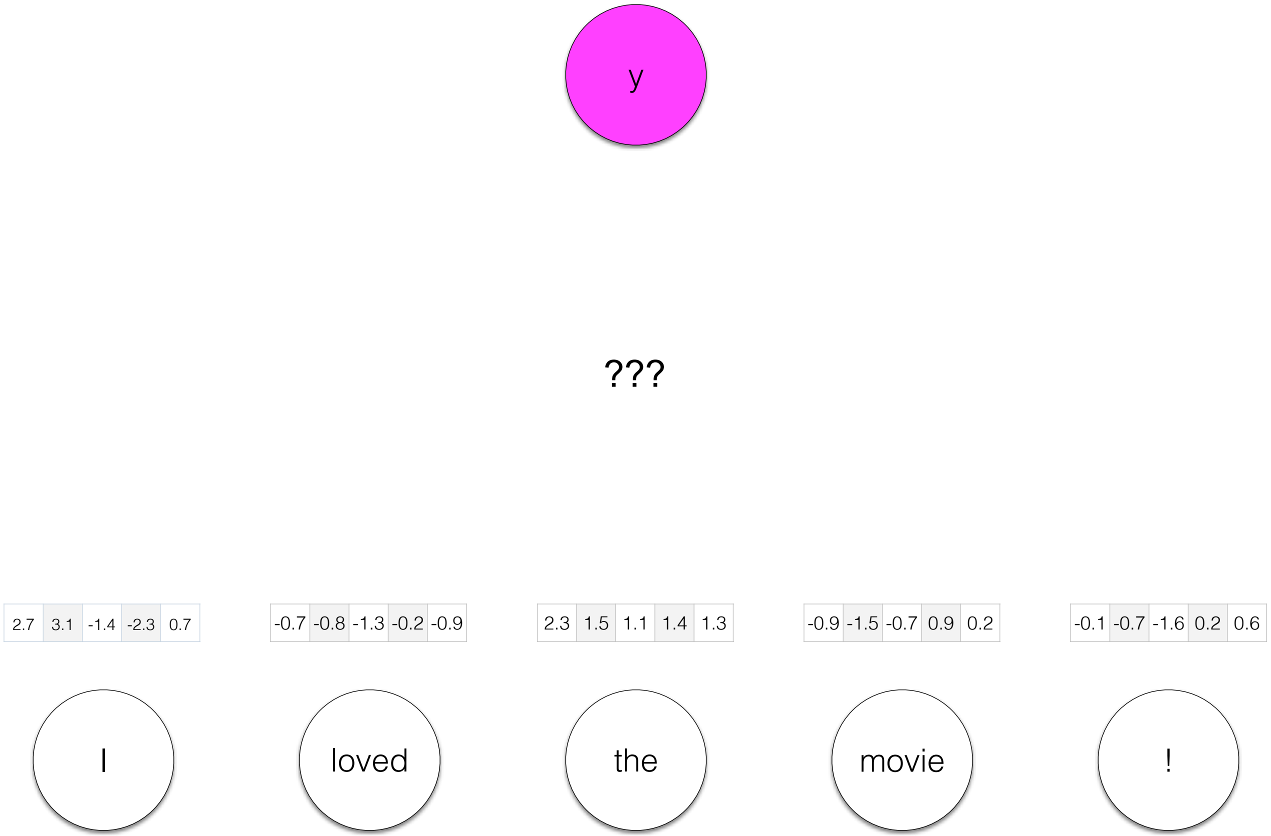
Applied Natural Language Processing

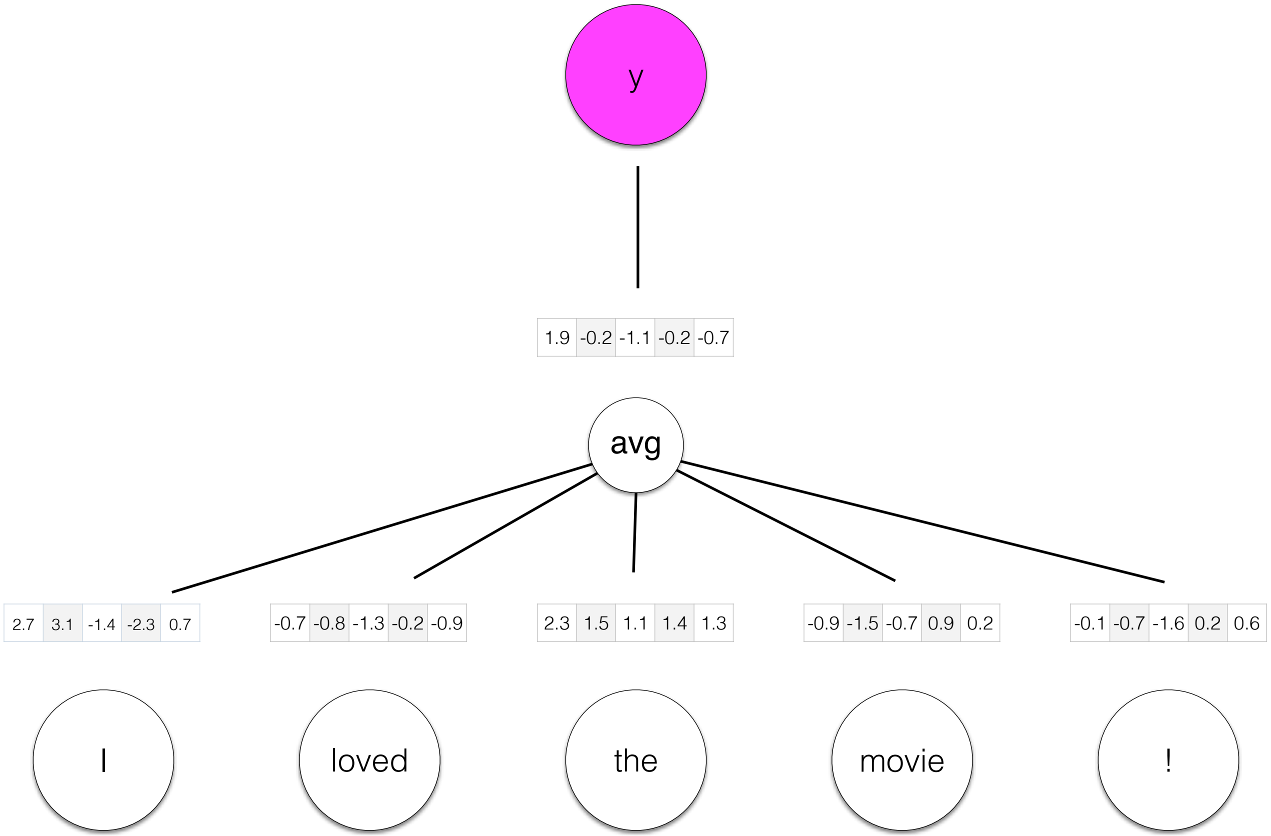
Info 256

Lecture 13: Transformers 2 (Oct 9, 2023)

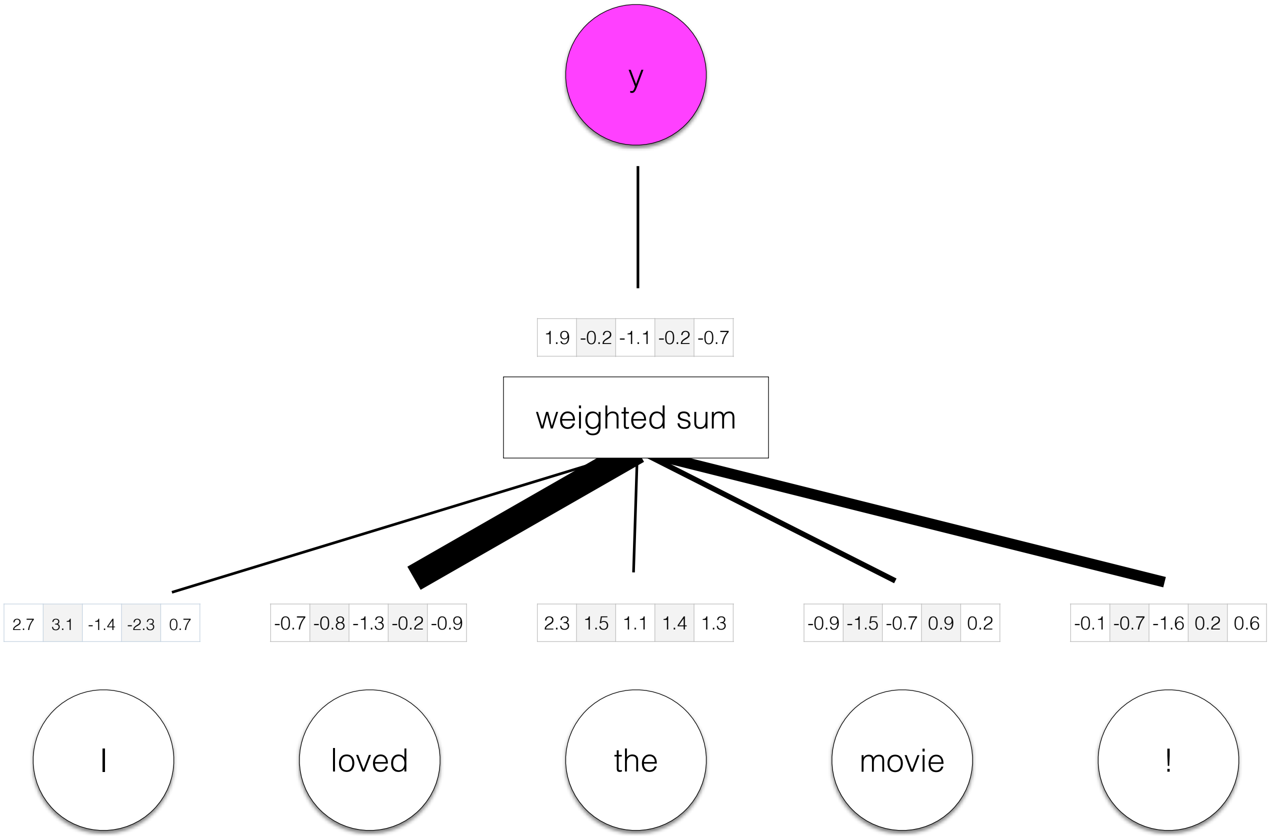
David Bamman, UC Berkeley

How do we use word embeddings for document classification?


y
???
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
!



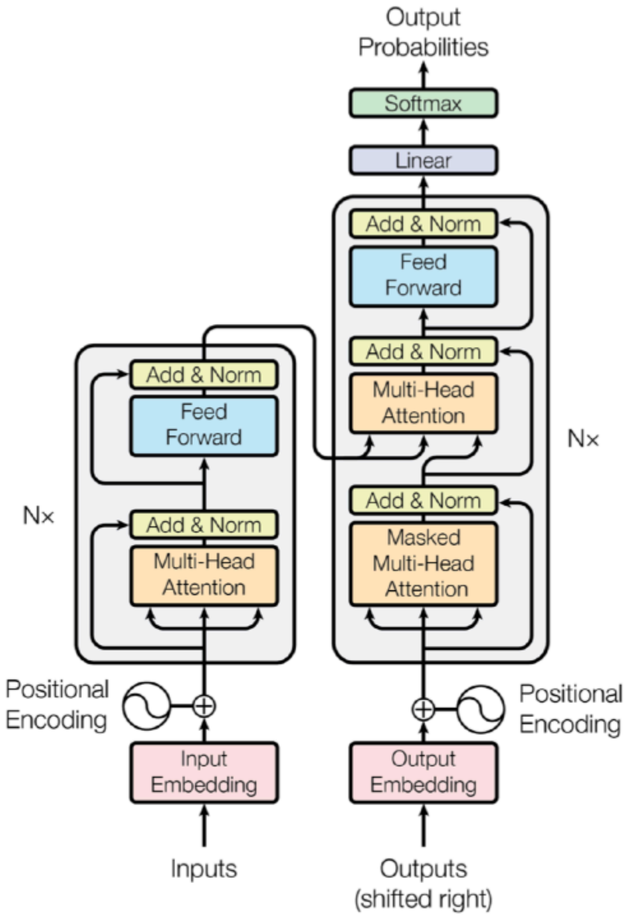
Iyyer et al. (2015), “Deep Unordered Composition Rivals Syntactic Methods for Text Classification” (ACL)


y
1.9 -0.2 -1.1 -0.2 -0.7
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
!

