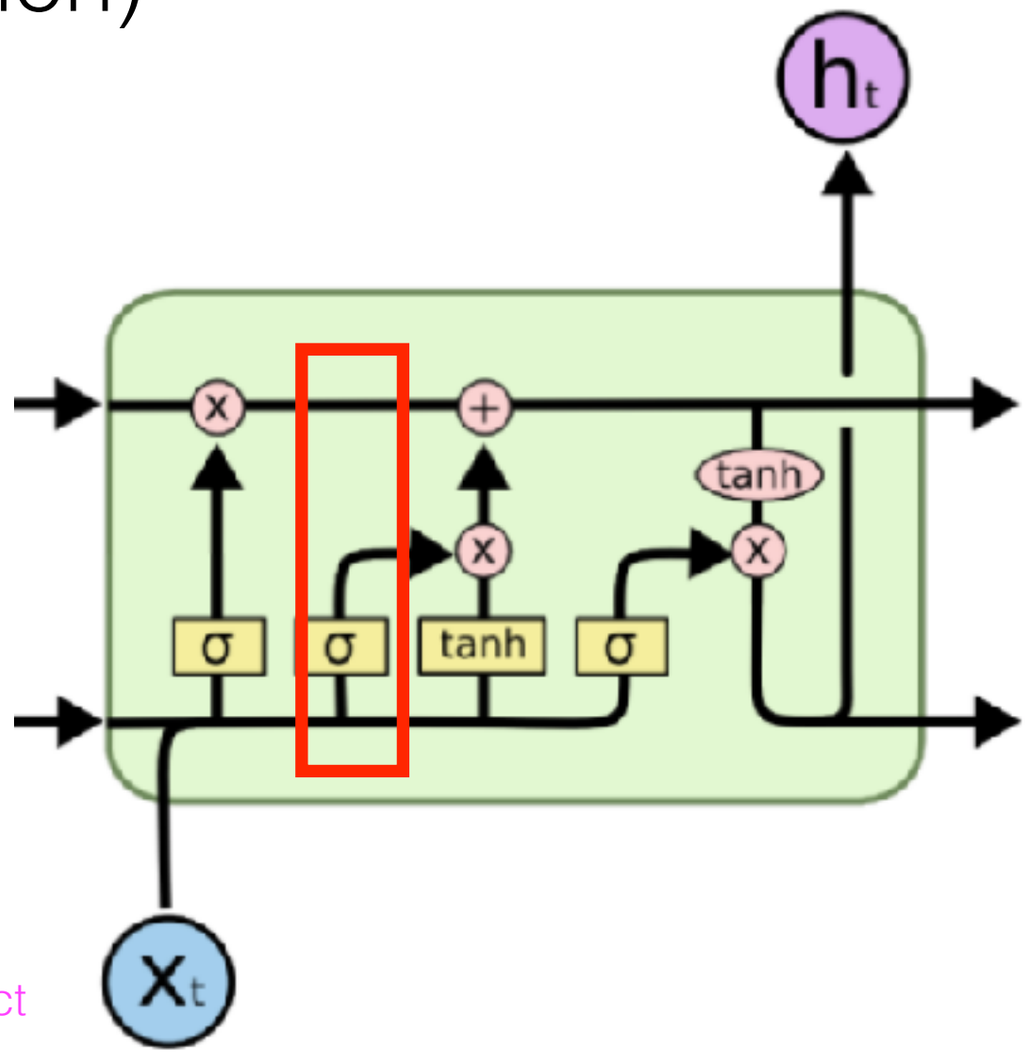
output

0.03	1.4	-0.35	-1.38	0.08

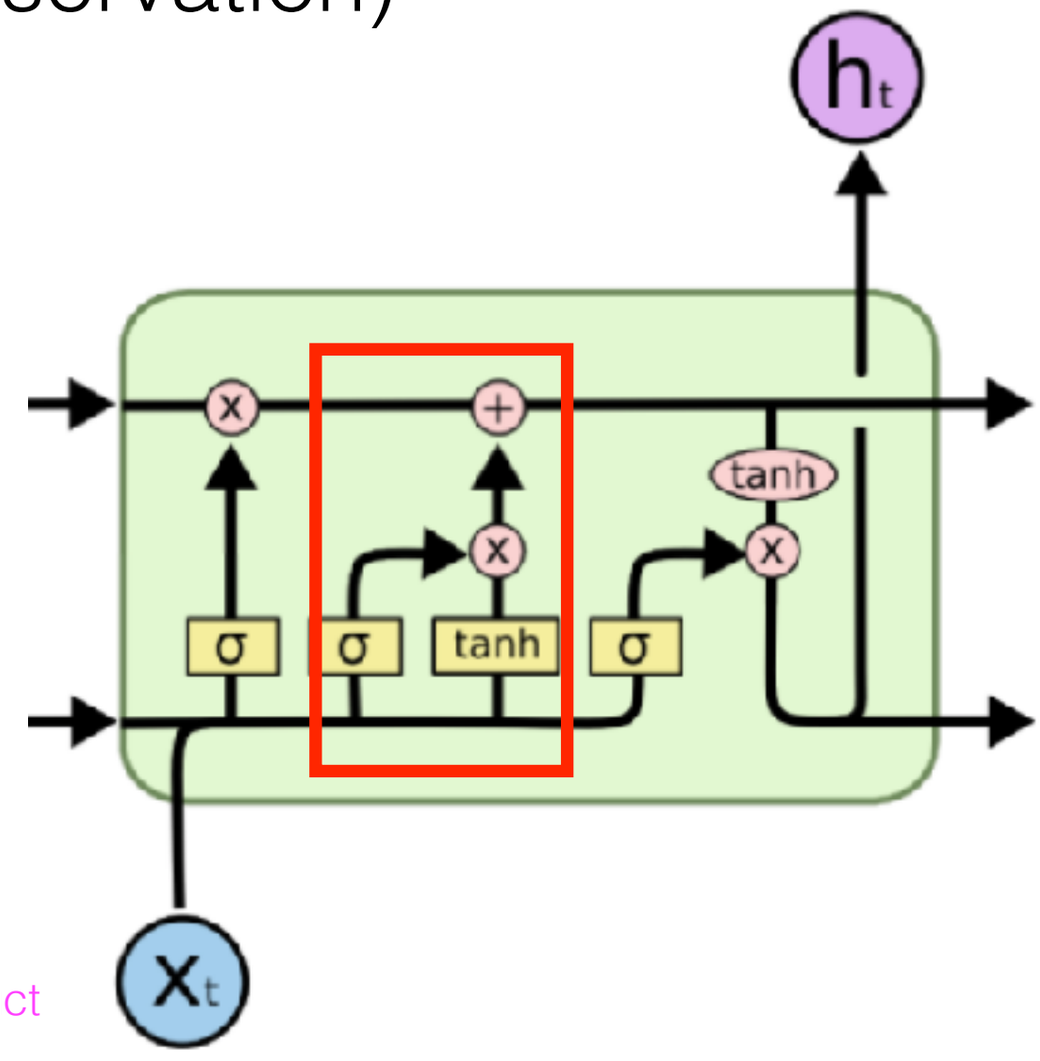
Forget gate: as a function of the previous hidden state and current input, forget information in the memory



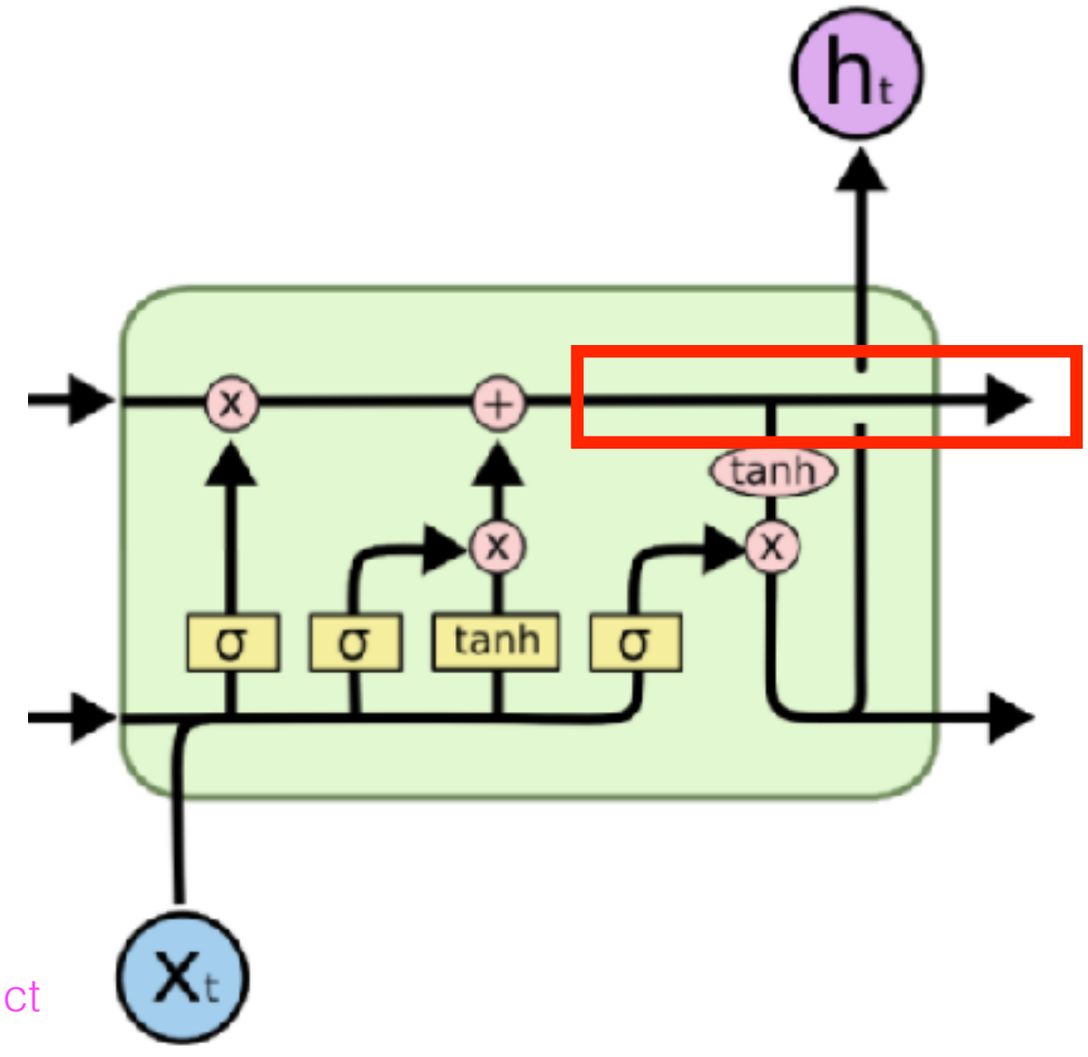
Input gate (but forget some information about the current observation)



