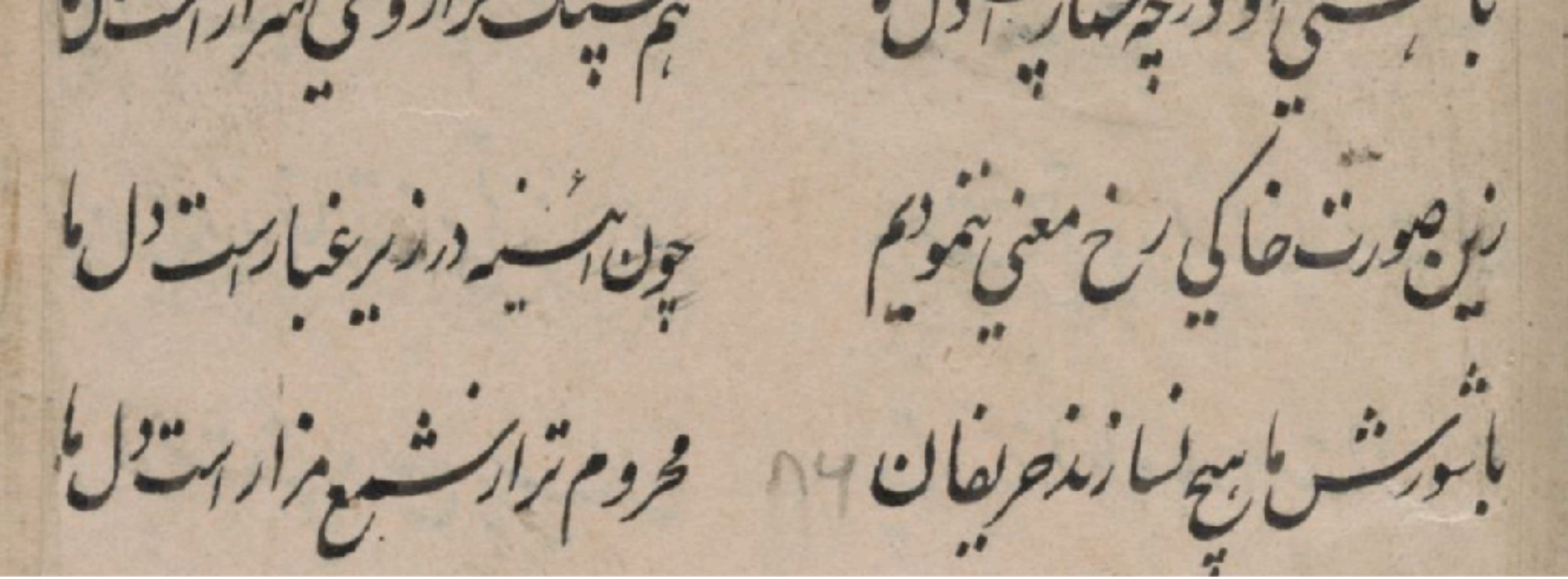


Natural Language Processing

Info 159/259

Lecture 9: Embeddings 2 (Feb 18, 2020)

David Bamman, UC Berkeley

Distributed representation

- Vector representation that encodes information about the distribution of contexts a word appears in

- Words that appear in similar contexts have similar representations (and similar meanings, by the distributional hypothesis).

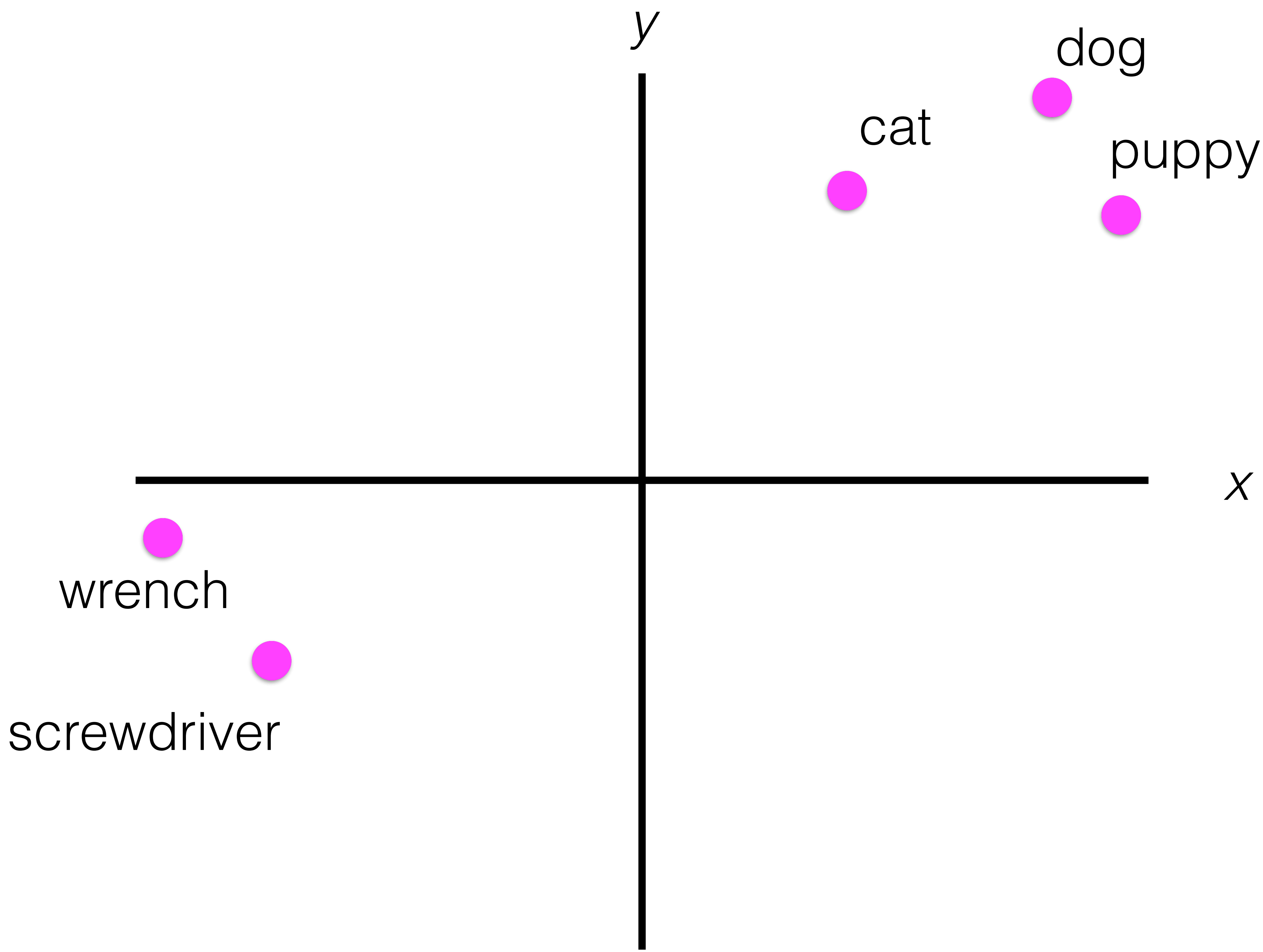
Dense vectors from prediction

- Learning low-dimensional representations of words by framing a predicting task: using context to predict words in a surrounding window

- Transform this into a supervised prediction problem; similar to language modeling but we're ignoring order within the context window

	1	2	3	4	…	50
the	0.418	0.24968	-0.41242	0.1217	…	-0.17862
,	0.013441	0.23682	-0.16899	0.40951	…	-0.55641
.	0.15164	0.30177	-0.16763	0.17684	…	-0.31086
of	0.70853	0.57088	-0.4716	0.18048	…	-0.52393
to	0.68047	-0.039263	0.30186	-0.17792	…	0.13228
…	…	…	…	…	…	…
chanty	0.23204	0.025672	-0.70699	-0.04547	…	0.34108
kronik	-0.60921	-0.67218	0.23521	-0.11195	…	0.85632
rolonda	-0.51181	0.058706	1.0913	-0.55163	…	0.079711
zsombor	-0.75898	-0.47426	0.4737	0.7725	…	0.84014
sandberger	0.072617	-0.51393	0.4728	-0.52202	…	0.23096

https://nlp.stanford.edu/projects/glove/


y
dog
cat
puppy
x
wrench
screwdriver

Word embeddings

- Pre-trained word embeddings great for words that appear frequently in data

- Unseen words are treated as UNKs and assigned zero or random vectors; everything unseen is assigned the same representation.

- supercalifragilisticexpialidocious
    - super, superior, supernatural
    - adventurous, fabulous, infamous
- supercalifragilisticexpialidociously
    - quickly, sadly, perfectly

Agglutinative languages

Muvaffakiyetsizleş(-mek)	(To) become unsuccessful
Muvaffakiyetsizleştir(-mek)	(To) make one unsuccessful
Muvaffakiyetsizleştirici	Maker of unsuccessful ones
Muvaffakiyetsizleştiricileş(-mek)	(To) become a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştir(-mek)	(To) make one a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriver(-mek)	(To) easily/quickly make one a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriverebil(-mek)	(To) be able to make one easily/quickly a maker of unsuccessful ones
Muvaffakiyetsizleştiricileştiriveremeyebileceklerimizdenmişsinizcesine	As though you happen to have been from among those whom we will not be able to easily/quickly make a maker of unsuccessful ones

Infinitive
danser

Past Participle
dansé

Gerund
dansant

Imperative
danse (tu)
dansons (nous)
dansez (vous)

Present
je danse
tu danses
il/elle danse
nous dansons
vous dansez
ils/elles dansent

Present Perfect
j'ai dansé
tu as dansé
il/elle a dansé
nous avons dansé
vous avez dansé
ils/elles ont dansé

Imperfect
je dansais
tu dansais
il/elle dansait
nous dansions
vous dansiez
ils/elles dansaient

Future
je danserai
tu danseras
il/elle dansera
nous danserons
vous danserez
ils/elles danseront

Conditional
je danserais
tu danserais
il/elle danserait
nous danserions
vous danseriez
ils/elles danseraient

Past Historic
je dansai
tu dansas
il/elle dansa
nous dansâmes
vous dansâtes
ils/elles dansèrent

Pluperfect
j'avais dansé
tu avais dansé
il/elle avait dansé
nous avions dansé
vous aviez dansé
ils/elles avaient dansé

Future Perfect
j'aurai dansé
tu auras dansé
il/elle aura dansé
nous aurons dansé
vous aurez dansé
ils/elles auront dansé

Past Anterior
j'eus dansé
tu eus dansé
il/elle eut dansé
nous eûmes dansé
vous eûtes dansé
ils/elles eurent dansé

Conditional Perfect
j'aurais dansé
tu aurais dansé
il/elle aurait dansé
nous aurions dansé
vous auriez dansé
ils/elles auraient dansé

Present Subjunctive
je danse
tu danses
il/elle danse
nous dansions
vous dansiez
ils/elles dansent

Imperfect Subjunctive
je dansasse
tu dansasses
il/elle dansât
nous dansassions
vous dansassiez
ils/elles dansassent

Present Perfect Subjunctive
j'aie dansé
tu aies dansé
il/elle ait dansé
nous ayons dansé
vous ayez dansé
ils/elles aient dansé

Pluperfect Subjunctive
j'eusse dansé
tu eusses dansé
il/elle eût dansé
nous eussions dansé
vous eussiez dansé
ils/elles eussent dansé

Shared structure

Even in languages like English that are not agglutinative and aren't highly inflected, words share important structure.

Even if we never see the word "unfriendly" in our data, we should be able to reason about it as:
un + friend + ly

friend
friended
friendless
friendly
friendship
unfriend
unfriendly

Subword models

- Rather than learning a single representation for each word type w, learn representations z for the set of ngrams $\mathcal{G}_w$ that comprise it [Bojanowski et al. 2017]

- The word itself is included among the ngrams (no matter its length).

- A word representation is the sum of those ngrams

$$w = \sum_{g \in \mathcal{G}_w} z_g$$

FastText

e(where) =

e(<wh)
+ e(whe)
+ e(her)
+ e(ere)
+ e(re>)

3-grams

+ e(<whe)
+ e(wher)
+ e(here)
+ e(ere>)

4-grams

+ e(<wher)
+ e(where)
+ e(here>)

5-grams

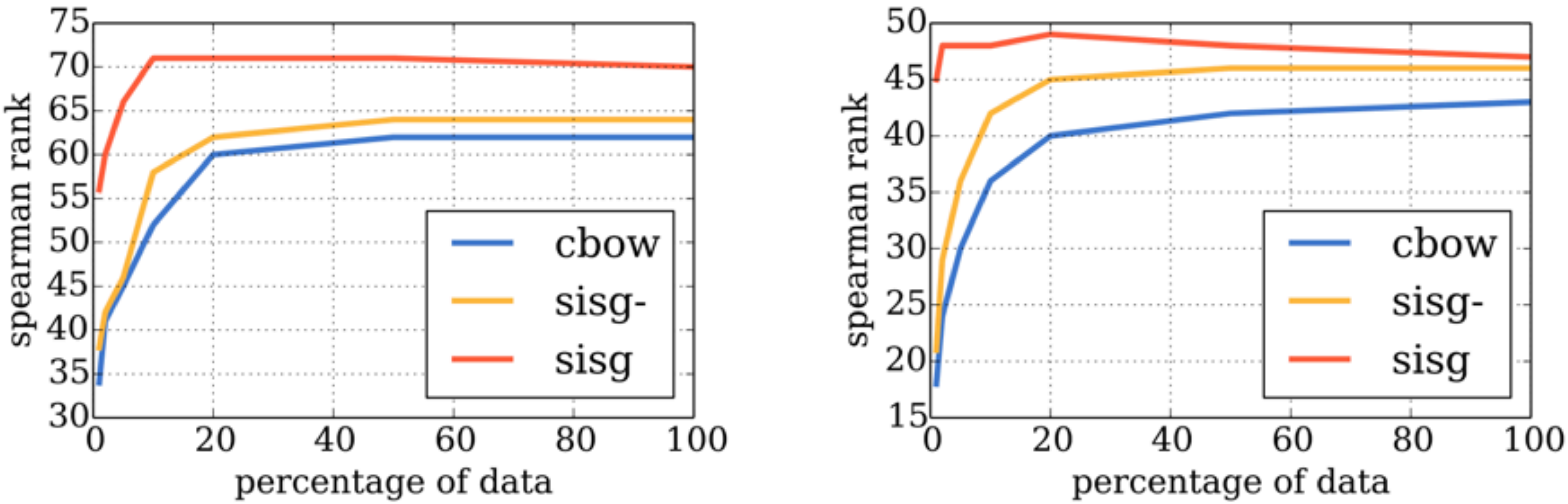
+ e(<where)
+ e(where>)

6-grams

+ e(<where>)

word

e(*) = embedding for *



(a) DE-GUR350

(b) EN-RW

1% = ~20M tokens

100% = ~1B tokens

- Subword models need less data to get comparable performance.

Low-dimensional distributed representations

- Low-dimensional, dense word representations are extraordinarily powerful (and are arguably responsible for much of gains that neural network models have in NLP).

- Lets your representation of the input share statistical strength with words that behave similarly in terms of their distributional properties (often synonyms or words that belong to the same class).

Using dense vectors

1. Trained word embeddings on a large collection (T tokens) of unlabeled text (Wikipedia, news, Twitter, books), preferably in the domain you want to use them for.

2. Use those pre-trained embeddings in a predictive model with S labeled examples

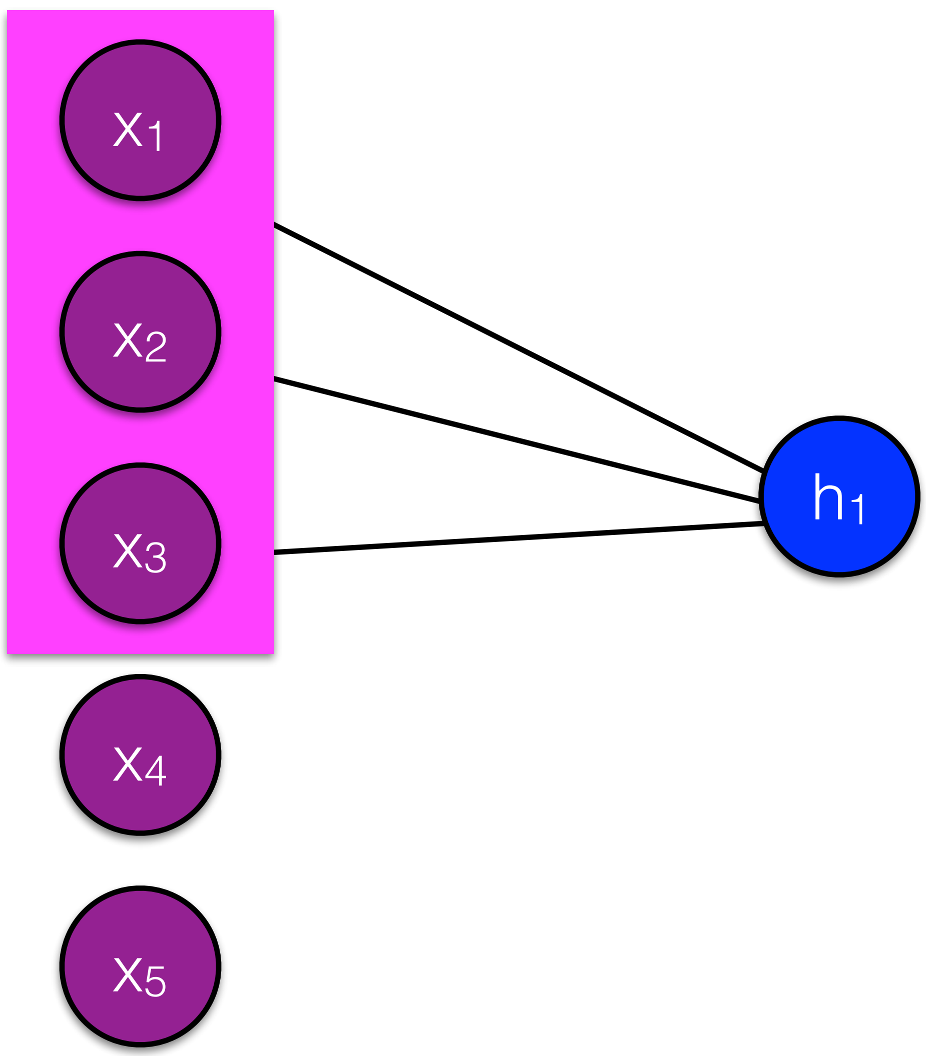
$T >> S$

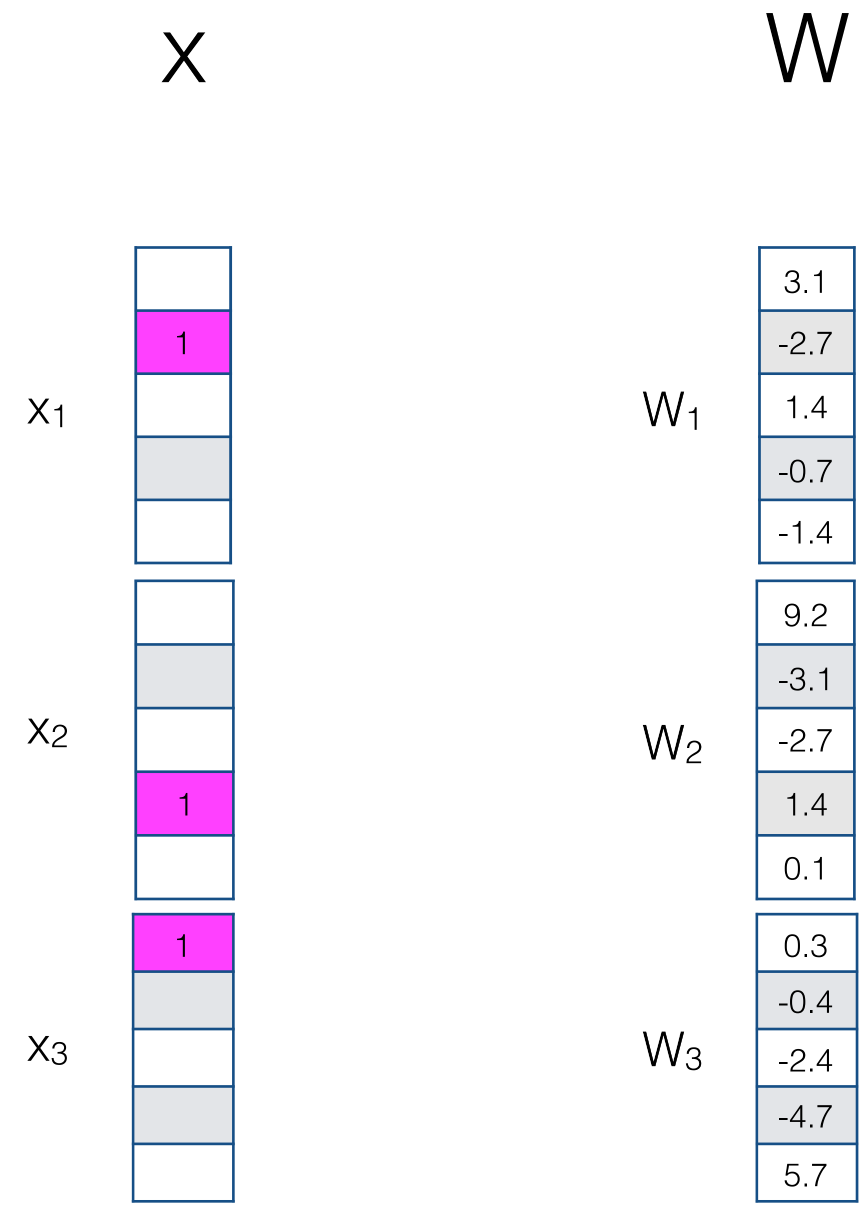
Using dense vectors

- In neural models (CNNs, RNNs, LM), replace the V-dimensional sparse vector with the much smaller K-dimensional dense one.