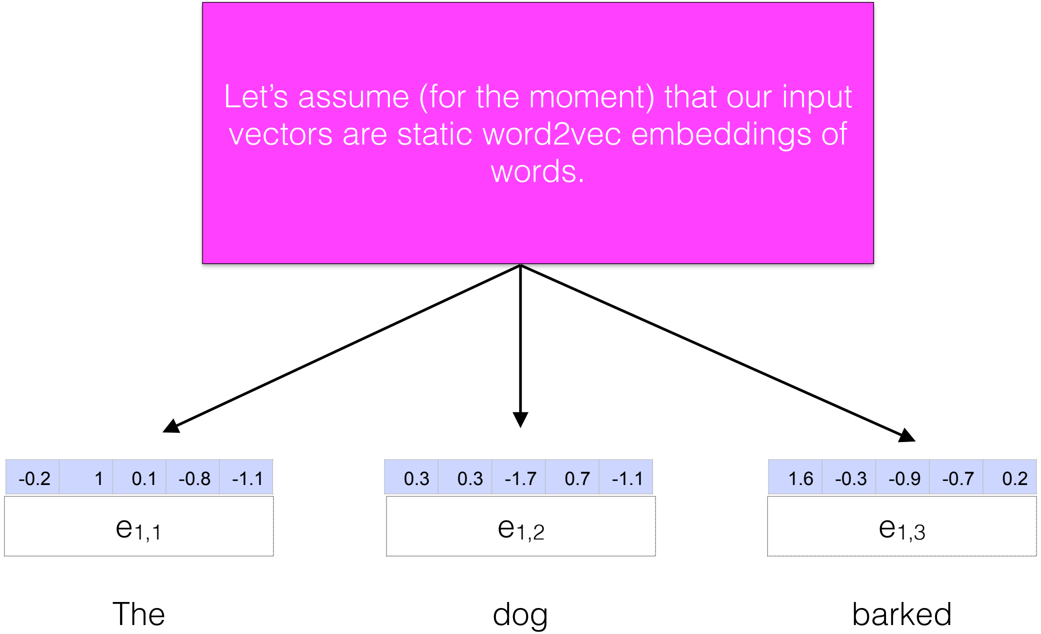
Transformers

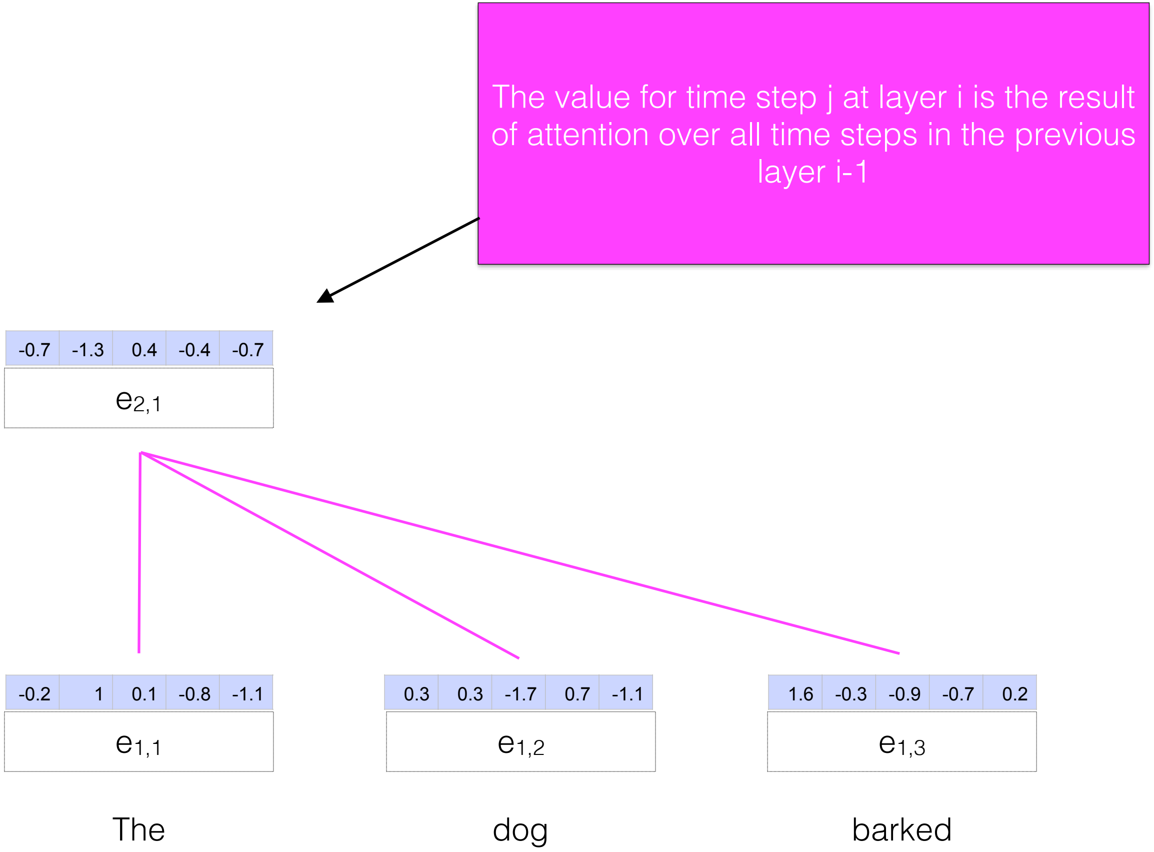
- Vaswani et al. 2017, “Attention is All You Need”

- Transforms map an input sequence of vectors to an output sequence of vectors of the same dimensionality



Self-Attention




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

- Let's separate out the different functions that an input vector has in attention by transforming it into separate representations for its role in a weighted sum (the value) from the roles used to assess compatibility (the query and key).

query $\quad q_{1,1} \in \mathbb{R}^{37} \quad (e_{1,1}W^Q)$

key $\quad k_{1,1} \in \mathbb{R}^{37} \quad (e_{1,1}W^K)$

value $\quad v_{1,1} \in \mathbb{R}^{100} \quad (e_{1,1}W^V)$

original value $\quad e_{1,1} \in \mathbb{R}^{100}$

$e_{1,1}$

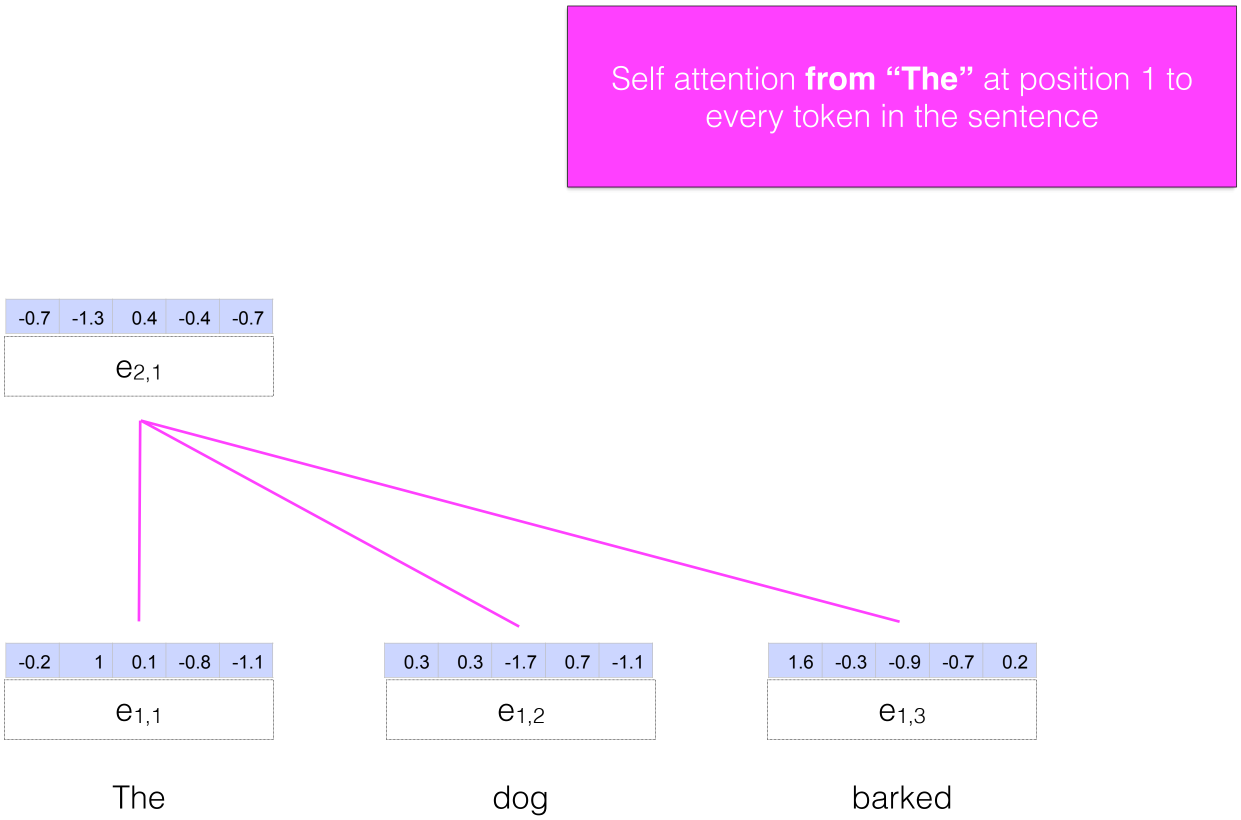
The

$$W^Q \in \mathbb{R}^{100\times 37}$$

$$W^K \in \mathbb{R}^{100\times 37}$$

$$W^V \in \mathbb{R}^{100\times 100}$$

These are all parameters we *learn*. 100 is the original input dimension; 37 is a hyper-parameter we choose.


Self attention **from “The”** at position 1 to every token in the sentence
-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

- The compatibility score between two words is the dot product between their respective query and key vectors.

$$score(e_i, e_j) = q_i \cdot k_j$$

	The	dog	barked	
a	0.07	0.58	0.35	a = softmax(scores)
scores	-1.4	0.64	0.14	
	$q_1 \cdot k_1$	$q_1 \cdot k_2$	$q_1 \cdot k_3$	