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



The memory passes directly to the next state

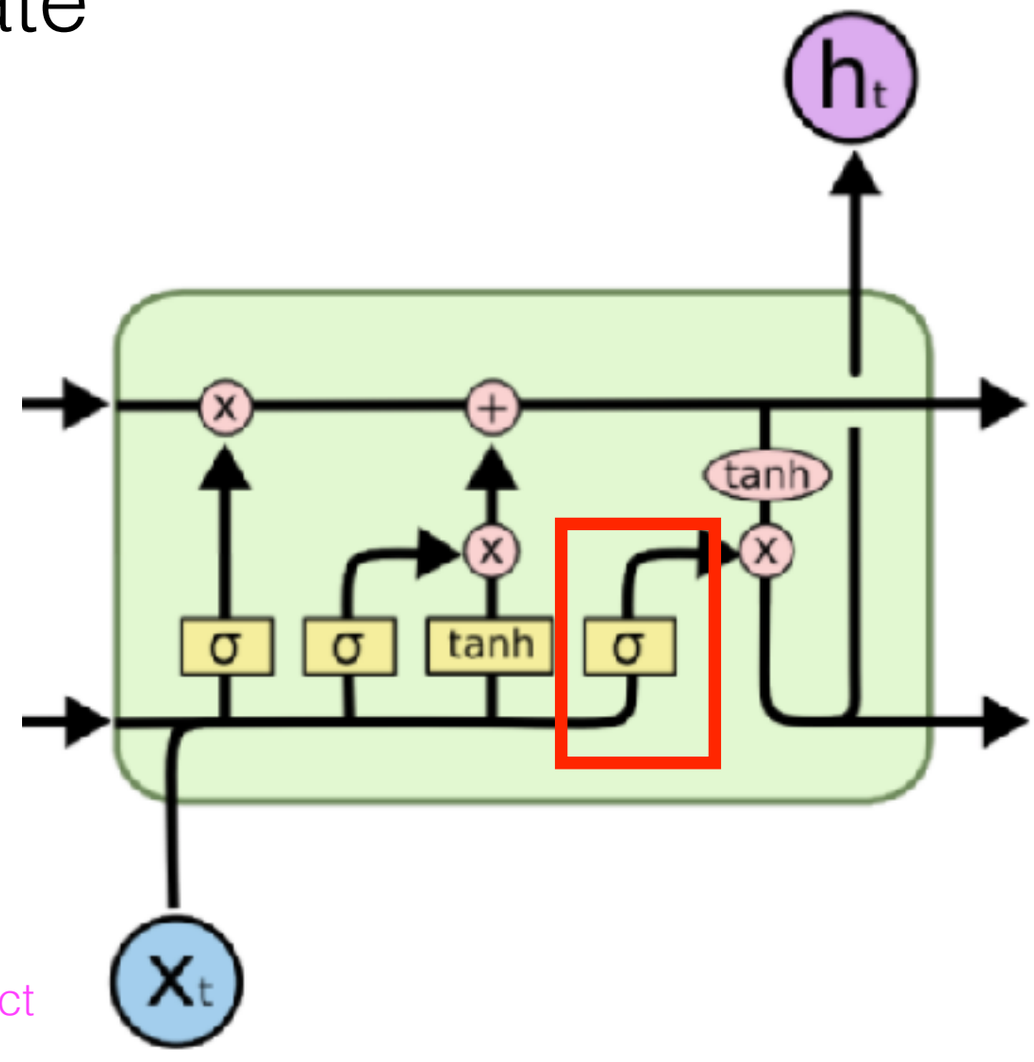


Output gate: forget some information
to send to the hidden state

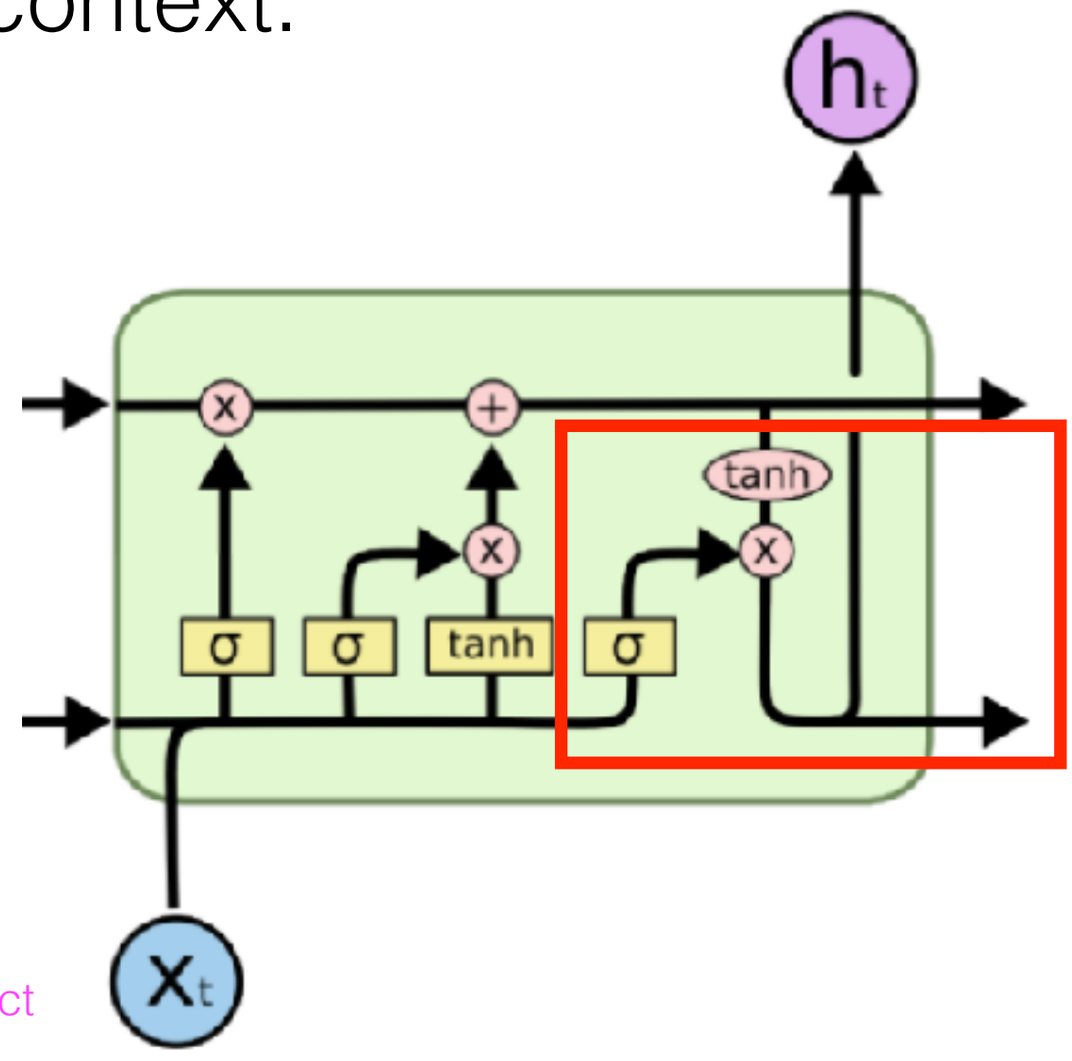
memory

hidden state

elementwise product



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



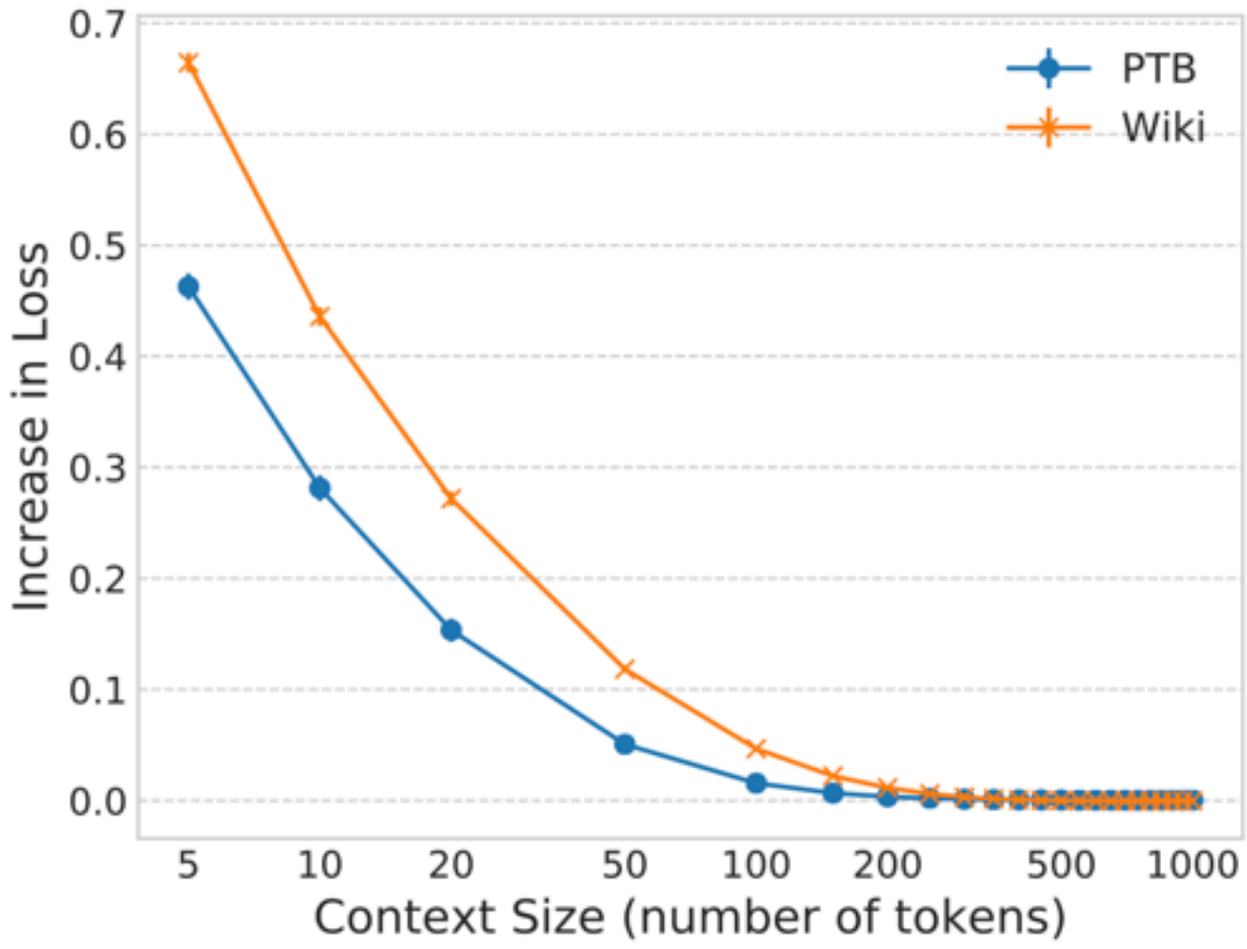
How much context?

