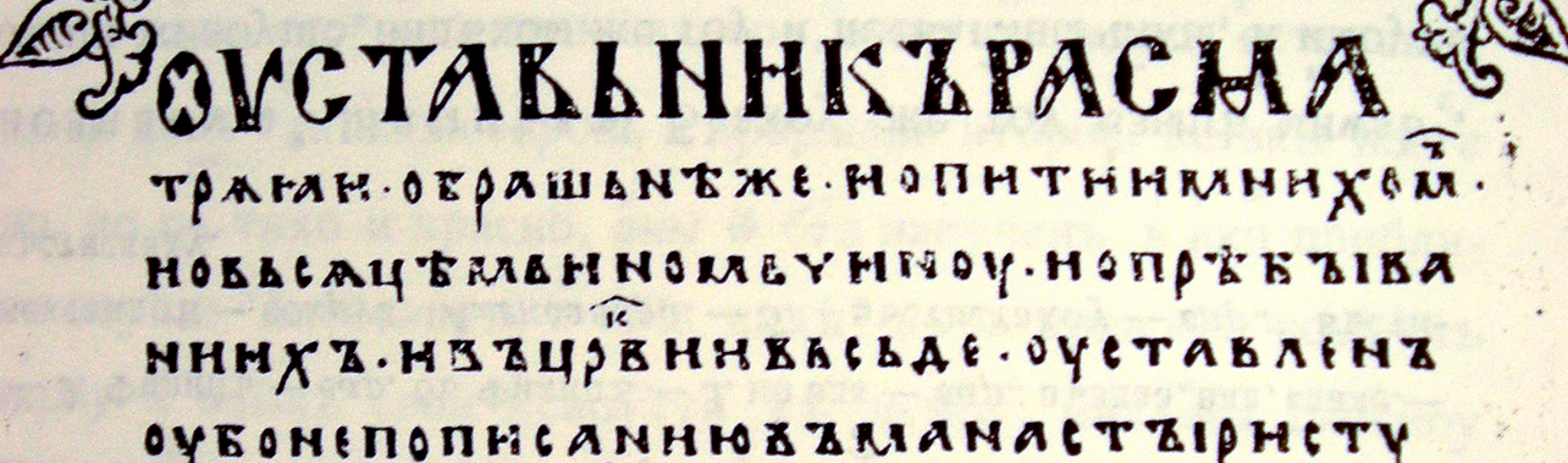




Natural Language Processing

Info 159/259

Lecture 8: Language models 2 (Feb 9, 2023)

David Bamman, UC Berkeley

Language Model

- Vocabulary $\mathcal{V}$ is a finite set of discrete symbols (e.g., words, characters); $V = |\mathcal{V}|$
- $\mathcal{V}^+$ is the infinite set of sequences of symbols from $\mathcal{V}$; each sequence ends with STOP
- $x \in \mathcal{V}^+$

Language Model

Language modeling is the task of estimating $P(w)$

Language Model

arma virumque cano

- arms man and I sing
- Arms, and the man I sing [Dryden]
- I sing of arms and a man

- When we have choices to make about different ways to say something, a language model gives us a formal way of operationalizing their fluency (in terms of how likely they are to exist in the language)

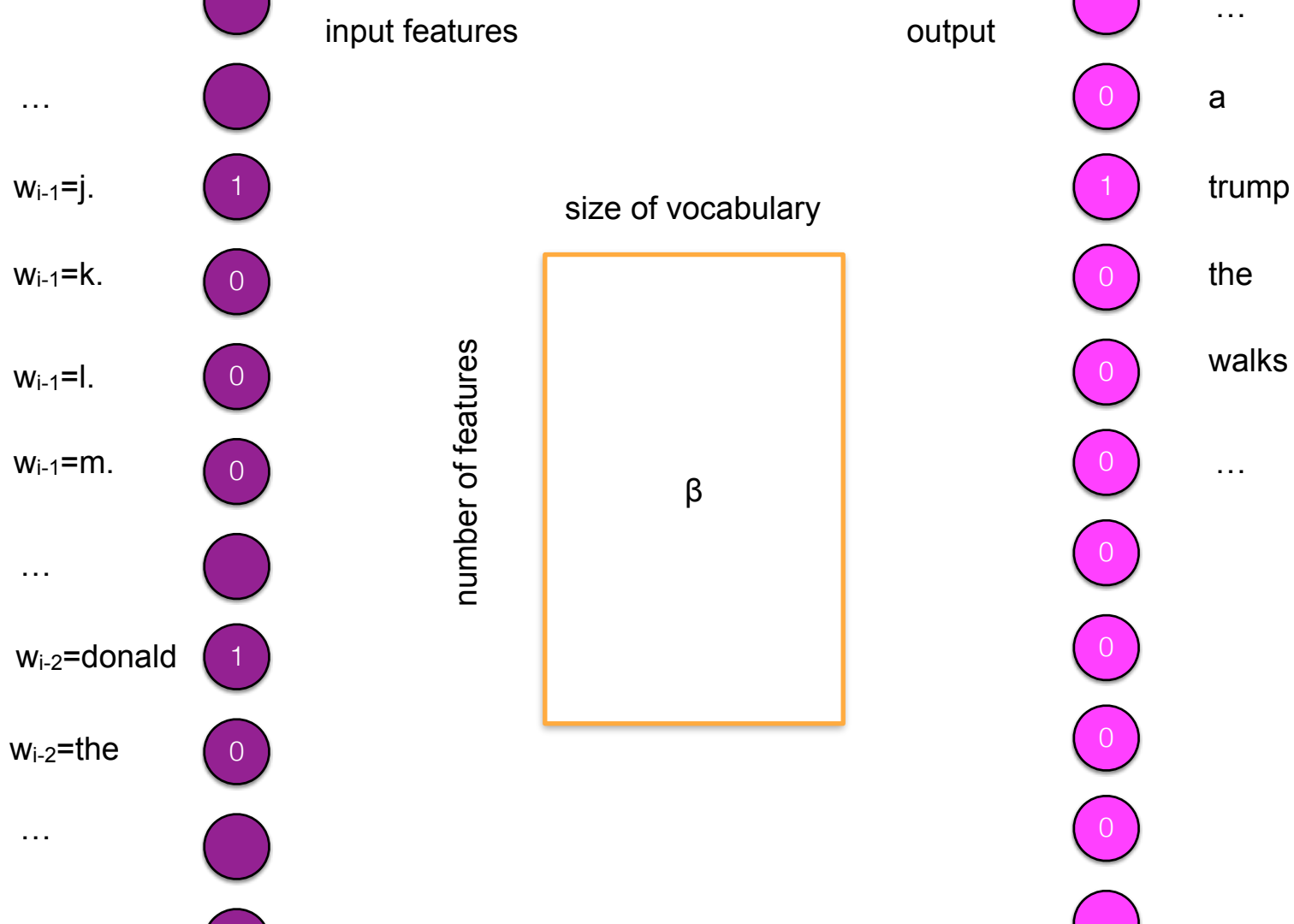
Language Model

- We can use multiclass logistic regression for language modeling by treating the vocabulary as the output space

$$\mathcal{Y} = \mathcal{V}$$

The United States Senate opens its second impeachment trial of former President Donald J. _____

feature classes	example
ngrams (w_{i-1} , w_{i-2} : w_{i-1} , w_{i-3} : w_{i-1})	w_{i-2} ="donald", w_{i-1} ="j."
gappy ngrams	w_1 ="impeachment" and w_2 ="donald"
spelling, capitalization	w_{i-1} is capitalized and w_i is capitalized
class/gazetteer membership	w_{i-1} in list of names and w_i in list of names



Tradeoffs

- Richer representations = more parameters, higher likelihood of overfitting
- Much slower to train than estimating the parameters of a classical model

$$P(Y = y \mid X = x; \beta) = \frac{\exp(x^\top \beta_y)}{\sum_{y' \in \mathcal{Y}} \exp(x^\top \beta_{y'})}$$

Neural LM

Simple feed-forward multilayer perceptron (e.g., one hidden layer)

input x = vector concatenation of a conditioning context of fixed size k



$$x = [v(w_1); \dots; v(w_k)]$$

$$X = [v(w_1); \dots; v(w_k)]$$

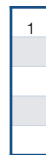
w_1 = tried

w_2 = to

w_3 = prepare

w_4 = residents

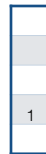
$v(w_1)$



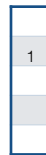
$v(w_2)$



$v(w_3)$



$v(w_4)$



$v(w_1)$



$v(w_2)$



$v(w_3)$



$v(w_4)$



one-hot encoding

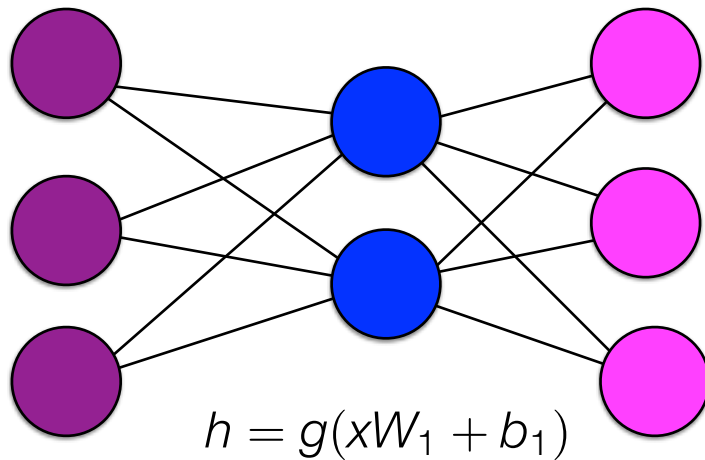
distributed
representation

$$W_1 \in \mathbb{R}^{kD \times H}$$

$$b_1 \in \mathbb{R}^H$$

$$W_2 \in \mathbb{R}^{H \times V}$$

$$b_2 \in \mathbb{R}^V$$



$$x = [v(w_1); \dots; v(w_k)]$$

$$\hat{y} = \text{softmax}(hW_2 + b_2)$$

Softmax

$$P(Y = y \mid X = x; \beta) = \frac{\exp(x^\top \beta_y)}{\sum_{y' \in \mathcal{Y}} \exp(x^\top \beta_{y'})}$$