-0.2	1	0.1	-0.8	-1.1

$e_{1,1}$

The

0.3	0.3	-1.7	0.7	-1.1

$e_{1,2}$

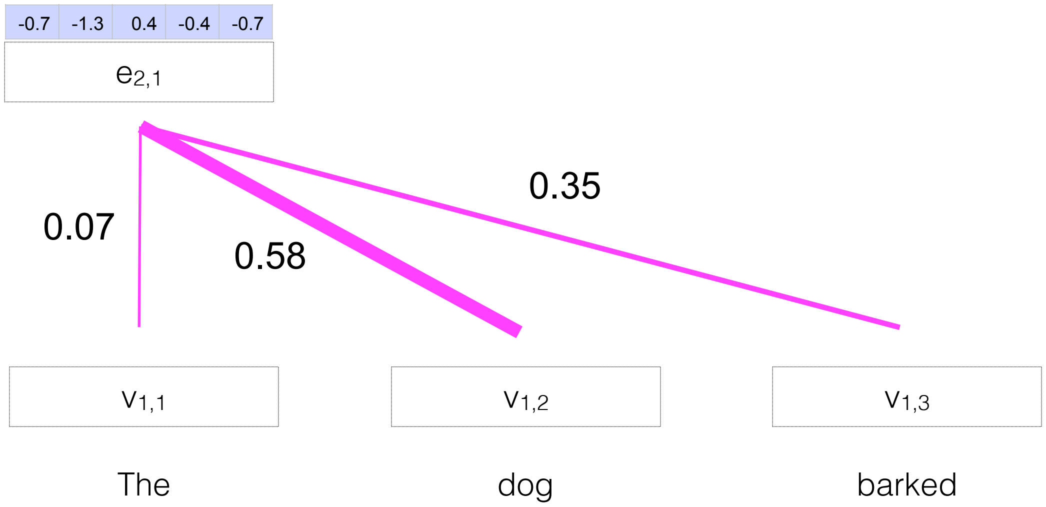
dog

1.6	-0.3	-0.9	-0.7	0.2

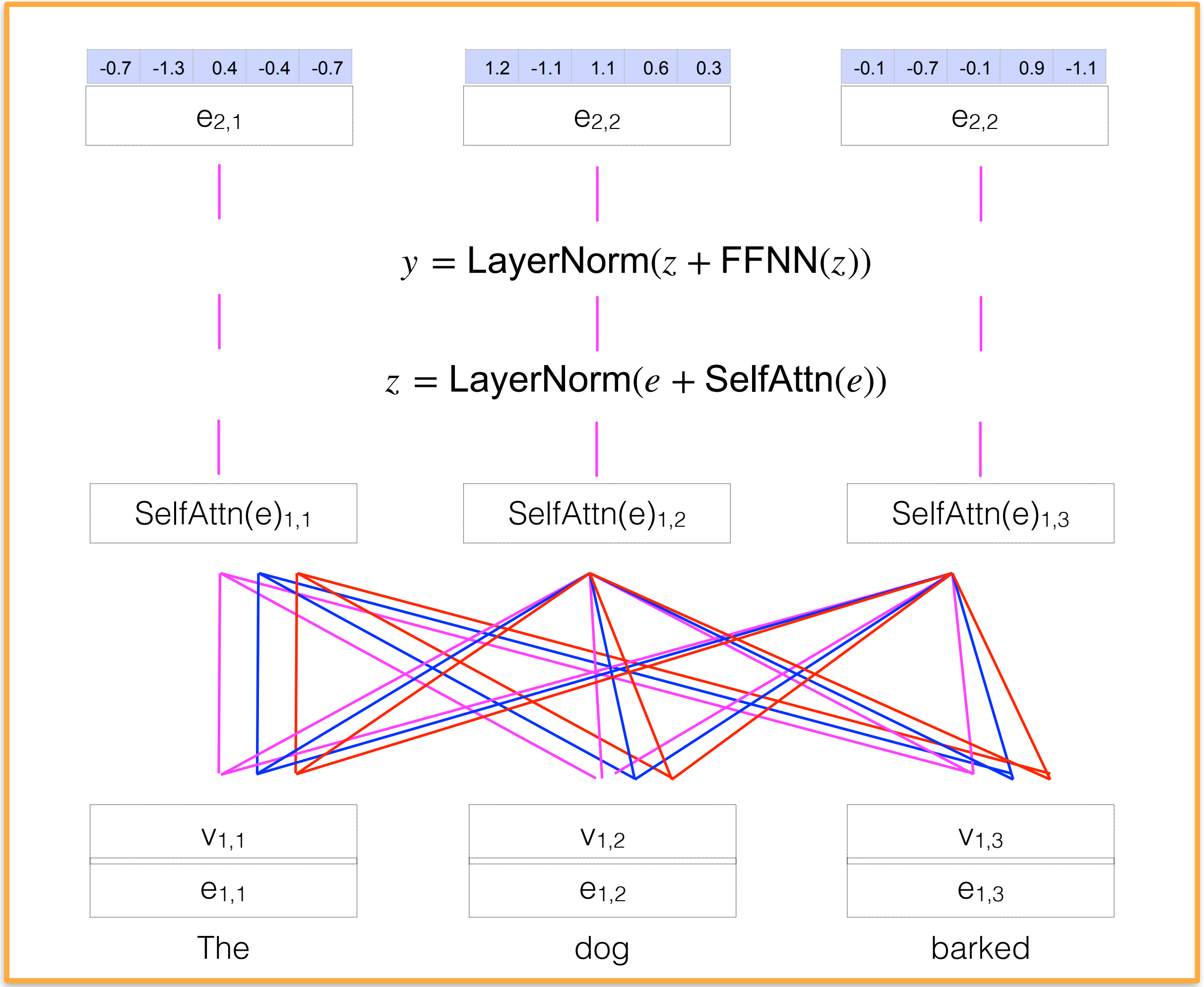
$e_{1,3}$

barked

- The output of attention is a weighted sum over the values of the previous layer.



Output

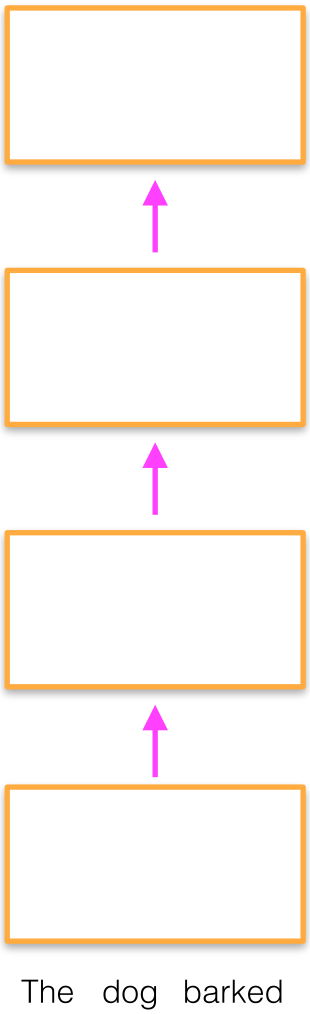


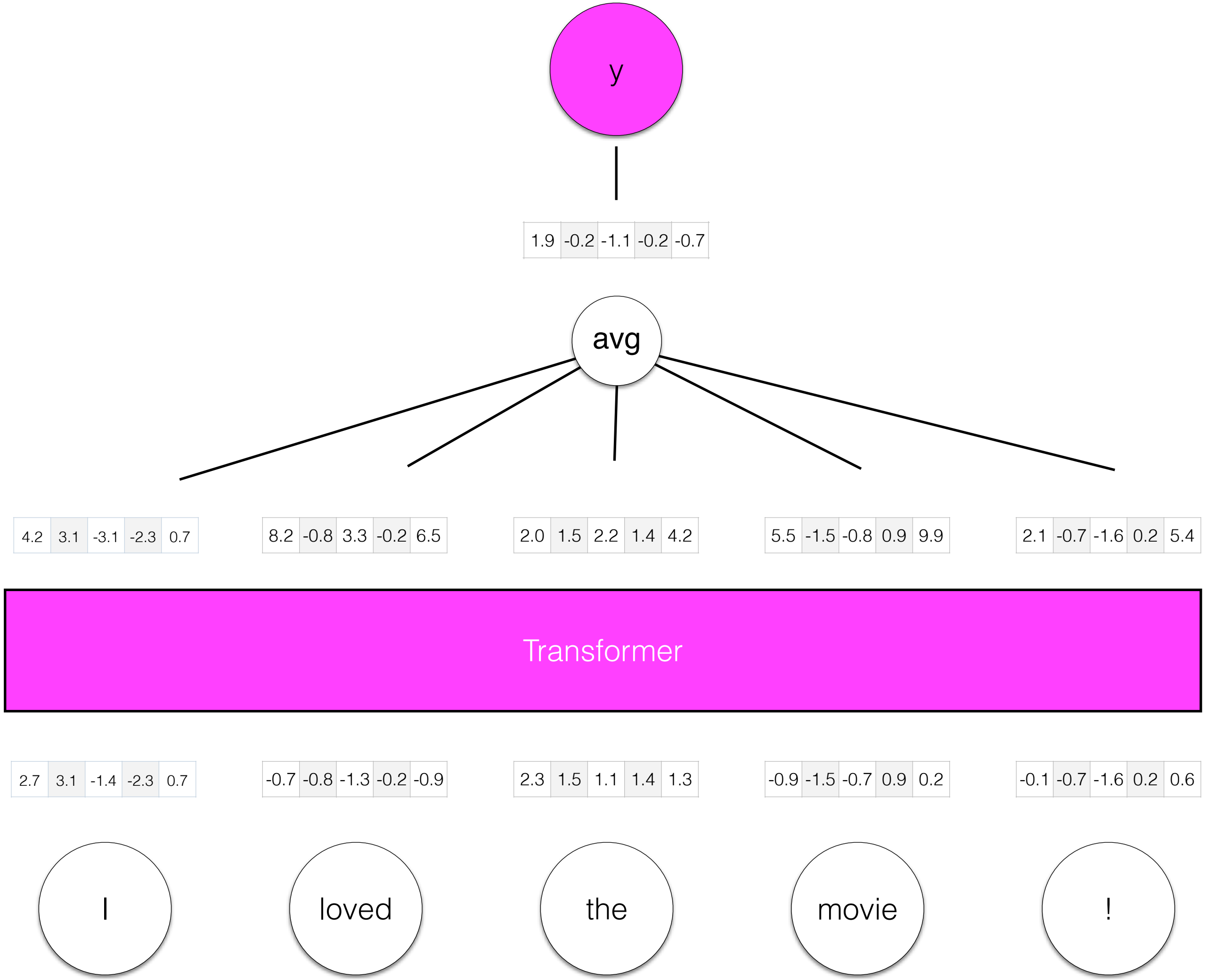
This whole process defines one attention block. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

Input

This whole process defines one attention block. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

Transformers can stack many such blocks; where the output from block b is the input to block b+1.




y
1.9 -0.2 -1.1 -0.2 -0.7
avg
4.2 3.1 -3.1 -2.3 0.7
8.2 -0.8 3.3 -0.2 6.5
2.0 1.5 2.2 1.4 4.2
5.5 -1.5 -0.8 0.9 9.9
2.1 -0.7 -1.6 0.2 5.4
Transformer
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
!

Data

- In this setup, a transformer has to learn *everything* from the labeled training data — including the fundamentals of the language (e.g., that “love” and “like” function similarly in determining sentiment).

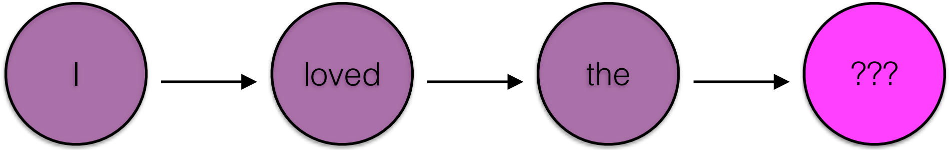
- If a word is not observed in the labeled training data (e.g., “adore”), then a model has no idea what to do with it (UNK).

Enter Language Models

- Language modeling is the task of estimating P(w)
- We considered these in the context of count-and-normalize LMs that make a Markov assumption in order to make estimation tractable.
- But there are many other models that we can use to perform language modeling.

Classical (causal) language model

Consider only the left context to predict the next word (i.e., the final word in a sequence is *masked*)



$$P\left(w_t \mid w_1, \ldots, w_{t-1}\right)$$

Markov LMs

bigram model
(first-order markov)

$$\prod_{i}^{n} P(w_i \mid w_{i-1}) \times P(\text{STOP} \mid w_n)$$

trigram model
(second-order markov)