- Can also take the derivative of the loss function with respect to those representations to optimize for a particular task.

CNNs

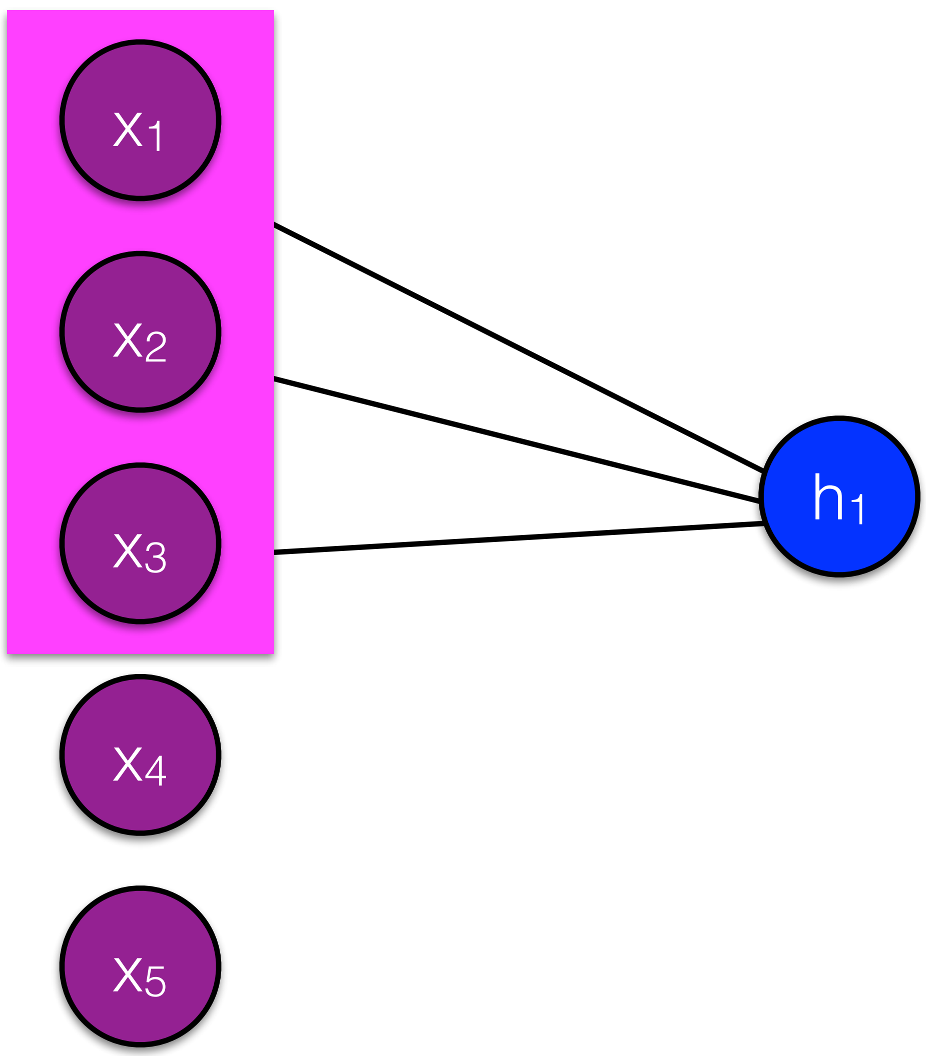




$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

In a CNN, we just replace the input one-hot representation with the embedding

CNNs



	X		W
x_1	0.4	W_1	3.1
	0.8		-2.7
	1.2		1.4
	-1.3		-0.7
	0.4		-1.4
x_2	0.2	W_2	9.2
	-5.3		-3.1
	-1.2		-2.7
	5.3		1.4
	0.4		0.1
x_3	2.6	W_3	0.3
	2.7		-0.4
	-3.2		-2.4
	6.2		-4.7
	1.9		5.7