Neural LM

conditioning context

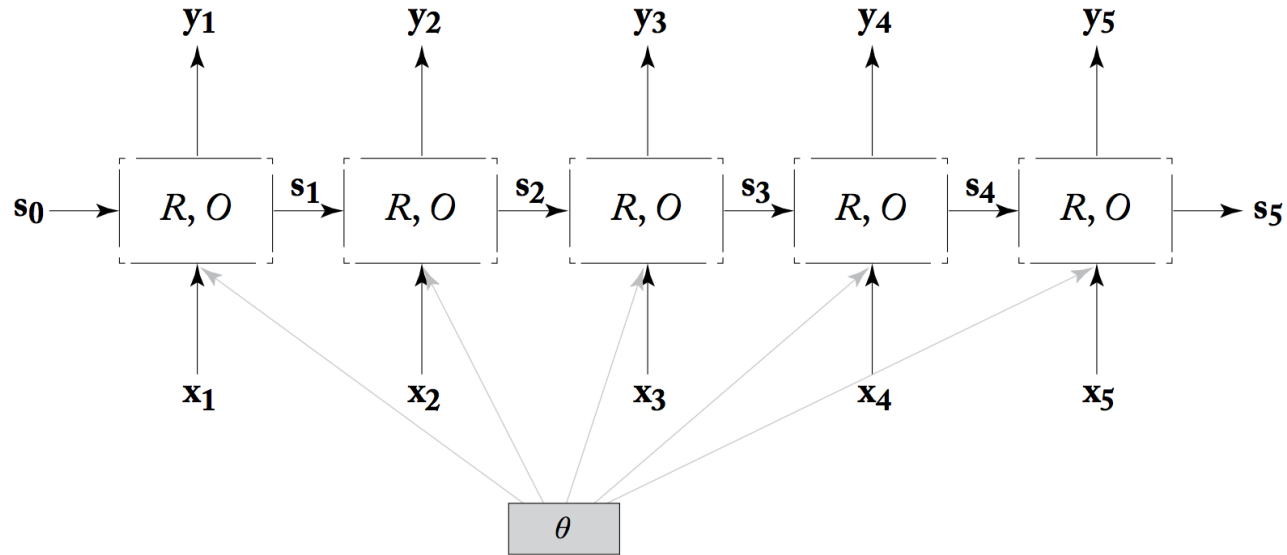
tried to prepare residents for the hardships of recovery from the

y

Recurrent neural network

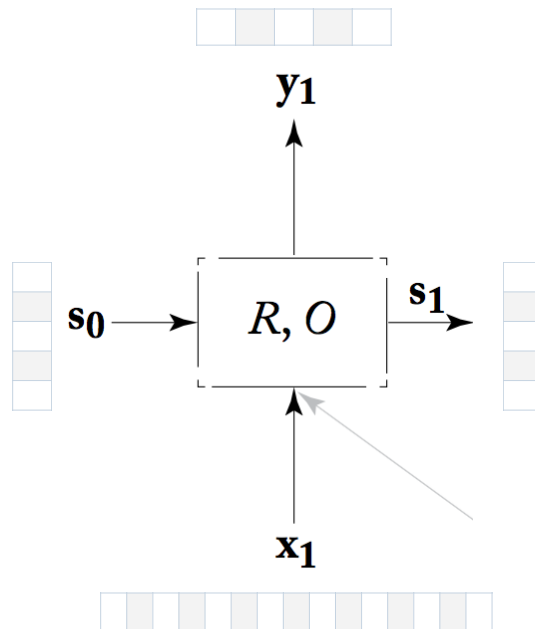
- RNN allow arbitrarily-sized conditioning contexts; condition on the entire sequence history.

Recurrent neural network



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



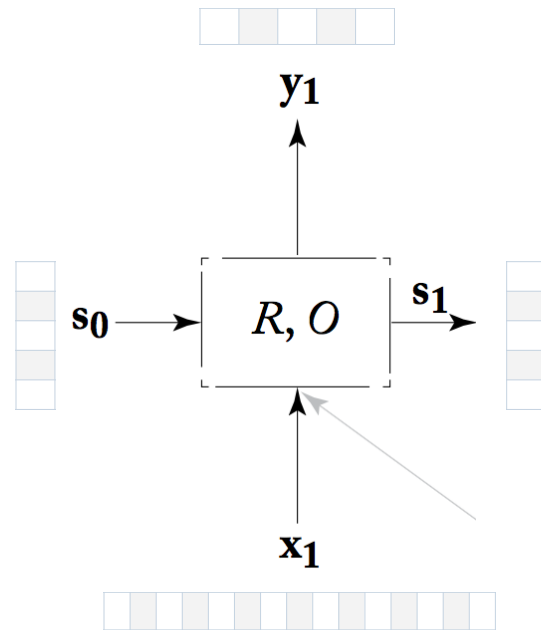
Recurrent neural network

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

R computes the output state as a function of the current input and previous state

$$y_i = O(s_i)$$

O computes the output as a function of the current output state



“Simple” RNN

$g = \tanh$ or relu

$$s_i = R(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b)$$

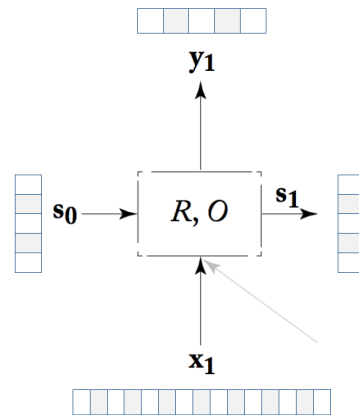
Different weight vectors W transform the previous state and current input before combining

$$W^s \in \mathbb{R}^{H \times H}$$

$$W^x \in \mathbb{R}^{D \times H}$$

$$b \in \mathbb{R}^H$$

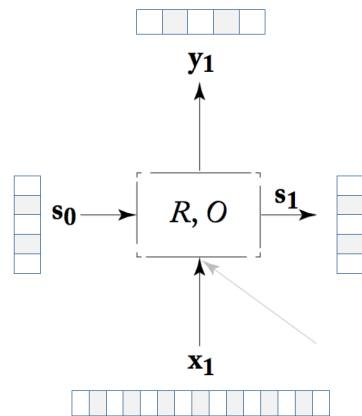
$$y_i = O(s_i) = s_i$$



RNN LM

- The output state s_i is an H-dimensional real vector; we can transfer that into a probability by passing it through an additional linear transformation followed by a softmax

$$y_i = O(s_i) = \text{softmax}(s_i W^o + b^o)$$



Training RNNs

- Given this definition of an RNN:

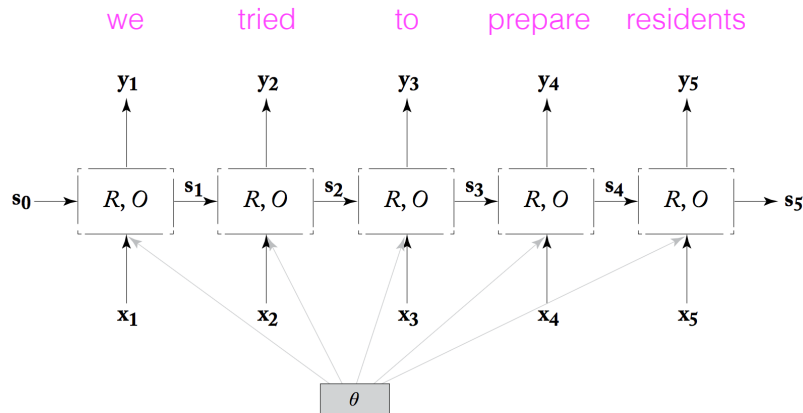
$$s_i = R(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b)$$

$$y_i = O(s_i) = \text{softmax}(s_iW^o + b^o)$$

- We have five sets of parameters to learn:

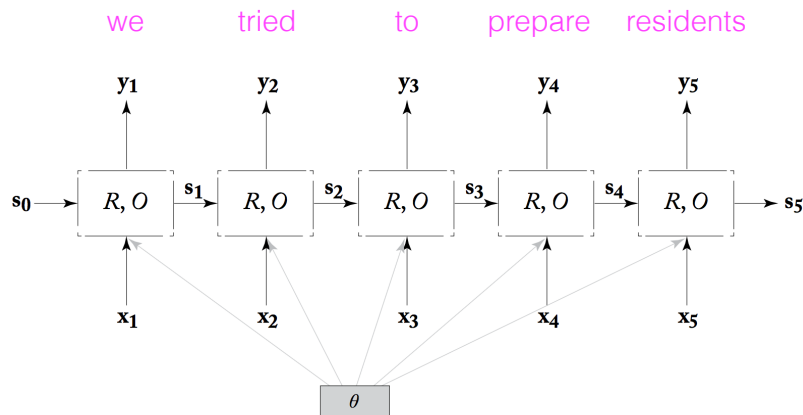
$$W^s, W^x, W^o, b, b^o$$

Training RNNs

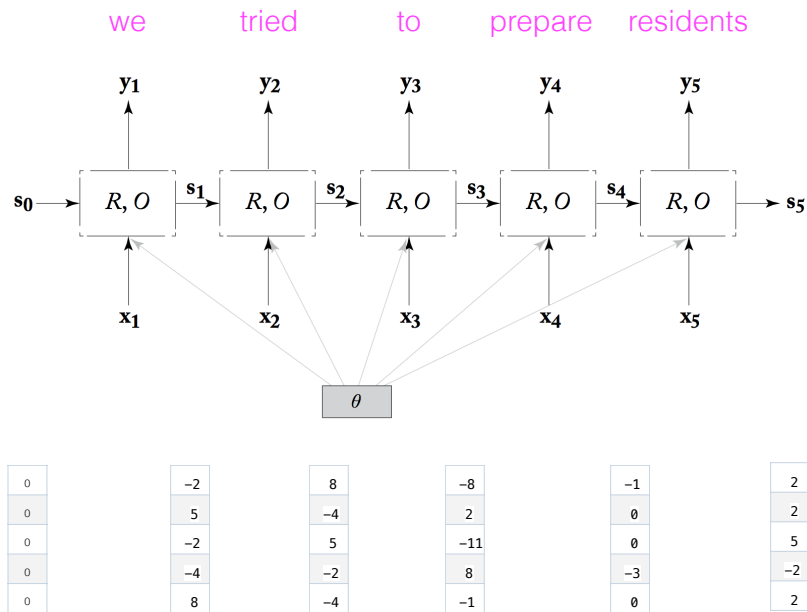


- At each time step, we make a prediction and incur a loss; we know the true y (the word we see in that position)

$$\frac{\partial L(\theta)_{y_1}}{\partial W^s} \quad \frac{\partial L(\theta)_{y_2}}{\partial W^s} \quad \frac{\partial L(\theta)_{y_3}}{\partial W^s} \quad \frac{\partial L(\theta)_{y_4}}{\partial W^s} \quad \frac{\partial L(\theta)_{y_5}}{\partial W^s}$$



- Training here is standard **backpropagation**, taking the derivative of the loss we incur at step t with respect to the parameters we want to update



Each state i encodes information seen until time i
and its structure is optimized to predict the next word

Character LM

- Vocabulary $\mathcal{V}$ is a finite set of discrete **characters**
- When the output space is small, you're putting a lot of the burden on the structure of the model
- Encode long-range dependencies (suffixes depend on prefixes, word boundaries etc.)