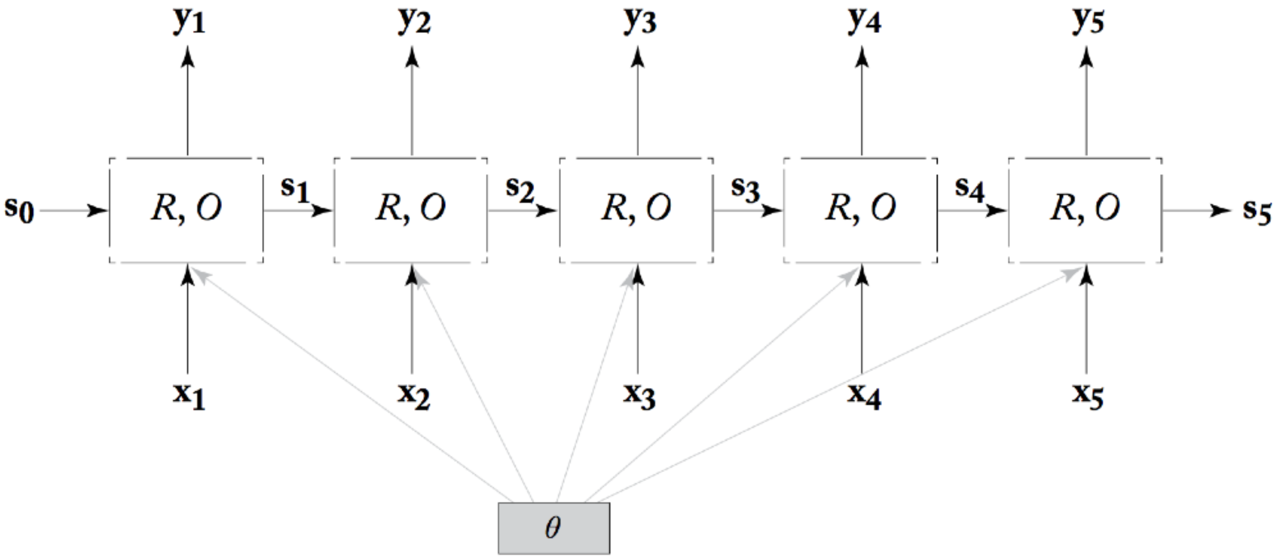
$$\prod_{i}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

$$\times P(\text{STOP} \mid w_{n-1}, w_n)$$

Recurrent neural network

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

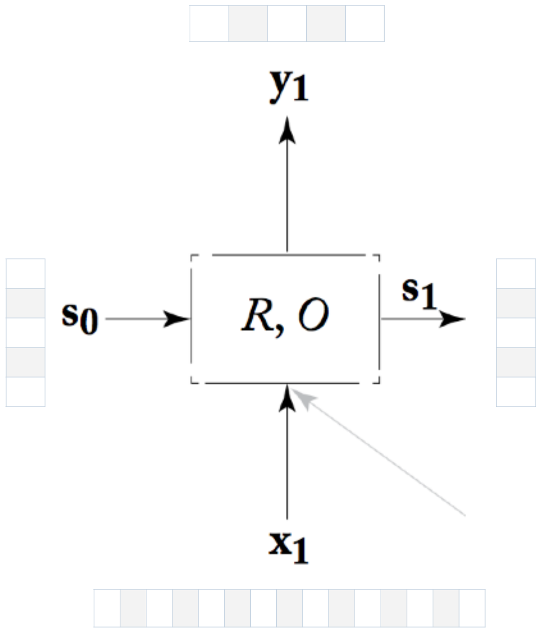
Recurrent neural network



Goldberg 2017

Recurrent neural network

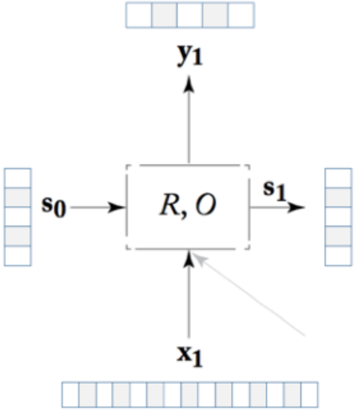
- Each time step has two inputs:
  - x_i (the observation at time step i); one-hot vector, feature vector or distributed representation.
  - s_{i-1} (the output of the previous state); base case: s_0 = 0 vector



RNN LM

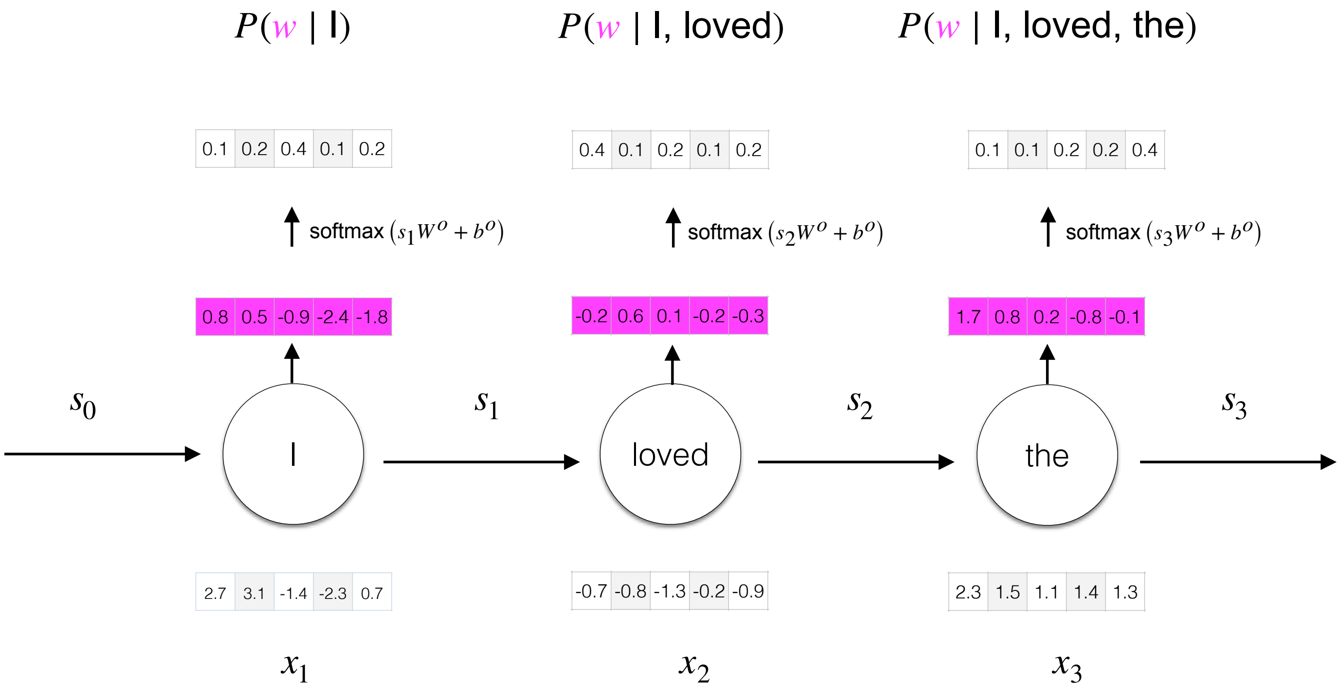
$$s_i = \text{relu}(\overbrace{s_{i-1}}^{\text{previous state}} \underbrace{W^s}_{\mathbb{R}^{H\times H}} + \overbrace{x_i}^{\text{current word}} \underbrace{W^x}_{\mathbb{R}^{D\times H}} + \underbrace{b}_{\mathbb{R}^{H}})$$

$$y_i = \text{softmax}(\overbrace{s_i}^{\text{current state}} \underbrace{W^o}_{\mathbb{R}^{H\times V}} + \underbrace{b^o}_{\mathbb{R}^{V}})$$



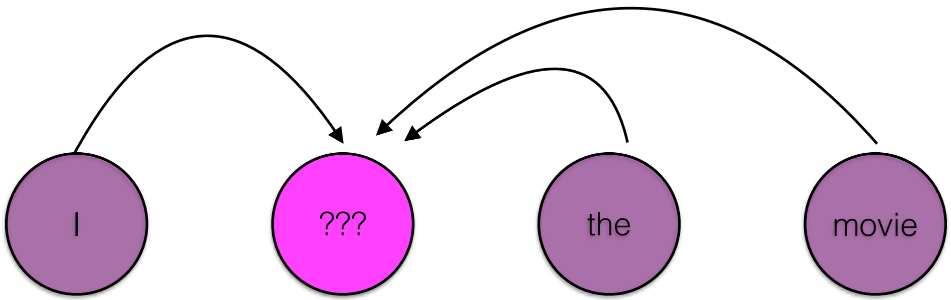
Elman 1990, Mikolov 2012

RNN Language model



Masked language model

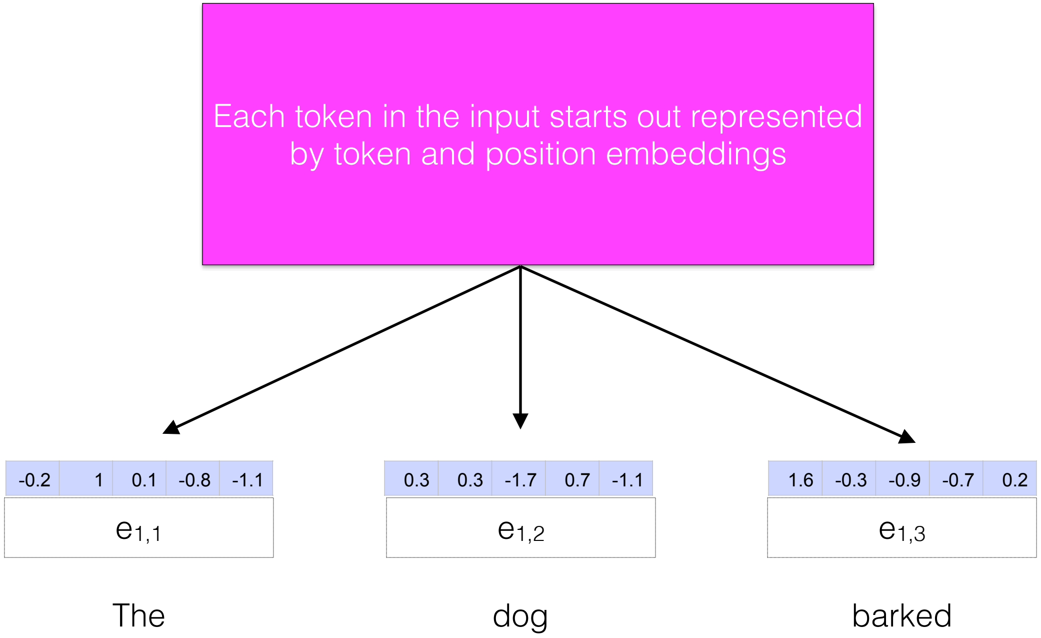
Use any context (left or right) to predict a masked word