- 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

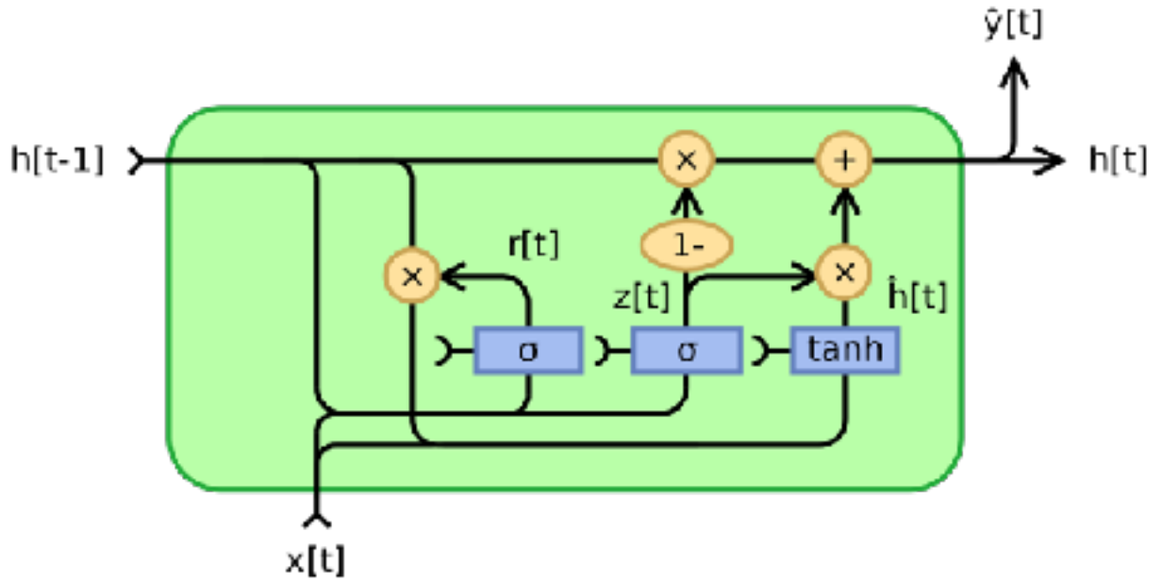
- Ignores word order beyond 50 words



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

GRU

- A gated recurrent unit adopts the same gating mechanism as an LSTM, but reduces the number of parameters to learn.

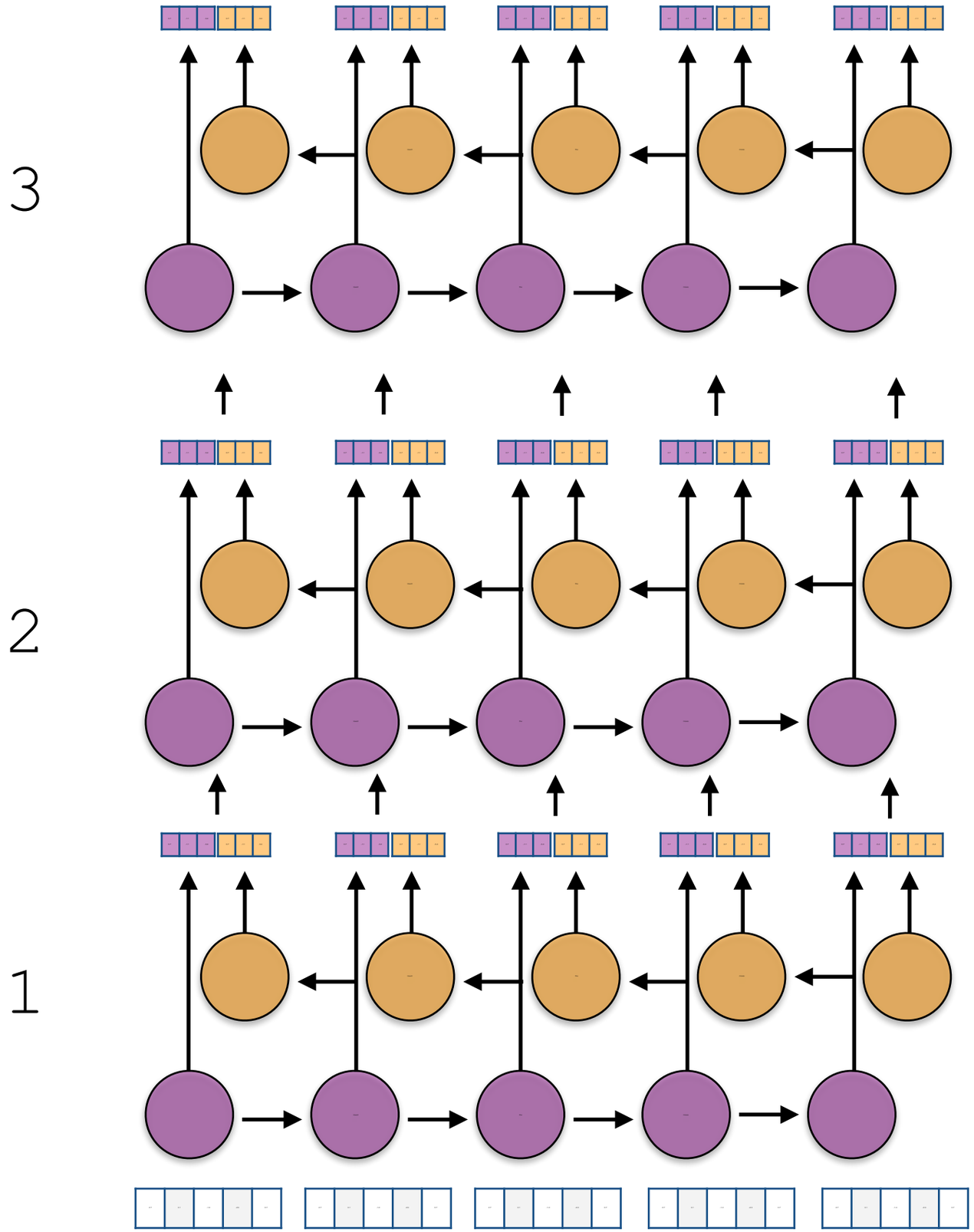


- Only one context vector (not a separate memory and hidden state vector) gets passed between timesteps.

- 2 gates (reset and update) instead of 3.

Stacked RNN

- Multiple RNNs/ LSTMs/GRUs, where the output of one layer becomes the input to the next.



Neural sequence labeling

- Large design space for exploration in these models:
    - RNN/LSTM/GRU
    - Stacking
    - Hidden dimension size
    - Training with dropout and other forms of regularization.

Padding

- Many neural network libraries require each sequence within the same batch to be the same length.

- We can make artificially make this so by padding shorter sequences with a special symbol not otherwise used (e.g. 0)

the	dog	ran		
he	ran	to	the	house
he	stopped			
he	went	inside		

1	3	4		
2	4	5	1	6
2	7			
2	8	9		

word embedding ids

1	3	4	0	0
2	4	5	1	6
2	7	0	0	0
2	8	9	0	0

word embedding ids

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)

Sequence labeling models

model	form	label dependency	rich features?
Hidden Markov Models	$\prod_{i=1}^{N} P(x_i \mid y_i)\, P(y_i \mid y_{i-1})$	Markov assumption	no
MEMM	$\prod_{i=1}^{N} P(y_i \mid y_{i-1}, x, \beta)$	Markov assumption	yes
CRF	$P(y \mid x, \beta)$	pairwise through entire sequence	yes
RNN	$\prod_{i=1}^{N} P(y_i \mid x_{1:i}, \beta)$	none	distributed

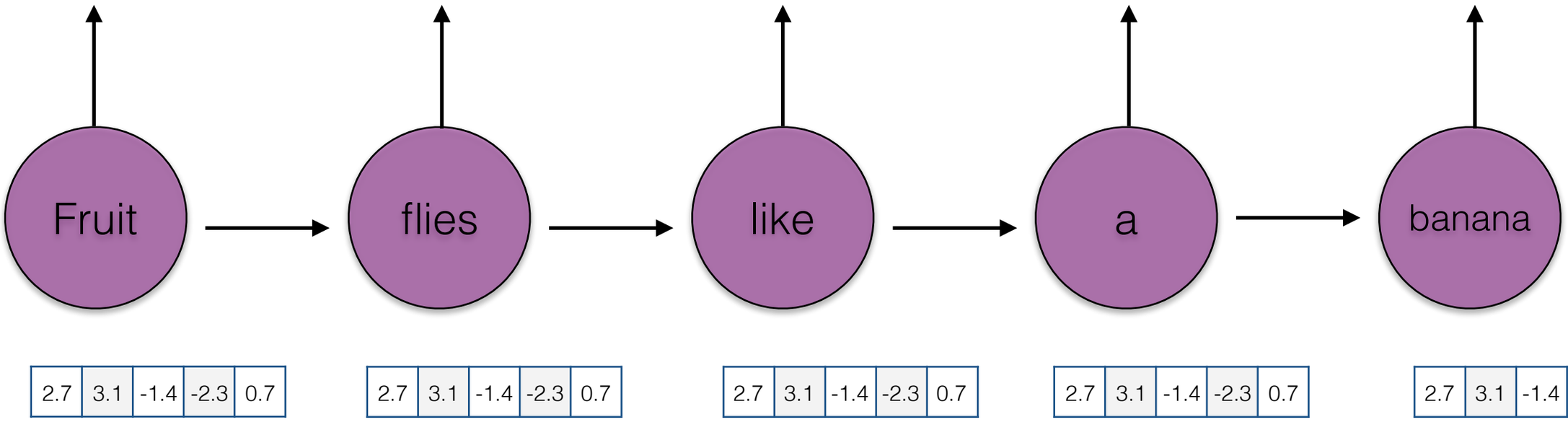
NN

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.