Character LM

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
```

Character LM

```
\begin{proof}
We may assume that  $\mathcal{I}$  is an abelian sheaf on  $\mathcal{C}$ .
\item Given a morphism  $\Delta : \mathcal{F} \rightarrow \mathcal{I}$ 
is an injective and let  $\mathcal{q}$  be an abelian sheaf on  $\mathcal{X}$ .
Let  $\mathcal{F}$  be a fibered complex. Let  $\mathcal{F}$  be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let  $\mathcal{F}$  be an abelian quasi-coherent sheaf on  $\mathcal{C}$ .
Let  $\mathcal{F}$  be a coherent  $\mathcal{O}_X$ -module. Then
 $\mathcal{F}$  is an abelian catenary over  $\mathcal{C}$ .
\item The following are equivalent
\begin{enumerate}
\item  $\mathcal{F}$  is an  $\mathcal{O}_X$ -module.
\end{enumerate}
\end{enumerate}
\end{lemma}
```

Character LM

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Drawbacks

- Very expensive to train (especially for large vocabulary, though tricks exists — cf. hierarchical softmax)
- Backpropagation through long histories leads to vanishing gradients (cf. **LSTMs** in a few weeks).
- But they consistently have strong performances in perplexity evaluations.

Count-and-normalize

Discriminative

Neural

Model	Perplexity			Entropy reduction over baseline	
	individual	+KN5	+KN5+cache	KN5	KN5+cache
3-gram, Good-Turing smoothing (GT3)	165.2	-	-	-	-
5-gram, Good-Turing smoothing (GT5)	162.3	-	-	-	-