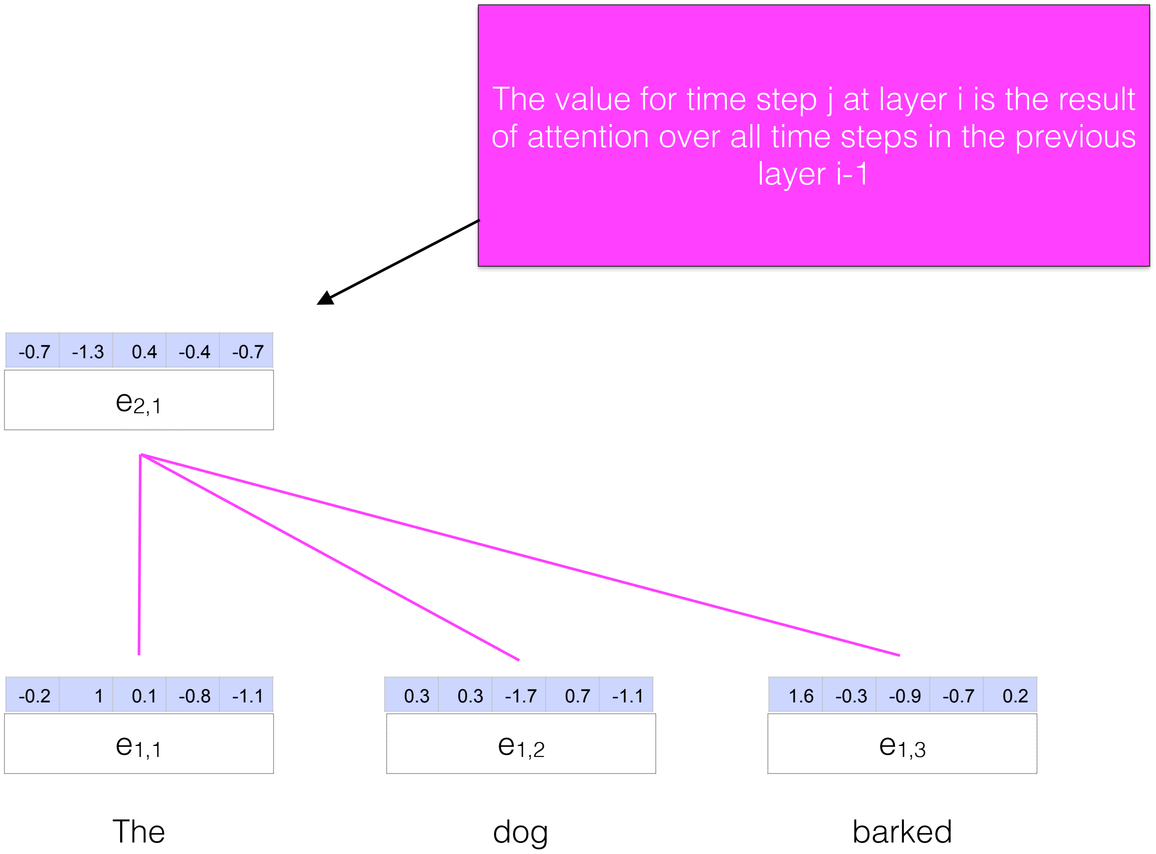
$$P\left(w_t \mid w_{\neg t}\right)$$

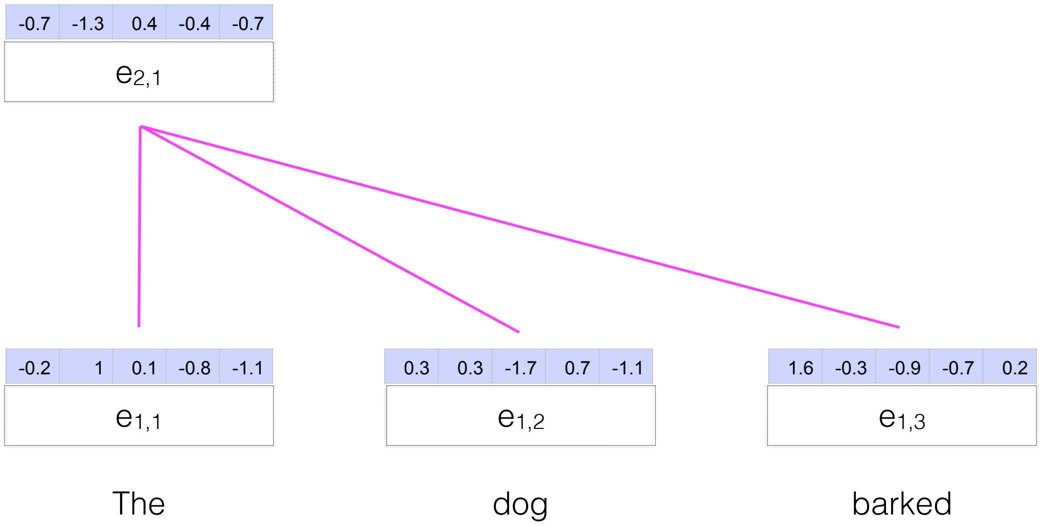
BERT

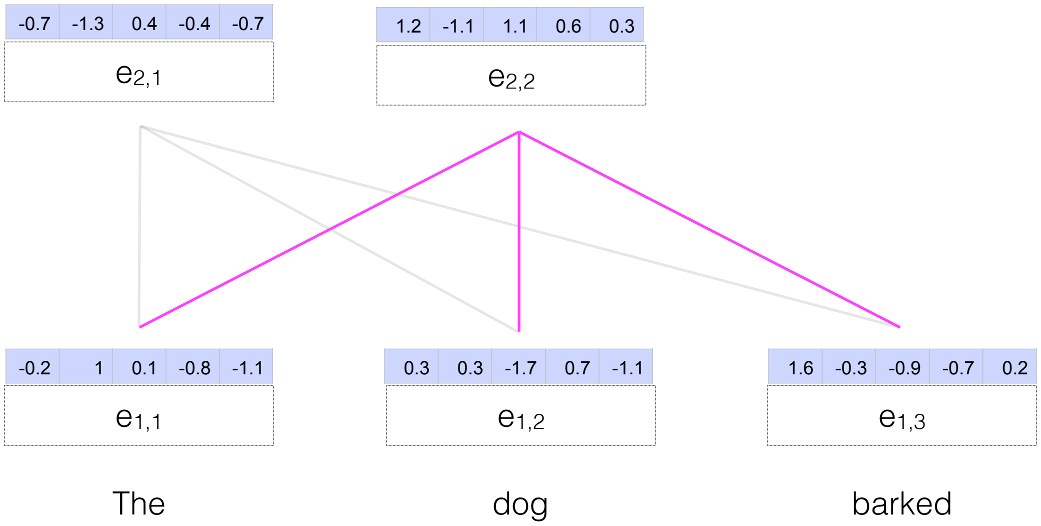
- Transformer-based model (Vaswani et al. 2017) to predict masked word using bidirectional context + next sentence prediction.
- Core idea: use attention mechanism to learn which words in the context to pay more attention to.
- Generates multiple layers of representations for each token sensitive to its context of use.



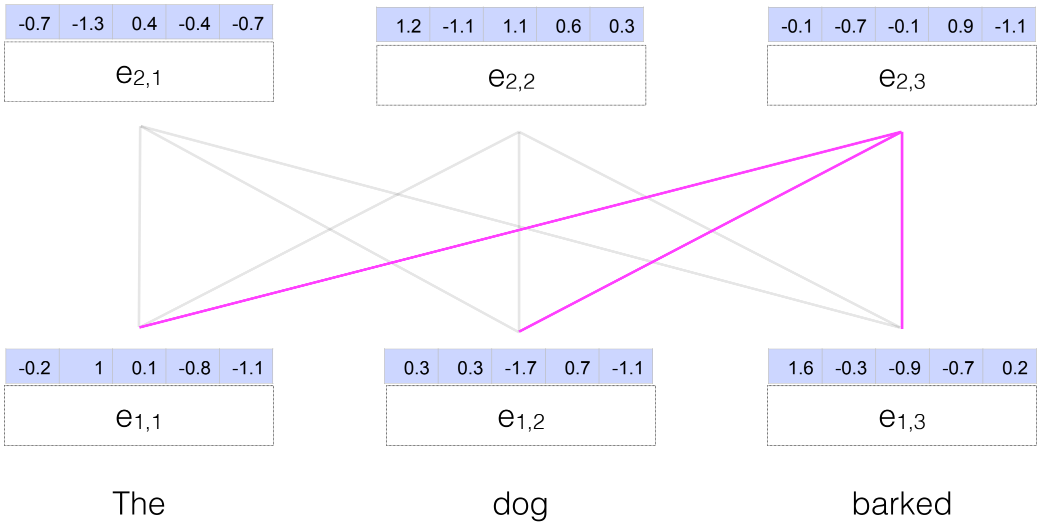
Each token in the input starts out represented by token and position embeddings
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


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

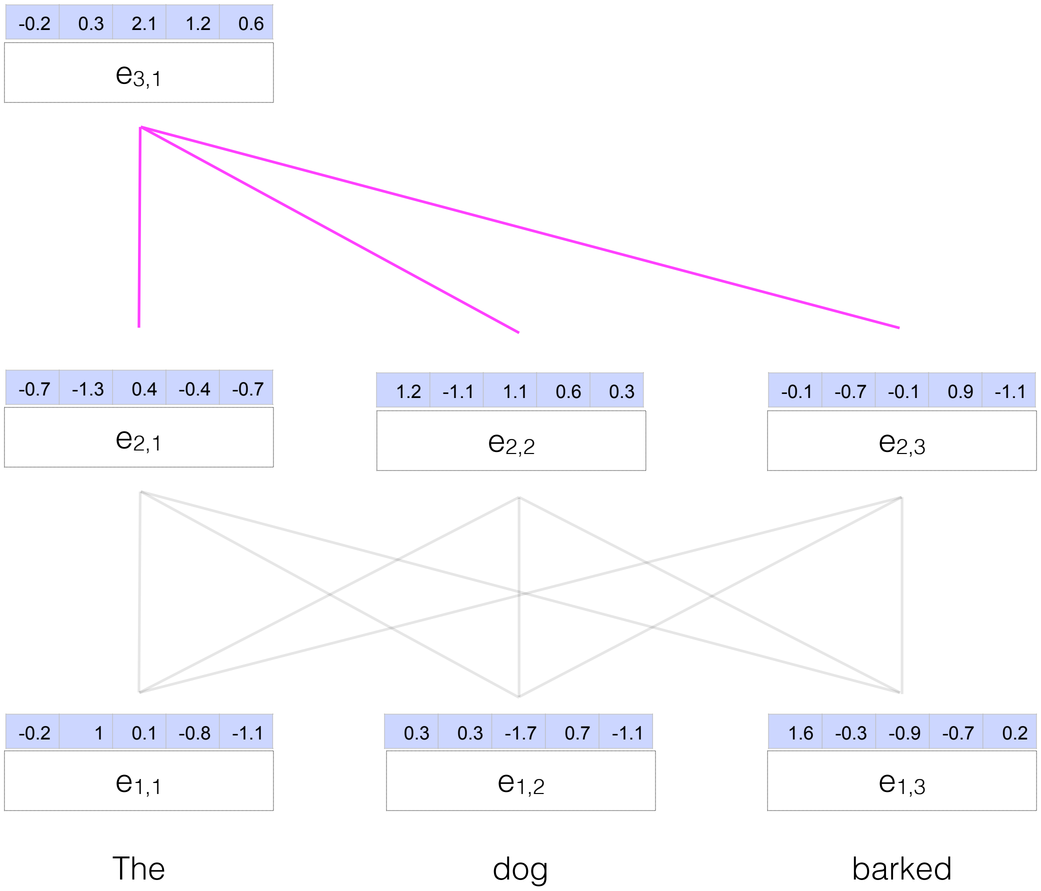

-0.7 -1.3 0.4 -0.4 -0.7
e2,1
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked



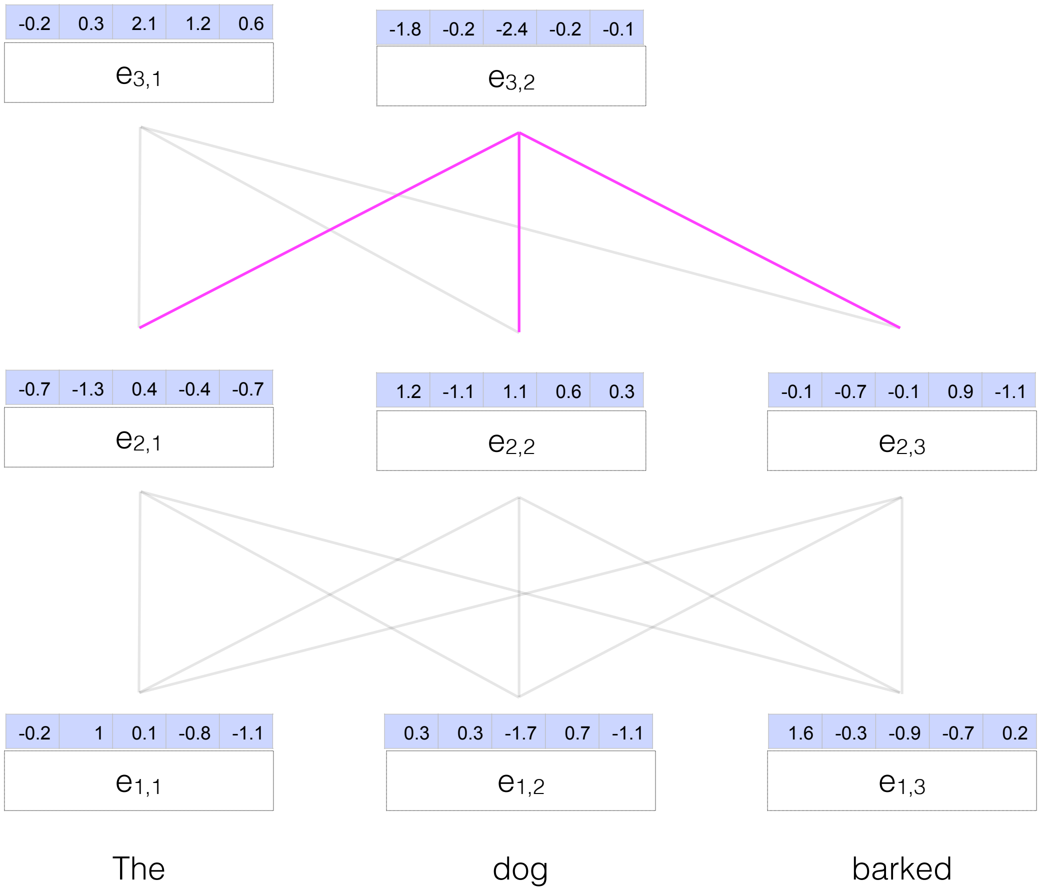
-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked