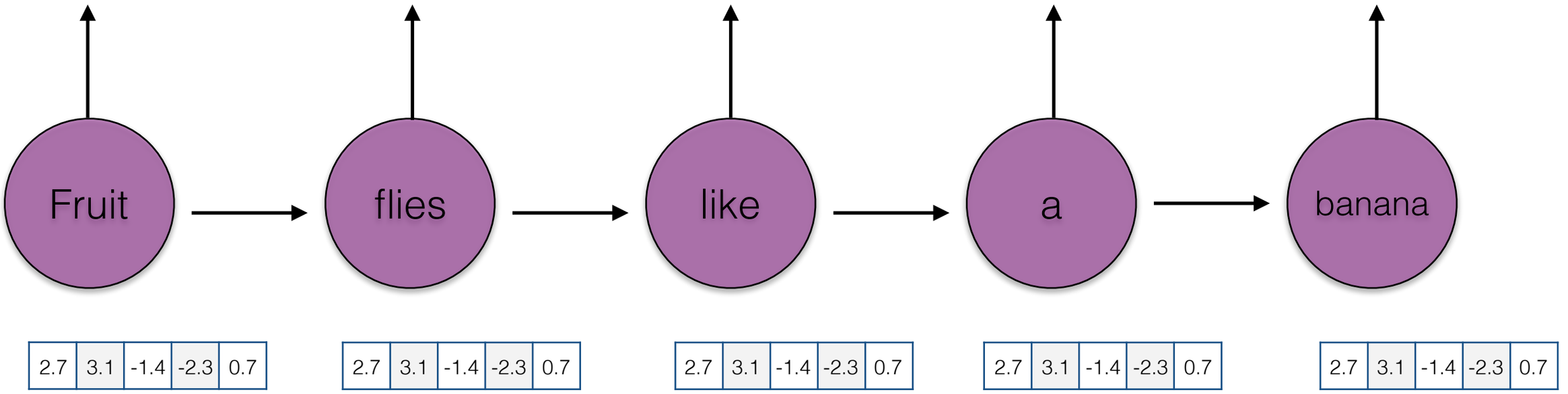
NN

VBZ

VB

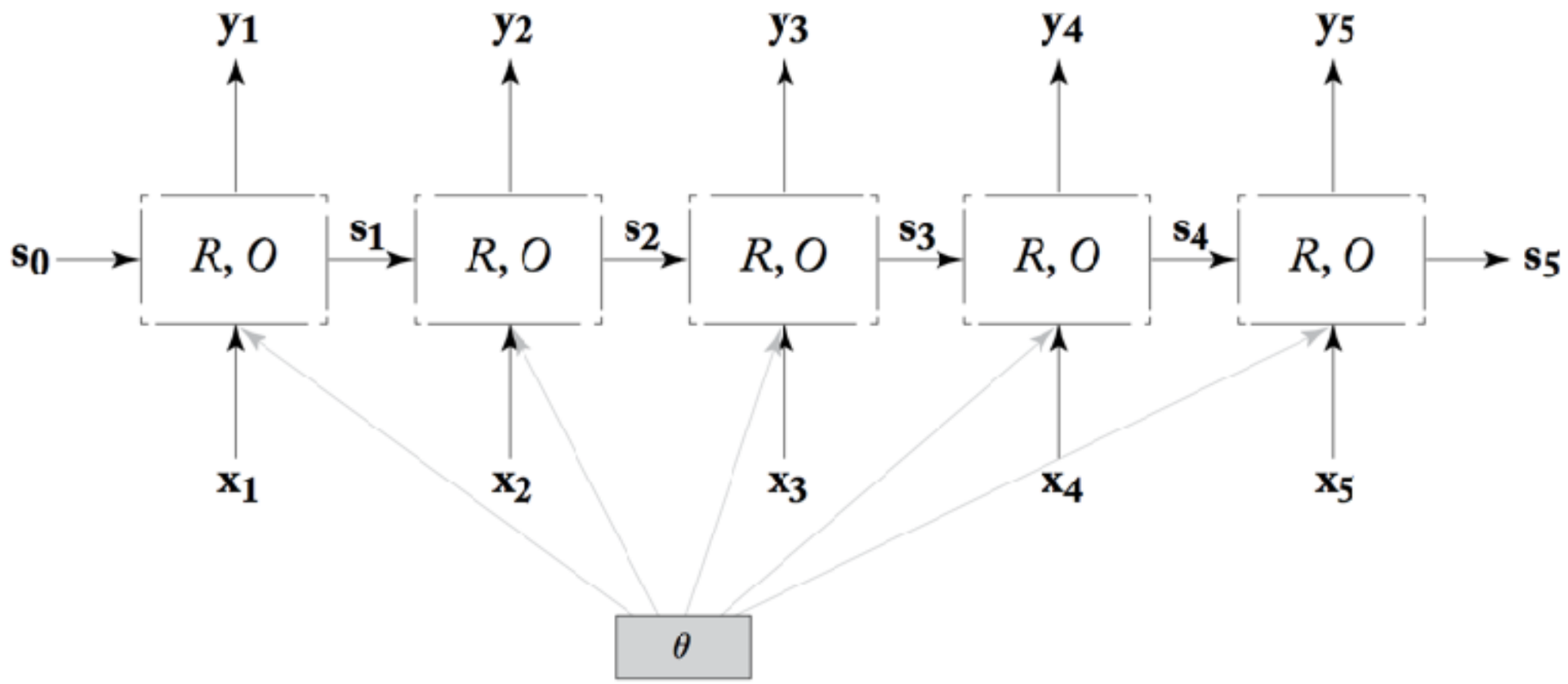
VBZ	0.51
NNS	0.48
JJ	0.01
NN	0
…	…

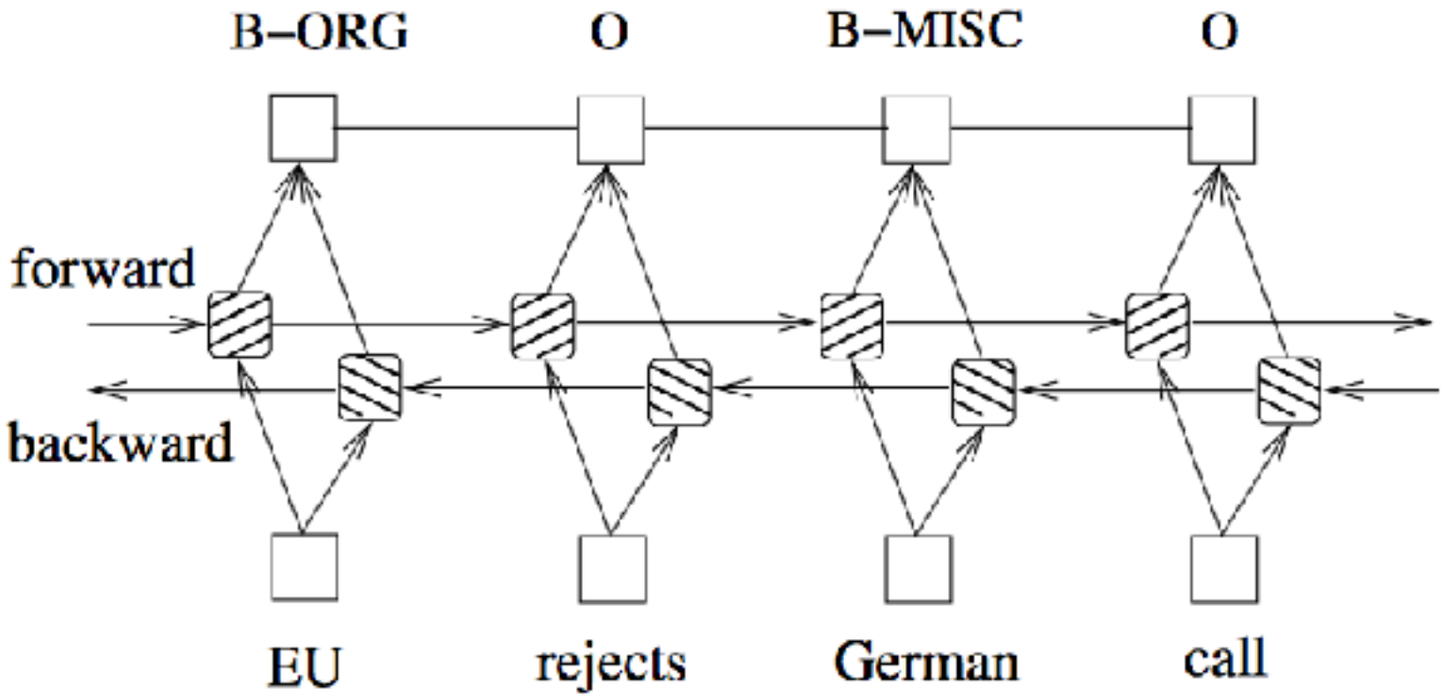
If we knew the categorical choice of VBZ at t_2, P(VB) at t_3 would be much lower.