3-gram, Kneser-Ney smoothing (KN3)	148.3	-	-	-	-
5-gram, Kneser-Ney smoothing (KN5)	141.2	-	-	-	-
5-gram, Kneser-Ney smoothing + cache	125.7	-	-	-	-
PAQ8o10t	131.1	-	-	-	-
Maximum entropy 5-gram model	142.1	138.7	124.5	0.4%	0.2%
Random clusterings LM	170.1	126.3	115.6	2.3%	1.7%
Random forest LM	131.9	131.3	117.5	1.5%	1.4%
Structured LM	146.1	125.5	114.4	2.4%	1.9%
Within and across sentence boundary LM	116.6	110.0	108.7	5.0%	3.0%
Log-bilinear LM	144.5	115.2	105.8	4.1%	3.6%
Feedforward neural network LM [50]	140.2	116.7	106.6	3.8%	3.4%
Feedforward neural network LM [40]	141.8	114.8	105.2	4.2%	3.7%
Syntactical neural network LM	131.3	110.0	101.5	5.0%	4.4%
Recurrent neural network LM	124.7	105.7	97.5	5.8%	5.3%
Dynamically evaluated RNNLM	123.2	102.7	98.0	6.4%	5.1%
Combination of static RNNLMs	102.1	95.5	89.4	7.9%	7.0%
Combination of dynamic RNNLMs	101.0	92.9	90.0	8.5%	6.9%

Model	Size	D	Valid	Test
Medium LSTM, Zaremba (2014)	10M	2	86.2	82.7
Large LSTM, Zaremba (2014)	24M	2	82.2	78.4
VD LSTM, Press (2016)	51M	2	75.8	73.2
VD LSTM, Inan (2016)	9M	2	77.1	73.9
VD LSTM, Inan (2016)	28M	2	72.5	69.0
VD RHN, Zilly (2016)	24M	10	67.9	65.4
NAS, Zoph (2016)	25M	-	-	64.0
NAS, Zoph (2016)	54M	-	-	62.4
LSTM	10M	1	61.8	59.6
LSTM		2	63.0	60.8
LSTM		4	62.4	60.1
RHN		5	66.0	63.5
LSTM	24M	1	61.4	59.5
LSTM		2	62.1	59.6
LSTM		4	60.9	58.3
RHN		5	64.8	62.2

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**).

Types and tokens