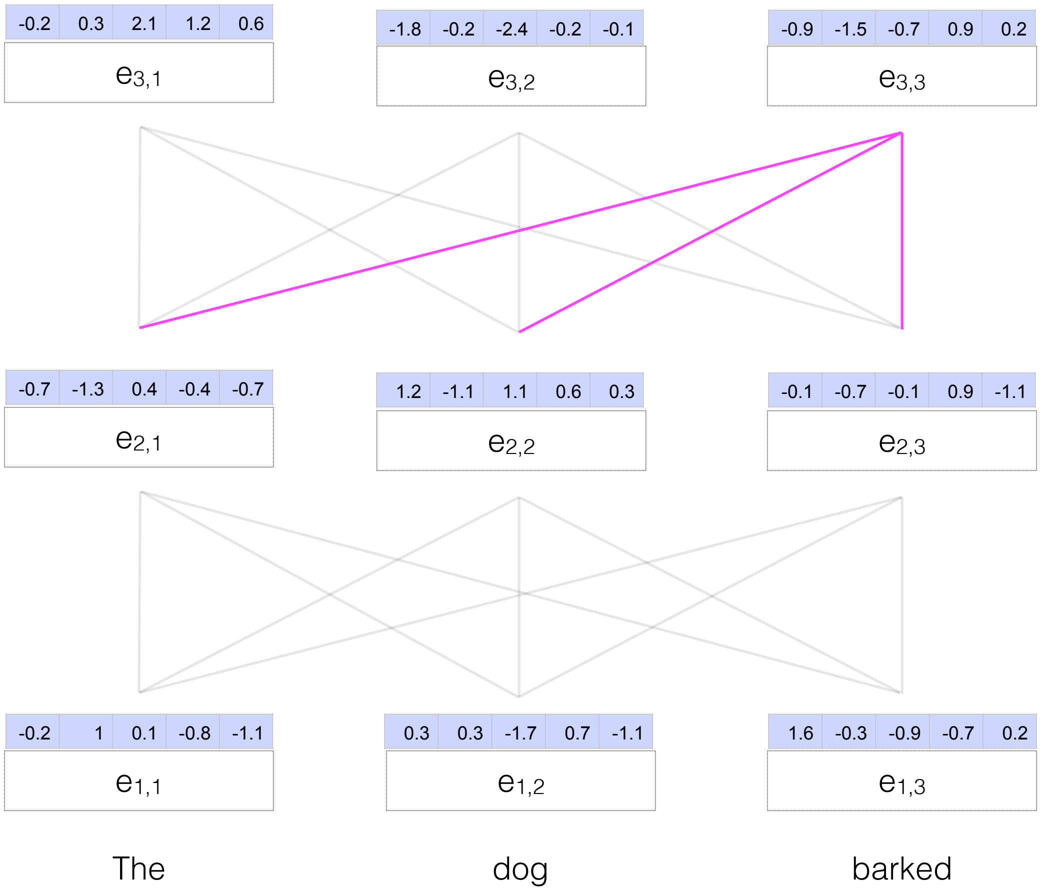
-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
1.2 -1.1 1.1 0.6 0.3
$e_{2,2}$
-0.1 -0.7 -0.1 0.9 -1.1
$e_{2,3}$
-0.2 1 0.1 -0.8 -1.1
$e_{1,1}$
0.3 0.3 -1.7 0.7 -1.1
$e_{1,2}$
1.6 -0.3 -0.9 -0.7 0.2
$e_{1,3}$
The
dog
barked



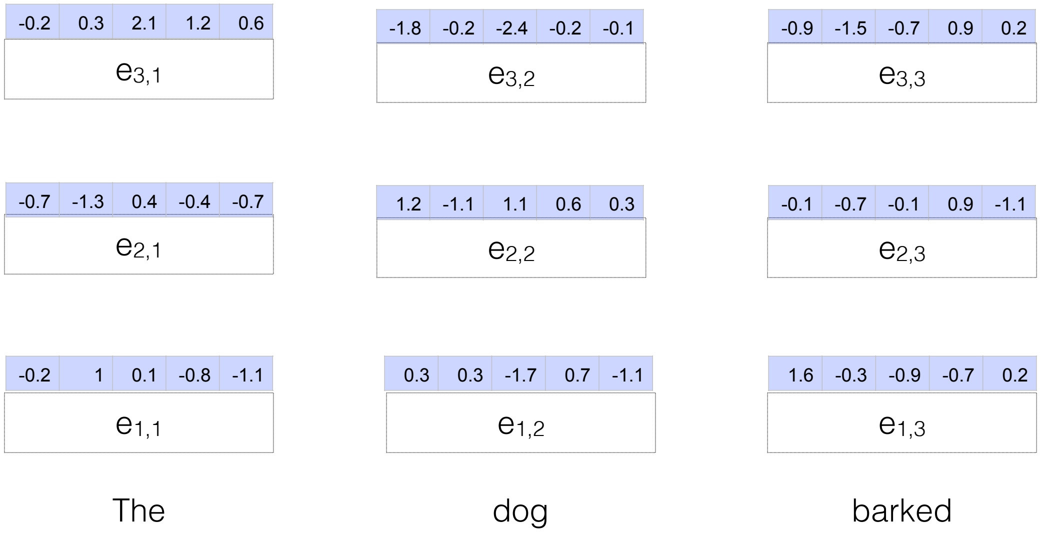
-0.2 0.3 2.1 1.2 0.6
e3,1
-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.1 -0.7 -0.1 0.9 -1.1
e2,3
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked

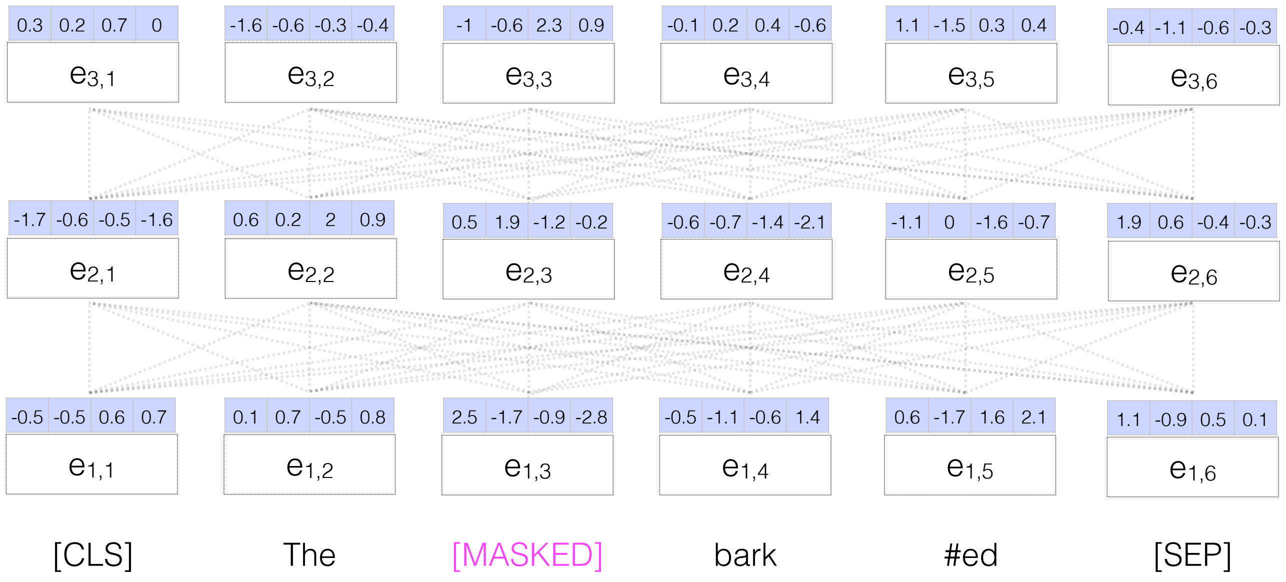


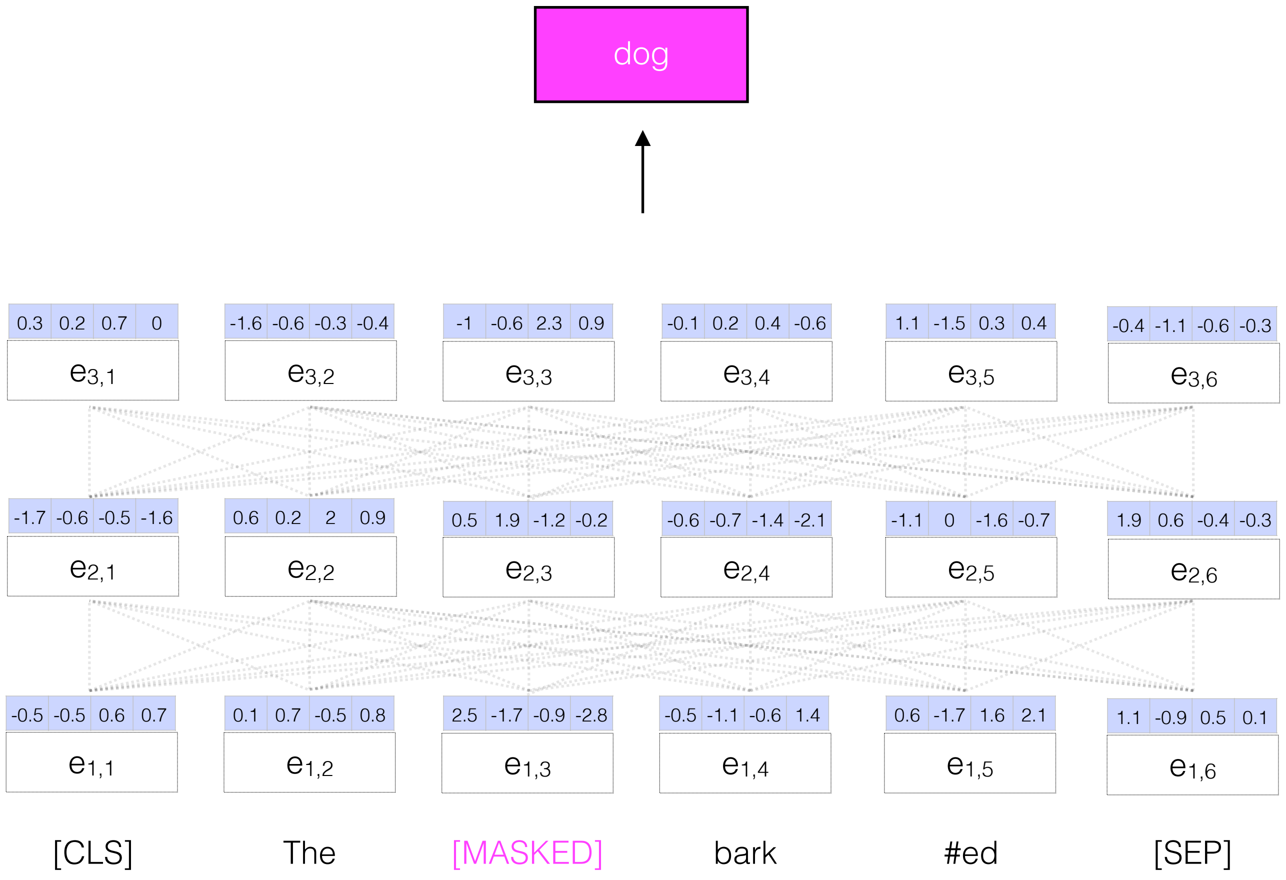
-0.2 0.3 2.1 1.2 0.6
$e_{3,1}$
-1.8 -0.2 -2.4 -0.2 -0.1
$e_{3,2}$
-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
1.2 -1.1 1.1 0.6 0.3
$e_{2,2}$
-0.1 -0.7 -0.1 0.9 -1.1
$e_{2,3}$
-0.2 1 0.1 -0.8 -1.1
$e_{1,1}$
0.3 0.3 -1.7 0.7 -1.1
$e_{1,2}$
1.6 -0.3 -0.9 -0.7 0.2
$e_{1,3}$
The
dog
barked


-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




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
-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
[MASKED]
bark
#ed
[SEP]

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).