Recurrent neural network

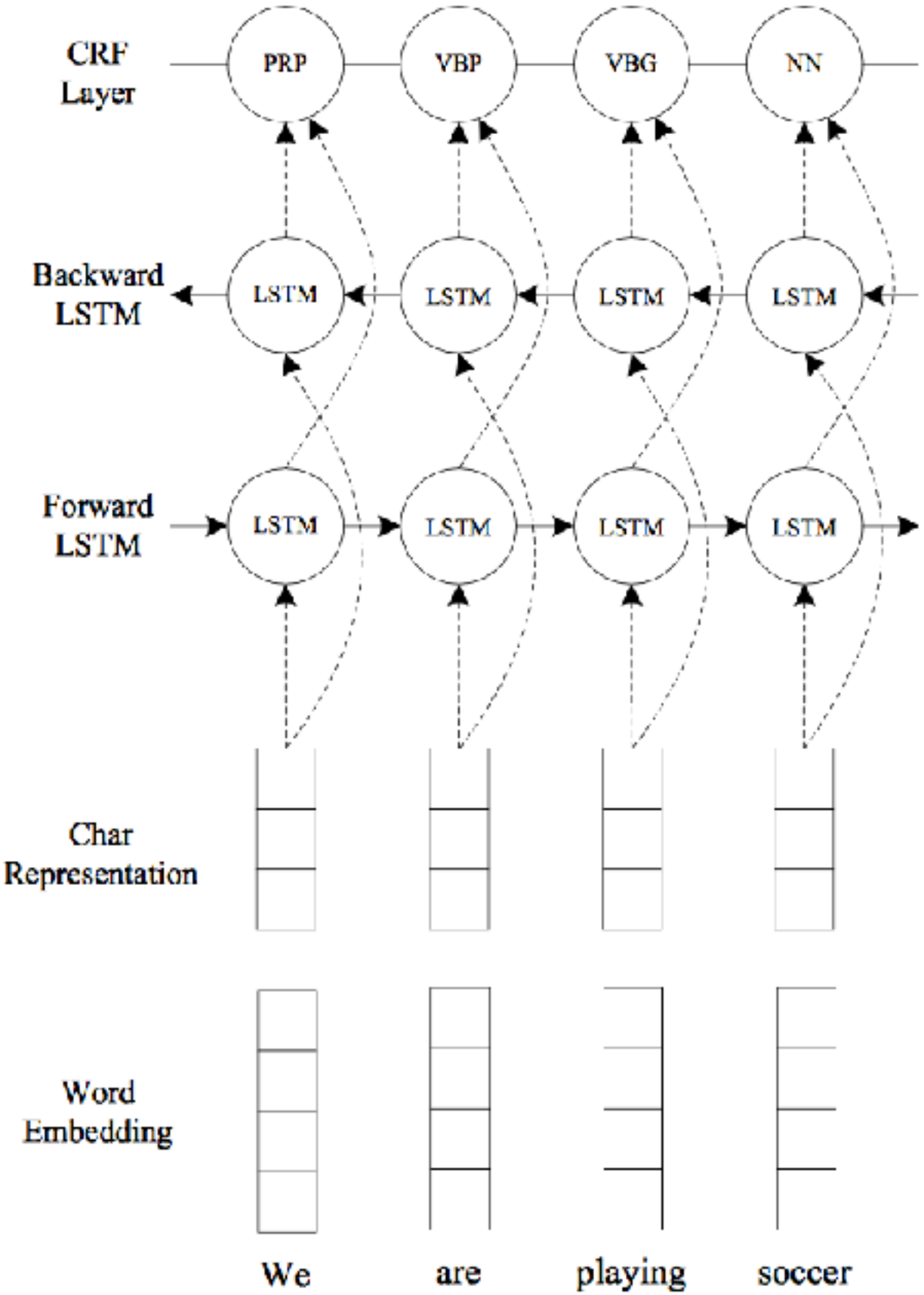
- How could we incorporate nearby label information into a single-direction RNN?





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

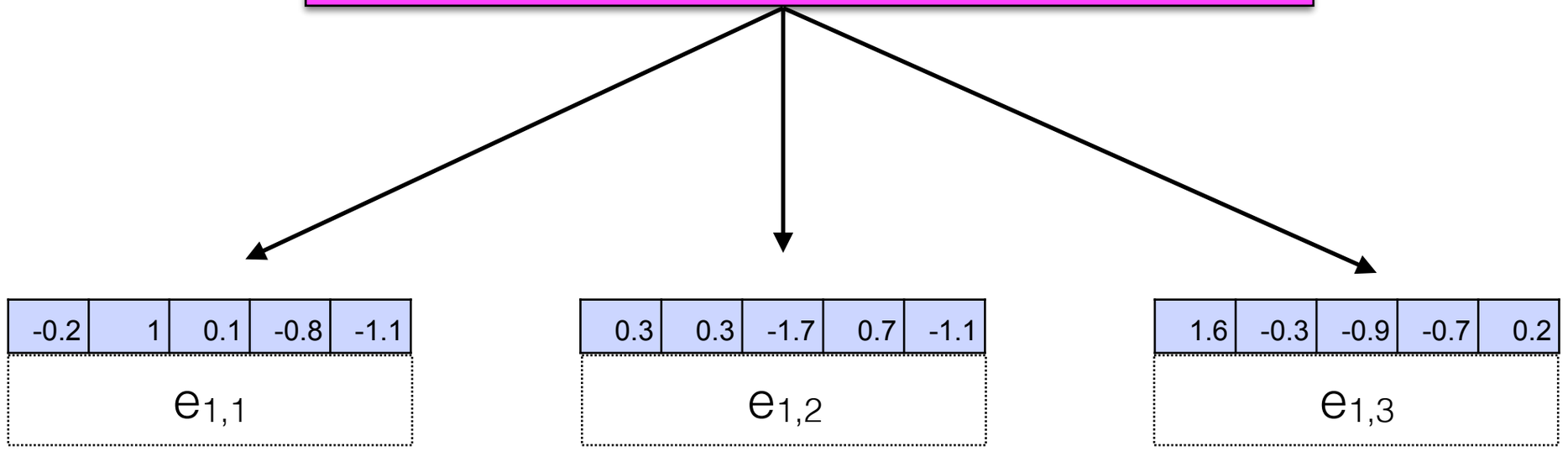
Ma and Hovy (2016), “End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF”

BERT

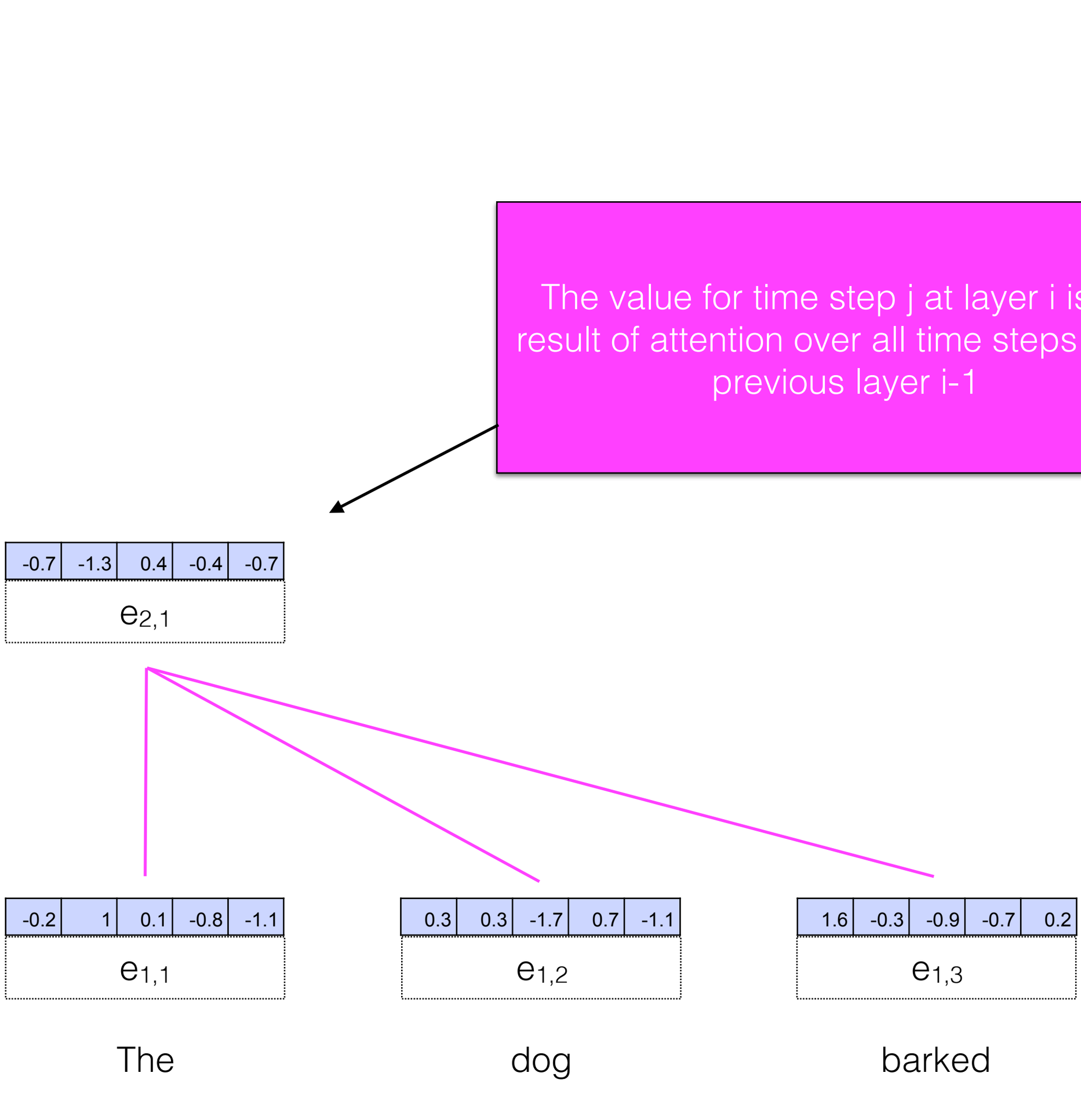
- Transformer-based model (Vaswani et al. 2017) to predict masked word using bidirectional context + next sentence prediction.