For dense input vectors (e.g., embeddings), full dot product

In a CNN, we just replace the input one-hot representation with the embedding

$$h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3)$$

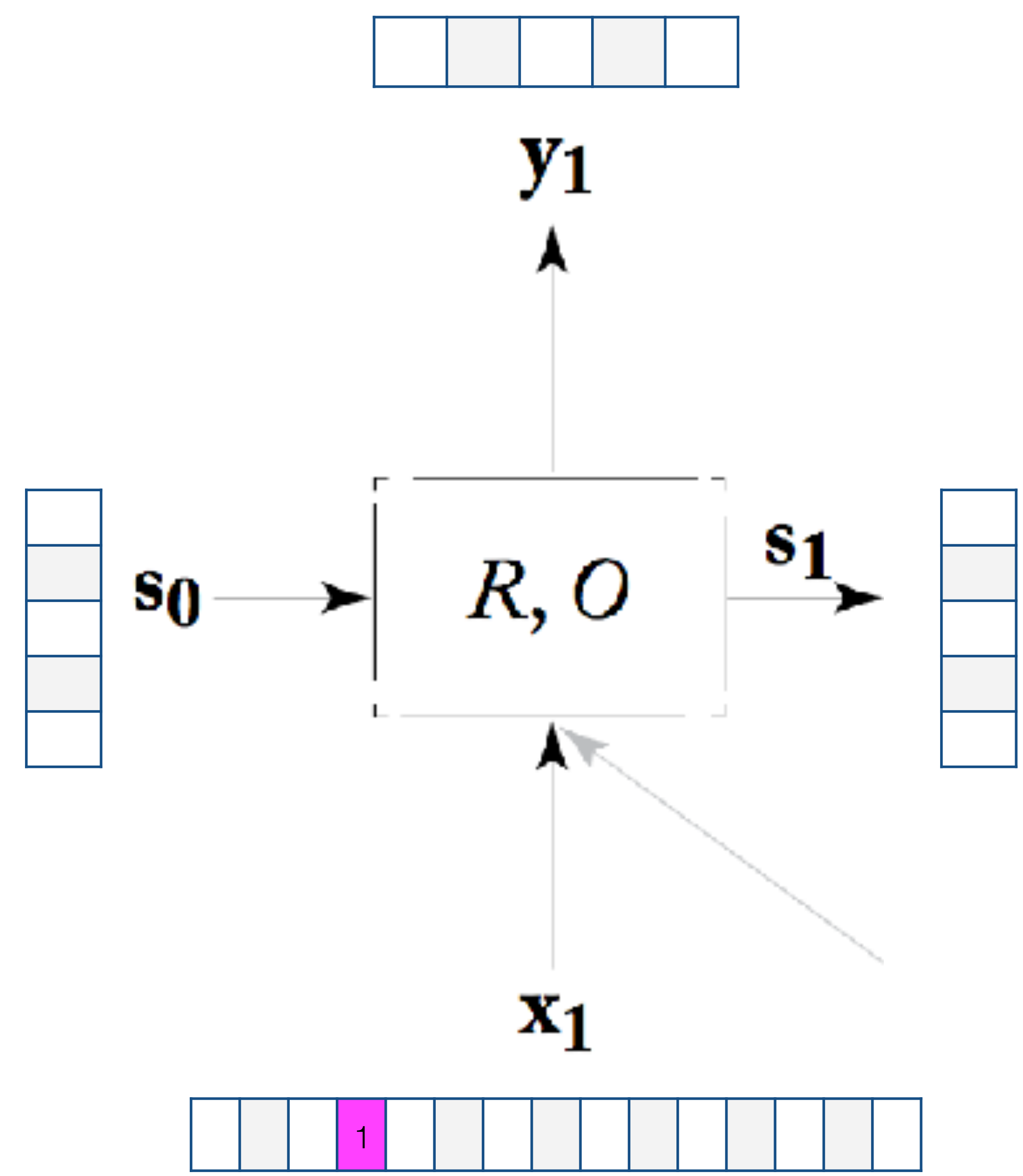
Recurrent neural network

$$s_i = R(x_i, s_{i-1})$$

R is some function of the current input and previous state

$$y_i = O(s_i)$$

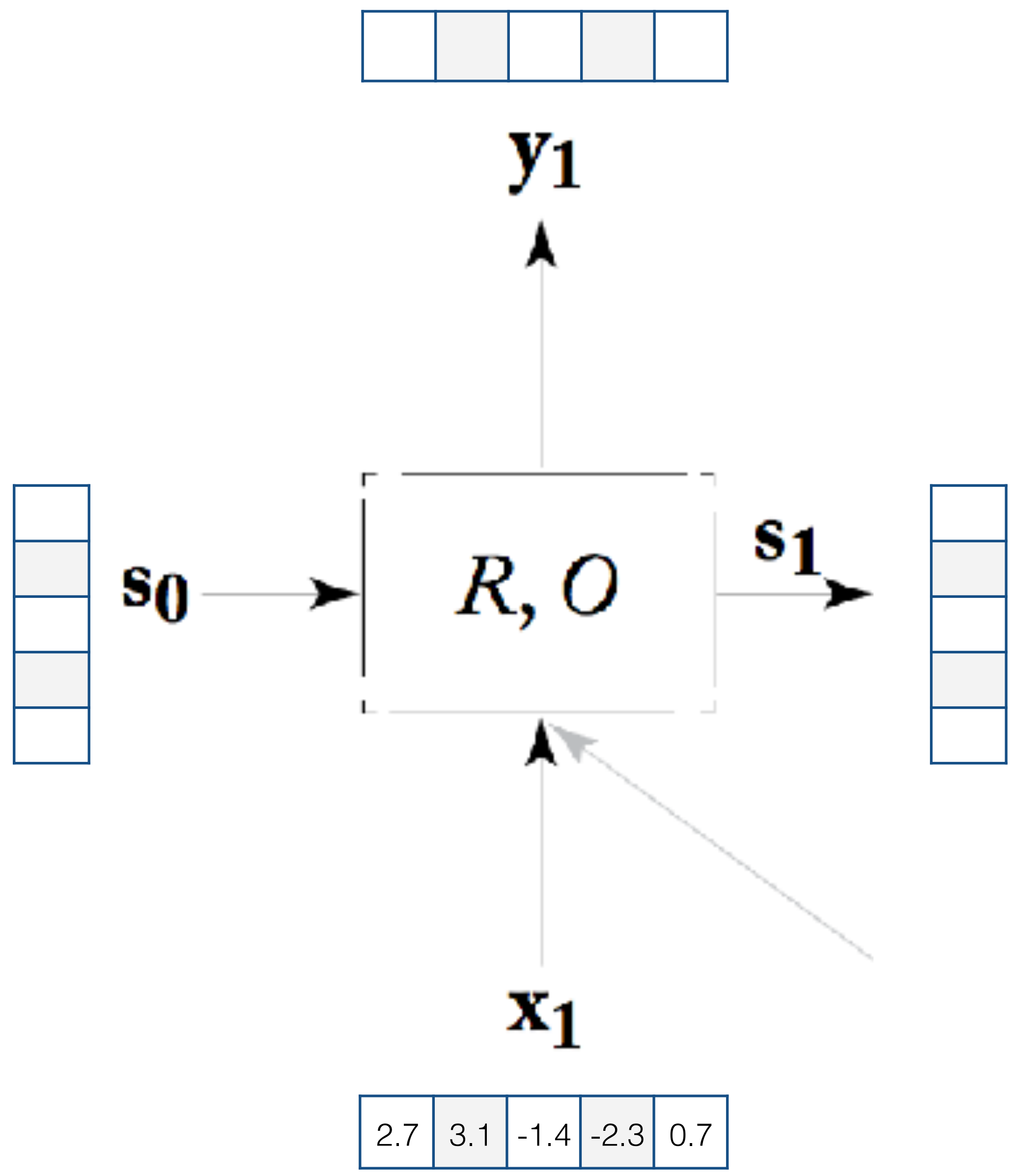
O is some function of the current state



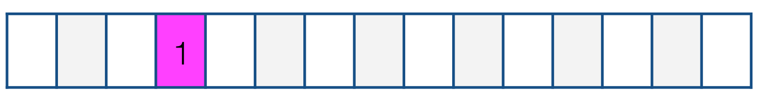
Recurrent neural network

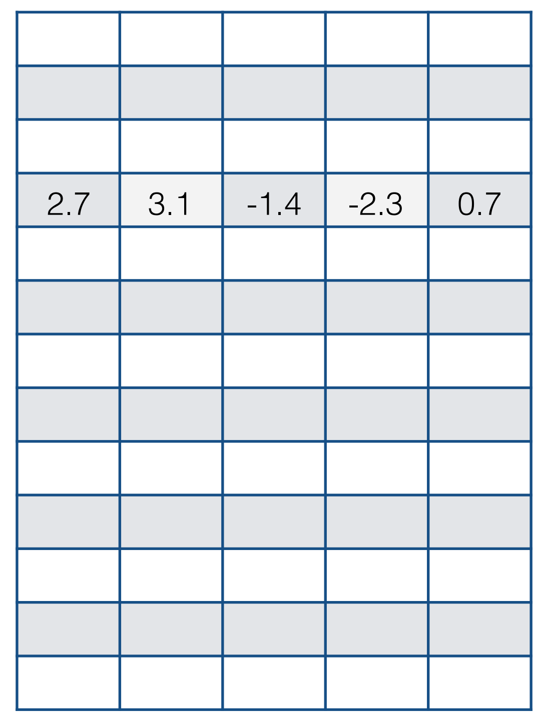
$$s_i = R(x_i, s_{i-1})$$

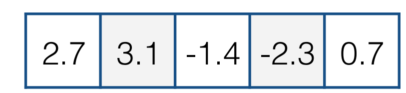
$$y_i = O(s_i)$$



Equivalently







sparse one-hot vector

$x \in \mathscr{R}^{V}$

embeddings matrix

$W \in \mathscr{R}^{V \times H}$

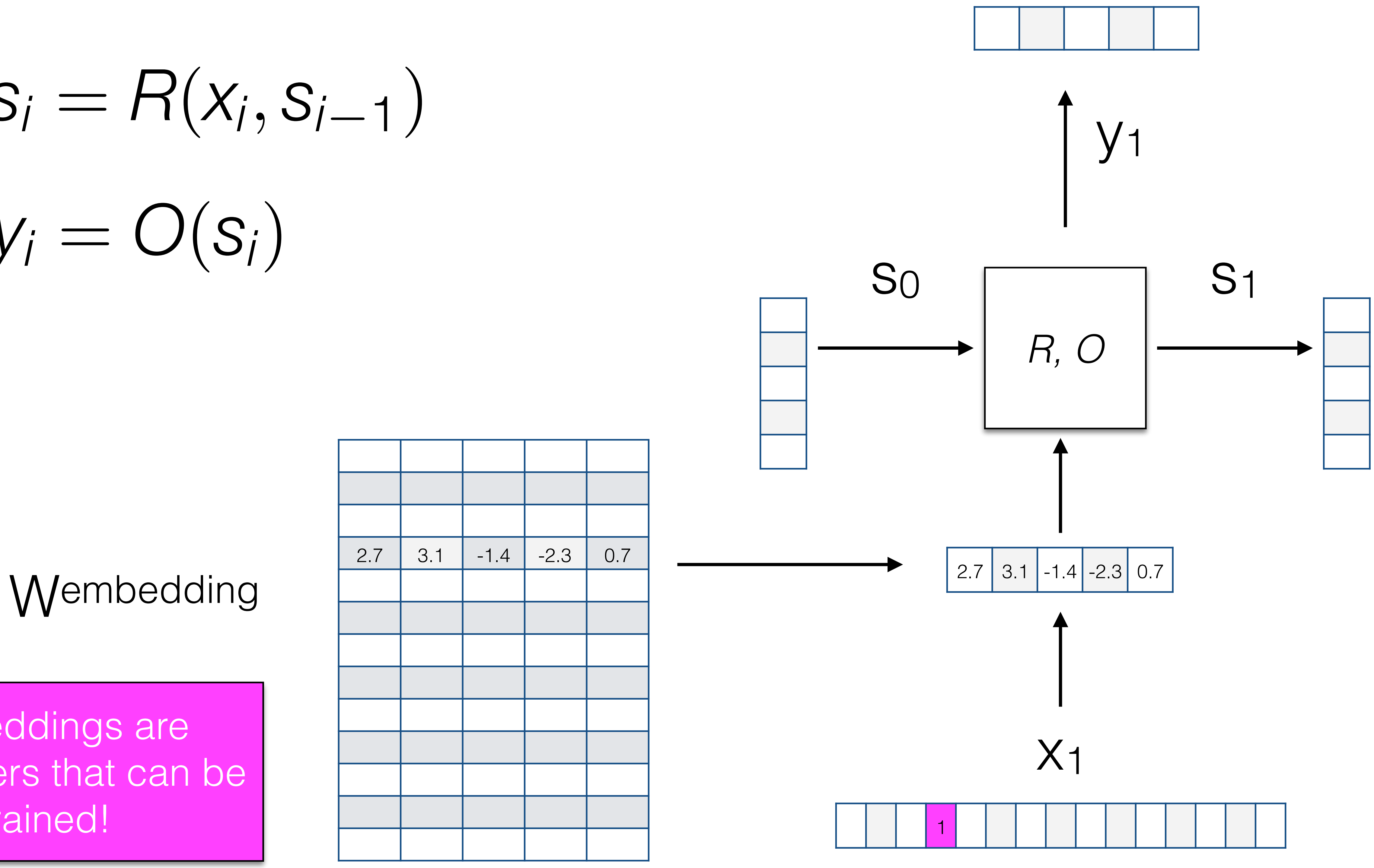
word embedding

$xW \in \mathscr{R}^{H}$

Recurrent neural network

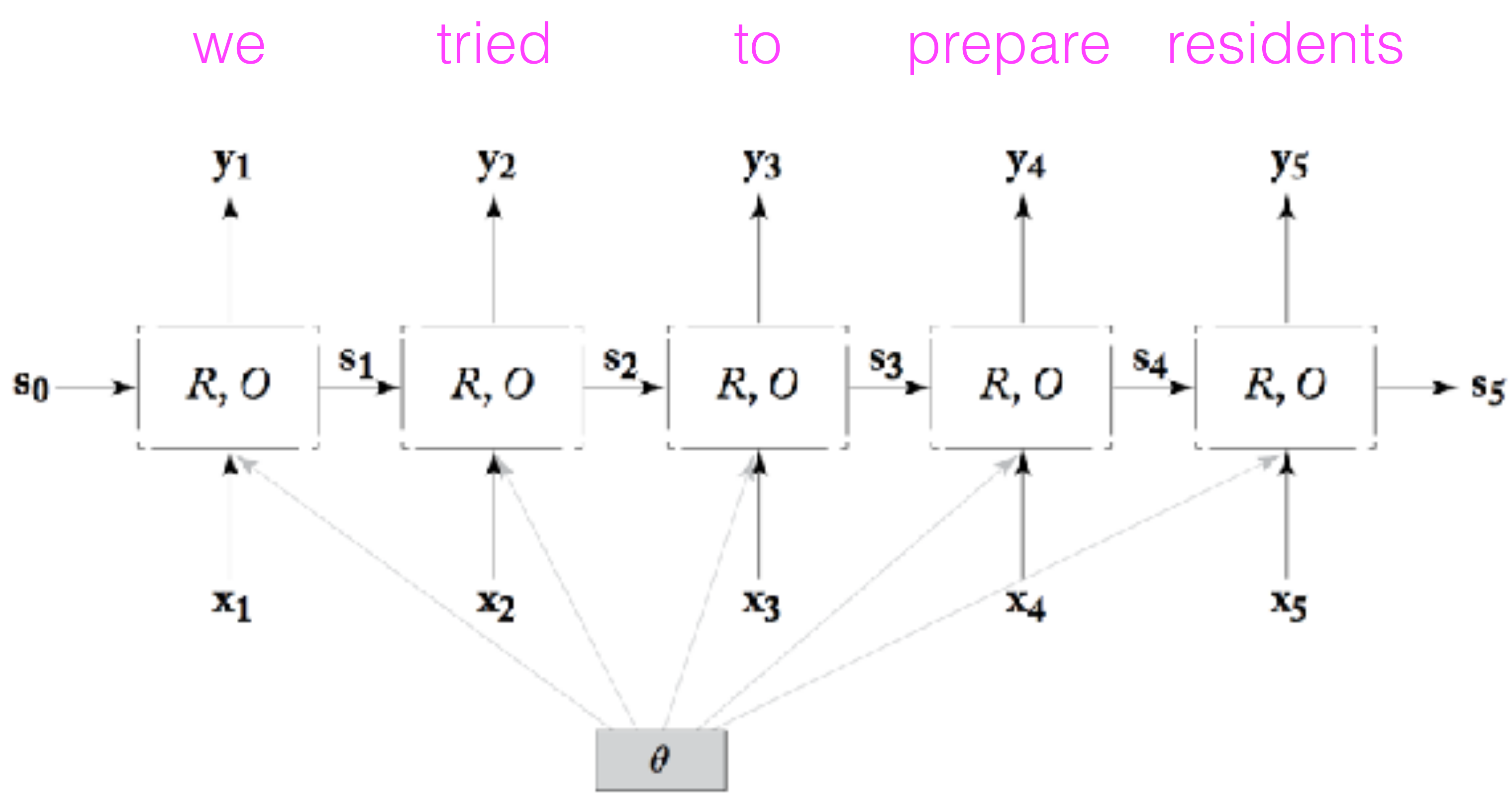
$$s_i = R(x_i, s_{i-1})$$

$$y_i = O(s_i)$$



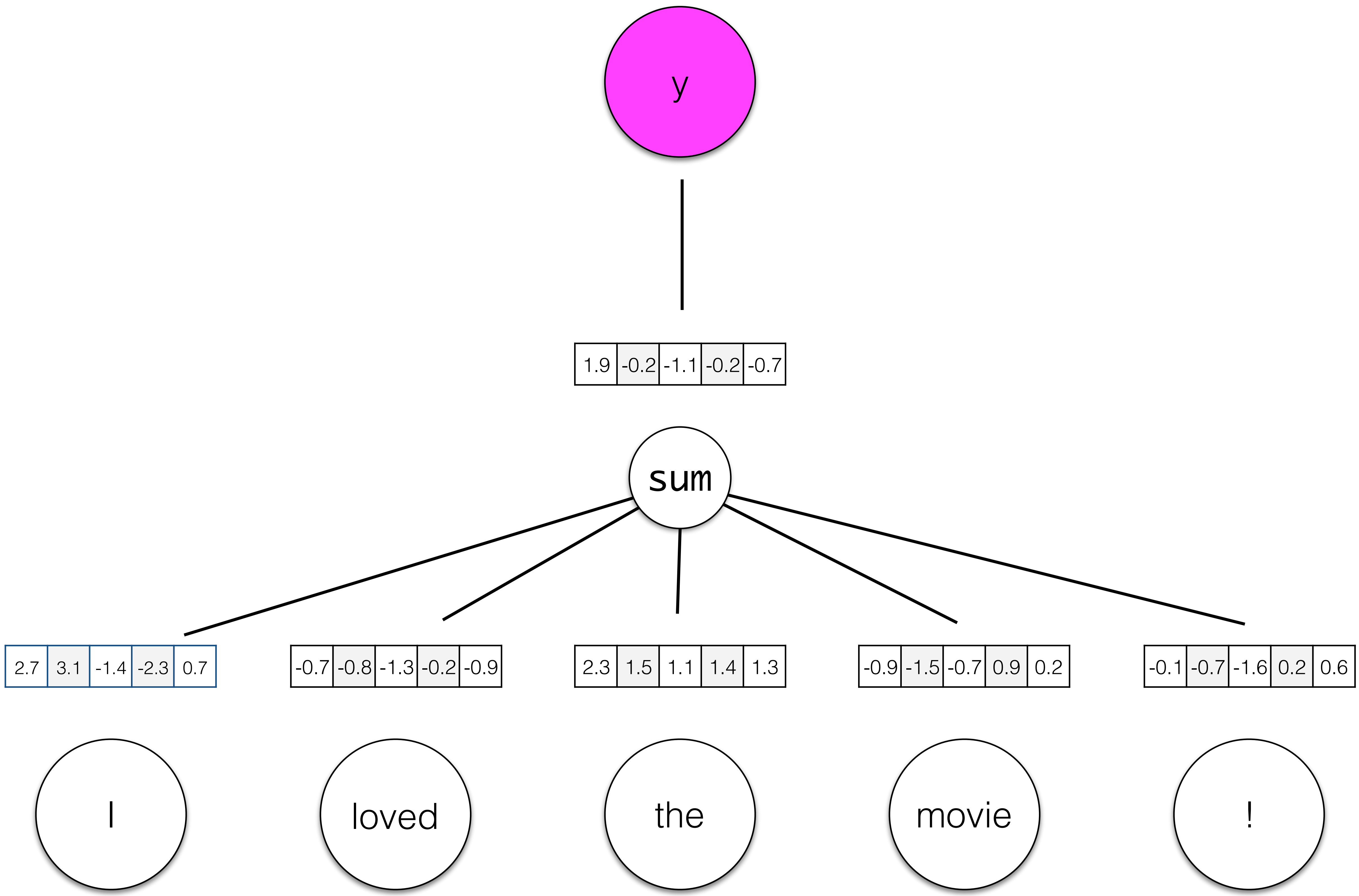
Embeddings are parameters that can be trained!

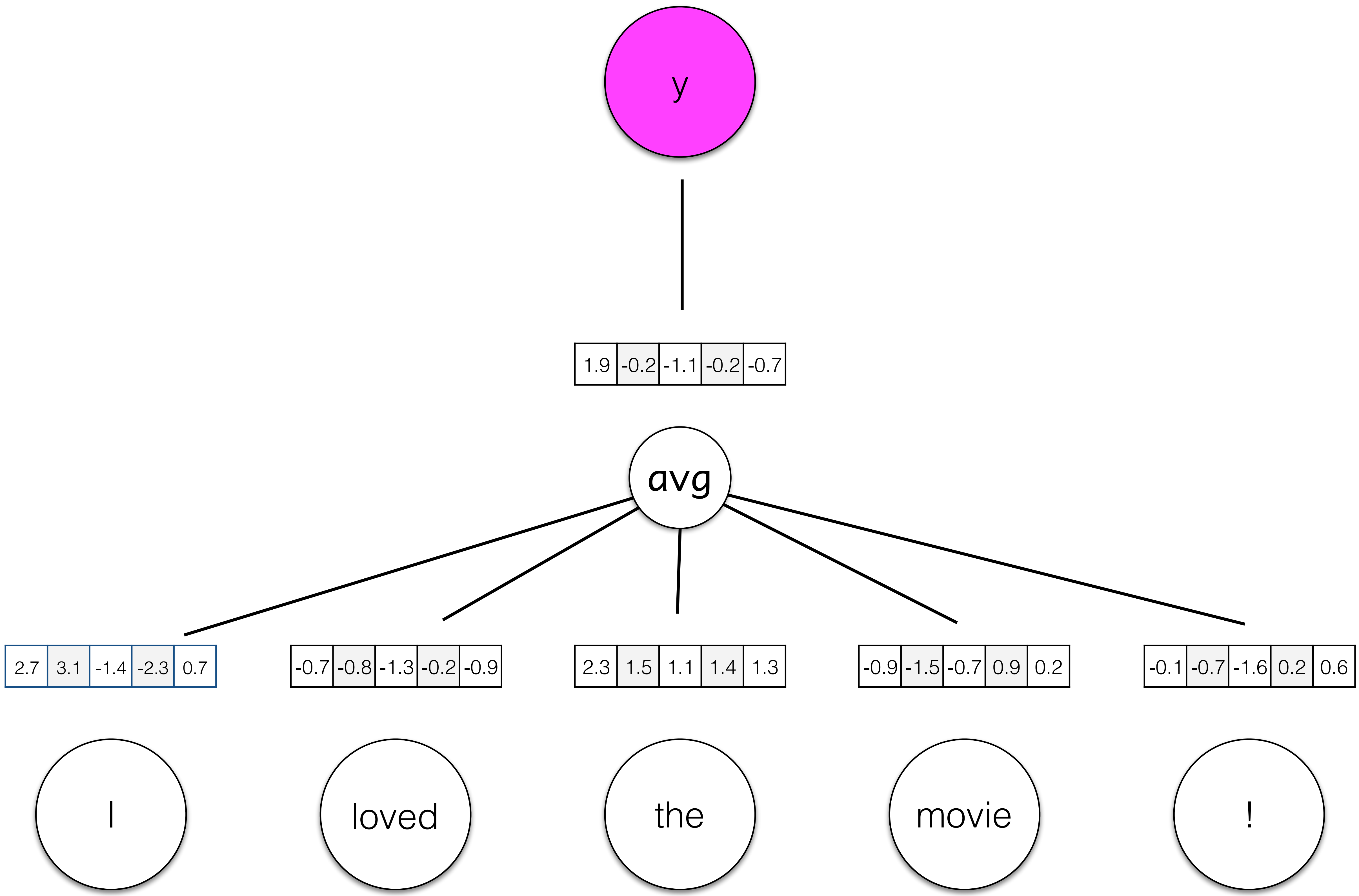
$$\frac{\partial L(\theta)_{y_1}}{\partial W^{embedding}}$$



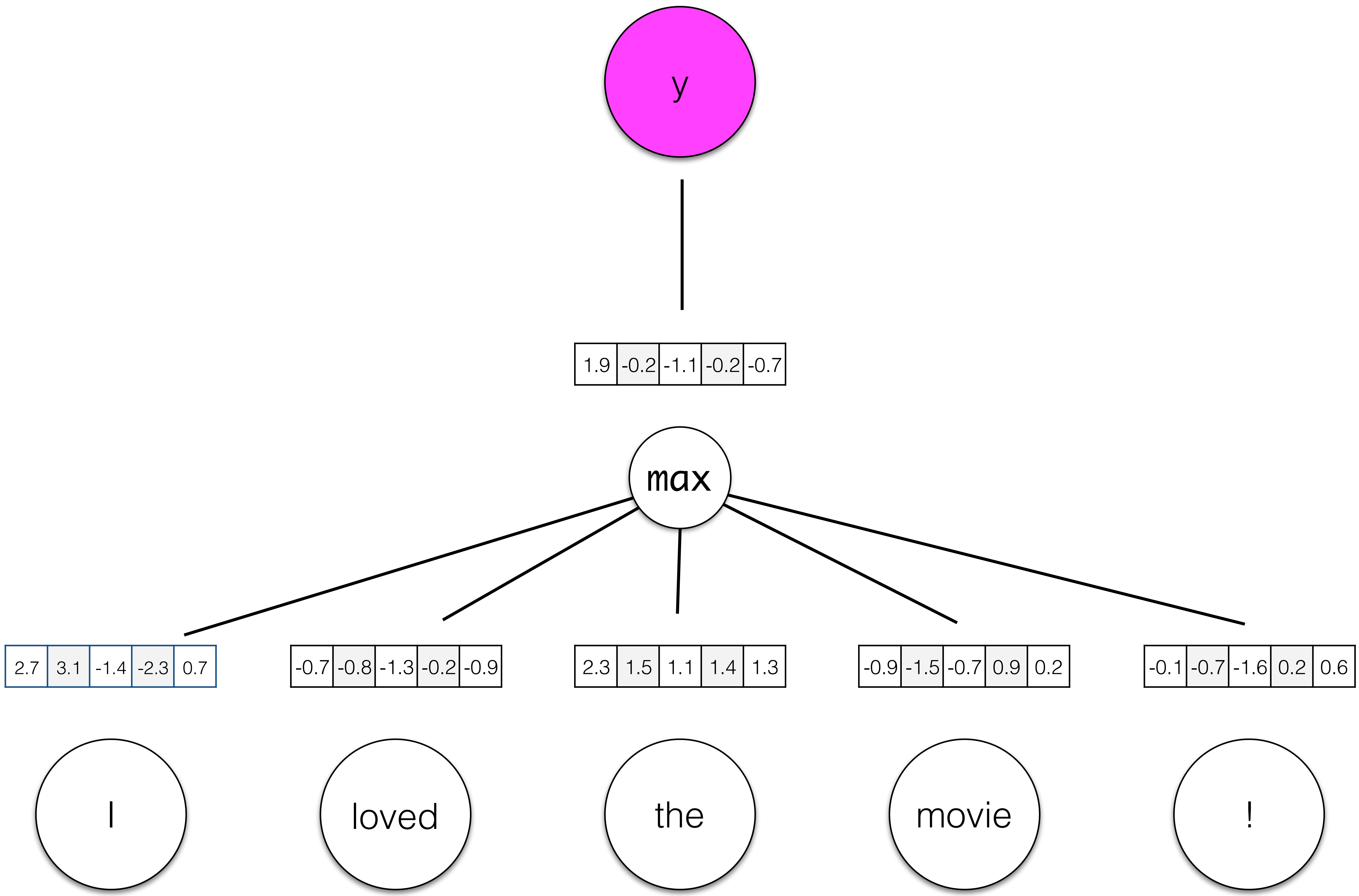
- We can optimize word embeddings for a specific task using by updating them using backpropagation as well.

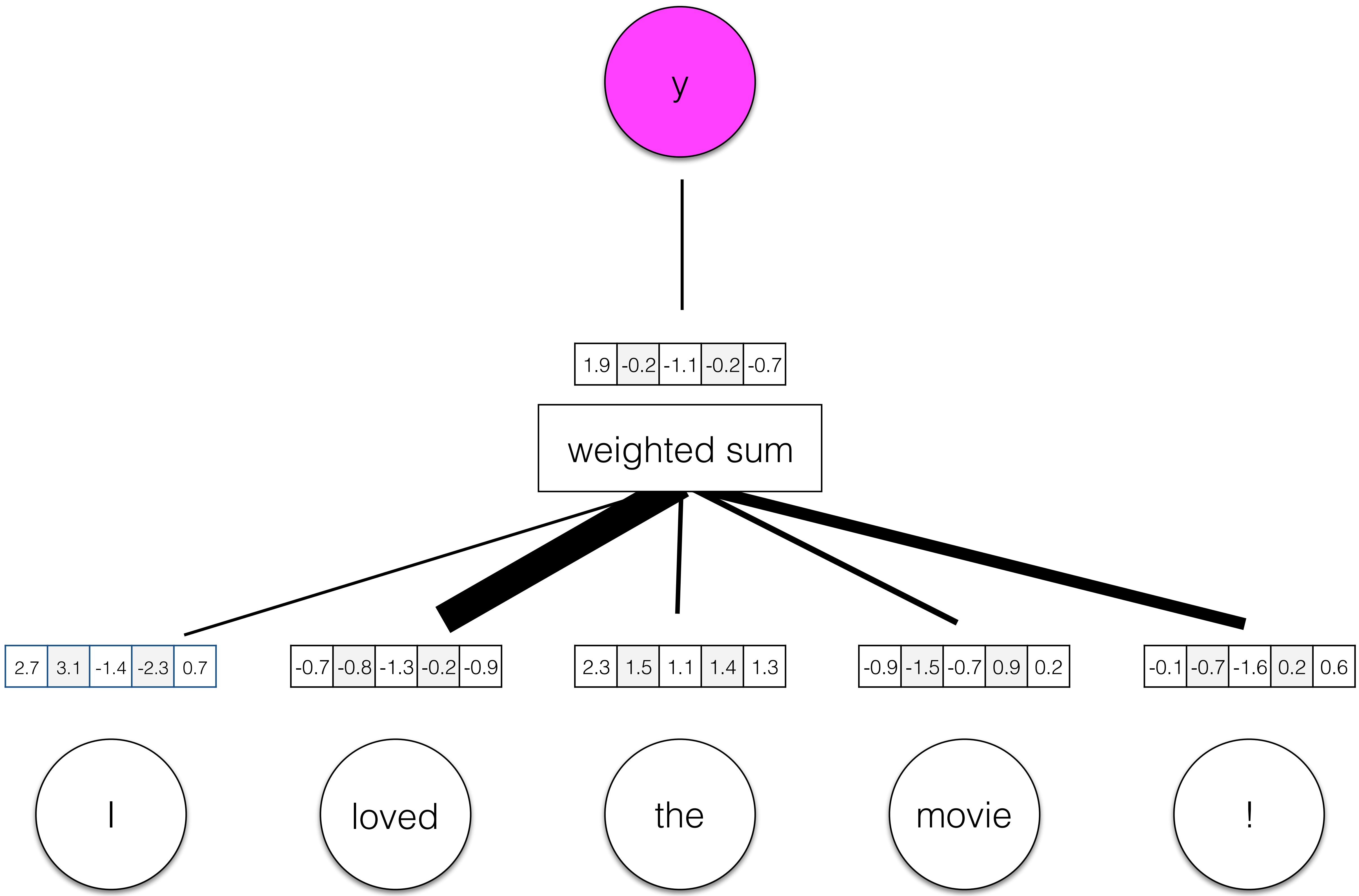
How do we use word embeddings for document classification?


y
1.9 -0.2 -1.1 -0.2 -0.7
sum
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
I
loved
the
movie
!



Iyyer et al. (2015), "Deep Unordered Composition Rivals Syntactic Methods for Text Classification" (ACL)


y
1.9 -0.2 -1.1 -0.2 -0.7
max
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
I
loved
the
movie
!


y
1.9 -0.2 -1.1 -0.2 -0.7
weighted sum
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
I
loved
the
movie
!

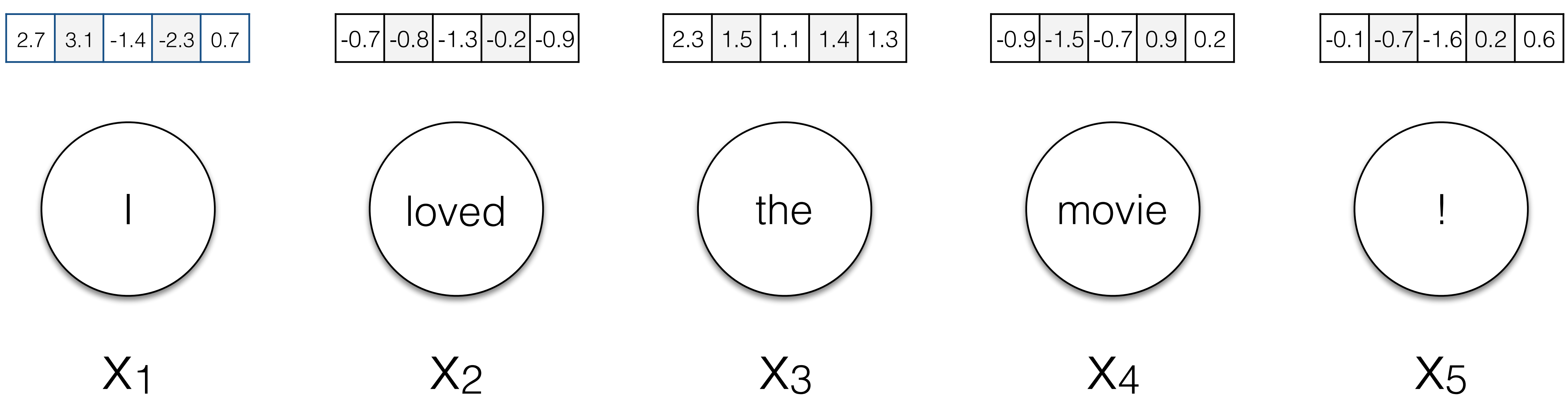
Attention

- Let's incorporate structure (and parameters) into a network that captures which elements in the input we should be attending to (and which we can ignore).

$$v \in \mathcal{R}^H$$

2.7	3.1	-1.4	-2.3	0.7

Define v to be a vector to be learned; think of it as an "important word" vector. The dot product here measures how similar each input vector is to that "important word" vector