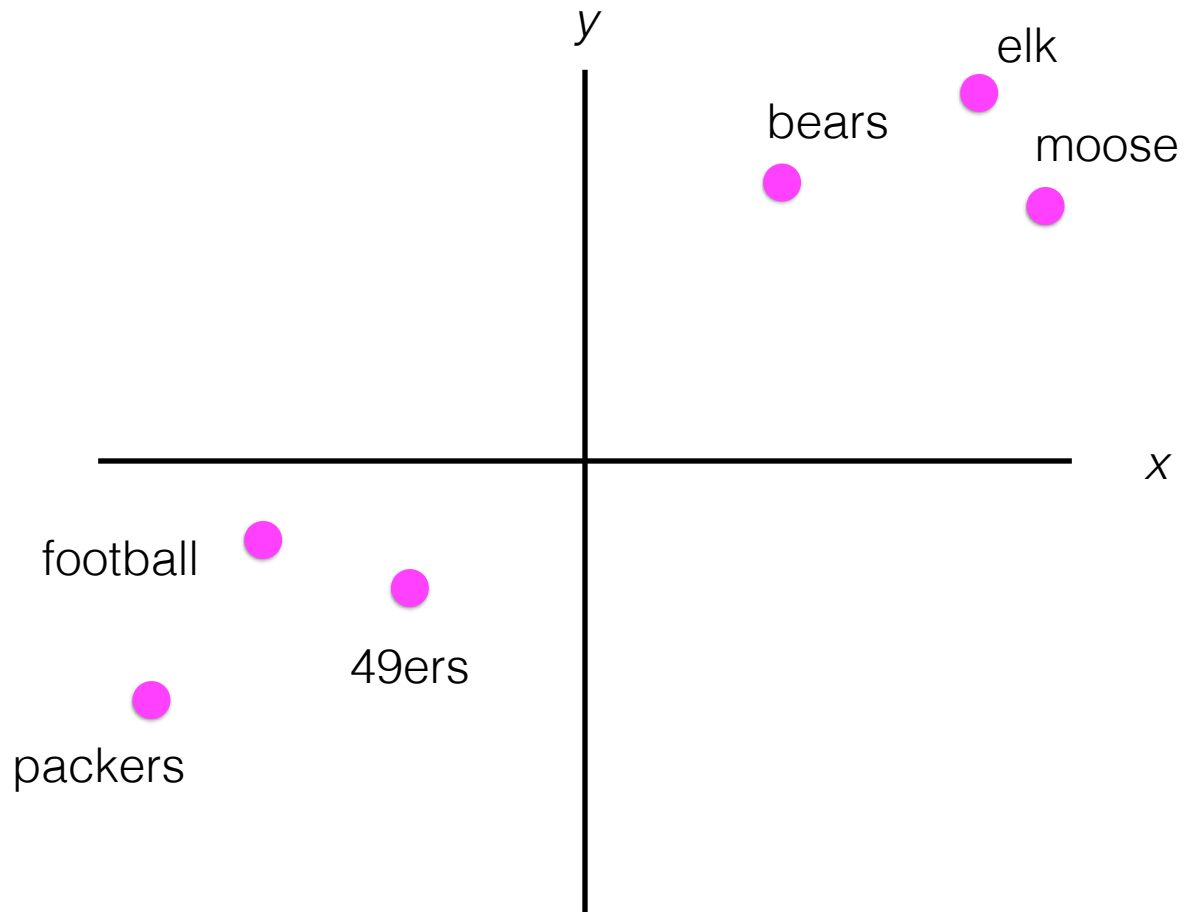
- Type: bears

- 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

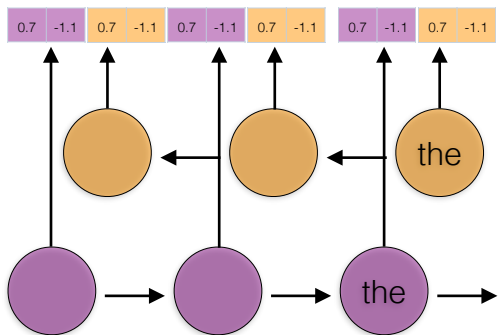


Contextualized word representations

- 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

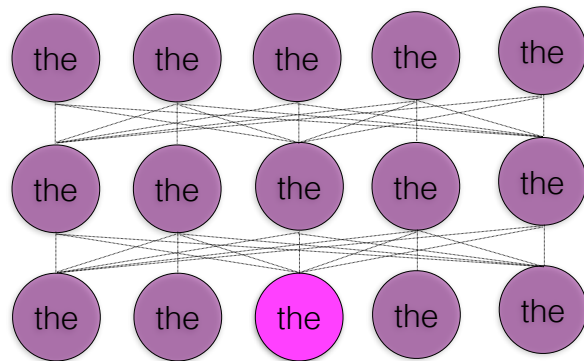
Stacked BiRNN trained to predict **next** word in language modeling task



Peters et al. 2018

BERT

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



Devlin et al. 2019

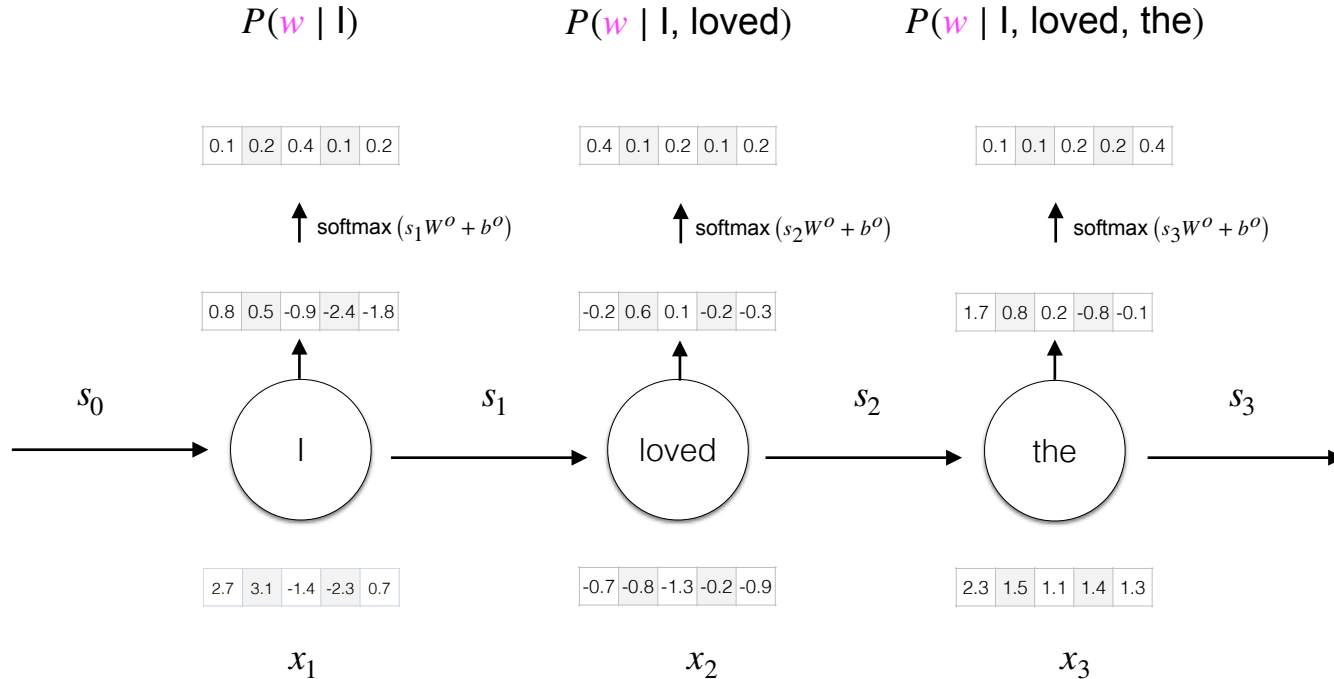
ELMo

- 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.

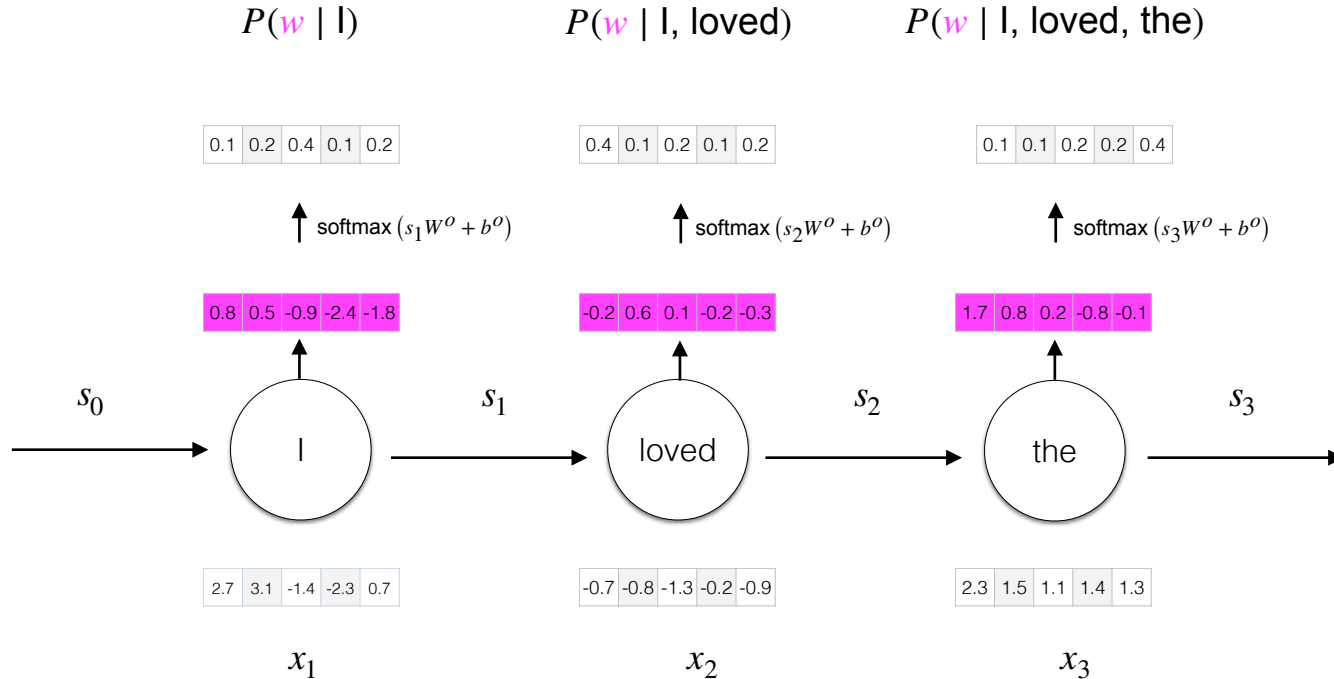
ELMo

- 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.

RNN Language model

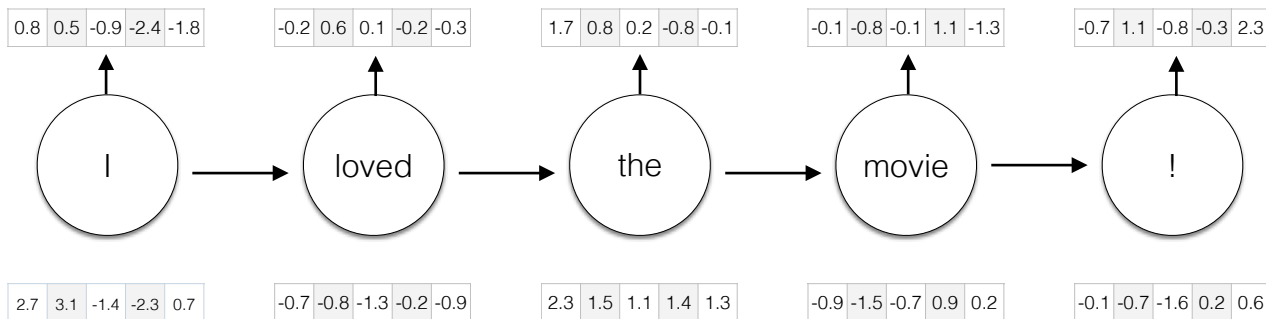


RNN Language model

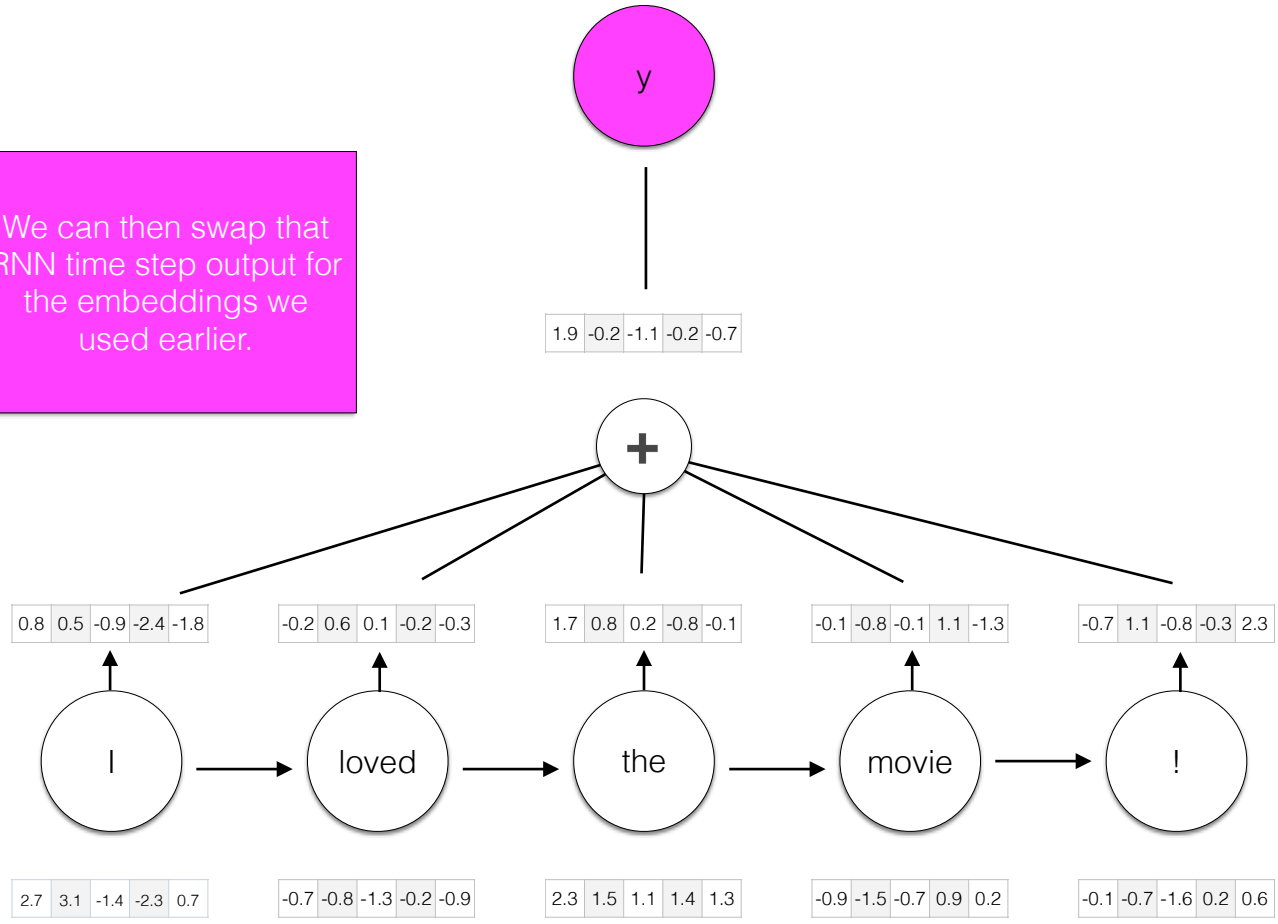


RNN

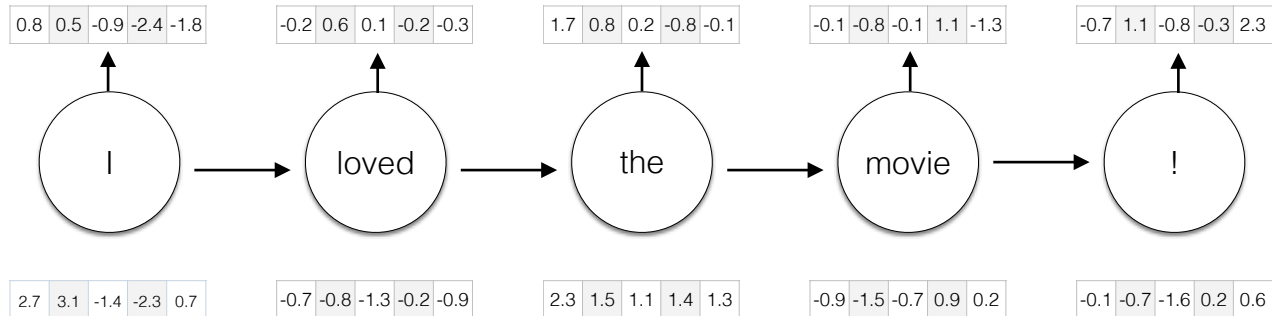
- With an RNN, we can generate a representation of the sequence as **seen through time t**.
- This encodes a representation of meaning specific to the local context a word is used in.



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



What about the **future** context?

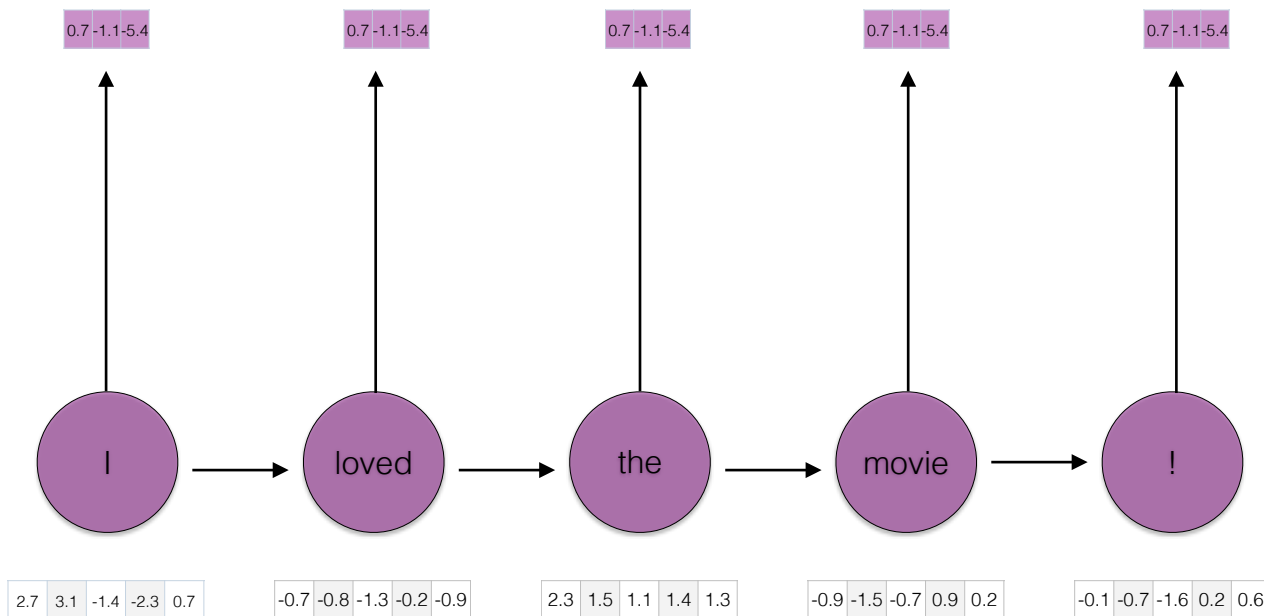


Bidirectional RNN

- 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

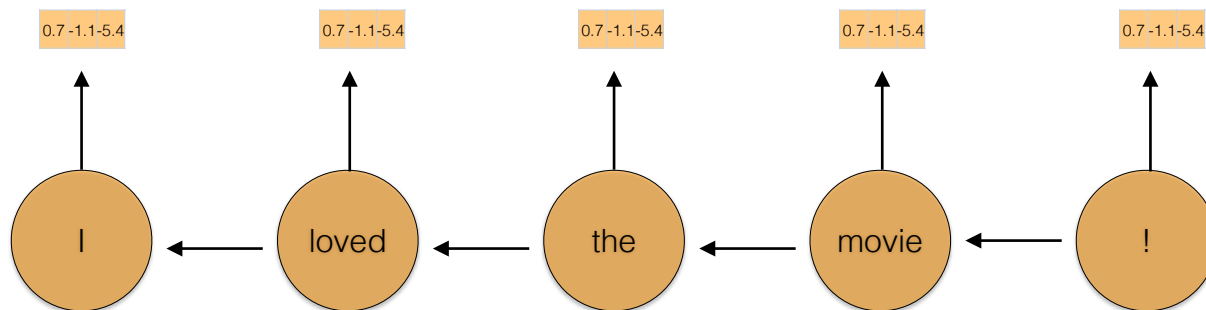
Bidirectional RNN

forward RNN



Bidirectional RNN

backward RNN



2.7	3.1	-1.4	-2.3	0.7
-----	-----	------	------	-----