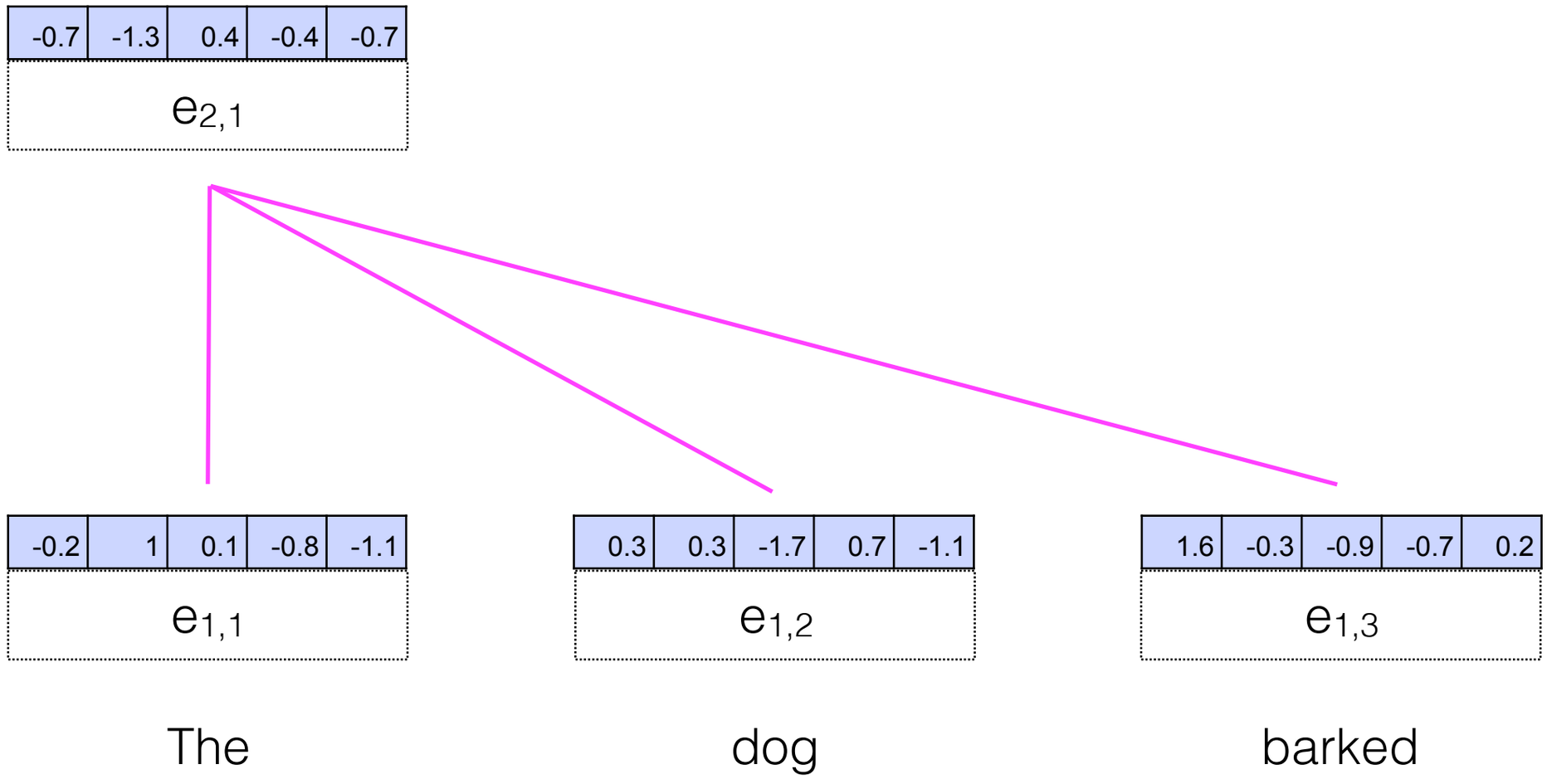
- Generates multiple layers of representations for each token sensitive to its context of use.

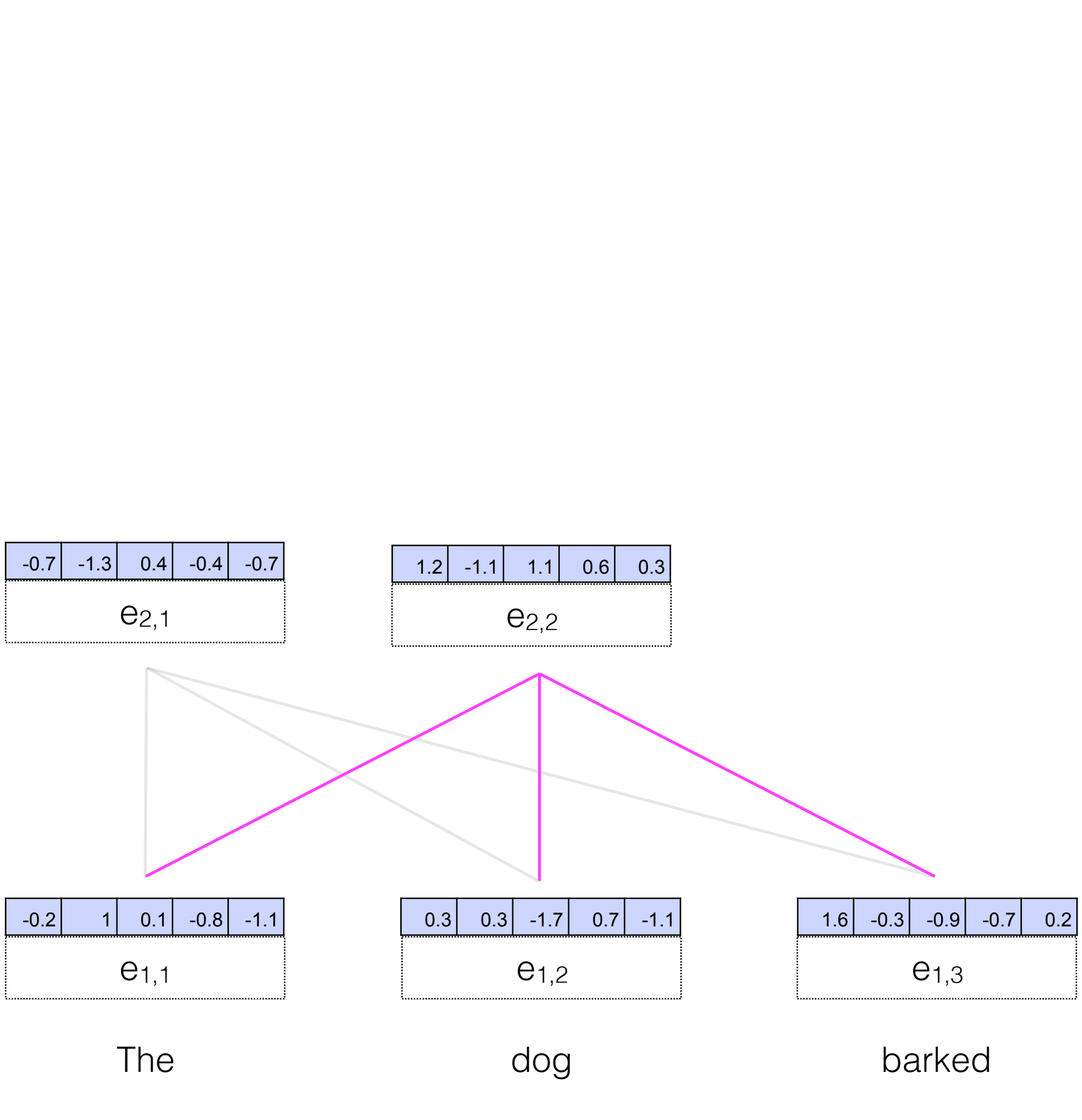
Each token in the input starts out represented by token and position embeddings