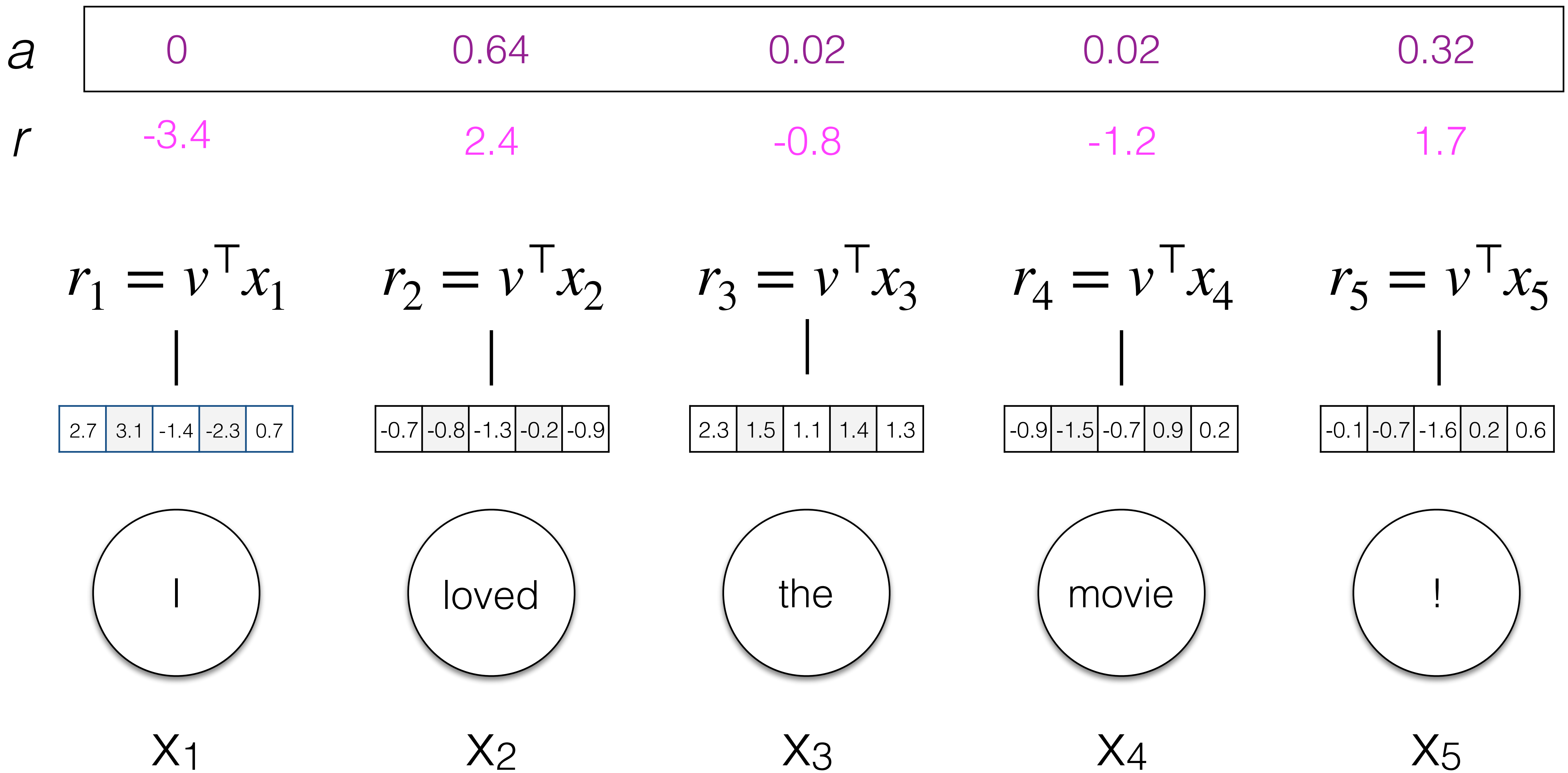
$$v \in \mathscr{R}^H$$

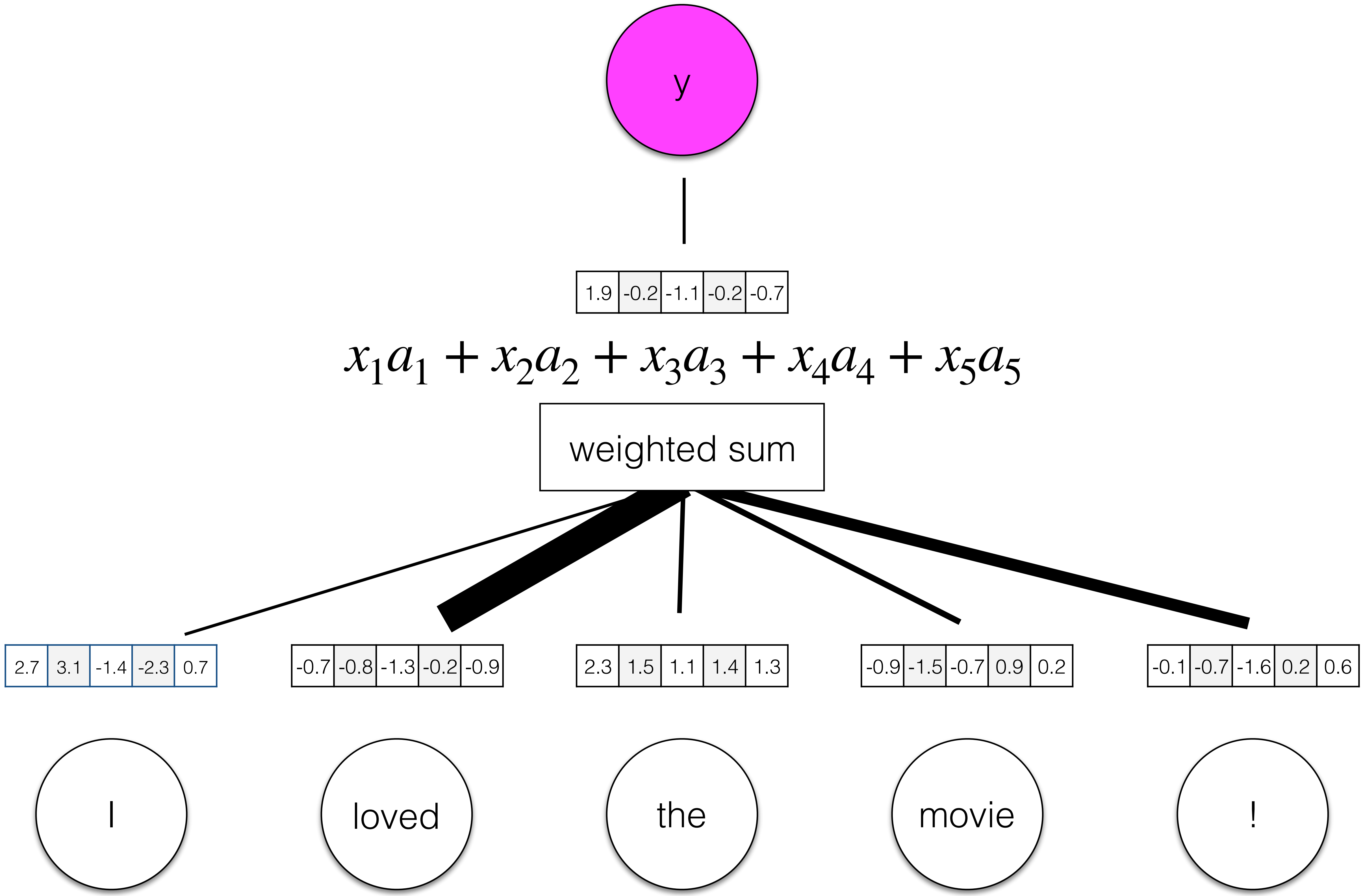
2.7	3.1	-1.4	-2.3	0.7

	x_1	x_2	x_3	x_4	x_5
score	-3.4	2.4	-0.8	-1.2	1.7
equation	$r_1 = v^\top x_1$	$r_2 = v^\top x_2$	$r_3 = v^\top x_3$	$r_4 = v^\top x_4$	$r_5 = v^\top x_5$
vector	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
word	I	loved	the	movie	!

Convert r into a vector of normalized weights that sum to 1.

$$a = \mathrm{softmax}(r)$$




y
1.9 -0.2 -1.1 -0.2 -0.7
$x_1a_1 + x_2a_2 + x_3a_3 + x_4a_4 + x_5a_5$
weighted sum
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
I
loved
the
movie
!

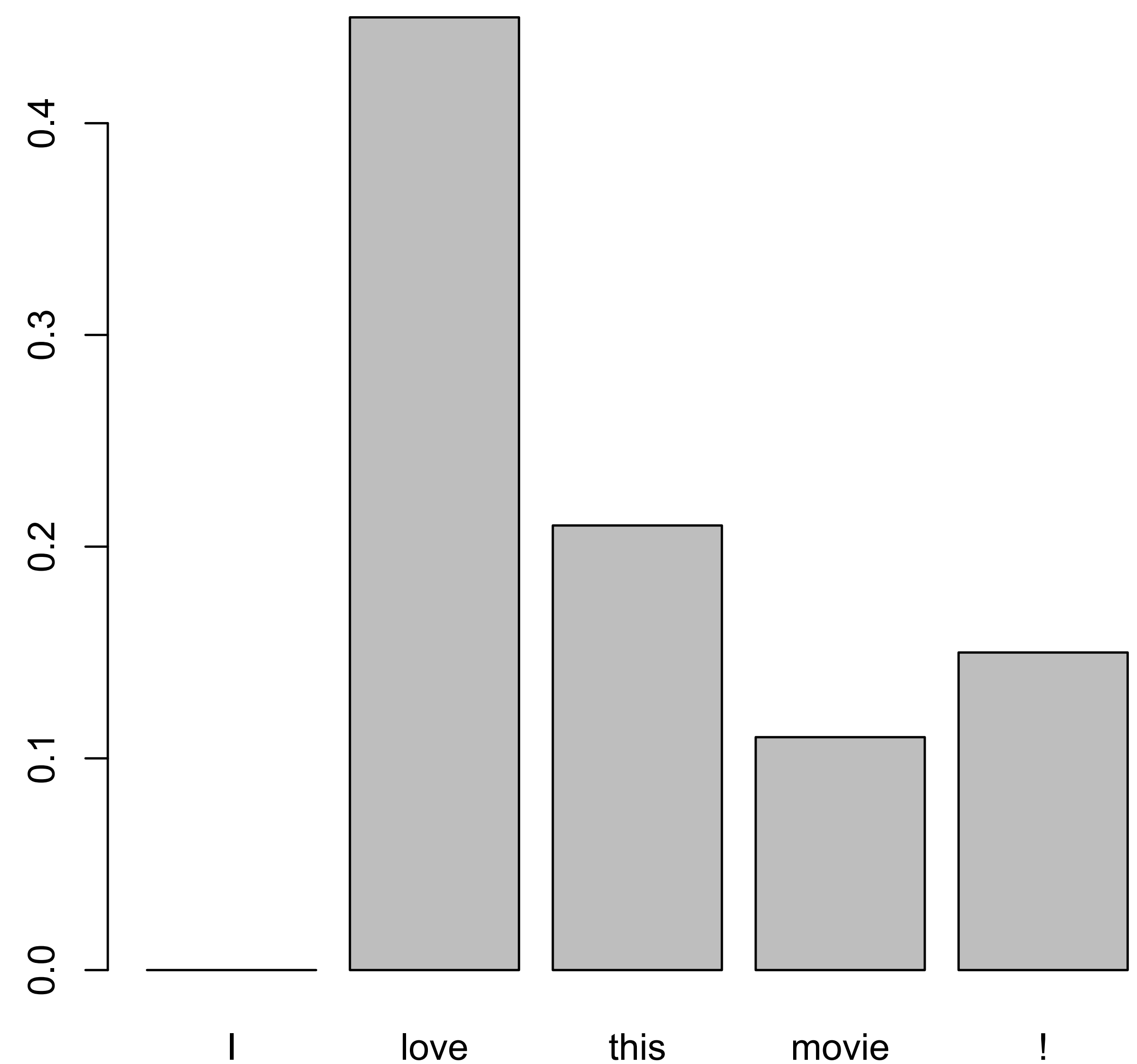
Attention

- Lots of variations on attention:
  - Linear transformation of x into before dotting with v
  - Non-linearities after each operation.
  - “Multi-head attention”: multiple v vectors to capture different phenomena that can be attended to in the input.
  - Hierarchical attention (sentence representation with attention over words + document representation with attention over sentences).

Attention

- Attention gives us a normalized weight for every token in a sequence that tells us how important that word was for the prediction

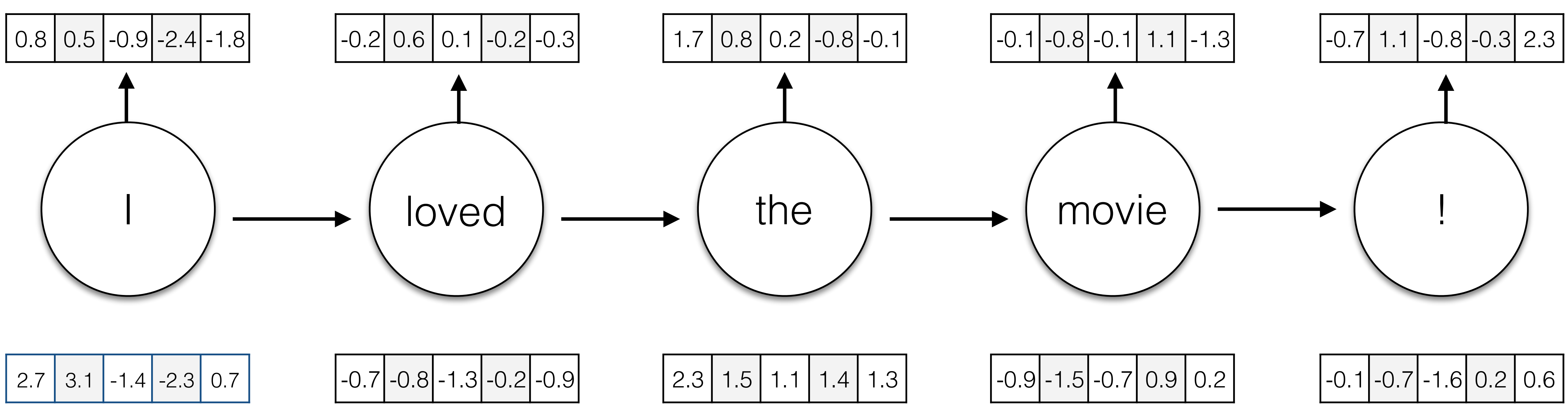
- This can be useful for visualization



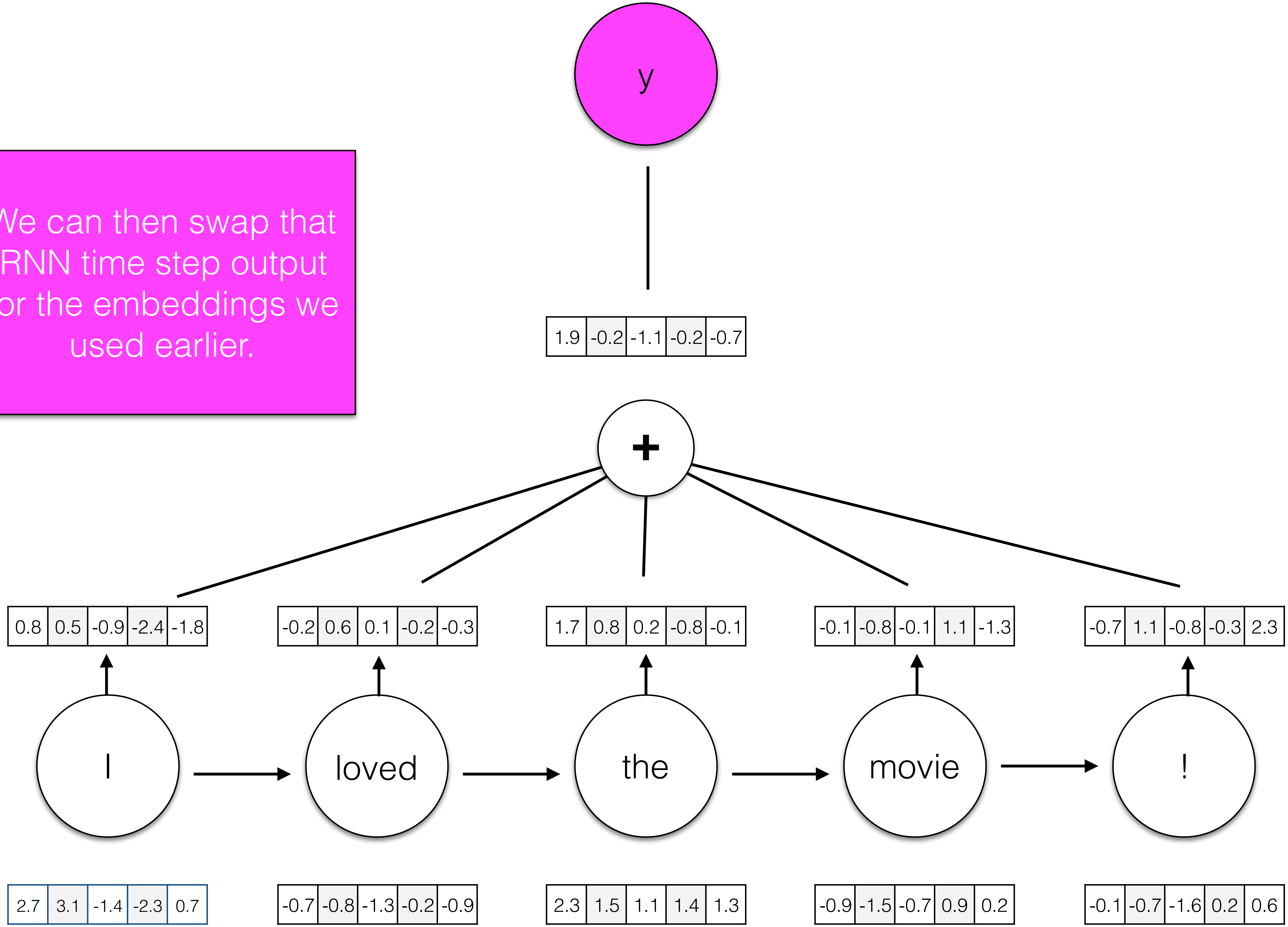
RNN

- With an RNN, we can generate a representation of the sequence as seen through time t.

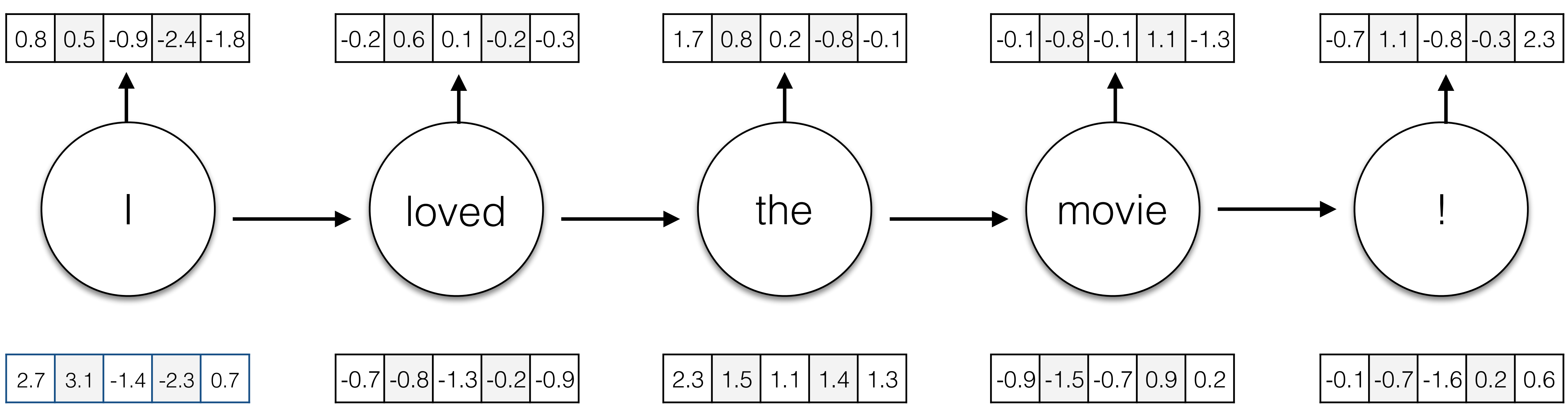
- This encodes a representation of meaning specific to the local context a word is used in.



We can then swap that RNN time step output for the embeddings we used earlier.



What about the future context?