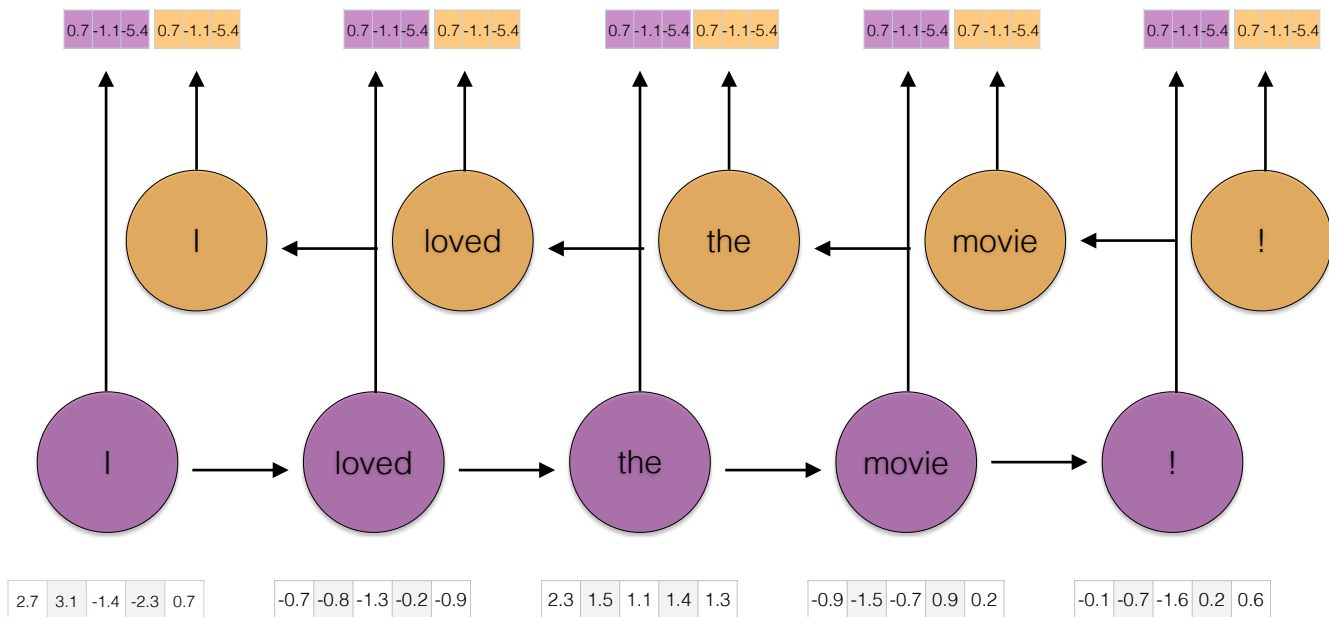
-0.7	-0.8	-1.3	-0.2	-0.9
------	------	------	------	------

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
------	------	------	-----	-----

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

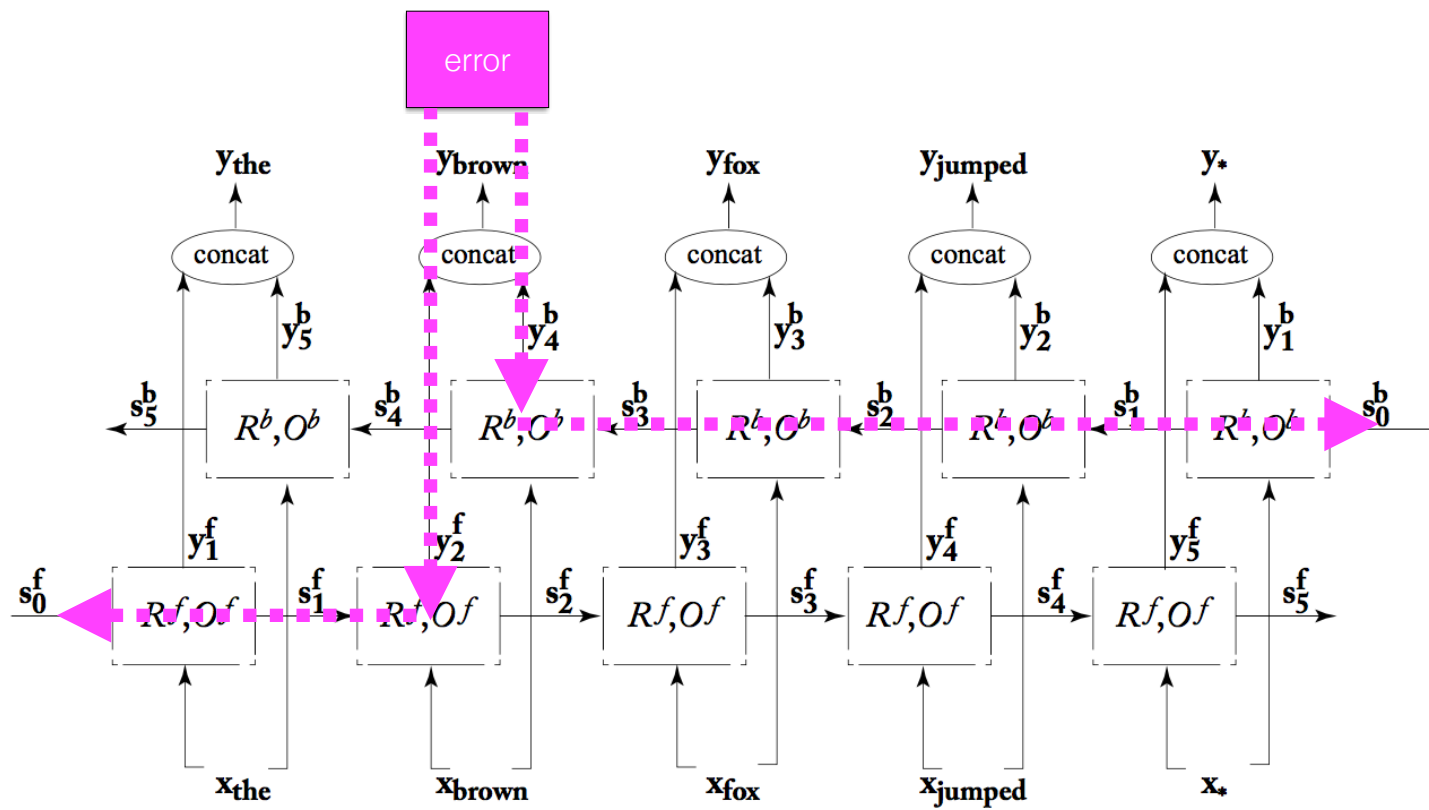
- Given this definition of an BiRNN:

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

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

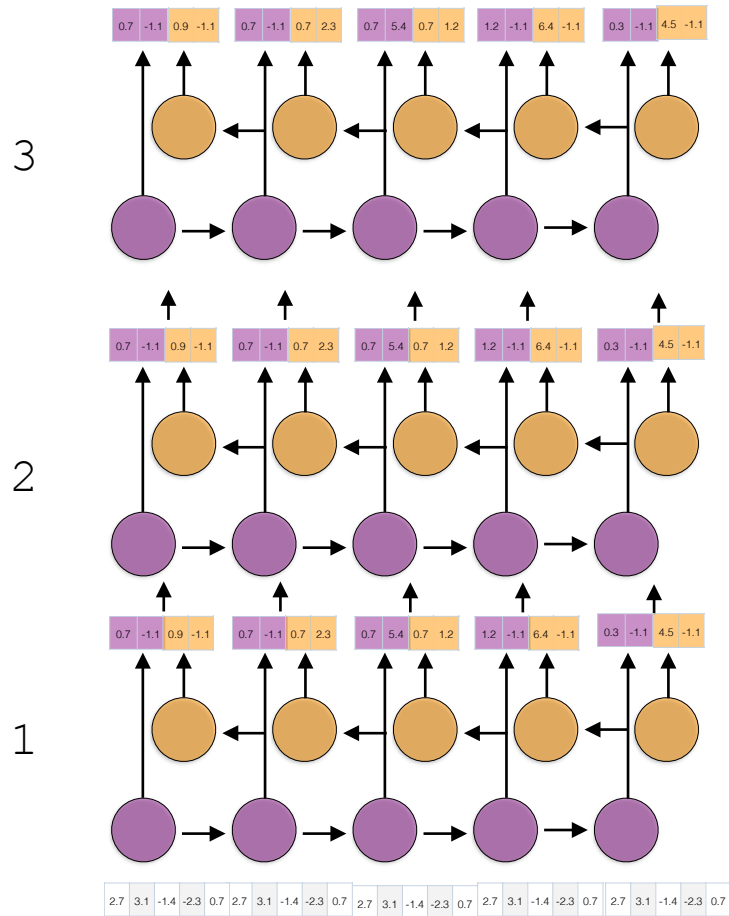
$$y_i = \text{softmax}([s_f^i; s_b^i]W^o + b^o)$$

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



Stacked RNN

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



ELMo

- 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.

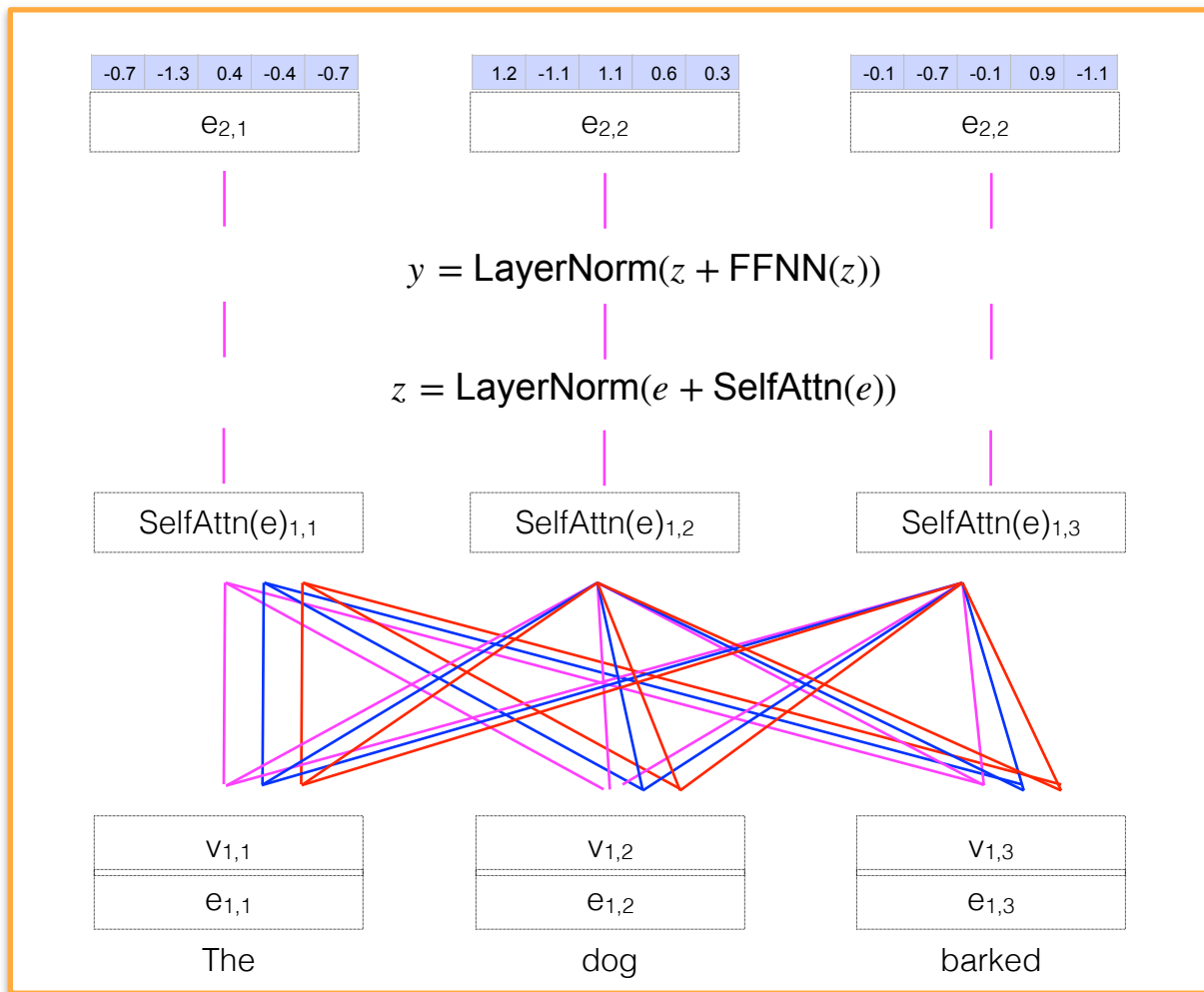
BERT

- 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.

Output

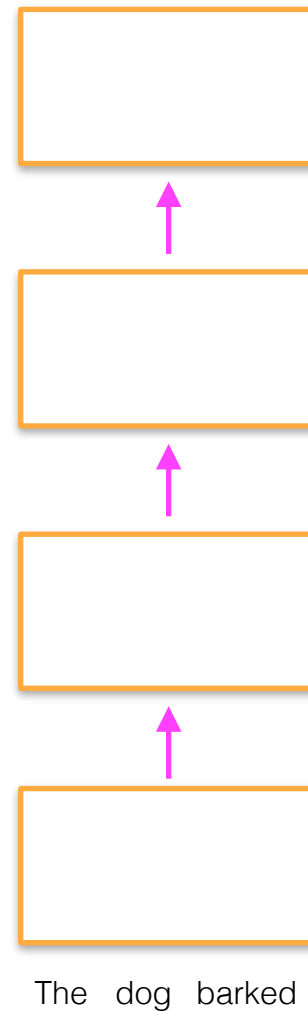
This whole process defines one attention **block**. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

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 $b+1$.



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

-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 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
$e_{1,1}$				

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

-0.7	-1.3	0.4	-0.4	-0.7
$e_{2,1}$				

-0.2	1	0.1	-0.8	-1.1
$e_{1,1}$				

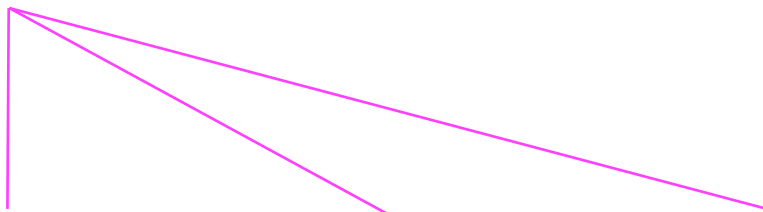
The

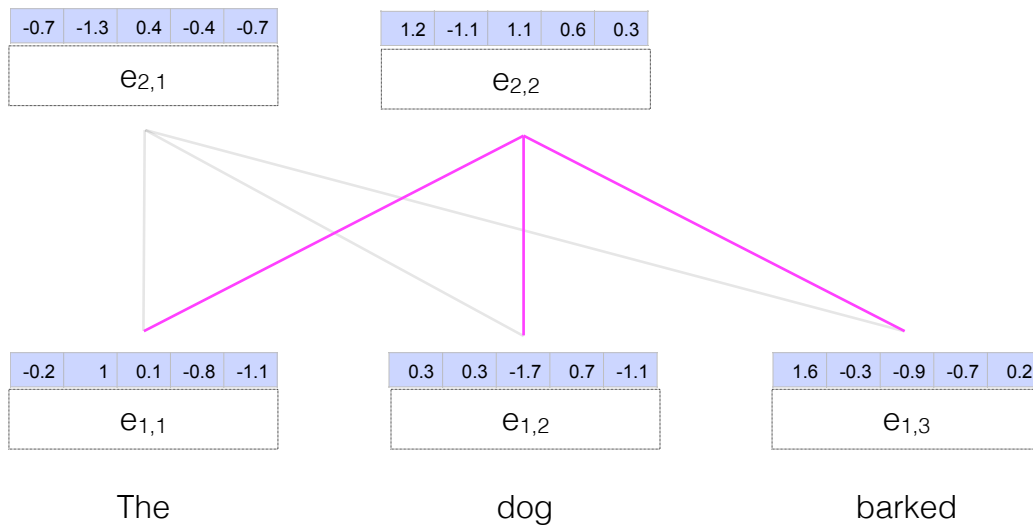