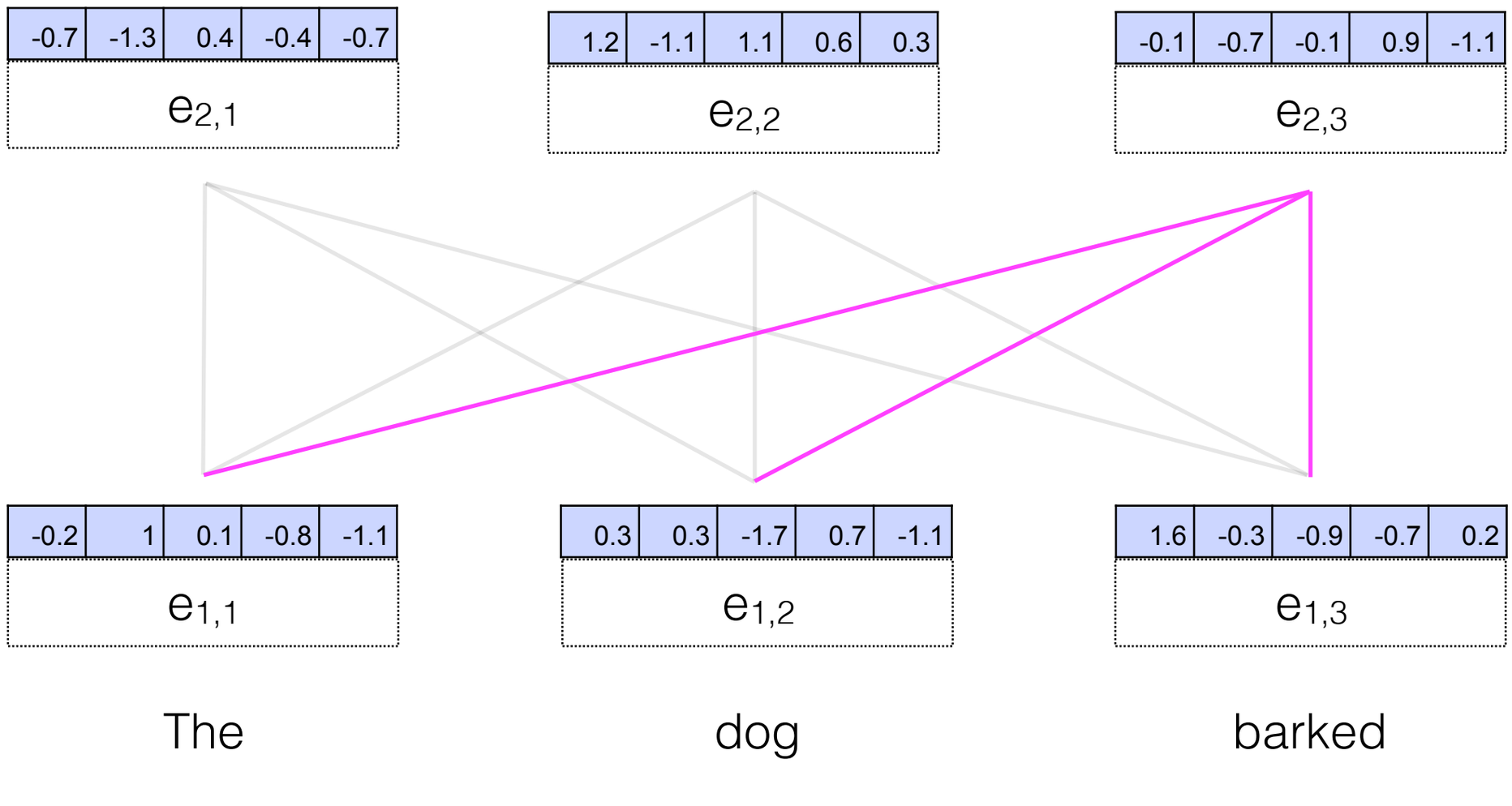


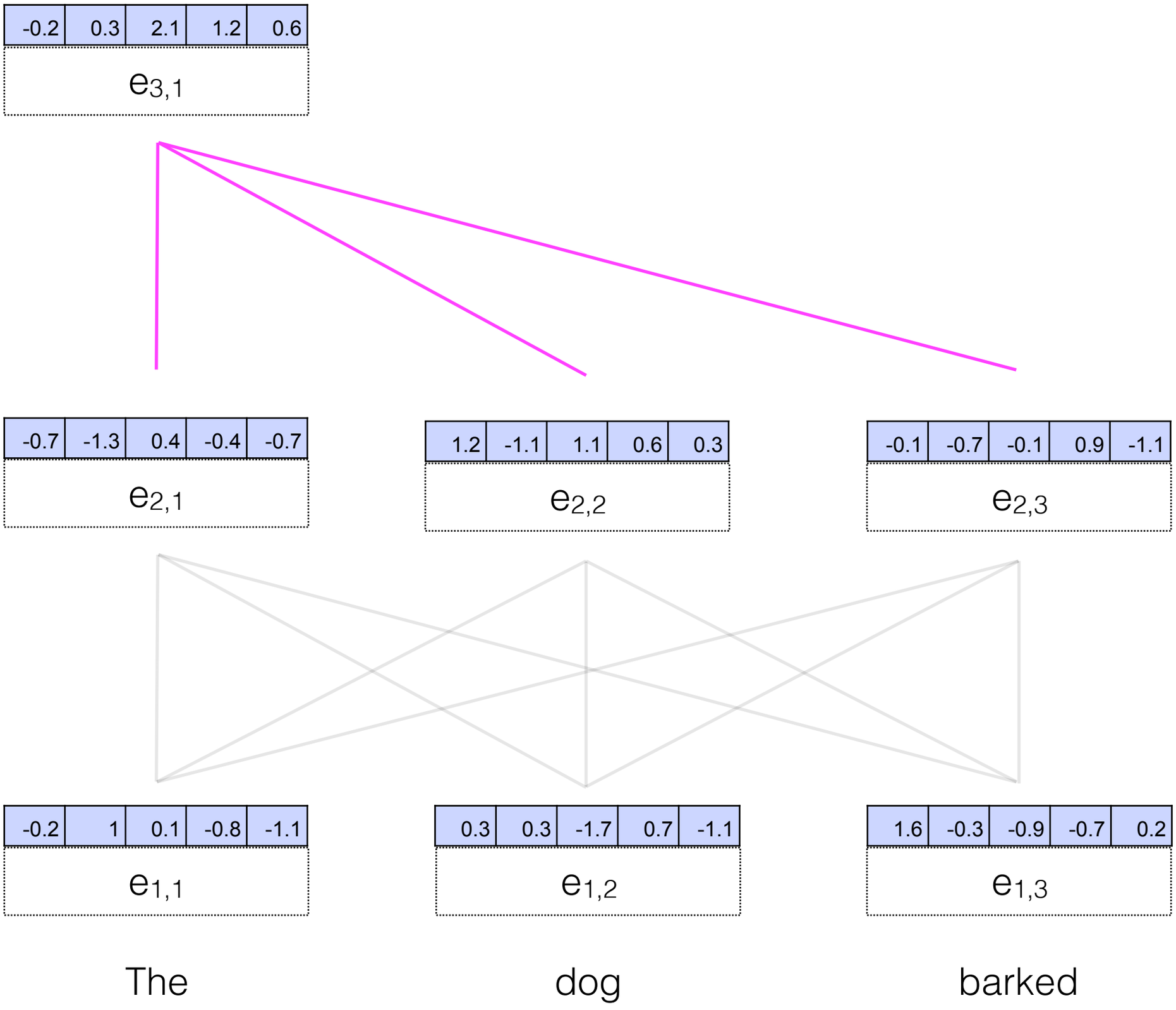
The value for time step j at layer i is the result of attention over all time steps in the previous layer i-1

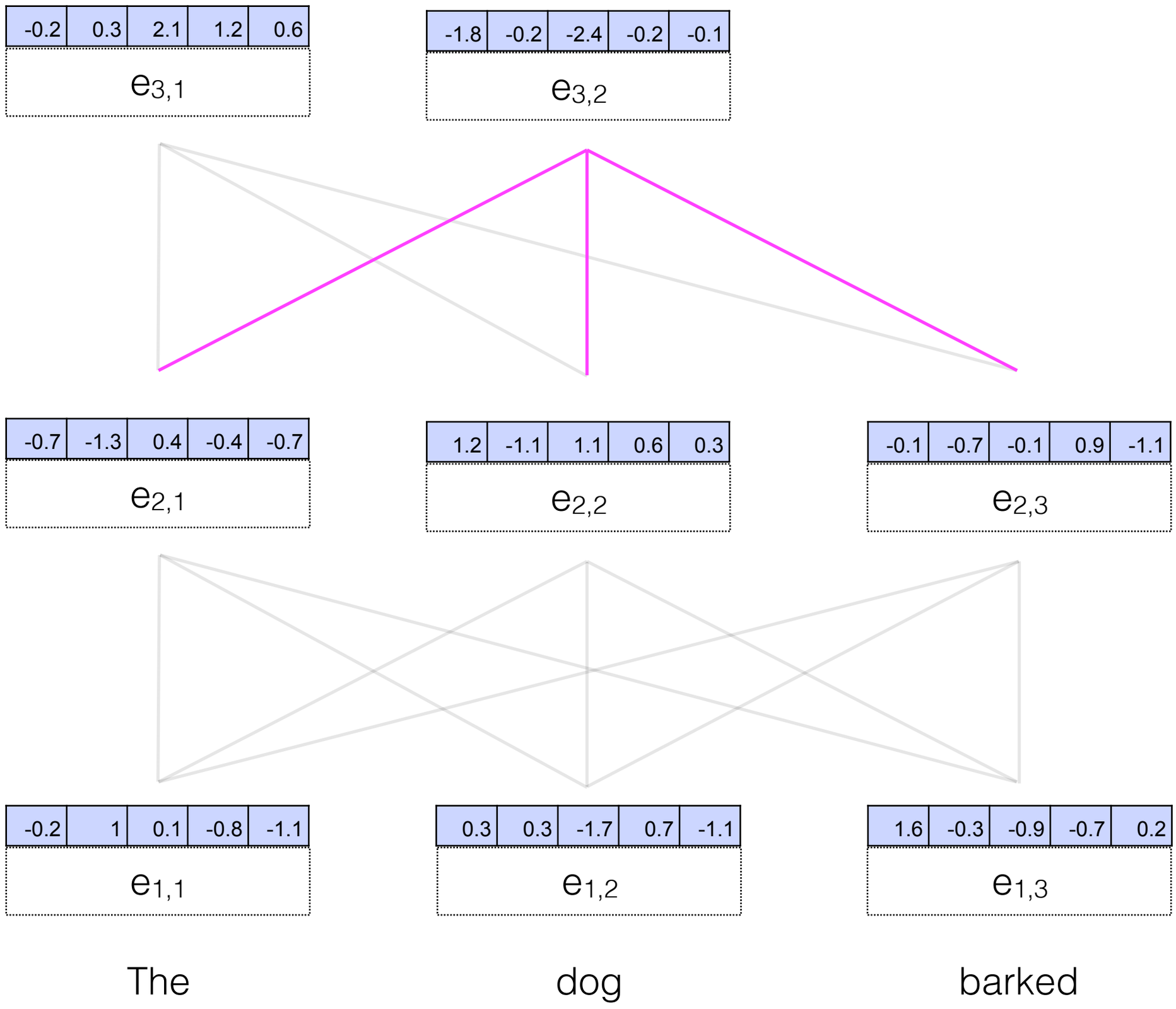



-0.7 -1.3 0.4 -0.4 -0.7
e2,1
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.1 -0.7 -0.1 0.9 -1.1
e2,3
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


-0.2 0.3 2.1 1.2 0.6
e3,1
-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.1 -0.7 -0.1 0.9 -1.1
e2,3
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


-0.2 0.3 2.1 1.2 0.6
e3,1
-1.8 -0.2 -2.4 -0.2 -0.1
e3,2
-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.1 -0.7 -0.1 0.9 -1.1
e2,3
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked