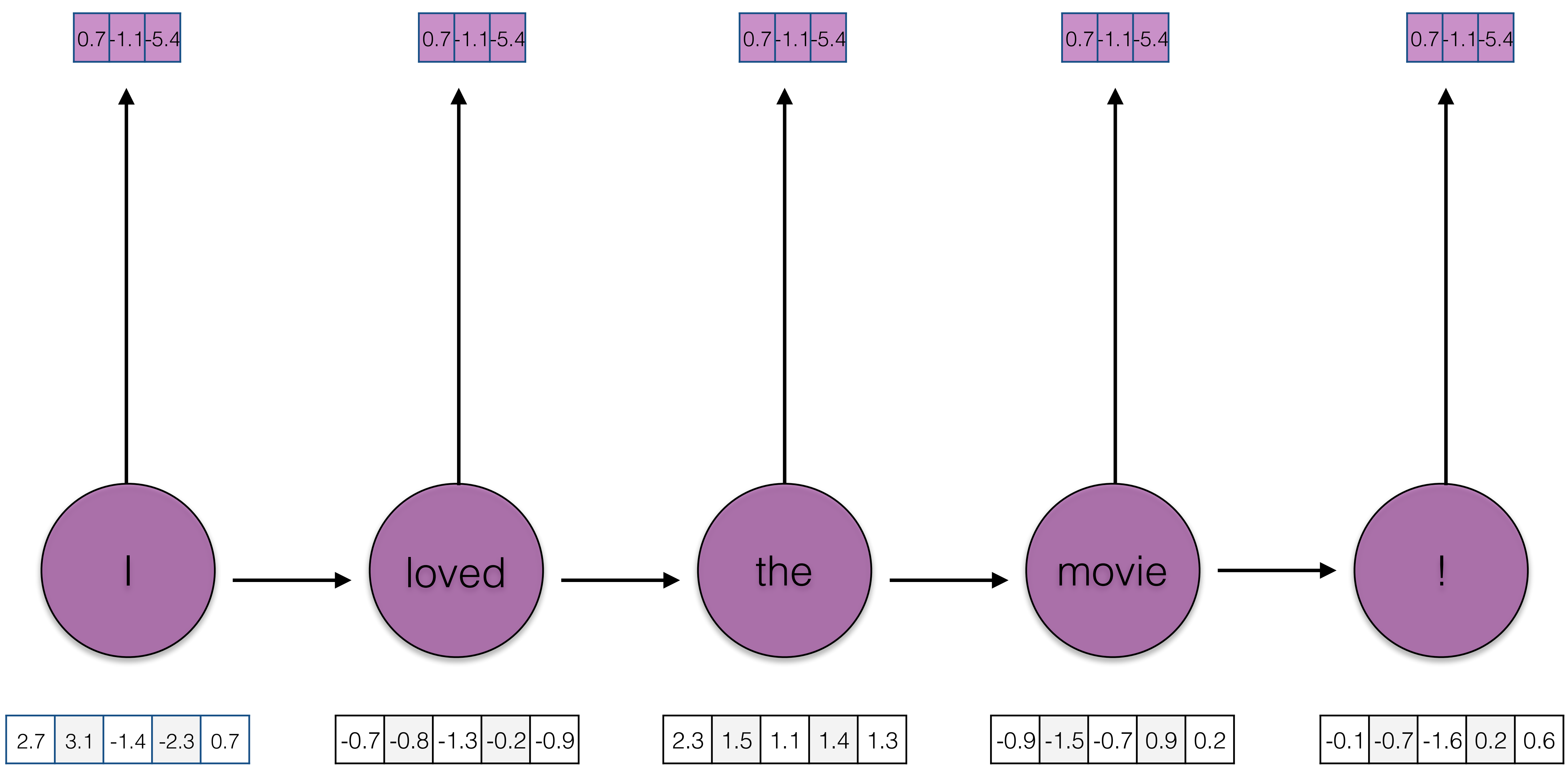


Bidirectional RNN

- 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

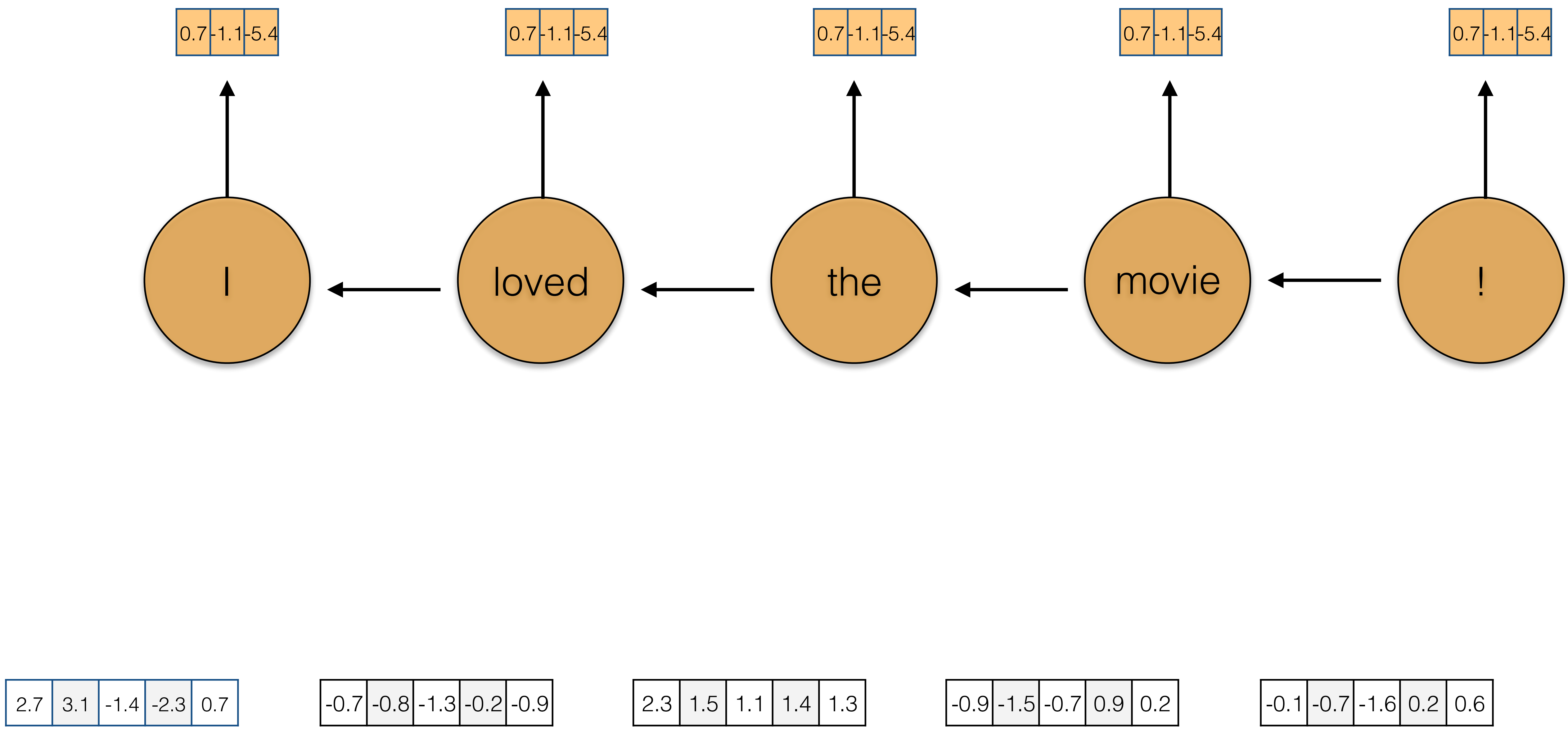
Bidirectional RNN

forward RNN

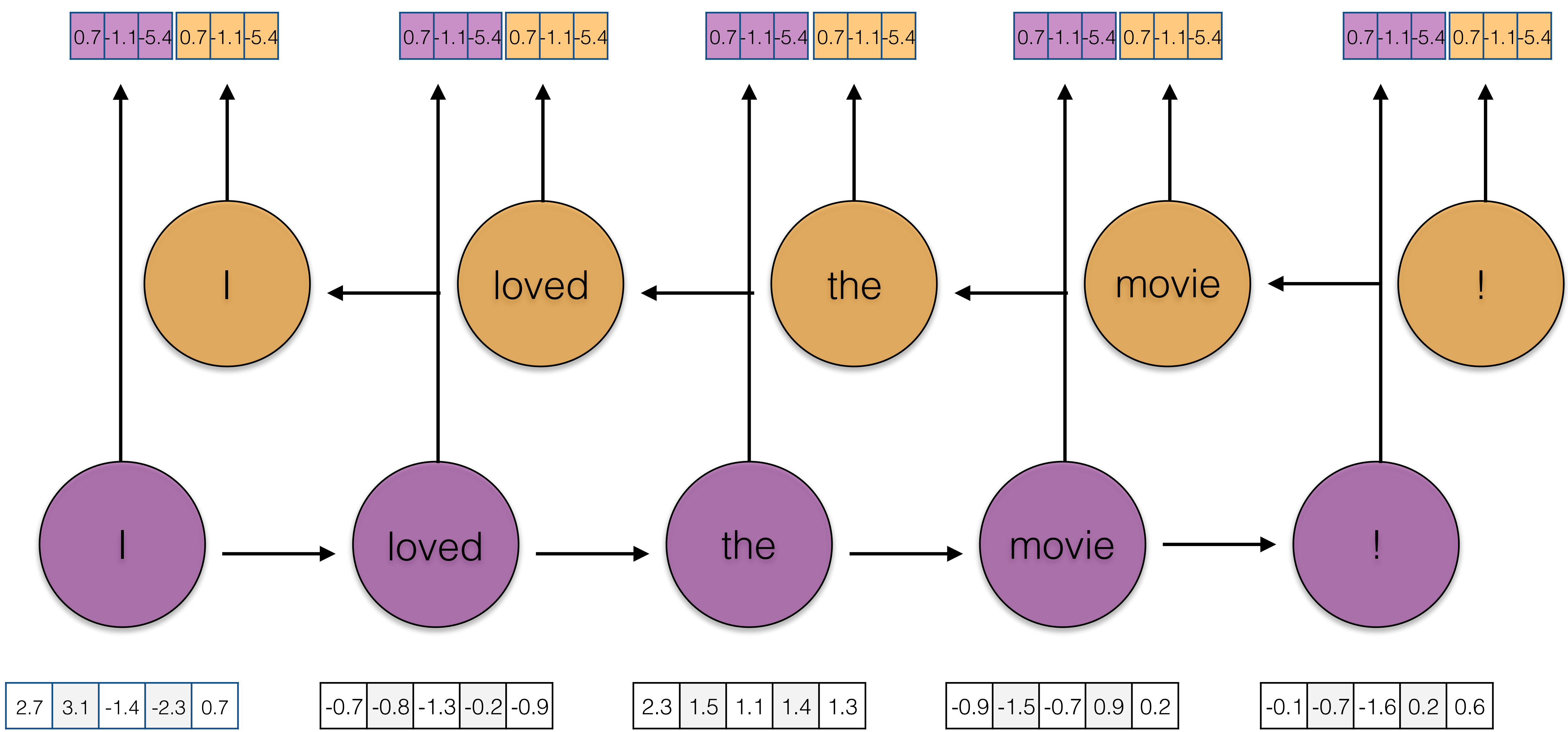


Bidirectional RNN

backward RNN



Bidirectional RNN



Bidirectional RNN

- The forward RNN and backward RNN each output a vector of size H at each time step, which we concatenate into a vector of size 2H.

- The forward and backward RNN each have separate parameters to be learned during training.

Training BiRNNs

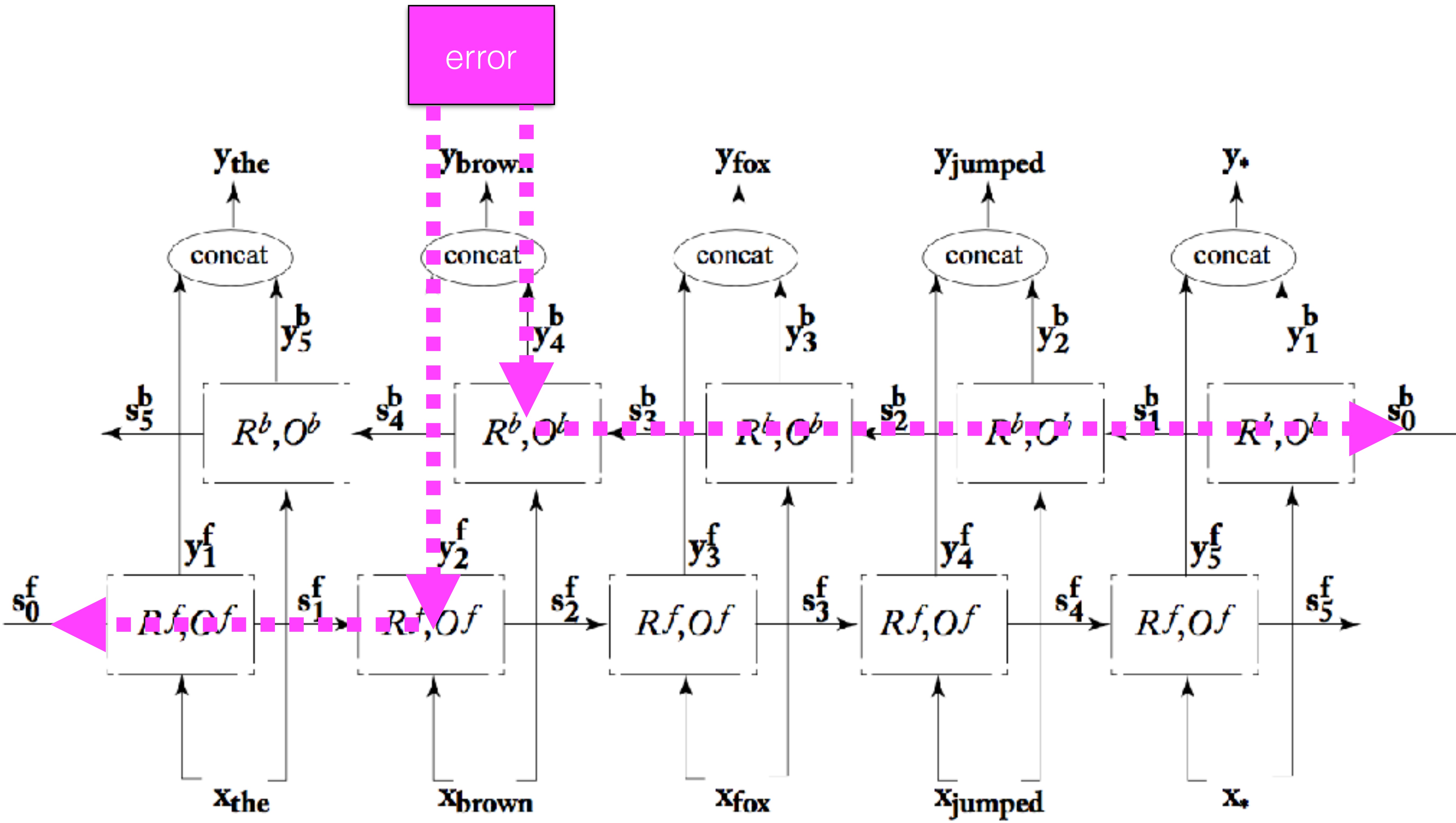
- Given this definition of an BiRNN:

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

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

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

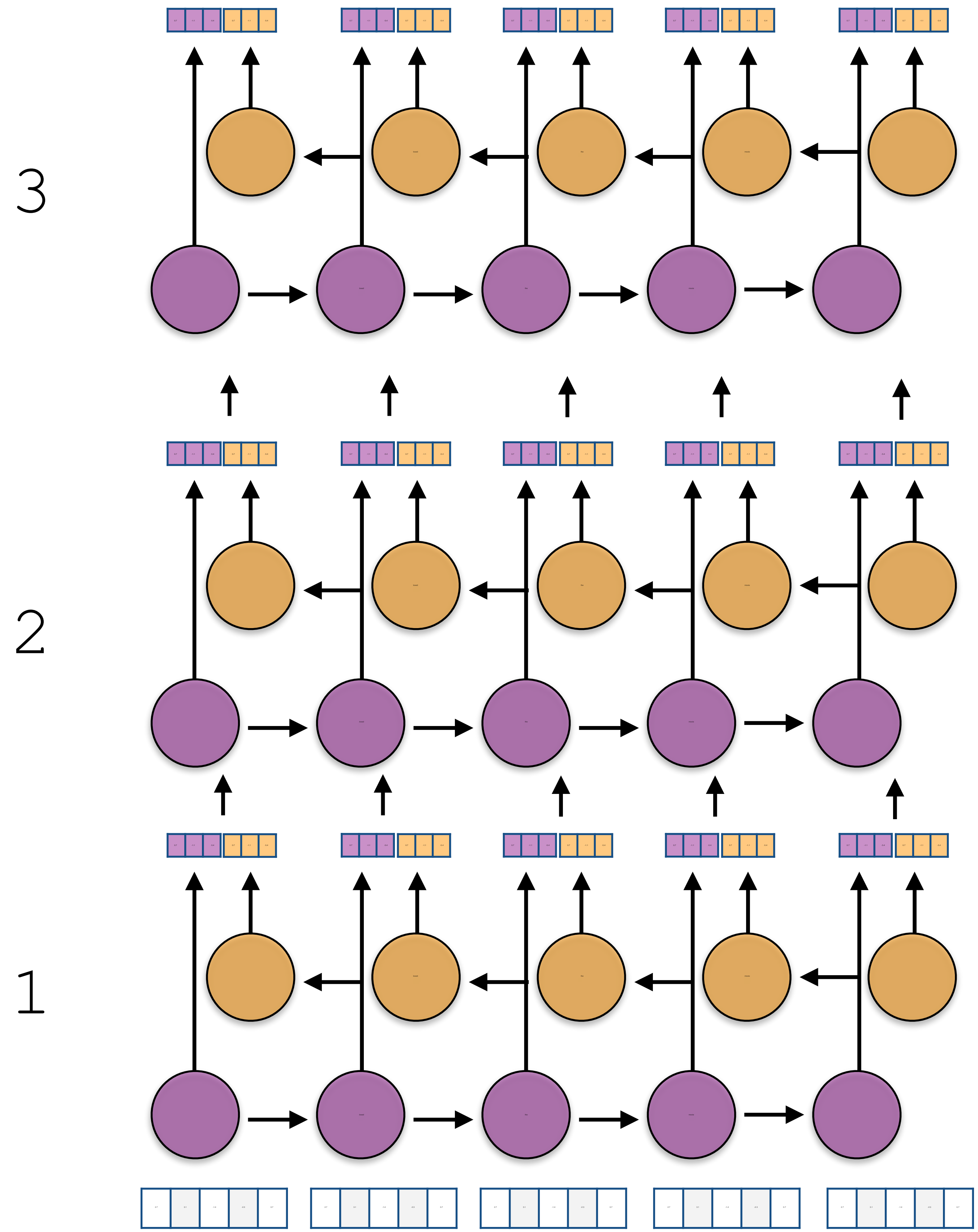
- We have 8 sets of parameters to learn (3 for each RNN + 2 for the final layer)



Goldberg 2017

Stacked RNN

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



Contextualized embeddings

- Models for learning static embeddings learn a single representation for a word type.

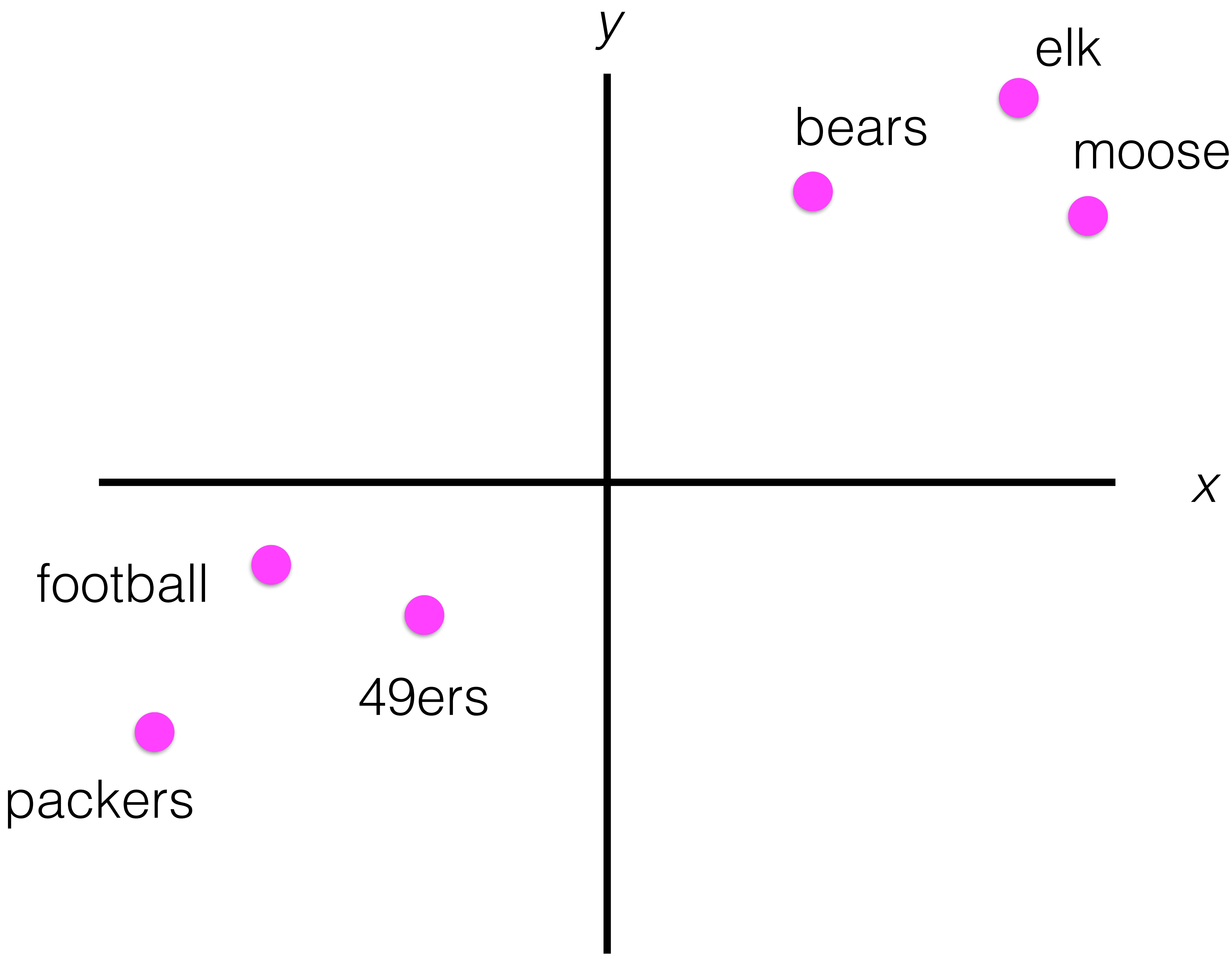
Types and tokens

- Type: bears

- Tokens:
    - The bears ate the honey
    - We spotted the bears from the highway
    - Yosemite has brown bears
    - The chicago bears didn’t make the playoffs

“bears”

3.1	1.4	-2.7	0.3
3.1	1.4	-2.7	0.3
3.1	1.4	-2.7	0.3
3.1	1.4	-2.7	0.3

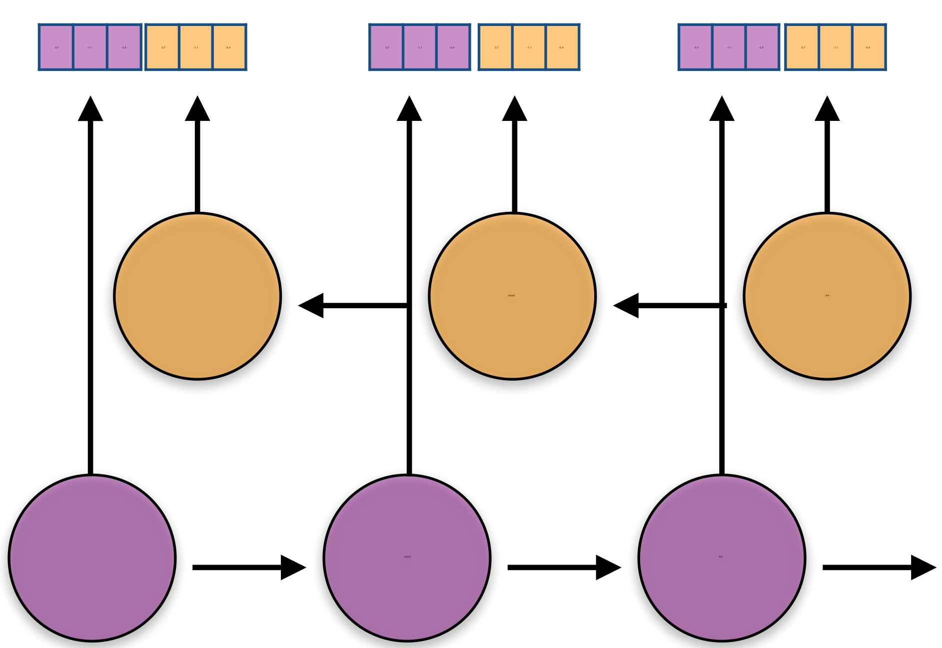

y
elk
bears
moose
x
football
49ers
packers

Contextualized word representations

- Big idea: transform the representation of a token in a sentence (e.g., from a static word embedding) to be sensitive to its local context in a sentence and trainable to be optimized for a specific NLP task.

ELMo

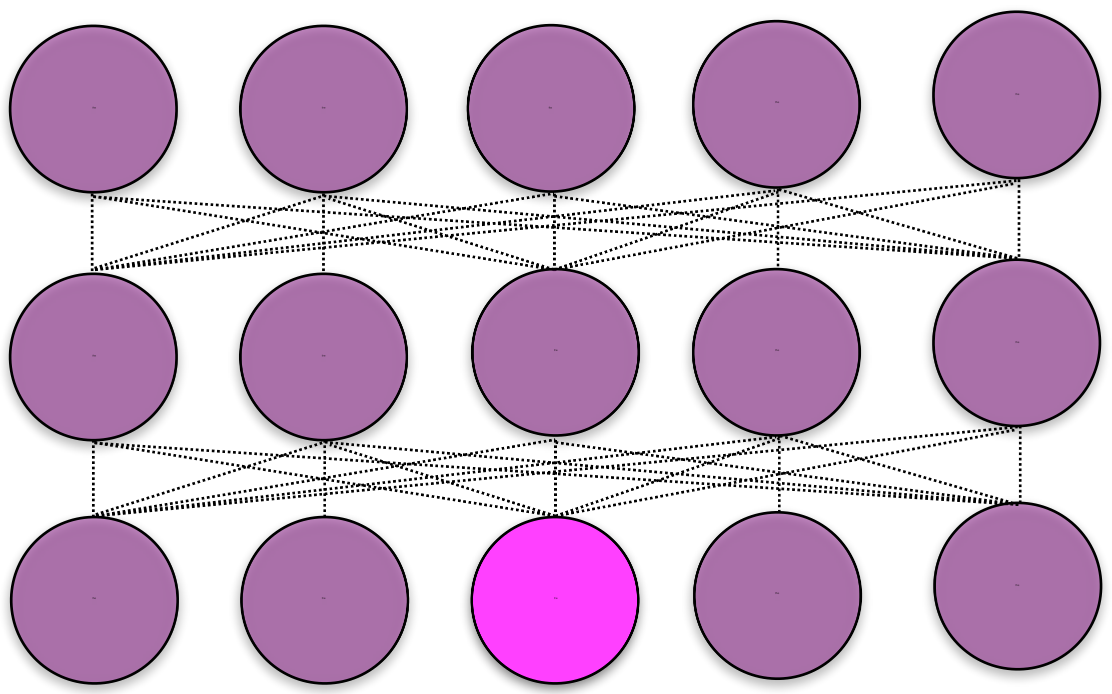
Stacked BiRNN trained to predict next word in language modeling task



Peters et al. 2018

BERT

Transformer-based model to predict masked word using bidirectional context + next sentence prediction.