0.3	0.3	-1.7	0.7	-1.1
$e_{1,2}$				

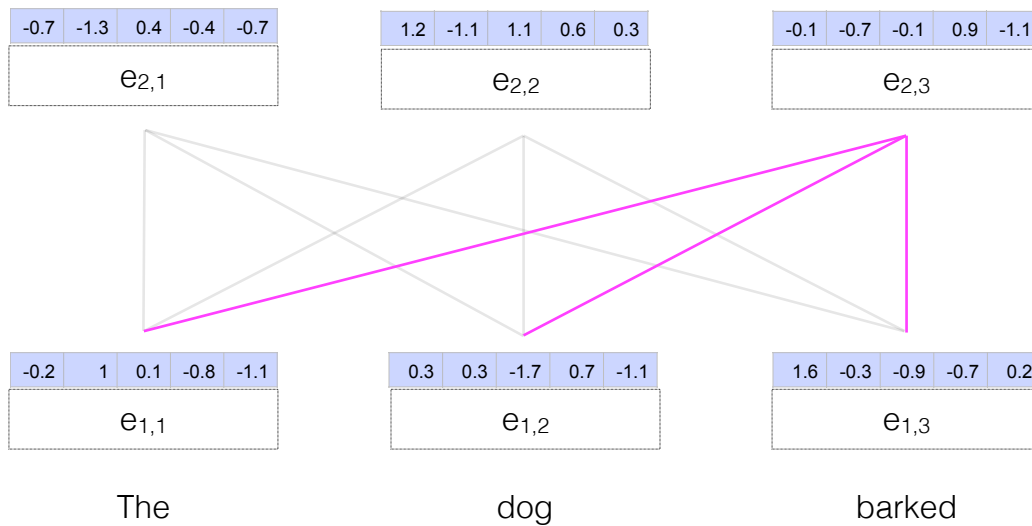
dog

1.6	-0.3	-0.9	-0.7	0.2
$e_{1,3}$				

barked







-0.2	0.3	2.1	1.2	0.6
$e_{3,1}$				

-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
$e_{1,2}$				

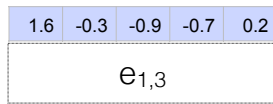
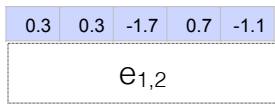
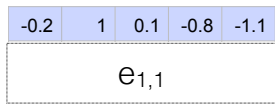
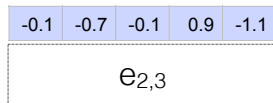
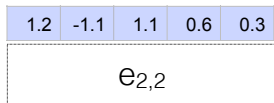
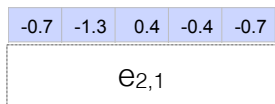
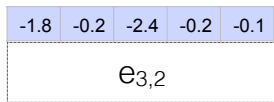
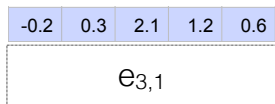
1.6	-0.3	-0.9	-0.7	0.2
$e_{1,3}$				

The

dog

barked

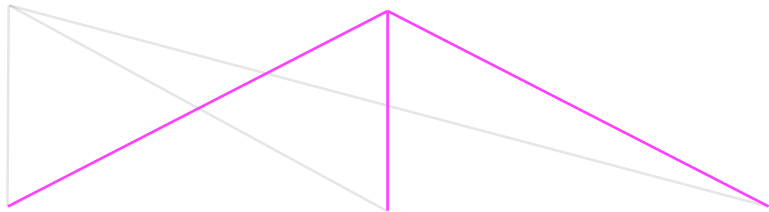




The

dog

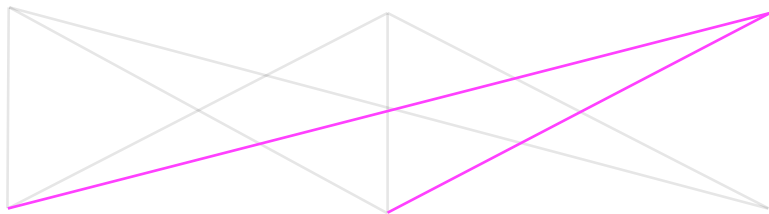
barked



-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

-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

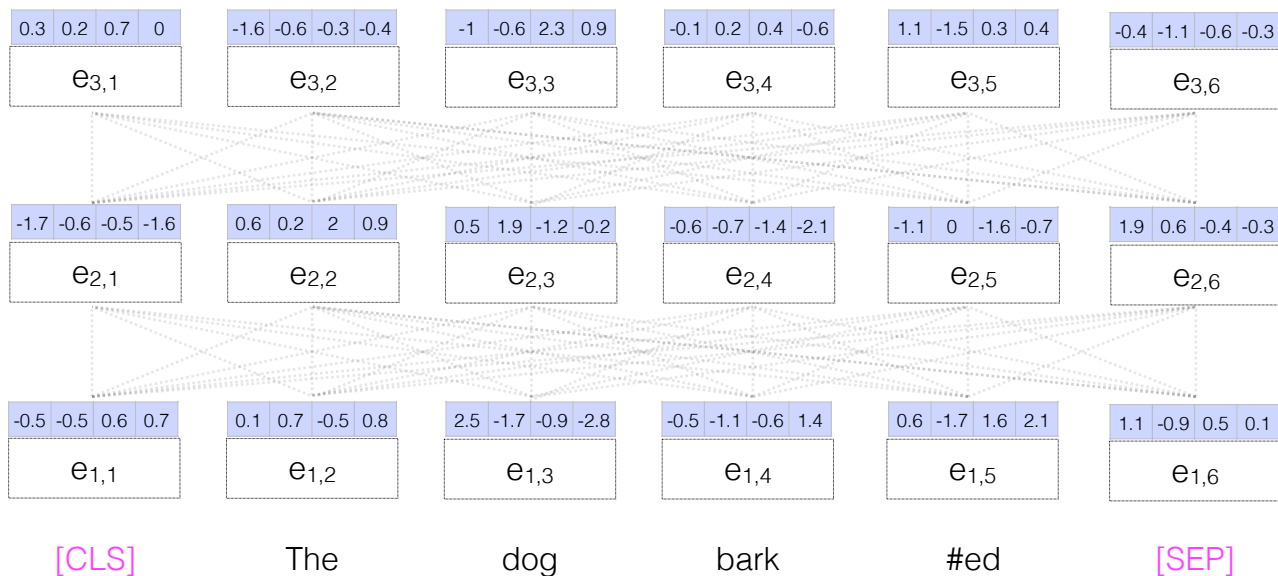
barked

WordPiece

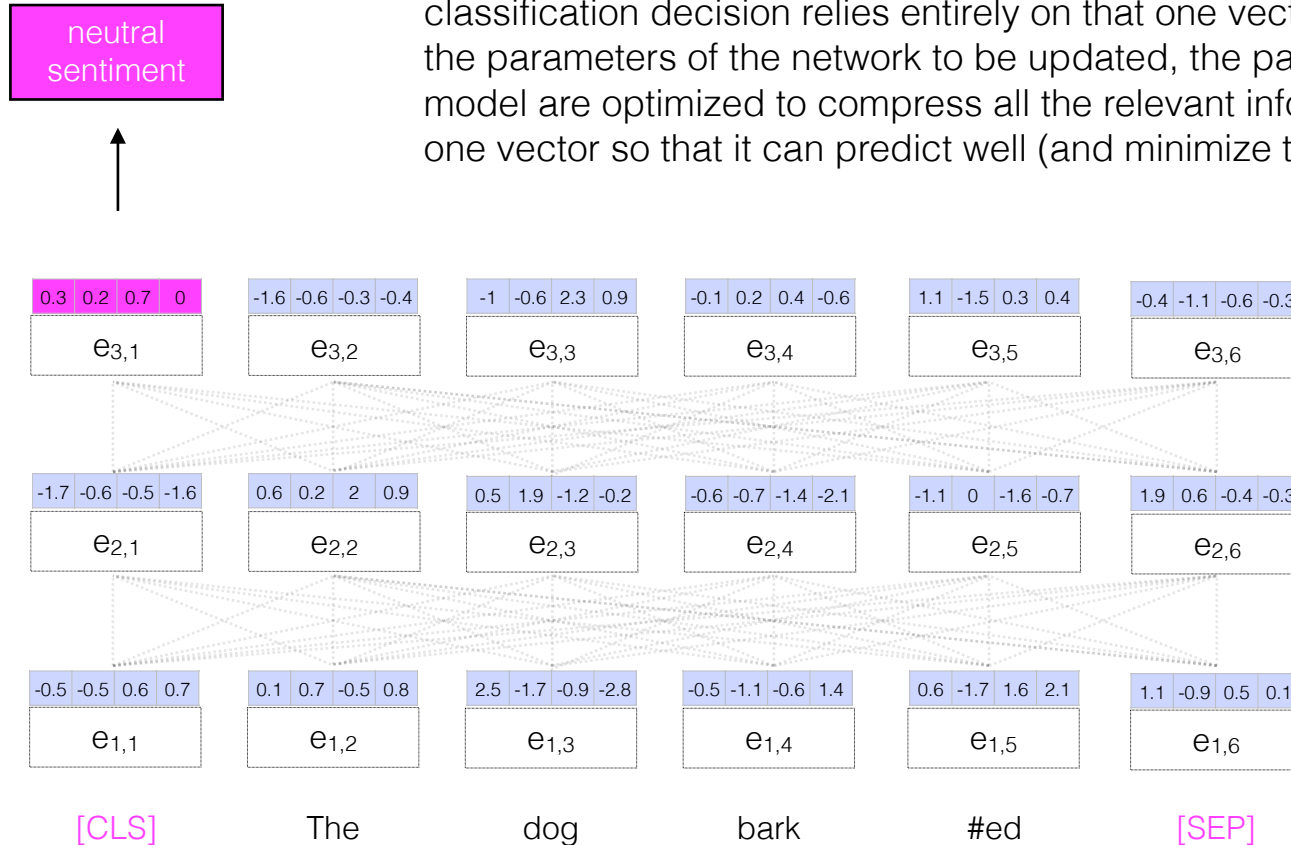
- BERT uses WordPiece tokenization, which segments some morphological structure of tokens
- Vocabulary size: 30,000

The	The
dog	dog
barked	bark #ed

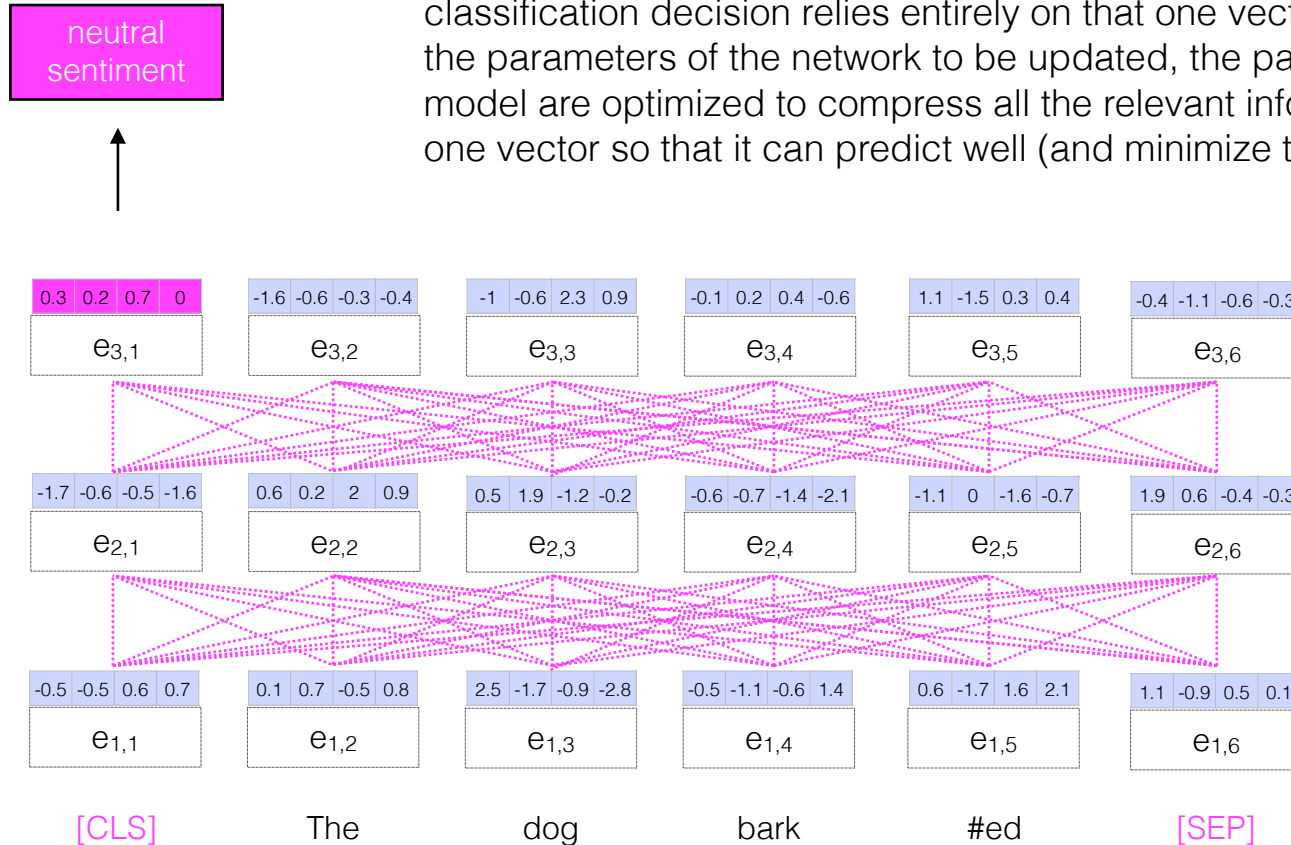
- BERT also encodes each sentence by appending a special token to the beginning ([CLS]) and end ([SEP]) of each sequence.
- This helps provides a single token that can be optimized to represent the entire sequence (e.g., for document classification)



- 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).