-0.2	0.3	2.1	1.2	0.6

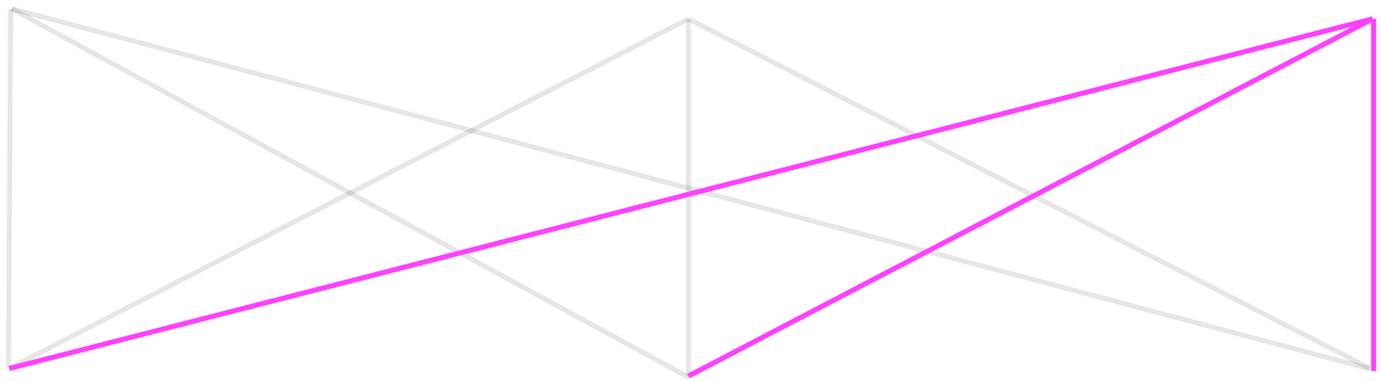
$e_{3,1}$

-1.8	-0.2	-2.4	-0.2	-0.1

$e_{3,2}$

-0.9	-1.5	-0.7	0.9	0.2

$e_{3,3}$



-0.7	-1.3	0.4	-0.4	-0.7

$e_{2,1}$

1.2	-1.1	1.1	0.6	0.3

$e_{2,2}$

-0.1	-0.7	-0.1	0.9	-1.1

$e_{2,3}$

-0.2	1	0.1	-0.8	-1.1

$e_{1,1}$

0.3	0.3	-1.7	0.7	-1.1

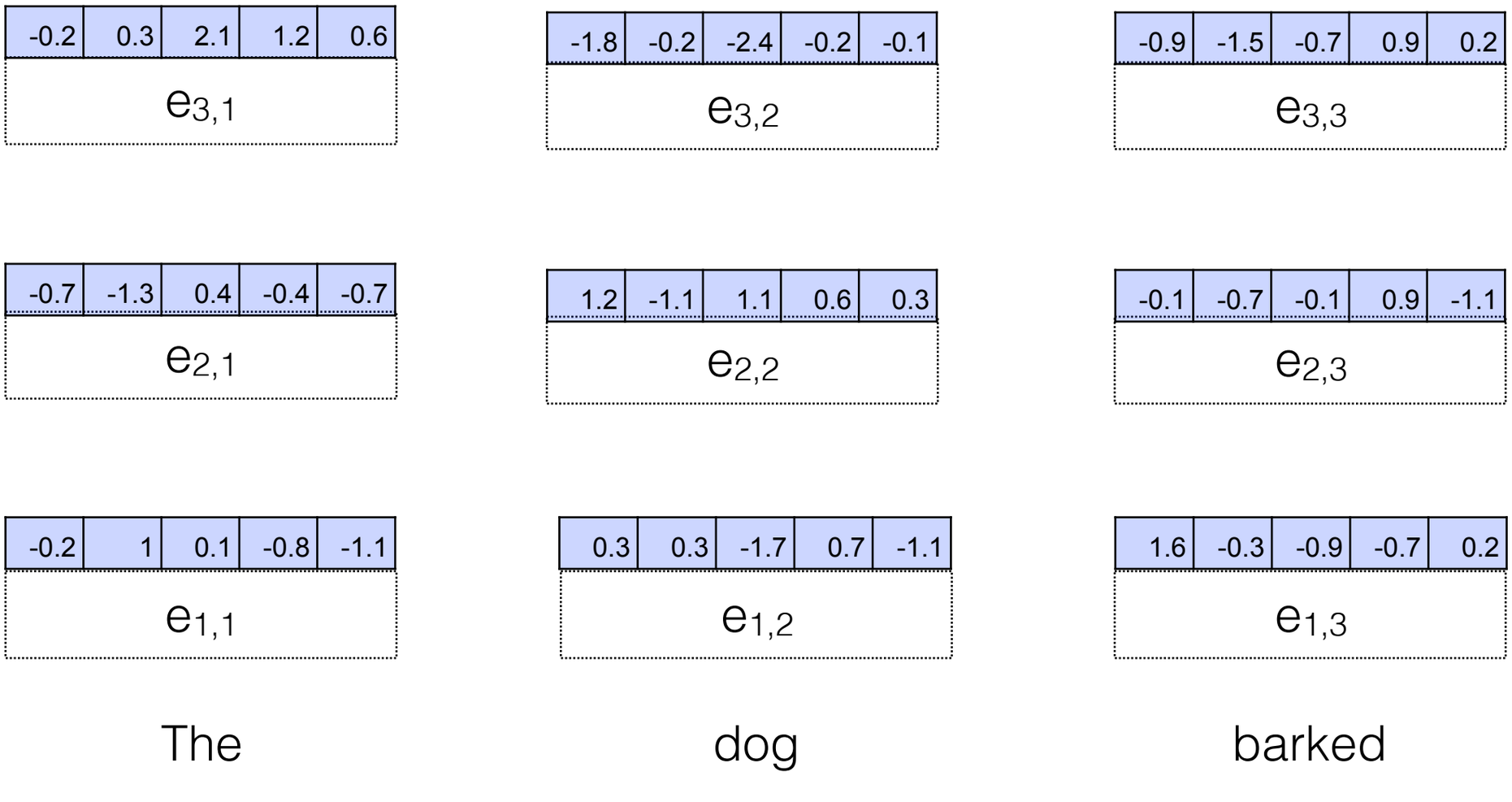
$e_{1,2}$

1.6	-0.3	-0.9	-0.7	0.2

$e_{1,3}$

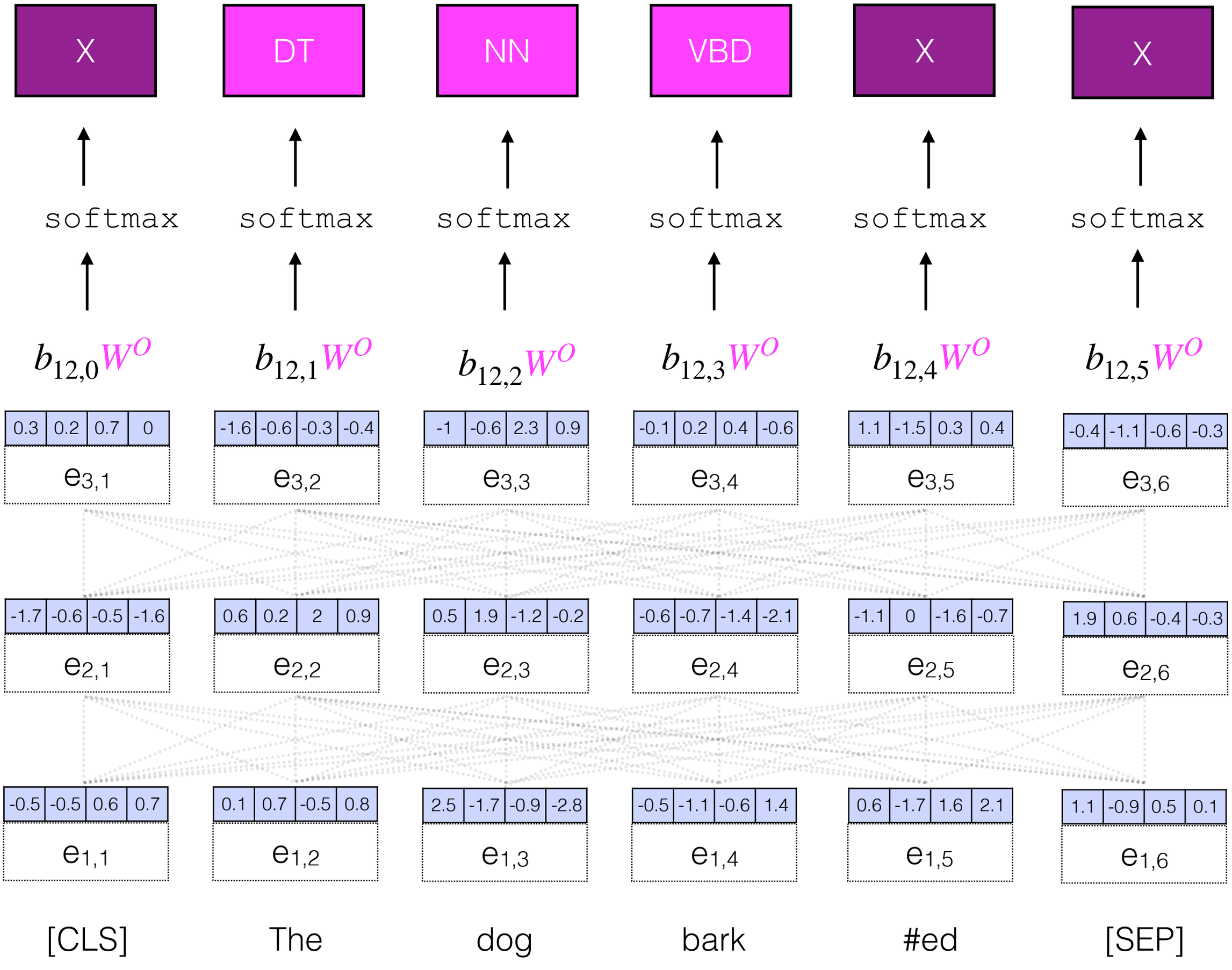
The dog barked

At the end of this process, we have one representation for each layer for each token



BERT

- BERT can be used not only as a language model to generate contextualized word representations, but also as a predictive model whose parameters are fine-tuned to a task.


X
DT
NN
VBD
X
X
softmax
softmax
softmax
softmax
softmax
softmax
$b_{12,0}W^O$
$b_{12,1}W^O$
$b_{12,2}W^O$
$b_{12,3}W^O$
$b_{12,4}W^O$
$b_{12,5}W^O$
0.3 0.2 0.7 0
-1.6 -0.6 -0.3 -0.4
-1 -0.6 2.3 0.9
-0.1 0.2 0.4 -0.6
1.1 -1.5 0.3 0.4
-0.4 -1.1 -0.6 -0.3
$e_{3,1}$
$e_{3,2}$
$e_{3,3}$
$e_{3,4}$
$e_{3,5}$
$e_{3,6}$
-1.7 -0.6 -0.5 -1.6
0.6 0.2 2 0.9
0.5 1.9 -1.2 -0.2
-0.6 -0.7 -1.4 -2.1
-1.1 0 -1.6 -0.7
1.9 0.6 -0.4 -0.3
$e_{2,1}$
$e_{2,2}$
$e_{2,3}$
$e_{2,4}$
$e_{2,5}$
$e_{2,6}$
-0.5 -0.5 0.6 0.7
0.1 0.7 -0.5 0.8
2.5 -1.7 -0.9 -2.8
-0.5 -1.1 -0.6 1.4
0.6 -1.7 1.6 2.1
1.1 -0.9 0.5 0.1
$e_{1,1}$
$e_{1,2}$
$e_{1,3}$
$e_{1,4}$
$e_{1,5}$
$e_{1,6}$
[CLS]
The
dog
bark
#ed
[SEP]

neutral

softmax

$b_{12,0}W^O$

0.3	0.2	0.7	0
$e_{3,1}$			

-1.6	-0.6	-0.3	-0.4
$e_{3,2}$			

-1	-0.6	2.3	0.9
$e_{3,3}$			

-0.1	0.2	0.4	-0.6
$e_{3,4}$			

1.1	-1.5	0.3	0.4
$e_{3,5}$			

-0.4	-1.1	-0.6	-0.3
$e_{3,6}$			

-1.7	-0.6	-0.5	-1.6
$e_{2,1}$			

0.6	0.2	2	0.9
$e_{2,2}$			

0.5	1.9	-1.2	-0.2
$e_{2,3}$			

-0.6	-0.7	-1.4	-2.1
$e_{2,4}$			

-1.1	0	-1.6	-0.7
$e_{2,5}$			

1.9	0.6	-0.4	-0.3
$e_{2,6}$			

-0.5	-0.5	0.6	0.7
$e_{1,1}$			

0.1	0.7	-0.5	0.8
$e_{1,2}$			

2.5	-1.7	-0.9	-2.8
$e_{1,3}$			

-0.5	-1.1	-0.6	1.4
$e_{1,4}$			

0.6	-1.7	1.6	2.1
$e_{1,5}$			

1.1	-0.9	0.5	0.1
$e_{1,6}$			

[CLS] The dog bark #ed [SEP]

BERT

- Pre-training: train BERT through masked language modeling and next-sentence prediction to learn the parameters of BERT layers. Trained on Wikipedia + BookCorpus.

- Task fine-tuning: add additional linear transformation + softmax to get distribution over output space. Trained on annotated data.