Devlin et al. 2019

ELMo

- Peters et al. (2018), “Deep Contextualized Word Representations” (NAACL)

- Big idea: transform the representation of a word (e.g., from a static word embedding) to be sensitive to its local context in a sentence and optimized for a specific NLP task.

- Output = word representations that can be plugged into just about any architecture a word embedding can be used.

ELMo

- Peters et al. (2018), “Deep Contextualized Word Representations” (NAACL)

- Train a bidirectional RNN language model with L layers on a bunch of text.

- Learn parameters to combine the RNN output across all layers for each word in a sentence for a specific task (NER, semantic role labeling, question answering etc.). Large improvements over SOTA for lots of NLP problems.

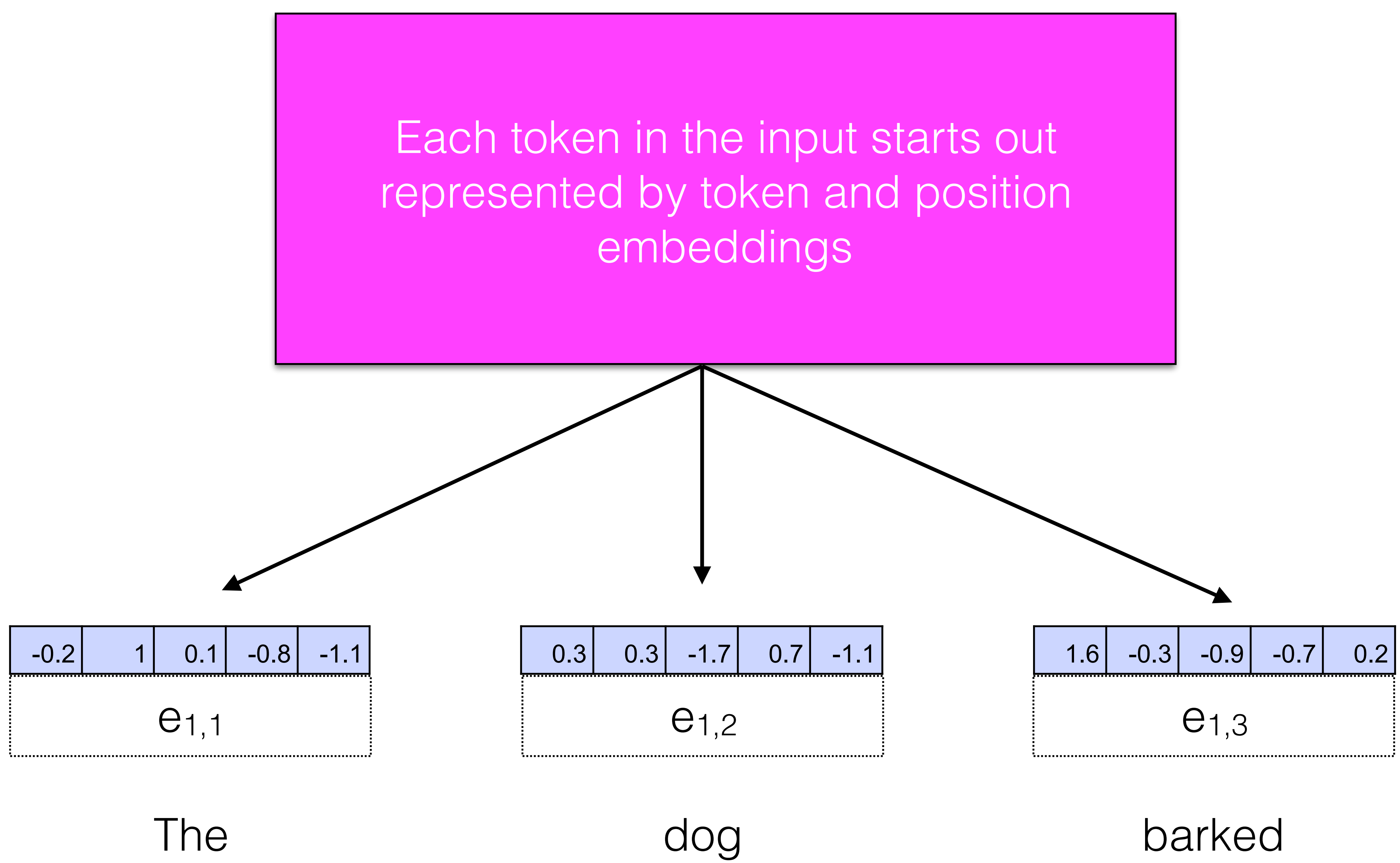
ELMo

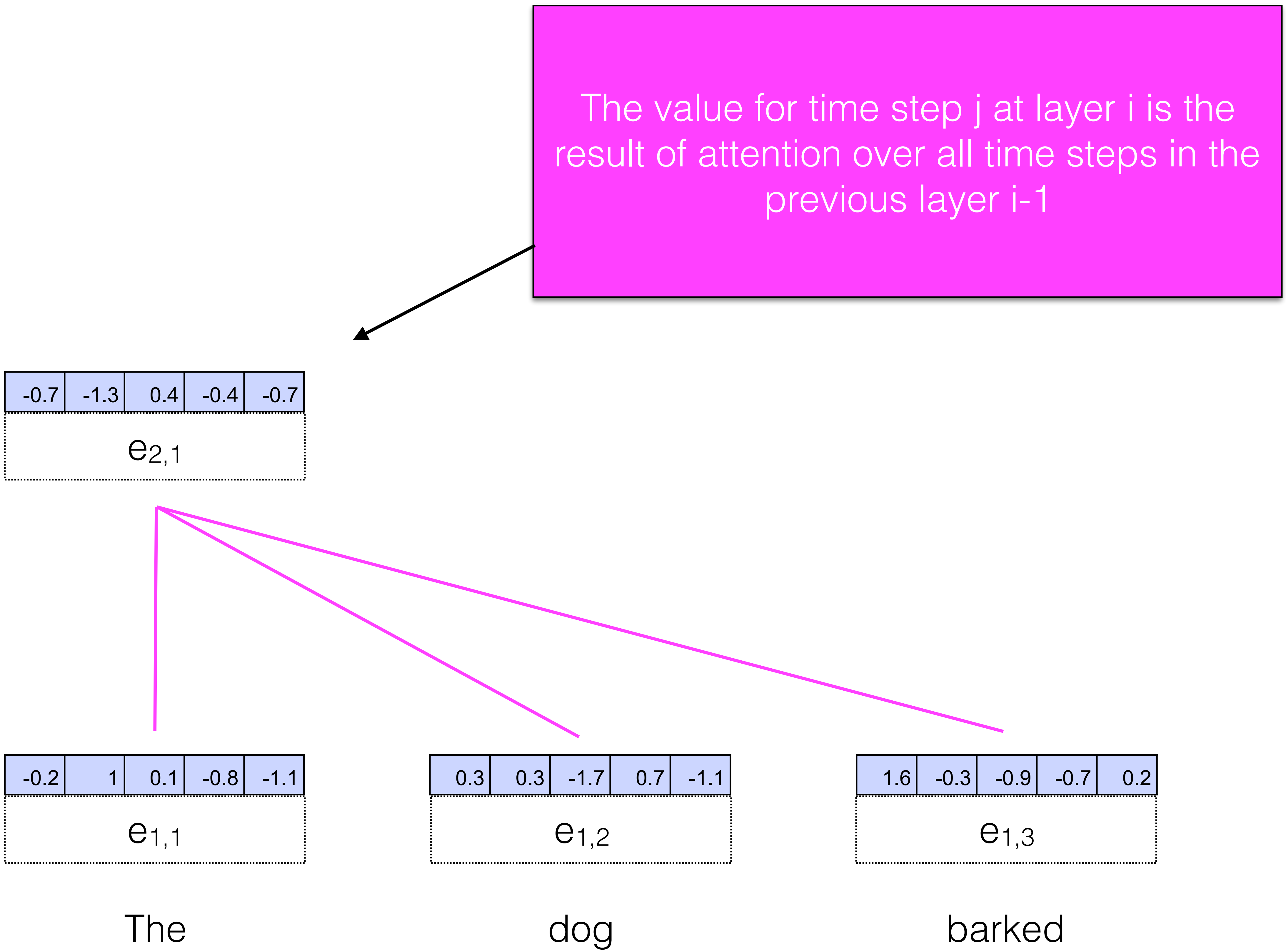
Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

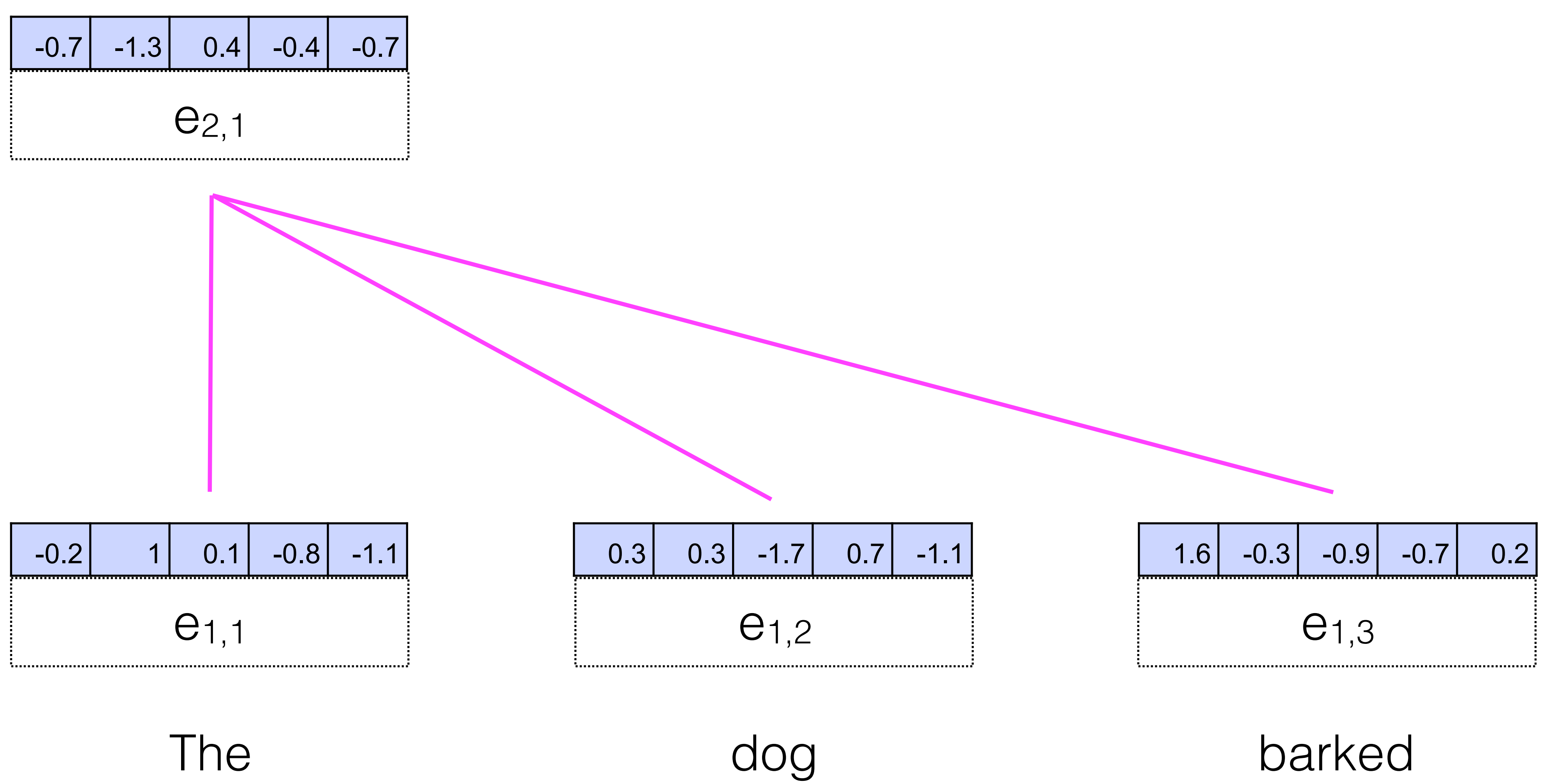
Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

BERT

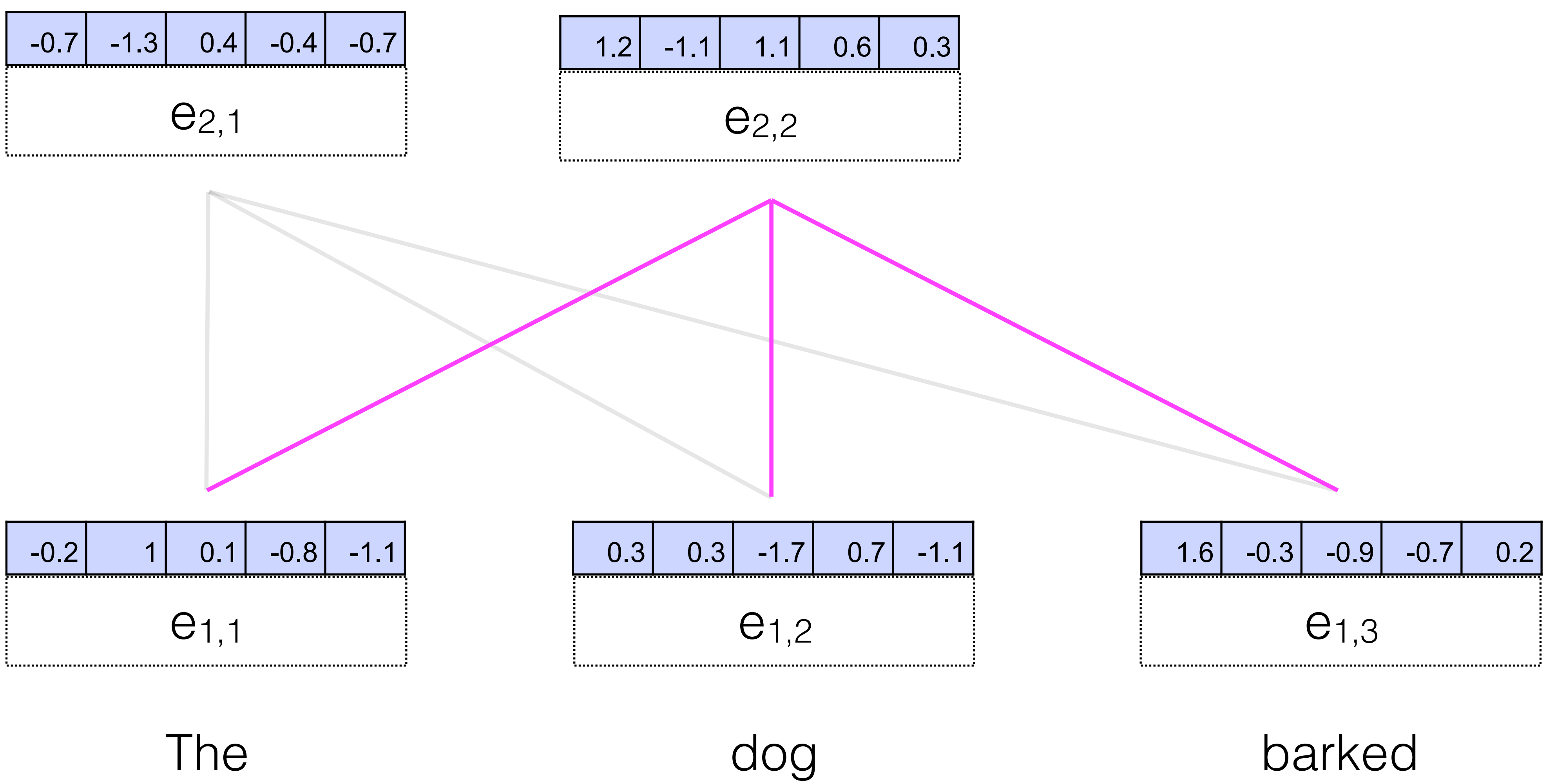
- Transformer-based model (Vaswani et al. 2017) to predict masked word using bidirectional context + next sentence prediction.
- Generates multiple layers of representations for each token sensitive to its context of use.


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

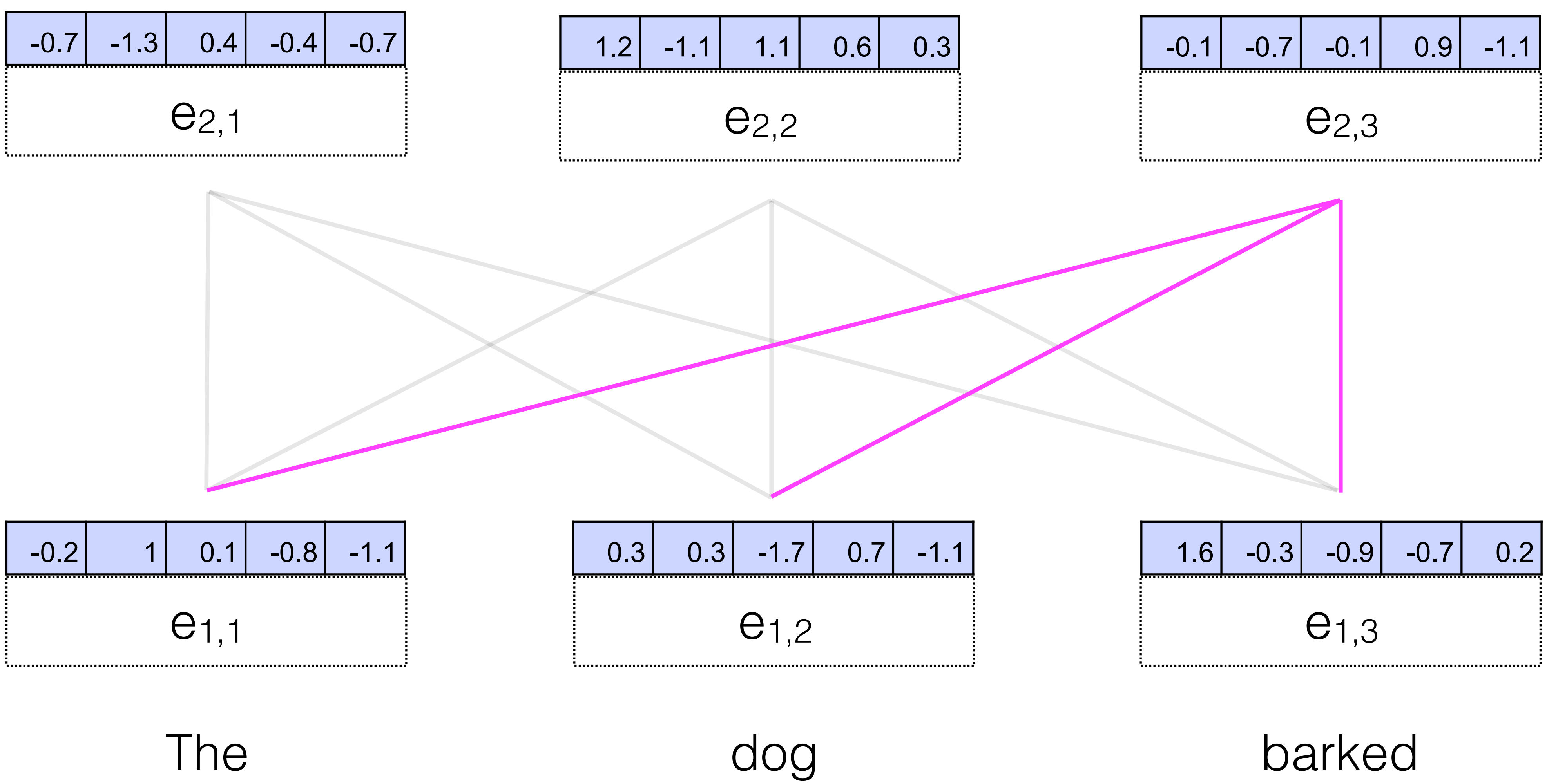

The value for time step j at layer i is the result of attention over all time steps in the previous layer i-1
-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
-0.2 1 0.1 -0.8 -1.1
$e_{1,1}$
0.3 0.3 -1.7 0.7 -1.1
$e_{1,2}$
1.6 -0.3 -0.9 -0.7 0.2
$e_{1,3}$
The
dog
barked