WordPiece

- BERT uses WordPiece tokenization, which segments some morphological structure of tokens
- Vocabulary size: 30,000

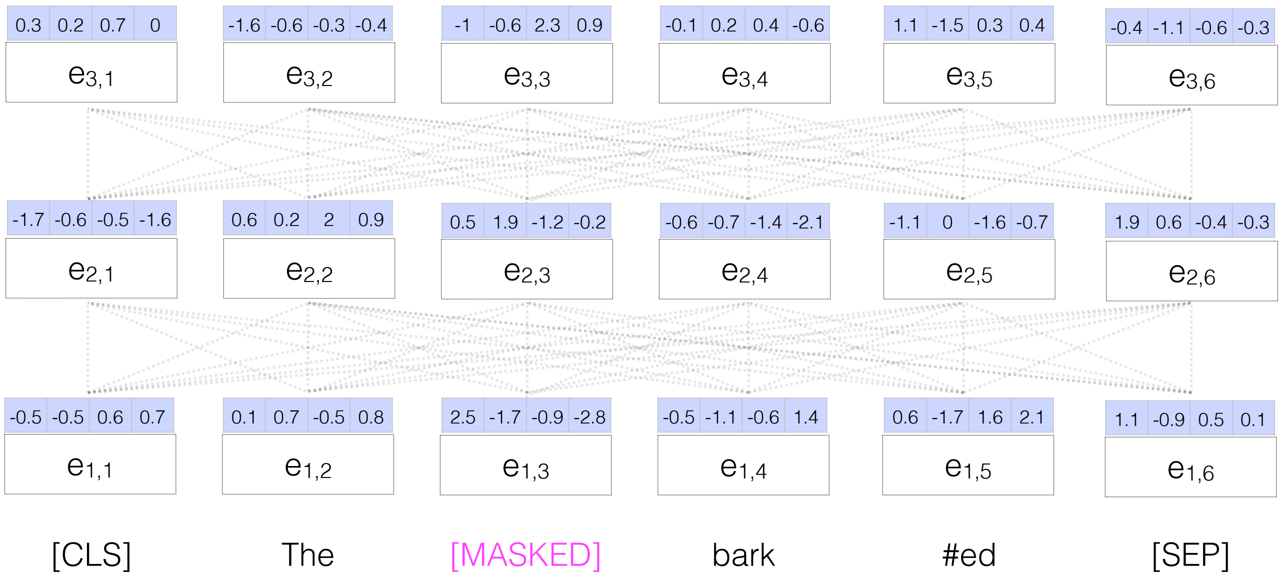
The	The
dog	dog
barked	bark #ed

BERT

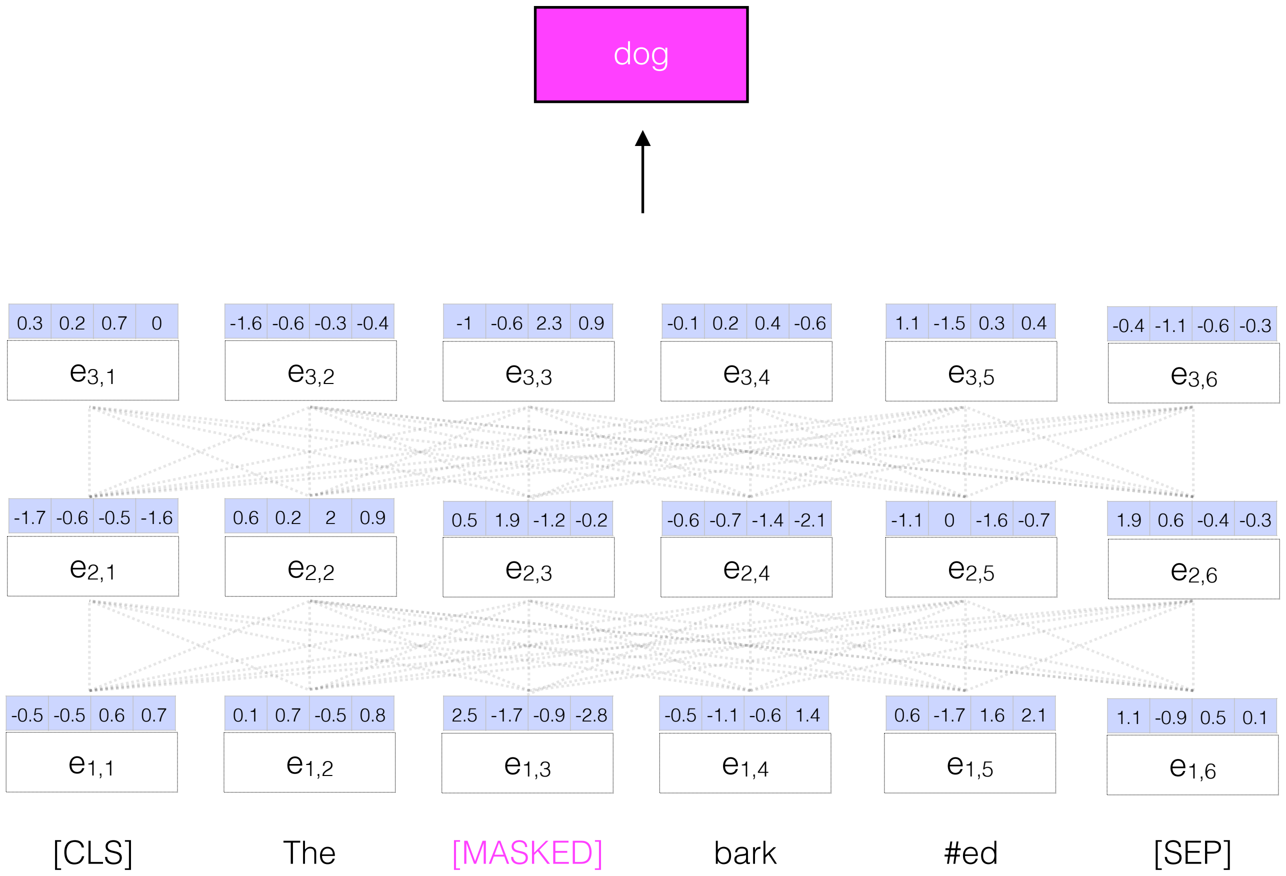
- Learn the parameters of this model with two objectives:
  - Masked language modeling
  - Next sentence prediction

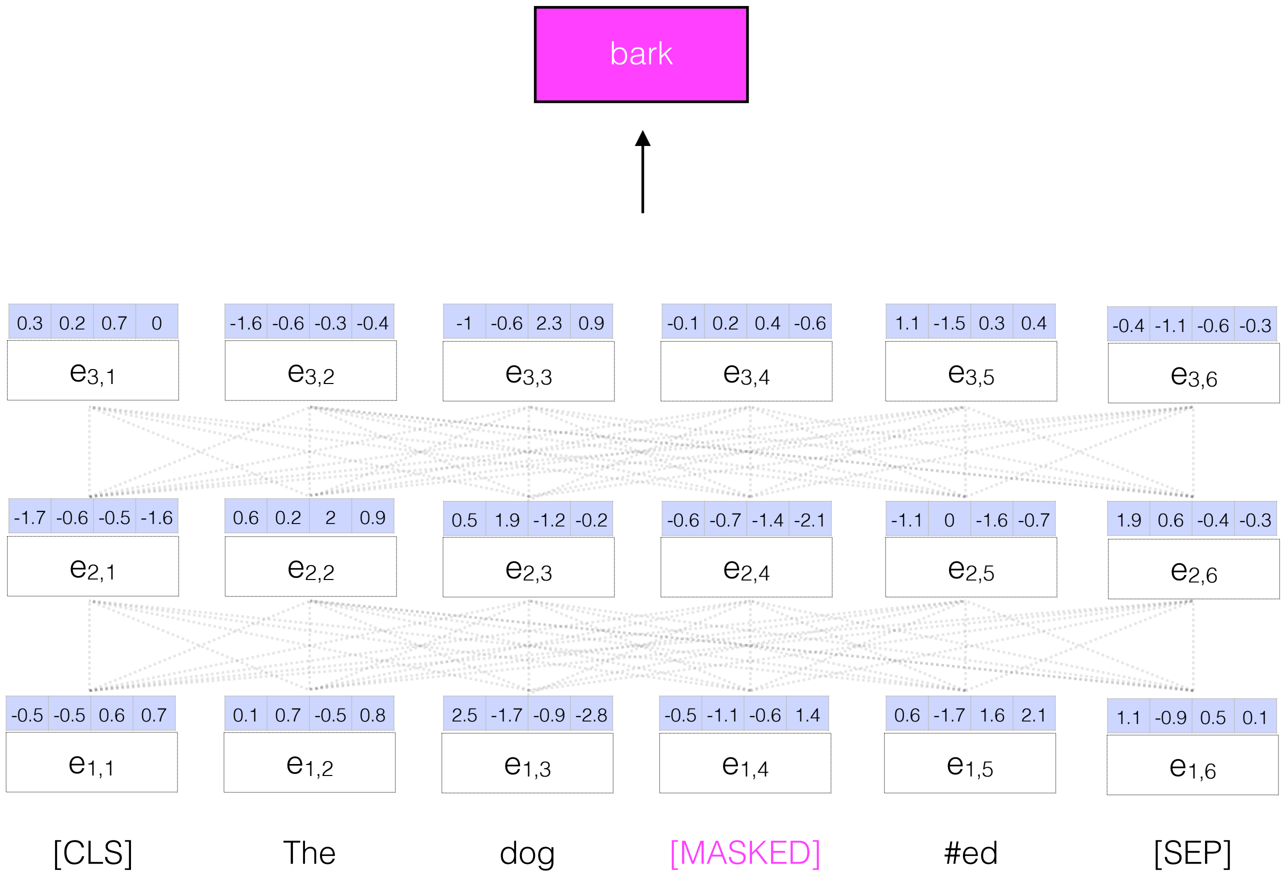
Masked LM

- Mask one word from the input and try to predict that word as the output
- More powerful than an RNN LM since it can reason about context on both sides of the word being predicted.



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
$e_{3,1}$
-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
[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]

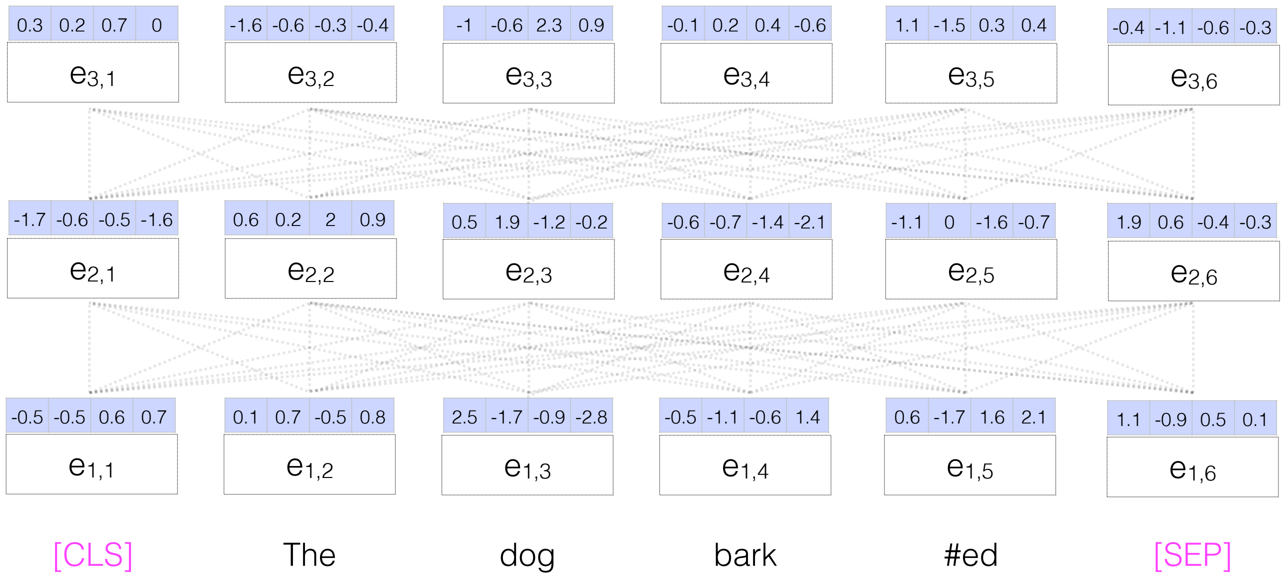
Next sentence prediction

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

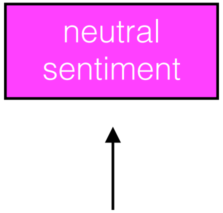
+ [CLS] The dog bark #ed [SEP] He was hungry