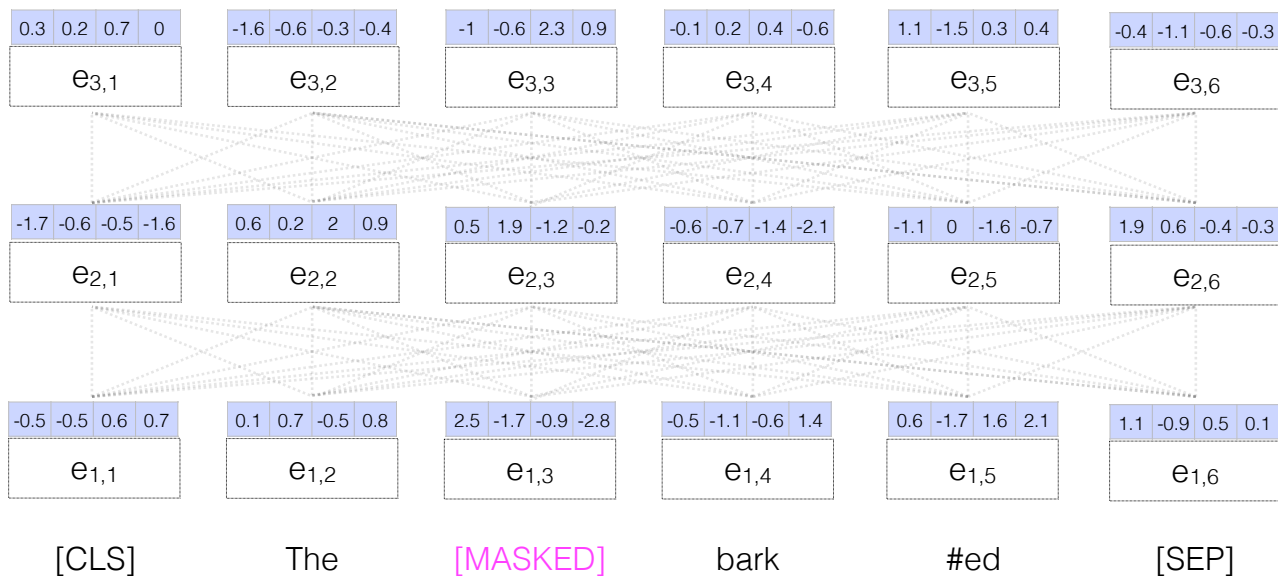


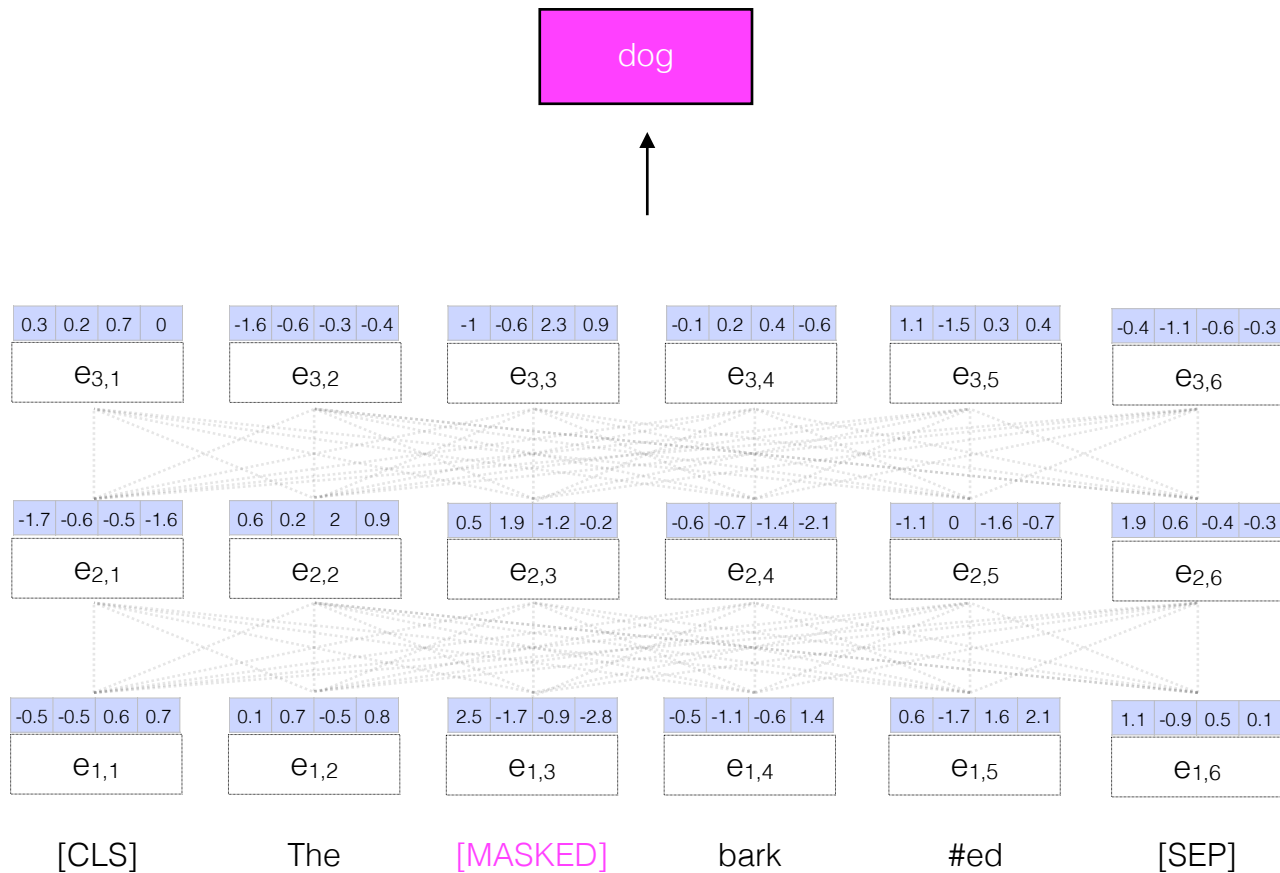
BERT

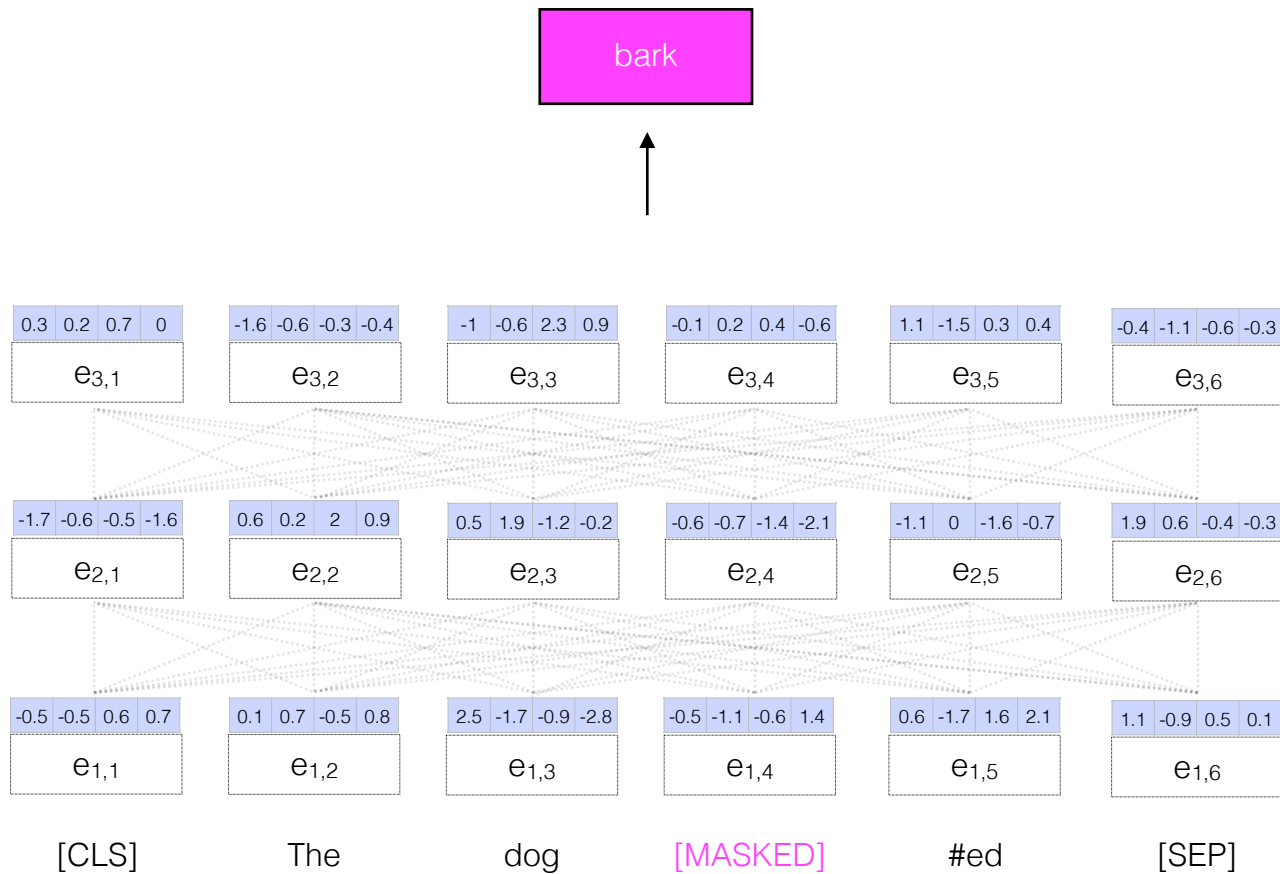
- Learn the parameters of this model with two objectives:
 - Masked language modeling
 - Next sentence prediction

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.







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!

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



52,449

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
	<code>bert-base-uncased</code>	12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.
	<code>bert-large-uncased</code>	24-layer, 1024-hidden, 16-heads, 336M parameters. Trained on lower-cased English text.

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



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](#))

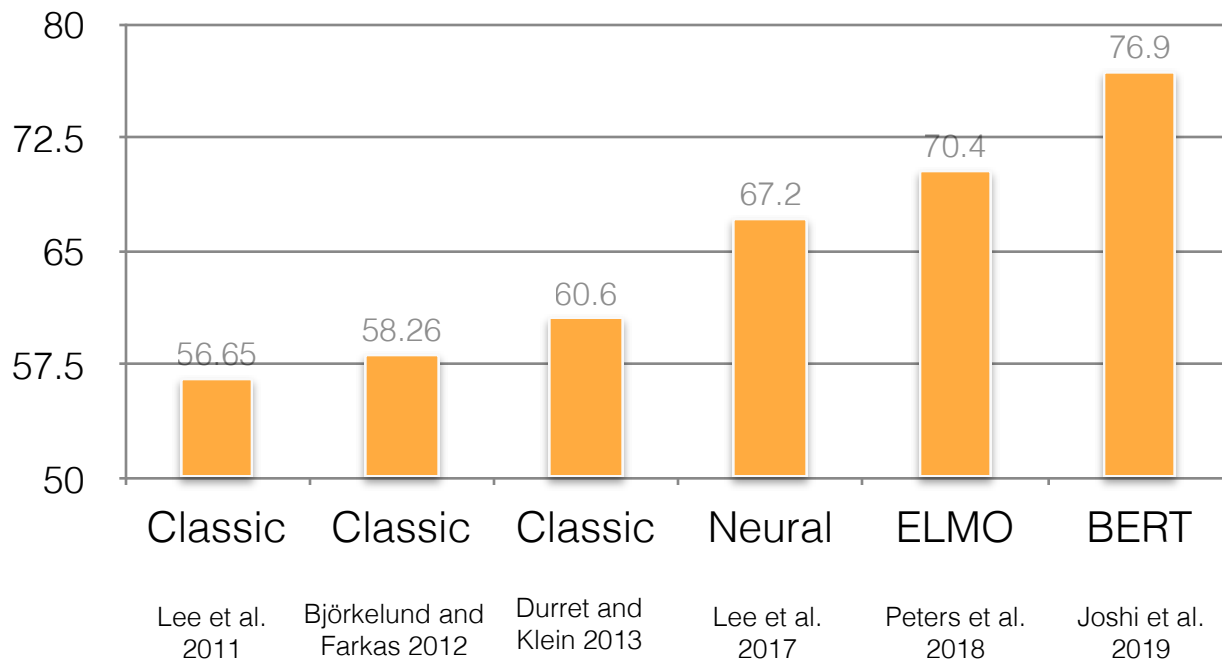
Curious which BERT model is the best for named entity recognition in Italian 🇮🇹? Just type *"Italian NER"* in the search bar!

Show 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	🔗 🔄

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>

Probing

- Even though BERT is mainly trained on a language modeling objective, it learns a lot about the structure of language — even without direct training data for specific linguistic tasks.
- Probing experiments uncover what—and where (in what layers)—pretrained BERT encodes this information.