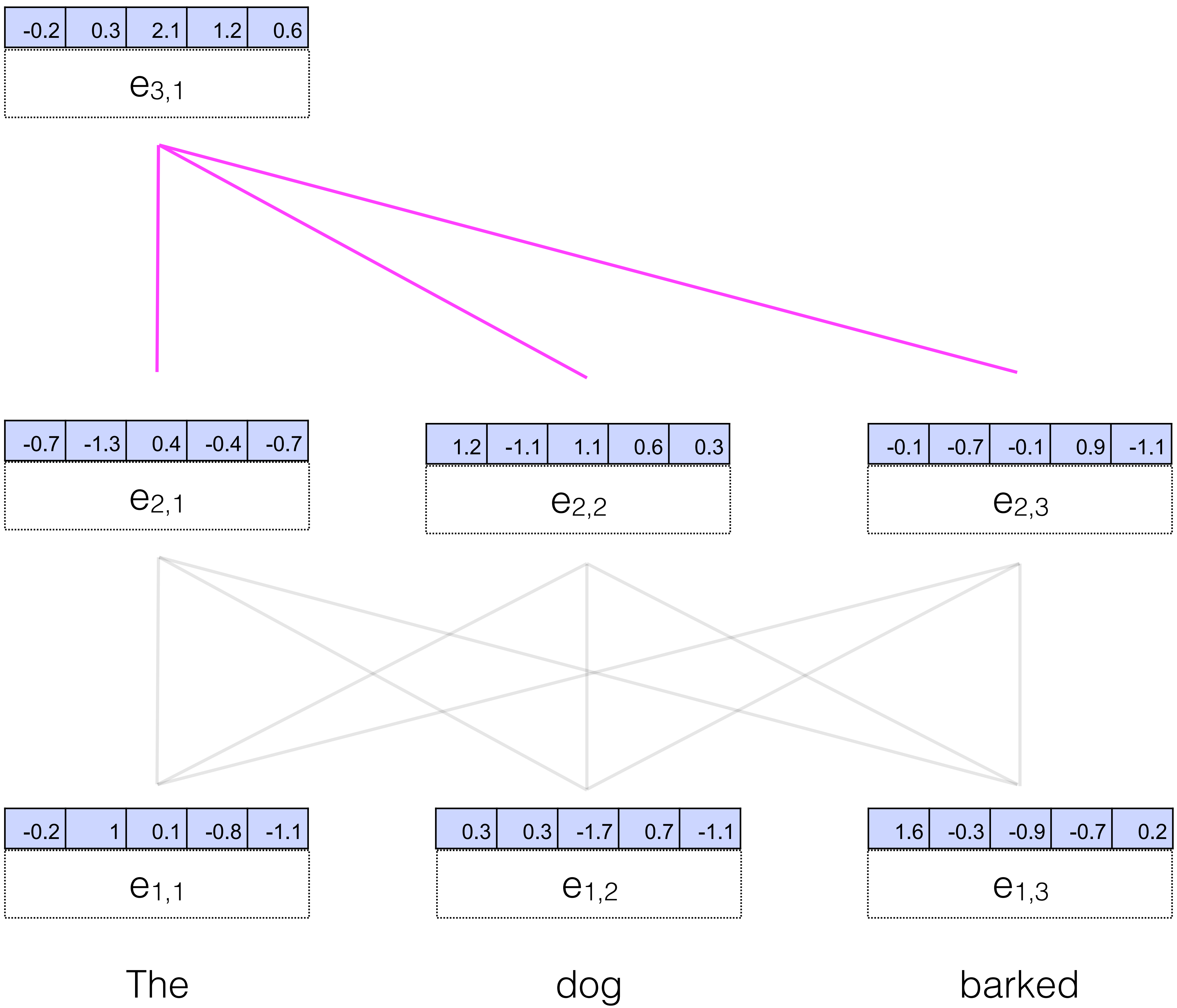


-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
-0.2 1 0.1 -0.8 -1.1
$e_{1,1}$
0.3 0.3 -1.7 0.7 -1.1
$e_{1,2}$
1.6 -0.3 -0.9 -0.7 0.2
$e_{1,3}$
The
dog
barked

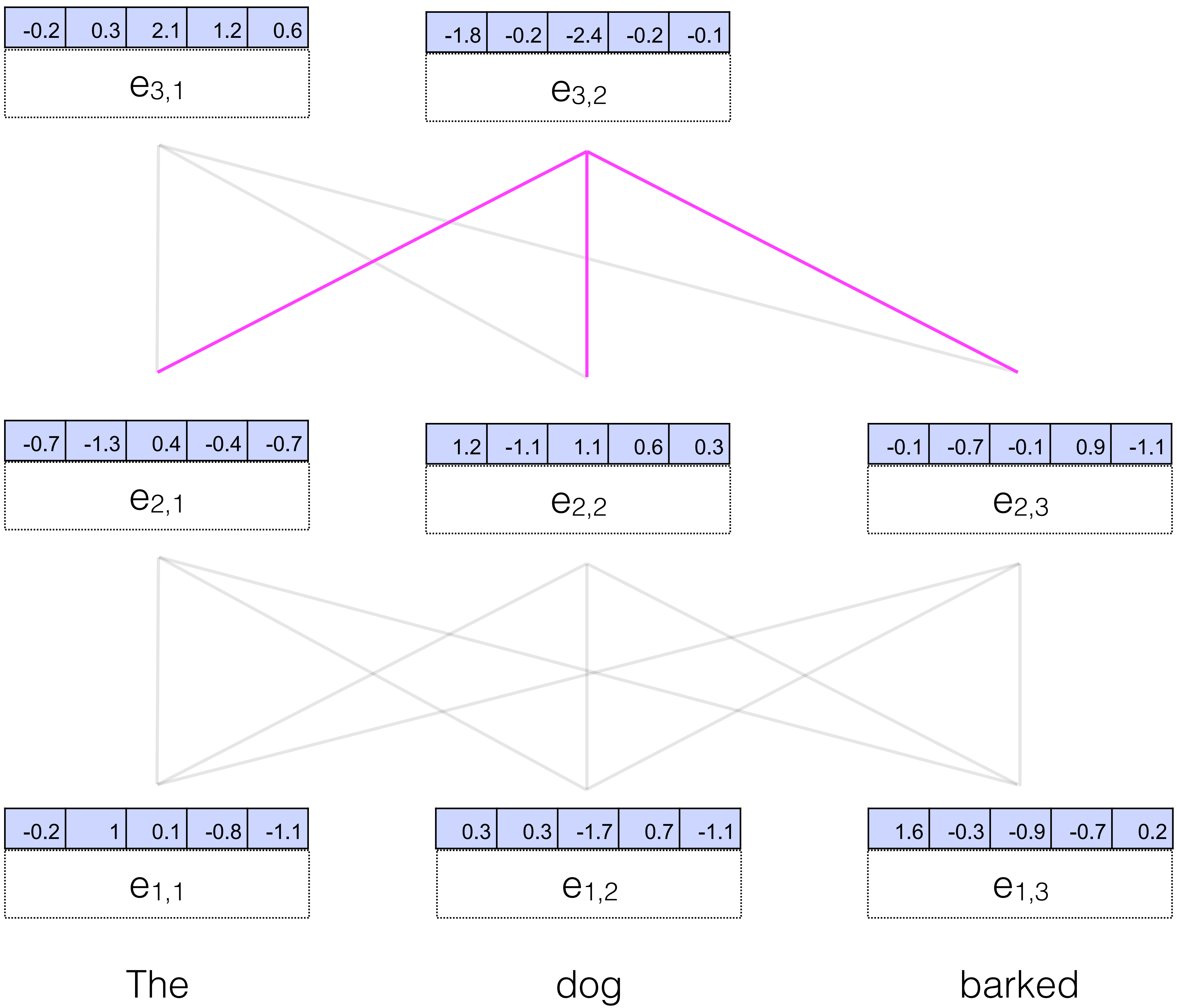


-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
1.2 -1.1 1.1 0.6 0.3
$e_{2,2}$
-0.2 1 0.1 -0.8 -1.1
$e_{1,1}$
0.3 0.3 -1.7 0.7 -1.1
$e_{1,2}$
1.6 -0.3 -0.9 -0.7 0.2
$e_{1,3}$
The
dog
barked

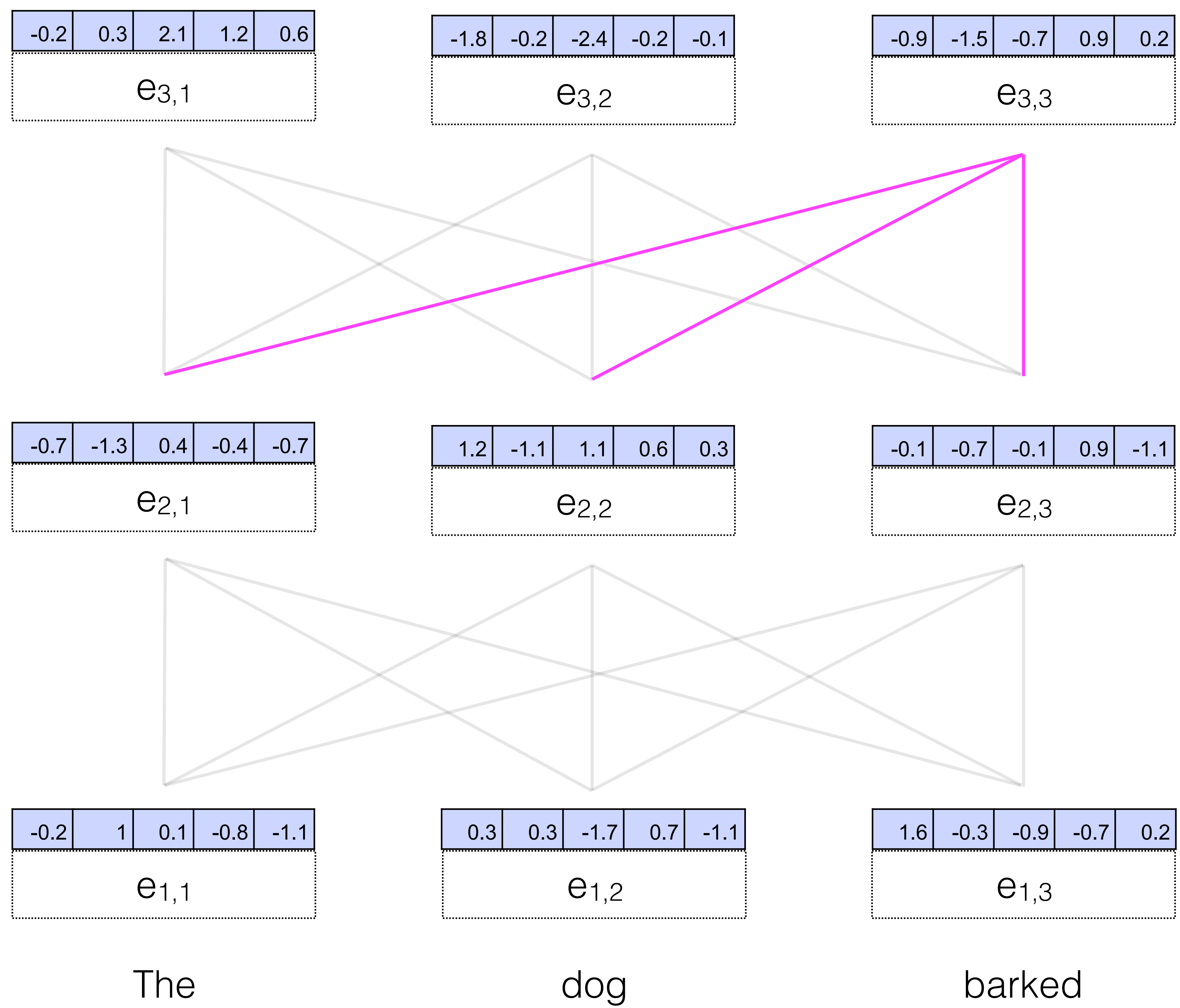

-0.7
-1.3
0.4
-0.4
-0.7
$e_{2,1}$
1.2
-1.1
1.1
0.6
0.3
$e_{2,2}$
-0.1
-0.7
-0.1
0.9
-1.1
$e_{2,3}$
-0.2
1
0.1
-0.8
-1.1
$e_{1,1}$
0.3
0.3
-1.7
0.7
-1.1
$e_{1,2}$
1.6
-0.3
-0.9
-0.7
0.2
$e_{1,3}$
The
dog
barked



-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

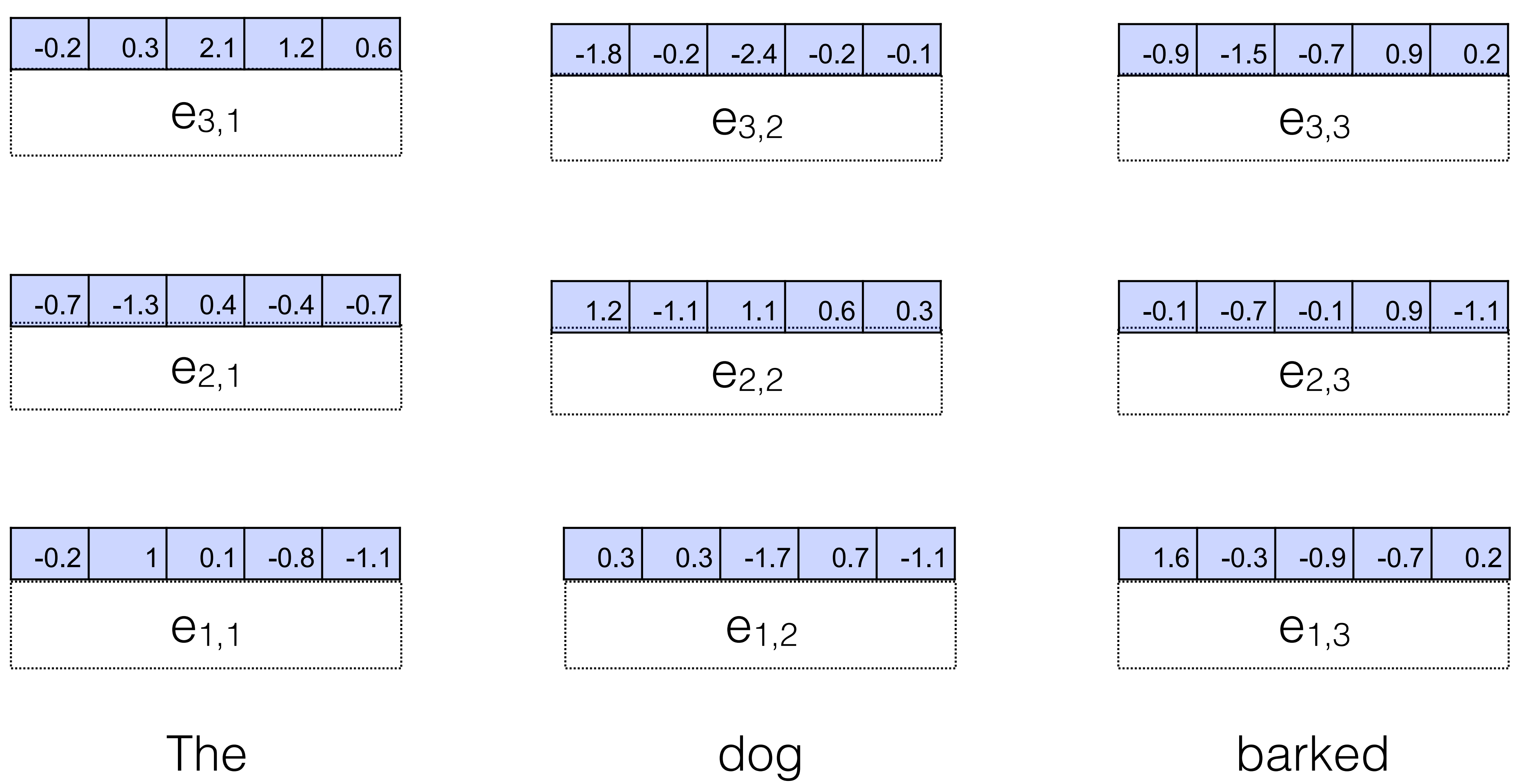


-0.2 0.3 2.1 1.2 0.6
$e_{3,1}$
-1.8 -0.2 -2.4 -0.2 -0.1
$e_{3,2}$
-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
1.2 -1.1 1.1 0.6 0.3
$e_{2,2}$
-0.1 -0.7 -0.1 0.9 -1.1
$e_{2,3}$
-0.2 1 0.1 -0.8 -1.1
$e_{1,1}$
0.3 0.3 -1.7 0.7 -1.1
$e_{1,2}$
1.6 -0.3 -0.9 -0.7 0.2
$e_{1,3}$
The
dog
barked



-0.2 0.3 2.1 1.2 0.6
e3,1
-1.8 -0.2 -2.4 -0.2 -0.1
e3,2
-0.9 -1.5 -0.7 0.9 0.2
e3,3
-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

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



WordPiece

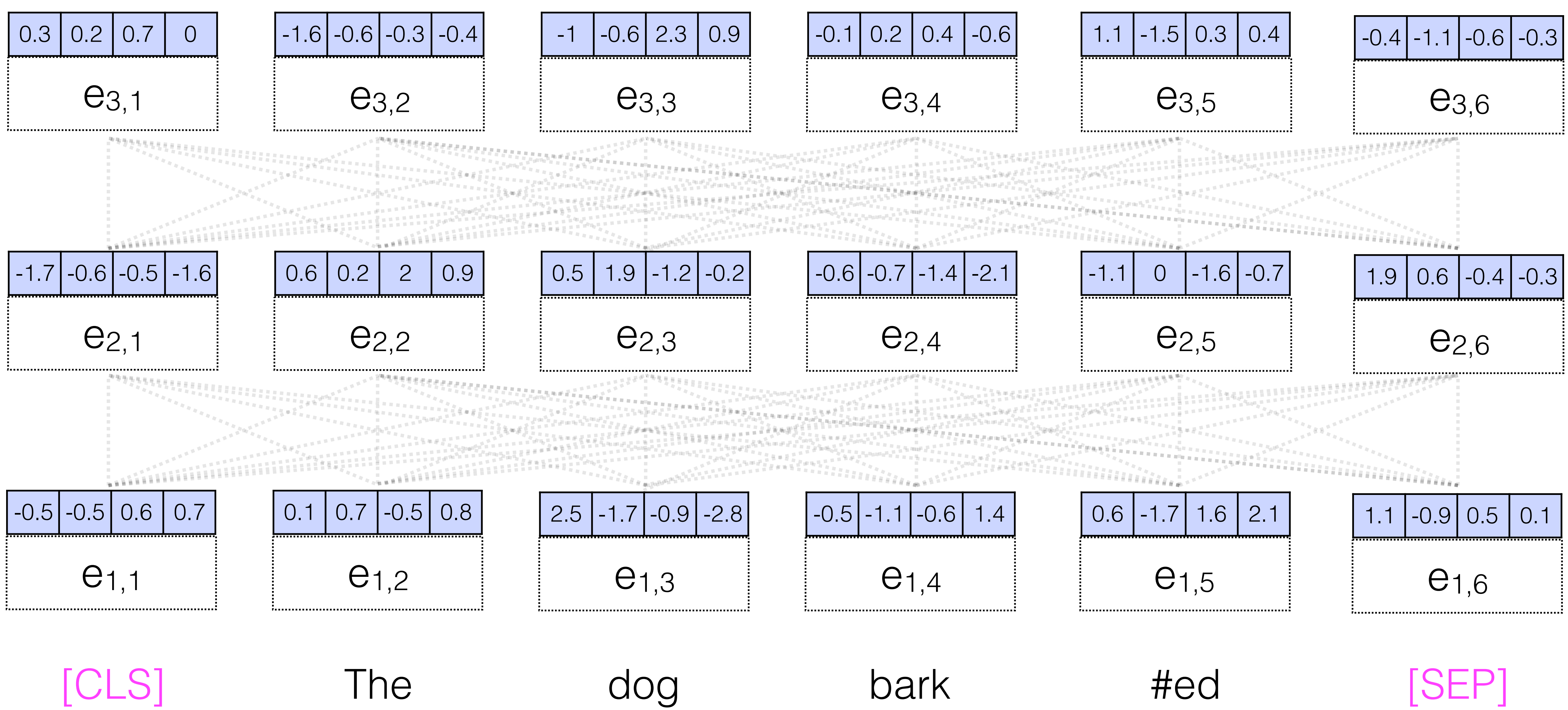
- BERT uses WordPiece tokenization, which segments some morphological structure of tokens

- Vocabulary size: 30,000

The	The
dog	dog
barked	bark #ed

- BERT also encodes each sentence by appending a special token to the beginning ([CLS]) and end ([SEP]) of each sequence.

- This helps provides a single token that can be optimized to represent the entire sequence.