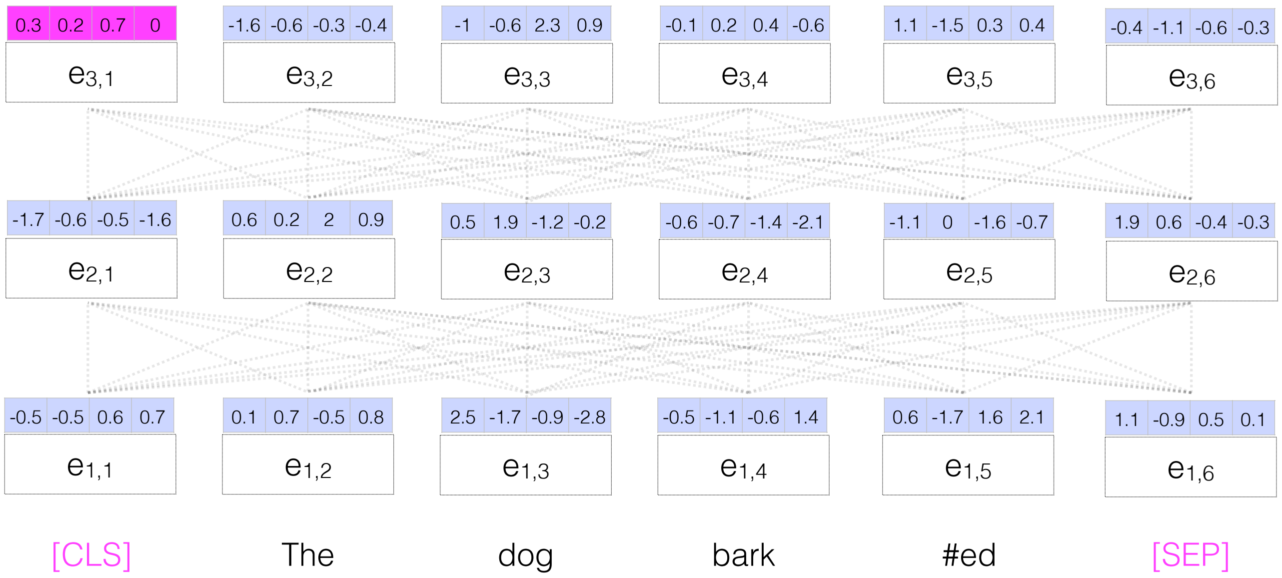
− [CLS] The dog bark #ed [SEP] Paris is in France

- BERT 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 (e.g., for document classification)

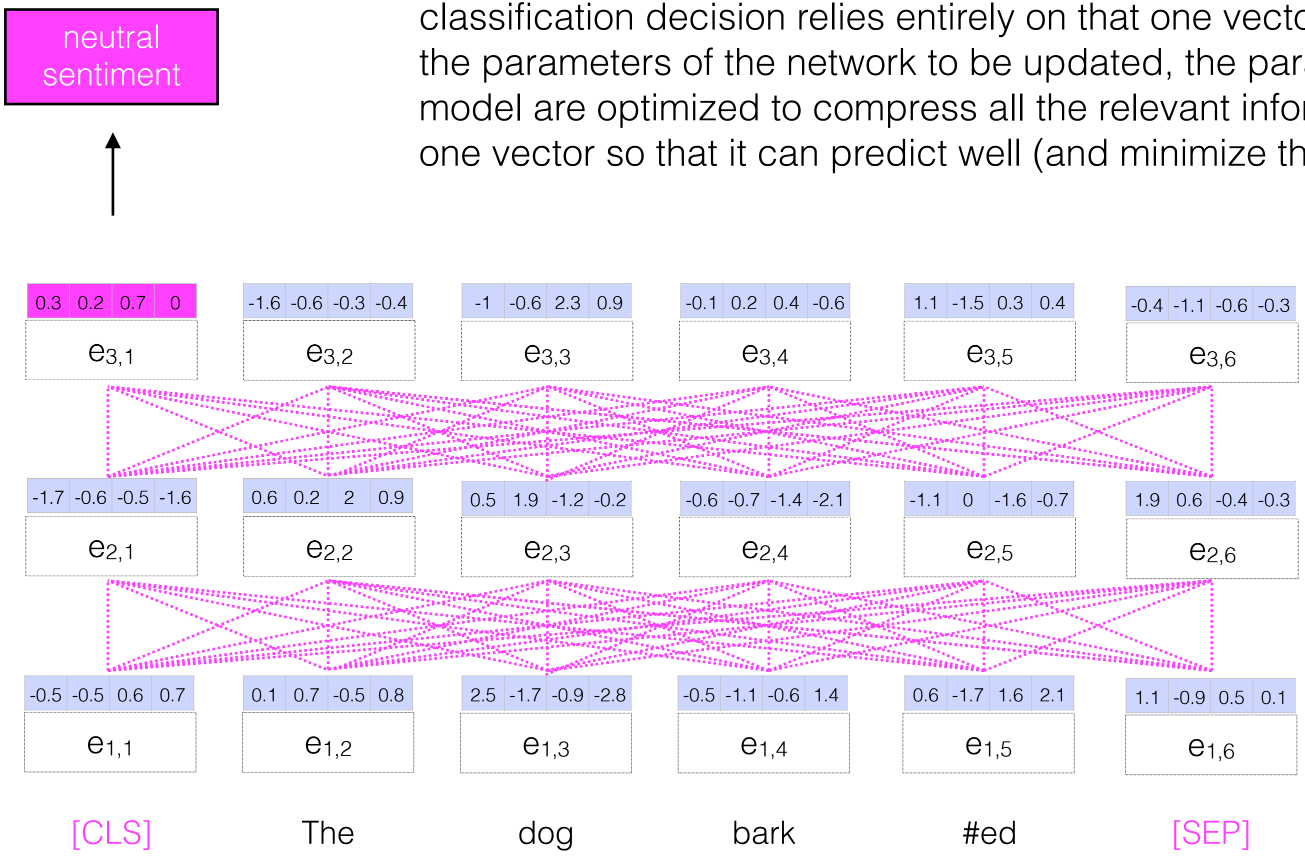


- We can represent the entire document with this *one* [CLS] vector
- Why does this work? When we design our network so that a classification decision relies entirely on that one vector and allow all the parameters of the network to be updated, the parameters of the model are optimized to compress all the relevant information into that one vector so that it can predict well (and minimize the loss).





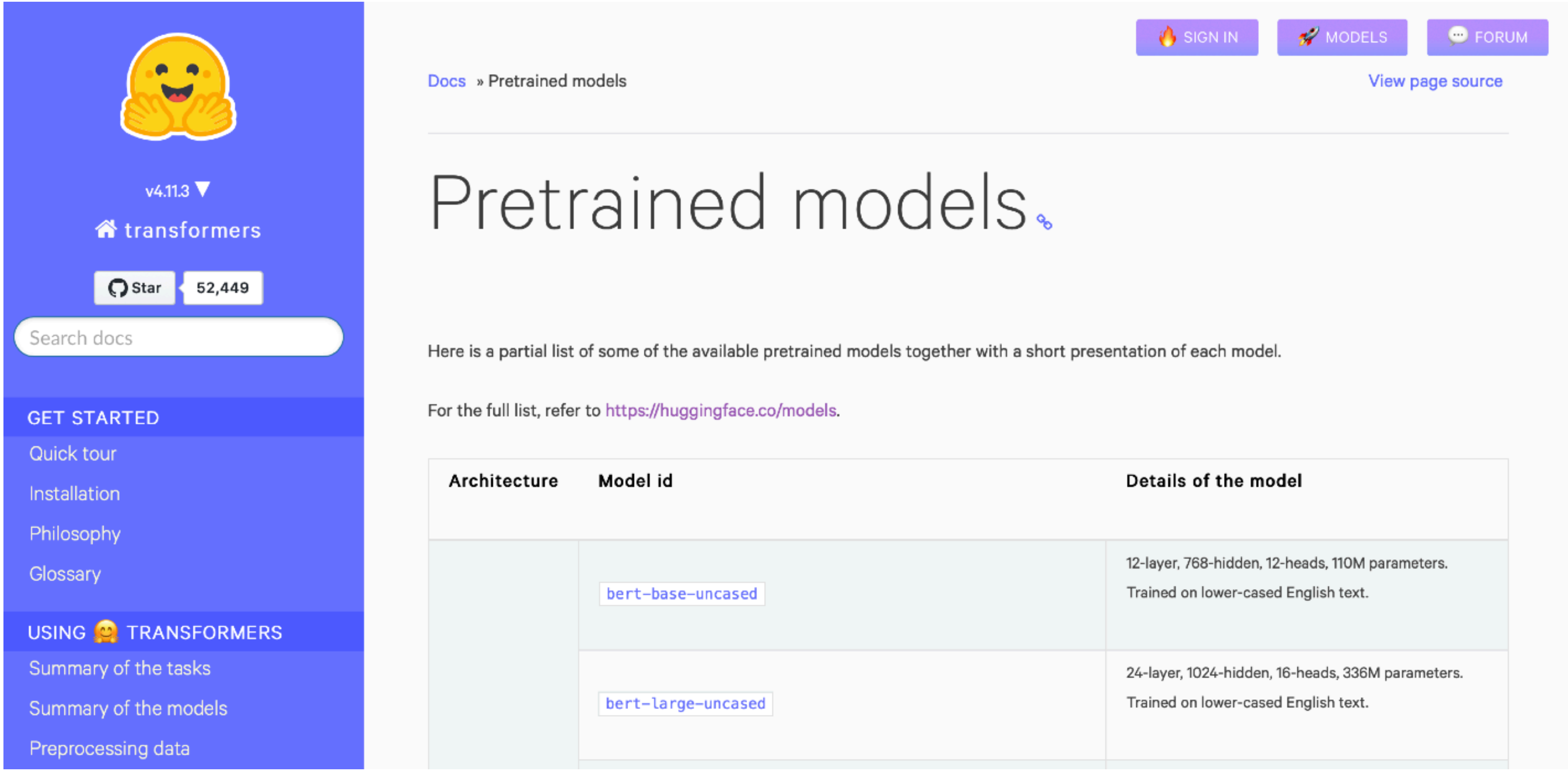
- We can represent the entire document with this *one* [CLS] vector
- Why does this work? When we design our network so that a classification decision relies entirely on that one vector and allow all the parameters of the network to be updated, the parameters of the model are optimized to compress all the relevant information into that one vector so that it can predict well (and minimize the loss).



BERT

	H=128	H=256	H=512	H=768
L=2	**2/128 (BERT-Tiny)**	2/256	2/512	2/768
L=4	4/128	**4/256 (BERT-Mini)**	**4/512 (BERT-Small)**	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	**8/512 (BERT-Medium)**	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	**12/768 (BERT-Base)**

https://github.com/google-research/bert



v4.11.3

transformers

Star 52,449

Search docs

GET STARTED

Quick tour

Installation

Philosophy

Glossary

USING 🤗 TRANSFORMERS

Summary of the tasks

Summary of the models

Preprocessing data

SIGN IN MODELS FORUM

Docs » Pretrained models

View page source

Pretrained models

Here is a partial list of some of the available pretrained models together with a short presentation of each model.

For the full list, refer to https://huggingface.co/models.

Architecture	Model id	Details of the model
	`bert-base-uncased`	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.
	`bert-large-uncased`	24-layer, 1024-hidden, 16-heads, 336M parameters. Trained on lower-cased English text.

https://huggingface.co/transformers/pretrained_models.html

123

BERT LANG STREET

Lost in (language-specific) BERT models? We are here to help!

We currently have indexed **31** BERT-based models, **19** Languages and **28** Tasks.

We have a total of **178** entries in this table; we also show **Multilingual Bert (mBERT)** results if available! (see our paper)

Curious which BERT model is the best for named entity recognition in Italian? Just type *"Italian NER"* in the search bar!

Show 10 entries

Search:

Language	Model	NLP Task	Dataset	Dataset-Domain	Measure	Performance	mBERT	Difference with mBERT	Source
Arabic	Arabert v1	SA	AJGT	twitter	Accuracy	93.8	83.6	10.2	
Arabic	Arabert v1	SA	HARD	hotel reviews	Accuracy	96.1	95.7	0.4	
Arabic	Arabert v1	SA	ASTD	twitter	Accuracy	92.6	80.1	12.5	
Arabic	Arabert v1	SA	ArSenTD-Lev	twitter	Accuracy	59.4	51.0	8.4	

https://bertlang.unibocconi.it

Activity

```
6.classification/
BERTClassification_TODO
```