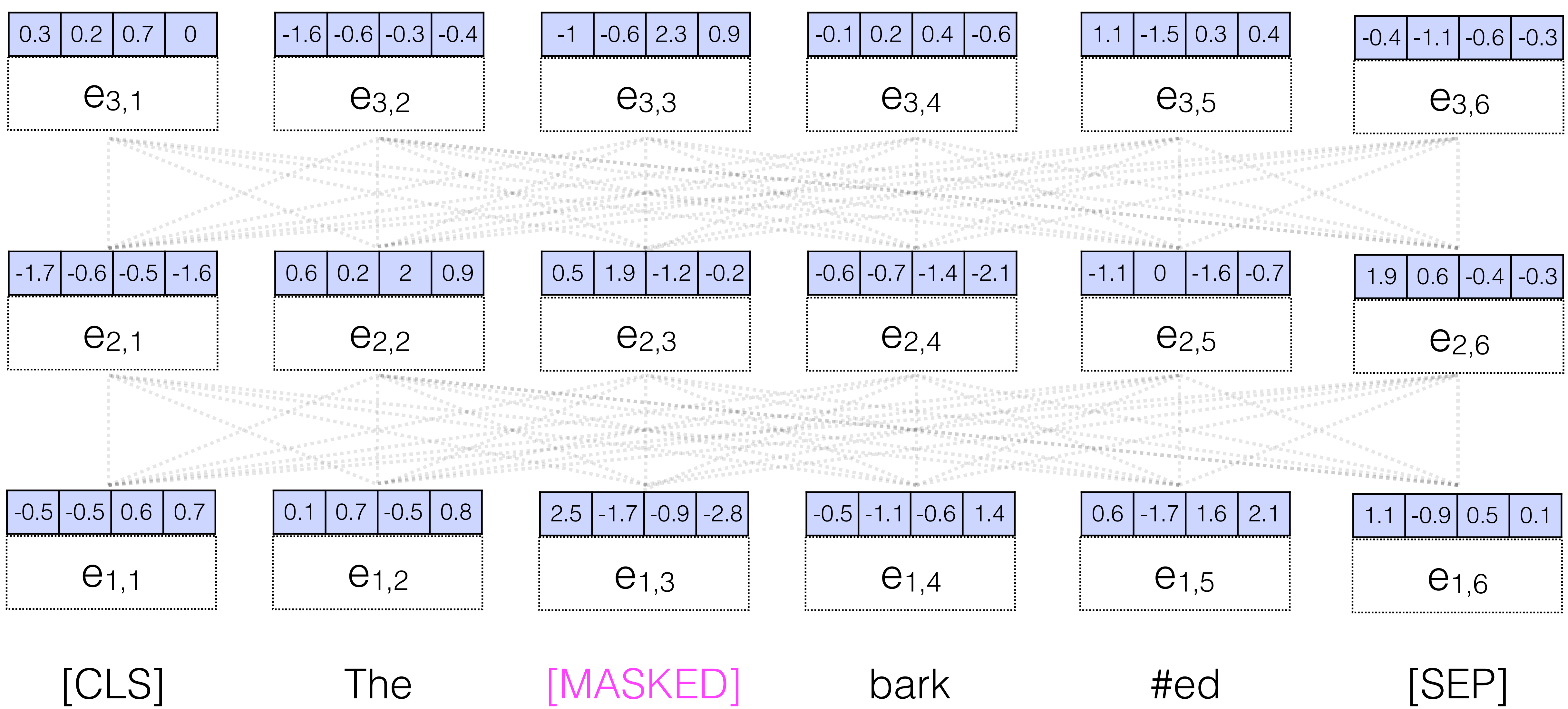


BERT

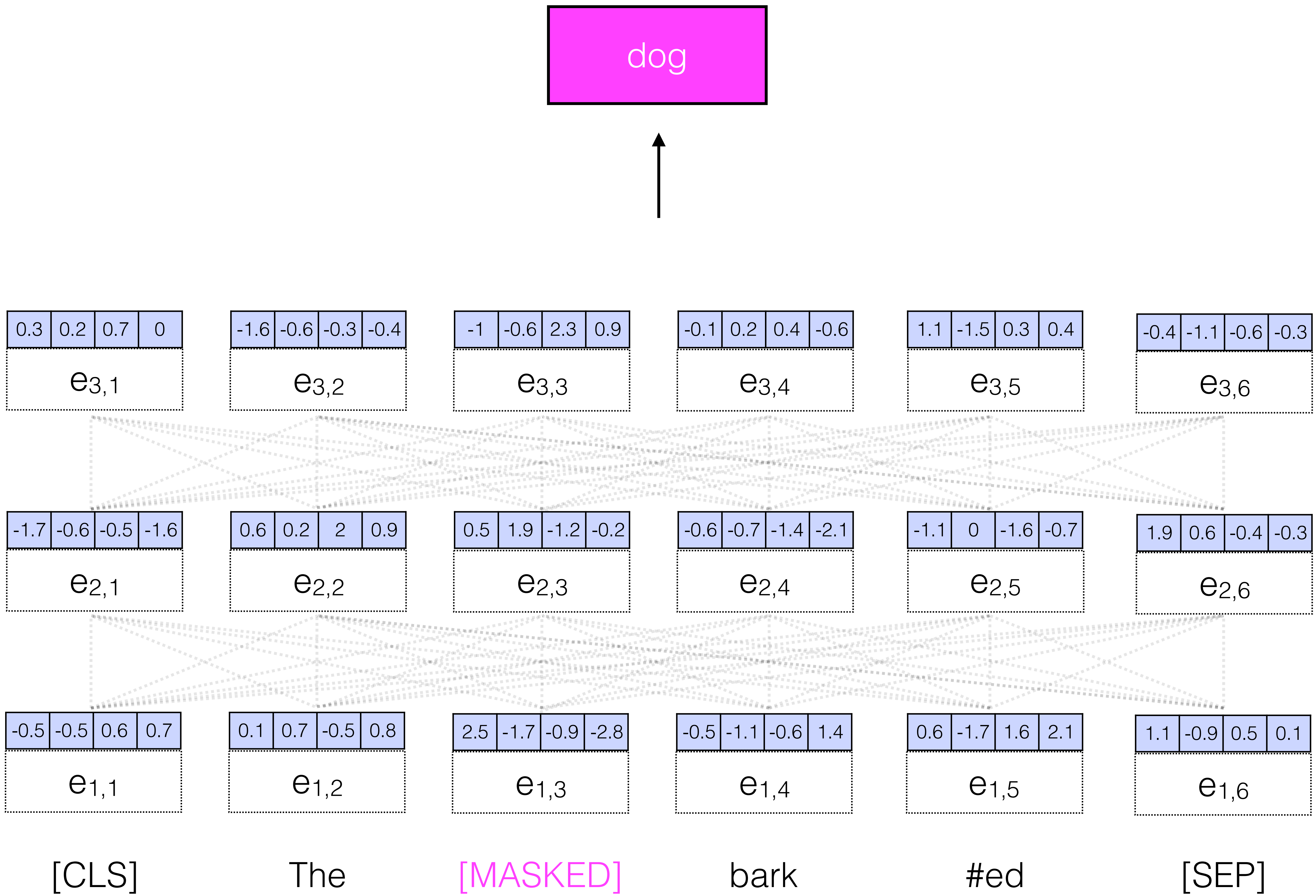
- Learn the parameters of this model with two objectives:
  - Masked language modeling
  - Next sentence prediction

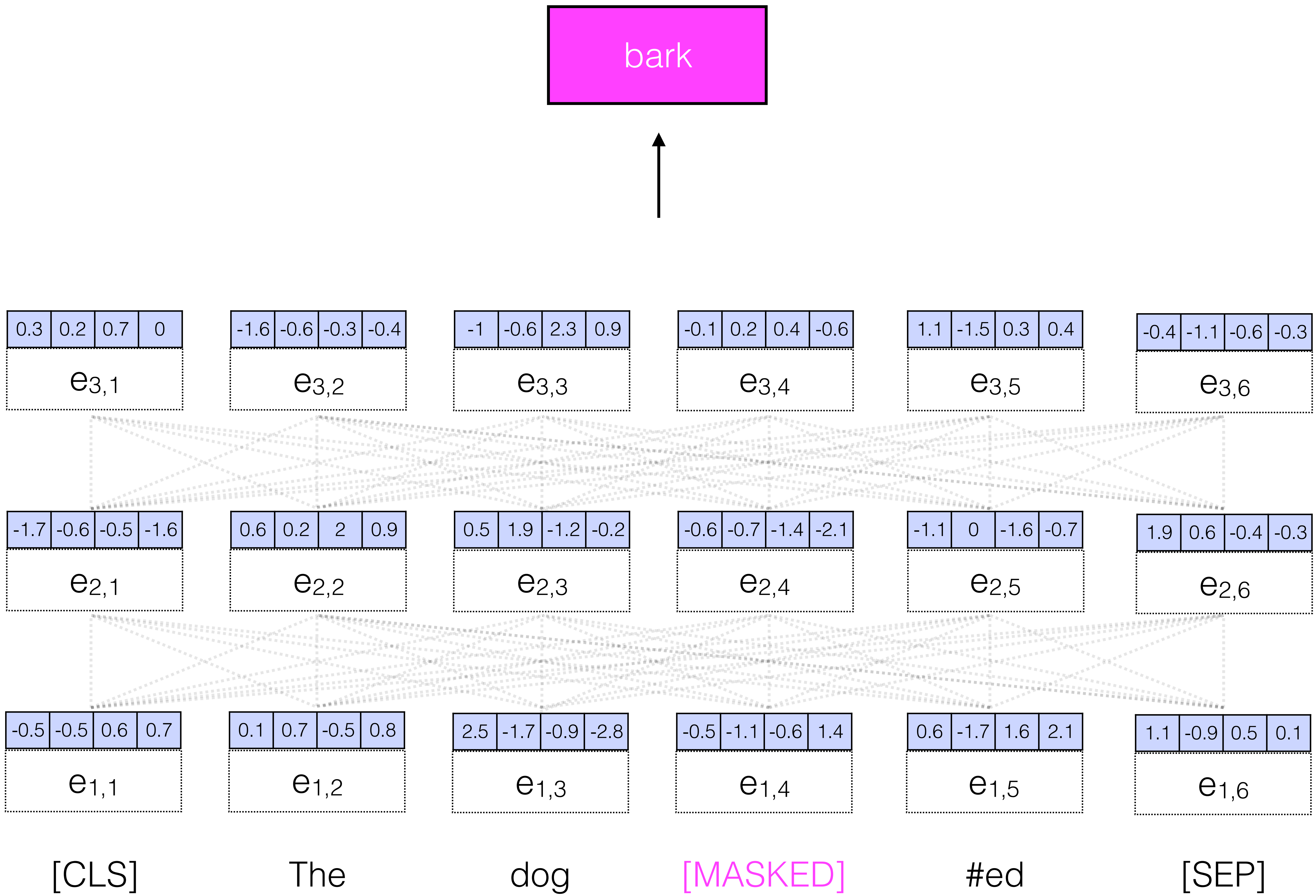
Masked LM

- Mask one word from the input and try to predict that word as the output
- More powerful than an RNN LM (or even a BiRNN LM) since it can reason about context on both sides of the word being predicted.
- A BiRNN models context on both sides, but each RNN only has access to information from one direction.



0.3 0.2 0.7 0
e3,1
-1.6 -0.6 -0.3 -0.4
e3,2
-1 -0.6 2.3 0.9
e3,3
-0.1 0.2 0.4 -0.6
e3,4
1.1 -1.5 0.3 0.4
e3,5
-0.4 -1.1 -0.6 -0.3
e3,6
-1.7 -0.6 -0.5 -1.6
e2,1
0.6 0.2 2 0.9
e2,2
0.5 1.9 -1.2 -0.2
e2,3
-0.6 -0.7 -1.4 -2.1
e2,4
-1.1 0 -1.6 -0.7
e2,5
1.9 0.6 -0.4 -0.3
e2,6
-0.5 -0.5 0.6 0.7
e1,1
0.1 0.7 -0.5 0.8
e1,2
2.5 -1.7 -0.9 -2.8
e1,3
-0.5 -1.1 -0.6 1.4
e1,4
0.6 -1.7 1.6 2.1
e1,5
1.1 -0.9 0.5 0.1
e1,6
[CLS]
The
[MASKED]
bark
#ed
[SEP]


dog
0.3 0.2 0.7 0
e3,1
-1.6 -0.6 -0.3 -0.4
e3,2
-1 -0.6 2.3 0.9
e3,3
-0.1 0.2 0.4 -0.6
e3,4
1.1 -1.5 0.3 0.4
e3,5
-0.4 -1.1 -0.6 -0.3
e3,6
-1.7 -0.6 -0.5 -1.6
e2,1
0.6 0.2 2 0.9
e2,2
0.5 1.9 -1.2 -0.2
e2,3
-0.6 -0.7 -1.4 -2.1
e2,4
-1.1 0 -1.6 -0.7
e2,5
1.9 0.6 -0.4 -0.3
e2,6
-0.5 -0.5 0.6 0.7
e1,1
0.1 0.7 -0.5 0.8
e1,2
2.5 -1.7 -0.9 -2.8
e1,3
-0.5 -1.1 -0.6 1.4
e1,4
0.6 -1.7 1.6 2.1
e1,5
1.1 -0.9 0.5 0.1
e1,6
[CLS]
The
[MASKED]
bark
#ed
[SEP]



bark
0.3 0.2 0.7 0
e3,1
-1.6 -0.6 -0.3 -0.4
e3,2
-1 -0.6 2.3 0.9
e3,3
-0.1 0.2 0.4 -0.6
e3,4
1.1 -1.5 0.3 0.4
e3,5
-0.4 -1.1 -0.6 -0.3
e3,6
-1.7 -0.6 -0.5 -1.6
e2,1
0.6 0.2 2 0.9
e2,2
0.5 1.9 -1.2 -0.2
e2,3
-0.6 -0.7 -1.4 -2.1
e2,4
-1.1 0 -1.6 -0.7
e2,5
1.9 0.6 -0.4 -0.3
e2,6
-0.5 -0.5 0.6 0.7
e1,1
0.1 0.7 -0.5 0.8
e1,2
2.5 -1.7 -0.9 -2.8
e1,3
-0.5 -1.1 -0.6 1.4
e1,4
0.6 -1.7 1.6 2.1
e1,5
1.1 -0.9 0.5 0.1
e1,6
[CLS]
The
dog
[MASKED]
#ed
[SEP]

Next sentence prediction

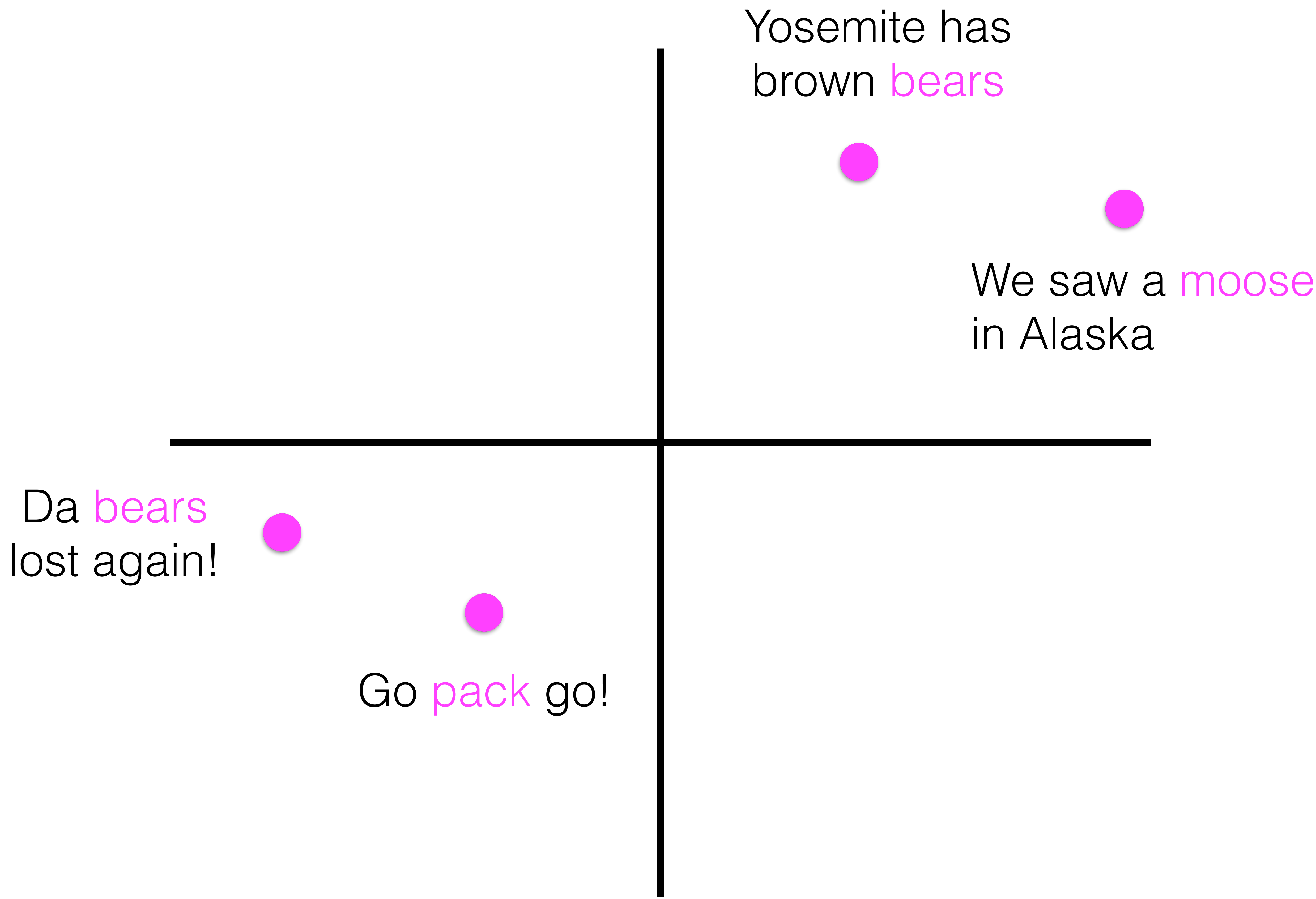
- For a pair of sentences, predict from [CLS] representation whether they appeared sequentially in the training data:

  + [CLS] The dog bark #ed [SEP] He was hungry

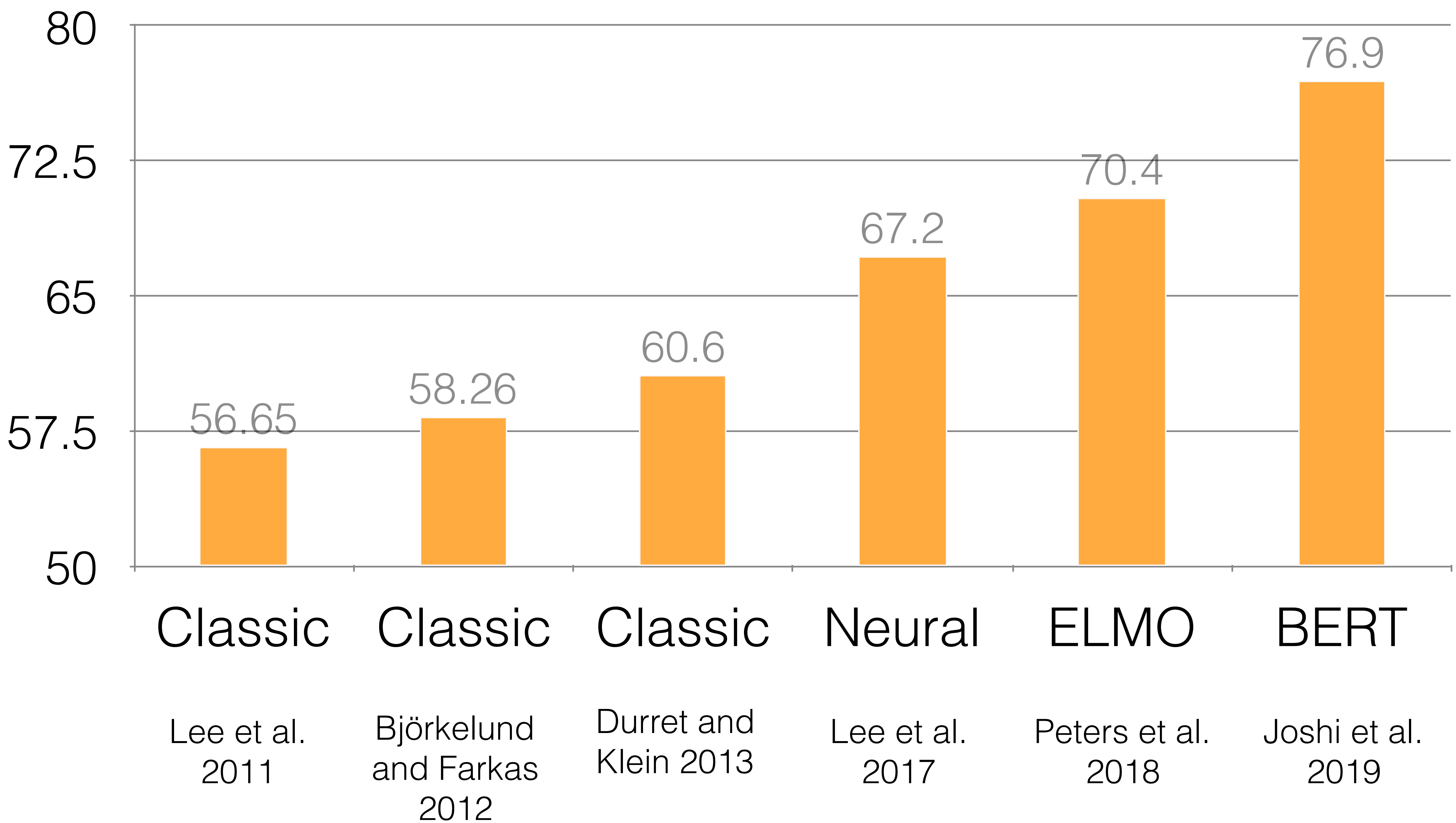
  − [CLS] The dog bark #ed [SEP] Paris is in France

BERT

- Deep layers (12 for BERT base, 24 for BERT large)
- Large representation sizes (768 per layer)
- Pretrained on English Wikipedia (2.5B words) and BooksCorpus (800M words)


Yosemite has brown bears
We saw a moose in Alaska
Da bears lost again!
Go pack go!

Progress — Coreference resolution



Bertology

- Hewitt et al. 2019
- Tenney et al. 2019
- McCoy et al. 2019
- Liu et al. 2019
- Clark et al. 2019
- Goldberg 2019
- Michel et al. 2019

Code

Pre-trained models for BERT, Transformer-XL, ALBERT, RoBERTa, DistilBERT, GPT-2, etc. for English, French, "Multilingual"

https://huggingface.co

- Word embeddings can be substituted for one-hot encodings in many models (MLP, CNN, RNN, logistic regression).

- Subword embeddings allow you to create embeddings for word not present in training data; require much less data to train.

- Attention gives us a mechanism to learn which parts of a sequence to pay attention to more in forming a representation of it.

- BiLSTMs can transform word embeddings to be sensitive to their use in context.

- Static word embeddings (word2vec, Glove) provide representations of word types; contextualized word representations (ELMo, BERT) provide representations of tokens in context.